Available online at www.sciencedirect.com

ScienceDirect

Comput. Methods Appl. Mech. Engrg. 400 (2022) 115506

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

# A multigrid preconditioner for spatially adaptive high-order meshless method on fluid–solid interaction problems

Zisheng Ye[a], Xiaozhe Hu[b], Wenxiao Pan[a,*]

[a] Department of Mechanical Engineering, University of Wisconsin–Madison, Madison, WI 53706, USA
[b] Department of Mathematics, Tufts University, Medford, MA 02155, USA

Received 29 March 2022; received in revised form 28 July 2022; accepted 2 August 2022
Available online xxxx

**Abstract**

We present a monolithic geometric multigrid preconditioner for solving fluid–solid interaction problems in Stokes limit. The problems are discretized by a spatially adaptive high-order meshless method, the generalized moving least squares (GMLS) with adaptive $h$-refinement. For solving fluid–solid interaction problems, we need to deal with a tightly coupled system consisting of the flow field and solid bodies, resulting in a linear system of equations with a block structure. In Stokes limit, solid kinematics can be dominated by the singularities governing the lubrication effects. Resolving those singularities with adaptive $h$-refinement can lead to an ill-conditioned linear system of equations. The key ingredients of the multigrid preconditioner include the interpolation and restriction operators and the smoothers. For constructing the interpolation and restriction operators, we utilize the geometric information of hierarchical sets of GMLS nodes generated in adaptive $h$-refinement. We build decoupled smoothers through physics-based splitting and then combine them via a multiplicative overlapping Schwarz approach. Through numerical examples with the inclusion of different numbers and shapes of solid bodies, we demonstrate the performance and assess the scalability of the designed preconditioner. As the total degrees of freedom and the number of solid bodies increase, the proposed monolithic geometric multigrid preconditioner can ensure convergence and good scalability when using the Krylov iterative method for solving the linear systems of equations generated from the spatially adaptive GMLS discretization. More specifically, for a fixed number of solid bodies, as the discretization resolution is incrementally refined, the number of iterations of the linear solver can be maintained at the same level, indicating nearly linear scalability of our preconditioner with respect to the total degrees of freedom. When the number of solid bodies $N_s$ increases, the number of iterations is nearly proportional to $\sqrt{N_s}$, implying the sublinear optimality with respect to the number of solid bodies.

*Keywords:* Multigrid preconditioner; Scalability; Meshless method; Generalized moving least squares; Adaptive refinement; Fluid–solid interactions

## 1. Introduction

In many applications, we need to deal with fluid–solid interactions, where freely moving solid bodies interact with fluid flow through bidirectional hydrodynamic couplings. In Stokes limit, solutions to fluid–solid interaction problems can exhibit singular pressures in the vicinity of sharp corners and in narrow lubrication gaps between

* Corresponding author.
*E-mail address:* wpan9@wisc.edu (W. Pan).

https://doi.org/10.1016/j.cma.2022.115506
0045-7825/© 2022 Elsevier B.V. All rights reserved.

solid bodies. If the concentration of solid bodies is dense, solid kinematics can be dominated by lubrication effects. Therefore, solving fluid–solid interaction problems can be challenging especially when there are many solid bodies and the moving solids have complex and feature-rich geometries and display large displacements and rotations. A numerical method must be able to handle the tight coupling between solid kinematics and Stokes flow and to resolve the singularities governing the lubrication effects without invoking *ad hoc* lubrication models. Furthermore, as the number of solid bodies increases, the numerical solver needs to be scalable such that practical applications involving large-scale systems can be accessible in simulations. In our previous work [1], we have developed a high-order accurate numerical method for solving fluid–solid interaction problems in Stokes limit. The method draws on the generalized moving least squares (GMLS) discretization and adaptive $h$-refinement. The GMLS discretization is built upon a rigorous approximation theory [2,3] and by local weighted least square optimization can obtain optimal recovery of target functional over a desired reproduction space directly from a point cloud. Thus, the GMLS discretization does not rely on mesh connectivity between discrete nodes, and hence can circumvent the cost of generating meshes and readily handle the insertion and deletion of nodes in $h$-adaptive refinement. Its polynomial reconstruction property allows high-order accuracy achievable by choosing appropriate polynomial bases. To ensure stable solutions, the div-grad operator in Stokes equations is approximated by a staggered discretization [1,4,5]; the flexibility of GMLS enables the $\epsilon$-ball graph of neighbor connectivity to be used as a surrogate for primal/dual meshes in, e.g., staggered finite volume methods. To be compatible with the divergence-free constraint for incompressible fluid, GMLS enables the reconstruction of velocity over a space of divergence-free polynomials, which can be efficiently performed locally in a least-squares sense [1,4], contrary to mesh-based methods necessitating costly global construction of divergence-free shape functions. To achieve adaptive $h$-refinement, an *a posteriori* recovery-based error estimator is defined and employed as the criterion to direct spatial adaptivity [1]. We have demonstrated that the resultant spatially adaptive GMLS method can achieve the same order of accuracy for both the velocity and pressure fields, recover optimal convergence in the presence of singularities, and resolve the singularities governing the lubrication effects without invoking any subgrid-scale lubrication model [1]. Thus, the spatially adaptive GMLS provides an $h$-adaptive, high-order, finite difference-like discretization with a staggered scheme for pressure approximation and a divergence-free velocity reconstruction. Similarly to the reproducing kernel particle method (RKPM) [6–9] and the element free Galerkin (EFG) method [10], the GMLS seeks a consistent high-order discretization by applying least-squares reconstruction to restore polynomial consistency. However, while the former are regarded as Galerkin meshfree methods, the GMLS is a collocation meshfree method. Our previous work [1] focused on the accuracy and stability of the solutions to fluid–solid interaction problems and examined benchmark problems with a few solid bodies. In this work, we scale up the spatially adaptive GMLS discretization for solving larger-scale problems and achieve scalability with increasing numbers of solid bodies, while preserving accuracy and stability.

The linear systems of equations generated from the GMLS discretization need to be solved properly to achieve scalability and efficiency. Krylov methods are usually used for solving large-scale linear systems of equations, for which robust preconditioners must be developed. Since we are dealing with a coupled system consisting of flow field and solid bodies, and the linear system has a block structure, the preconditioner needs to be specially designed. There are mainly two categories of preconditioners for block-structured linear systems: those based on block-factorization approaches (e.g., [11–16]) and those based on monolithic multigrid (MG) methods (e.g., [17–24]). For the GMLS discretization of the Stokes equations, preconditioners based on block factorization were studied in [1,4]. The block-factorization-based approaches have shown good scalability on pure fluid problems and can handle cases with inclusion of a small number of solid bodies. However, from our in-house numerical experiments, those approaches scale badly with respect to the number of solid bodies. And their convergence become fragile with increasing numbers of solid bodies and levels of refinement. On the other hand, to the best of our knowledge, monolithic multigrid (MG) methods, especially geometric MG (GMG) methods, have not yet been developed for the GMLS discretization, although they have been developed for finite-element methods [17–24]. In a monolithic GMG method, smoothers are usually used for handling all unknowns simultaneously to deal with the block structure of a linear system. The interpolations/restrictions for each unknown are constructed based on a series of hierarchical geometric grids. Such a geometric approach leads to good performance for block-structured problems such as Stokes equations [19–21,23,24] and the fluid–solid interaction problems [25–27].

Due to its meshless nature, there are no hierarchical geometric *meshes* in the GMLS discretization, making the development of a monolithic GMG method challenging. Thus, in our previous works [1,4], the algebraic multigrid

(AMG) method was used in conjunction with the block-factorization-based preconditioners. In particular, since AMG methods are developed mainly for scalar Poisson-like problems, they are used to invert the diagonal blocks that appear in the block preconditioners. However, due to the complexity of the fluid–solid interaction problems and the limitation of block-factorization-based preconditioners, such AMG-based preconditioning strategies cannot ensure convergence and achieve scalability. Therefore, in this work, we propose to use the geometric information of the adaptively refined GMLS nodes to build the hierarchical structure and develop a monolithic GMG-based preconditioner for solving the linear systems arising from the GMLS discretization.

The two main ingredients of an MG method are: (1) the interpolation/restriction operators that allow transferring the velocity and pressure values between different resolution levels, and (2) the smoothers that can efficiently damp the high-frequency error components on each resolution level. The interplay between these two ingredients determines the performance of a MG method. We propose a monolithic MG method in the geometric setting in this work. Although we do not have hierarchical meshes, the iterative adaptive *h*-refinement process provides us with hierarchical sets of GMLS nodes whose geometric information can be used to construct the interpolation and restriction operators. More specifically, taking the fine-level nodes as the target nodes and the coarse-level nodes as the source, we use the local GMLS approximations from the source to the target to construct the interpolation operator. In addition, the divergence-free property of velocity, which usually plays an essential role in designing efficient MG methods for solving Stokes equations, is preserved by the GMLS approximations in the interpolation operator. On the other hand, the restriction operator is constructed by an averaging approach, i.e., the value of a (parent) GMLS node with the coarse-level resolution is the averaged value of its child nodes (with one level finer resolution) generated from a new iteration of adaptive refinement.

For the smoothers, we build decoupled smoothers through physics-based splitting; namely, we handle the fluid domain and the solid bodies separately. For the fluid domain, the corresponding submatrix has a block structure due to the two field variables, velocity and pressure. We use a node-wise block Gauss–Seidel smoother to handle the coupling between the two variables. The smoother for the solid bodies handles each solid body separately. For each solid body, due to the coupling between the solid body and nearby fluids, we assemble the submatrix corresponding to the solid body *and* its neighboring GMLS nodes in the fluid domain. Such a submatrix also has a block structure but is relatively small. So we use a Schur complement approach to invert it approximately based on the block splitting between the fluid and solid degrees of freedoms (DOFs). Finally, the smoother for the fluid domain and the smoother for the solid bodies are combined through a two-stage approach, which can be considered as a multiplicative overlapping Schwarz method.

Built on the proposed interpolation and restriction operators and smoothers, we develop a monolithic GMG preconditioner for using the Krylov iterative method, e.g., GMRES, to solve the linear systems of equations generated from the GMLS discretization. With the help of the geometric information of the GMLS nodes, the proposed monolithic GMG preconditioner ensures convergence for complex fluid–solid interaction problems and achieves scalability. Specifically, according to our numerical experiments, for a fixed number of solid bodies, the number of iterations maintains in the same level as we incrementally refine the discretization resolution, which indicates that our preconditioner scales nearly linearly with respect to the number of total DOFs. In addition, when the number of solid bodies $N_s$ increases, the number of iterations is nearly proportional to $\sqrt{N_s}$, which implies the sublinear optimality with respect to the number of solid bodies. Furthermore, the parallel implementation of the proposed monolithic GMG preconditioner is developed by leveraging PETSc [28]. A series of numerical tests are designed and performed to demonstrate the parallel scalability of our proposed preconditioner for the inclusion of different shapes of solid bodies. In particular, our parallel implementation achieves weak scalability for both pure fluid flow and fluid–solid interaction problems. Moreover, for the problem in an artificial vascular network, the numerical results are consistent with the strong scalability predicted by Amdahl's law up to 240 cores, which demonstrates the efficiency of our parallel implementation.

The rest of the paper is organized as follows. Section 2 provides the necessary preliminaries on the governing equations of the fluid–solid interaction problems considered and the spatially adaptive GMLS discretization. In Section 3, we present the proposed monolithic GMG method for solving the linear system generated from the GMLS discretization. We describe the parallel implementation in Section 4. Our numerical tests and results are discussed in Section 5. Finally, we summarize our main contributions and results and suggest directions of future work in Section 6.

## 2. Preliminary

### 2.1. Governing equations

We consider the coupled dynamics of steady Stokes flow of incompressible fluid and freely moving solid bodies suspended in the fluid. Hence, the computational domain contains the fluid and $N_s$ solid bodies. Denote $\Omega_f \subset \mathbb{R}^d$ as the sub-domain occupied by the fluid, $\Gamma_0$ as the outer wall boundary of the whole domain, and $\Gamma_n, n = 1, 2, \dots N_s$ as the boundaries of $N_s$ freely moving solids in $d$-dimensional physical space. Assume $\Gamma_n \cap \Gamma_m = \emptyset, m, n = 0, 1, 2, \dots N_s$ always hold during the whole dynamics process, i.e., there is finite separation between any two boundaries. Each solid body is assumed to follow rigid-body dynamics, and its motion is tracked by its center of mass (COM) position $\mathbf{X}_n$ and orientation $\boldsymbol{\Theta}_n$. Its translational and angular velocities are accordingly donated as $\dot{\mathbf{X}}_n$ and $\dot{\boldsymbol{\Theta}}_n$, respectively. The fluid flow is coupled to the motions of all solid bodies through the following steady Stokes problem:

$$\begin{cases} \dfrac{\nabla p}{\rho} - \nu \nabla^2 \mathbf{u} = \mathbf{f} & \forall \mathbf{x} \in \Omega_f \\ \nabla \cdot \mathbf{u} = 0 & \forall \mathbf{x} \in \Omega_f \\ \mathbf{u} = \mathbf{w} & \forall \mathbf{x} \in \Gamma_0 \\ \mathbf{u} = \dot{\mathbf{X}}_n + \dot{\boldsymbol{\Theta}}_n \times (\mathbf{x} - \mathbf{X}_n) & \forall \mathbf{x} \in \Gamma_n, n = 1, \dots, N_s \\ \mathbf{n} \cdot \dfrac{\nabla p}{\rho} - \nu \mathbf{n} \cdot \nabla^2 \mathbf{u} = \mathbf{n} \cdot \mathbf{f} & \forall \mathbf{x} \in \Gamma = \Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cdots \cup \Gamma_{N_s} , \end{cases} \tag{1}$$

where $\nu$ is the kinematic viscosity of fluid; $\rho$ is the density of the fluid; $\mathbf{f}$ denotes the body force exerted on the fluid; $\mathbf{w}$ is the velocity of the wall boundary; and $\mathbf{n}$ is the unit normal vector outward facing at boundary $\Gamma$.

For a divergence-free velocity field ($\nabla \cdot \mathbf{u} = 0$), Eq. (1) is equivalent to:

$$\begin{cases} \dfrac{\nabla p}{\rho} + \nu \nabla \times \nabla \times \mathbf{u} = \mathbf{f} & \forall \mathbf{x} \in \Omega_f \\ -\dfrac{\nabla^2 p}{\rho} = -\nabla \cdot \mathbf{f} & \forall \mathbf{x} \in \Omega_f \\ \mathbf{u} = \mathbf{w} & \forall \mathbf{x} \in \Gamma_0 \\ \mathbf{u} = \dot{\mathbf{X}}_n + \dot{\boldsymbol{\Theta}}_n \times (\mathbf{x} - \mathbf{X}_n) & \forall \mathbf{x} \in \Gamma_n, n = 1, \dots, N_s \\ \mathbf{n} \cdot \dfrac{\nabla p}{\rho} + \nu \mathbf{n} \cdot \nabla \times \nabla \times \mathbf{u} = \mathbf{n} \cdot \mathbf{f} & \forall \mathbf{x} \in \Gamma = \Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cdots \cup \Gamma_{N_s} , \end{cases} \tag{2}$$

where we have utilized the vector identity $\nabla^2 \mathbf{u} = -\nabla \times \nabla \times \mathbf{u}$ for $\nabla \cdot \mathbf{u} = 0$. Solving Eq. (2) instead of Eq. (1) can avoid the saddle-point structure of the Stokes operator. However, it necessitates a numerical reconstruction of velocity faithful to the divergence-free constraint.

In Stokes limit, the inertia effect of solids can be neglected. Hence, each solid body's translational and angular motions are subject to the force-free and torque-free constraints as:

$$\begin{cases} \displaystyle\int_{\Gamma_n} \boldsymbol{\sigma} \cdot \mathrm{d}\boldsymbol{\mathcal{A}} + \mathbf{f}_{e,n} = \mathbf{0} \\ \displaystyle\int_{\Gamma_n} (\mathbf{x} - \mathbf{X}_n) \times (\boldsymbol{\sigma} \cdot \mathrm{d}\boldsymbol{\mathcal{A}}) + \boldsymbol{\tau}_{e,n} = \mathbf{0} , \end{cases} \tag{3}$$

where $n = 1, \dots, N_s$; $\boldsymbol{\sigma} = -p\mathbf{I} + \mu[\nabla \mathbf{u} + (\nabla \mathbf{u})^{\intercal}]$ is the stress exerted by the fluid on the boundary of each solid body; $\mathbf{f}_{e,n}$ and $\boldsymbol{\tau}_{e,n}$ represent the external force and torque applied on each solid body, respectively.

Therefore, by solving Eqs. (2) and (3) concurrently as a monolithic system, we obtain each solid's translational and angular velocities $\{\dot{\mathbf{X}}_n, \dot{\boldsymbol{\Theta}}_n\}_{n=1,2,\dots,N_s}$ as well as the fluid's velocity ($\mathbf{u}$) and pressure ($p$) fields at each instant time. Given $\{\dot{\mathbf{X}}_n, \dot{\boldsymbol{\Theta}}_n\}_{n=1,2,\dots,N_s}$ and the initial value of $\{\mathbf{X}_n, \boldsymbol{\Theta}_n\}_{n=1,2,\dots,N_s}$, the solid bodies' new positions and orientations $\{\mathbf{X}_n, \boldsymbol{\Theta}_n\}_{n=1,2,\dots,N_s}$ are updated by invoking a temporal integrator. In this work, we adopt the 5th order Runge–Kutta integrator with adaptive time stepping [29], also known as ODE45 [30], and implement it in our solver. To achieve adaptive time stepping, this integrator needs to specify an initial time step and relative error tolerance, which are set as $\Delta t_0 = 0.2$ s and $10^{-5}$, respectively, in this work.

## 2.2. Spatially adaptive GMLS discretization

For numerically solving Eqs. (2) and (3), we employ the spatially adaptive GMLS method, developed in our previous work [1]. The linear system resulting from the adaptive GMLS discretization is solved by the proposed scalable linear solver with multigrid preconditioner. Before we introduce the proposed linear solver in Section 3, we briefly review the key ingredients of the adaptive GMLS method in this section, including the GMLS approximation, the divergence-free GMLS reconstruction for the velocity field, the staggered discretization of the div-grad operator, and the adaptive refinement scheme. The improvements beyond our previous work [1] are also discussed.

### 2.2.1. GMLS approximation

The fluid domain $\Omega_f$ is discretized by a set of collocation points (referred to as interior GMLS nodes). All the boundaries $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cdots \cup \Gamma_{N_s}$ are discretized by another set of discrete points (referred to as boundary GMLS nodes), where the boundary conditions (BCs) are imposed. For a GMLS node at $\mathbf{x}_i$ and a given scalar function $\psi$ evaluated at its neighbor domain: $\psi_j = \psi(\mathbf{x}_j)$, a polynomial $\psi_h(\mathbf{x})$ of order $m$ is sought to approximate $\psi$ and its derivatives $(D^\alpha \psi)$ at $\mathbf{x}_i$. That is, $\psi_h(\mathbf{x}) = \mathbf{P}^\mathsf{T}(\mathbf{x})\mathbf{c}^*$ with a polynomial basis $\mathbf{P}(\mathbf{x})$ and coefficient vector $\mathbf{c}^*$ such that the following weighted residual functional is minimized:

$$J(\mathbf{x}_i) = \sum_{j \in \mathcal{N}_{\epsilon_i}} \left[\psi_j - \mathbf{P}_i^\mathsf{T}(\mathbf{x}_j)\mathbf{c}_i\right]^2 W_{ij} \ . \tag{4}$$

For this quadratic programming optimization problem, its solution can be easily given as:

$$\mathbf{c}_i^* = \left(\sum_{k \in \mathcal{N}_{\epsilon_i}} \mathbf{P}_i(\mathbf{x}_k) W_{ik} \mathbf{P}_i^\mathsf{T}(\mathbf{x}_k)\right)^{-1} \left(\sum_{j \in \mathcal{N}_{\epsilon_i}} \mathbf{P}_i^\mathsf{T}(\mathbf{x}_j) W_{ij} \psi_j\right) . \tag{5}$$

Note that for $\psi \in \operatorname{span}(\mathbf{P}(\mathbf{x}))$, this approximation ensures $\psi$ to be exactly reconstructed. This property of polynomial reconstruction grants high-order accuracy achievable for the GMLS approximation by taking large $m$, e.g. $m = 4, 6$ [2]. An arbitrary $\alpha$th order derivative of the function can then be approximated by:

$$D^\alpha \psi(\mathbf{x}) \approx D_h^\alpha \psi_h(\mathbf{x}) := (D^\alpha \mathbf{P}(\mathbf{X}))^\mathsf{T} \mathbf{c}^*. \tag{6}$$

The weight function involved in Eqs. (4)–(5) is defined as $W_{ij} = W(r_{ij})$ with $r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ and $W(r) = 1 - \left(\frac{r}{\epsilon}\right)^4$ for $r < \epsilon$, or otherwise, $W(r) = 0$. Here, $\epsilon$ is the compact support, and hence, it is only necessary to include GMLS nodes within an $\epsilon$-neighborhood of the $i$th GMLS node, i.e., $j \in \mathcal{N}_{\epsilon_i} = \{\mathbf{x}_j \text{ s.t. } \|\mathbf{x}_i - \mathbf{x}_j\| < \epsilon_i\}$. Therefore, the approximation errors in both the interpolant and derivatives of $\psi$ depend on the value of $\epsilon$, or on the normalized support size $\epsilon/\Delta x$ if the characteristic nodal distance $\Delta x$ within the support is fixed. $\epsilon/\Delta x$ must be large enough to ensure unisolvency over the reconstruction space and thereby a well-posed solution to Eq. (4). Larger $\epsilon/\Delta x$ also leads to a more accurate least-square approximation. However, too large $\epsilon/\Delta x$ can yield very dense linear operators and ill-conditioned linear systems. Denote $N_{m,p}$ the minimal number of neighbors required to solve Eq. (4) for the polynomial basis $\mathbf{P}(\mathbf{x})$ of order $m$. In practice, to ensure a well-posed and accurate solution of Eq. (4), we determine the size of compact support $\epsilon_i$ for each node by taking the minimum value of $\epsilon_i/\Delta x_i$ such that $|\mathcal{N}_{\epsilon_i}| \geq 2^{d/2} N_{m,p}$, where $|\mathcal{N}_{\epsilon_i}|$ denotes the number of nodes in the set $\mathcal{N}_{\epsilon_i}$; $d$ represents the physical dimension.

### 2.2.2. Divergence-free GMLS reconstruction for the velocity field

We apply the GMLS approximation discussed above to both the velocity and pressure fields. However, the polynomial basis used to approximate the velocity field $\mathbf{u}$ is chosen from the space of $m$th order divergence-free vector polynomials $\mathbf{P}^{\text{div}}(\mathbf{x})$ to enforce compatibility with the divergence-free constraint on the velocity. Thus, the following polynomial reconstruction is built for $\mathbf{u}$ and for discretizing its gradient and curl–curl operators:

$$\begin{aligned} &\mathbf{u}_h(\mathbf{x}_i) = (\mathbf{P}_i^{\text{div}})^\mathsf{T}(\mathbf{x}_i)\mathbf{c}_i^{\text{div}*} \qquad \text{with } \mathbf{c}_i^{\text{div}*} = \mathbf{M}_i^{\text{div}^{-1}} \left(\sum_{j \in \mathcal{N}_{\epsilon_i}} \mathbf{P}_i^{\text{div}}(\mathbf{x}_j)\mathbf{u}(\mathbf{x}_j) W_{ij}\right) , \\ &\nabla_h \mathbf{u}_h(\mathbf{x}_i) = (\nabla \mathbf{P}_i^{\text{div}})^\mathsf{T}(\mathbf{x}_i)\mathbf{c}_i^{\text{div}*} , \quad (\nabla \times \nabla \times)_h \mathbf{u}_h(\mathbf{x}_i) = (\nabla \times \nabla \times \mathbf{P}_i^{\text{div}})^\mathsf{T}(\mathbf{x}_i)\mathbf{c}_i^{\text{div}*} , \end{aligned} \tag{7}$$

where

$$\mathbf{M}_i^{\text{div}} = \sum_{j\in\mathcal{N}_{\epsilon_i}} \mathbf{P}_i^{\text{div}}(\mathbf{x}_j) W_{ij} (\mathbf{P}_i^{\text{div}})^{\mathsf{T}}(\mathbf{x}_j) \ .$$

The complete divergence-free polynomial basis $\mathbf{P}_i^{\text{div}}(\mathbf{x})$ of different orders $m$ in both 2D ($d = 2$) and 3D ($d = 3$) are provided in Appendix A.

To impose the Dirichlet (no-slip) BCs in Eq. (2) for $\mathbf{u}$, the boundary GMLS nodes take the velocity of the corresponding boundary $\Gamma_n$ that they belong to.

### 2.2.3. Staggered discretization of div-grad operator

To ensure compatibility and thereby numerical stability, we employ the staggered GMLS discretization [1,5] for the div-grad operator of pressure $p$ in Eq. (2). It builds upon each $\epsilon$-neighborhood graph that plays as a local primal–dual complex for each GMLS node. We refer to it as virtual cell, where a set of primal edges are constructed as: $\mathbf{E}_i = \{\mathbf{x}_i - \mathbf{x}_j | \mathbf{x}_j \in \mathcal{N}_{\epsilon_i}\}$, each associated with a midpoint $\mathbf{x}_{ij} = \frac{\mathbf{x}_i+\mathbf{x}_j}{2}$ and a virtual dual face at the midpoint and normal to the edge. The staggered GMLS discretization of the div-grad operator is then constructed from a topological gradient over primal edges and a GMLS divergence recovered from local virtual dual faces. Thus, we have:

$$\begin{aligned} &p_h(\mathbf{x}_i) = \mathbf{P}_i^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_i^* \ , \qquad \nabla_h p_h(\mathbf{x}_i) = \frac{1}{2}(\nabla \mathbf{P}_i)^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_i^* \ , \qquad \nabla_h^2 p_h(\mathbf{x}_i) = \frac{1}{4}(\nabla^2 \mathbf{P}_i)^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_i^* \\ &\text{with } \mathbf{c}_i^* = \left(\sum_{j\in\mathcal{N}_{\epsilon_i}} \mathbf{P}_i(\mathbf{x}_{ij}) W_{ij} \mathbf{P}_i^{\mathsf{T}}(\mathbf{x}_{ij})\right)^{-1} \left(\sum_{j\in\mathcal{N}_{\epsilon_i}} \mathbf{P}_i(\mathbf{x}_{ij}) W_{ij} (p_i - p_j)\right) \ , \end{aligned} \tag{8}$$

where the polynomial basis $\mathbf{P}$ can be the $\epsilon_i$-scaled Taylor $m$th order monomials; and the coefficient $\mathbf{c}_i^*$ is the solution of the quadratic program:

$$\mathbf{c}_i(\mathbf{x}) = \arg\min_{\mathbf{c}_i} \left\{ \sum_{j\in\mathcal{N}_{\epsilon_i}} \left[(p_i - p_j) - \mathbf{P}_i^{\mathsf{T}}(\mathbf{x}_{ij})\mathbf{c}_i\right]^2 W_{ij} \right\} . \tag{9}$$

Note that pressure $p$ is subject to an inhomogeneous Neumann BC in Eq. (2), written in the form of $\partial_{\mathbf{n}}|_{\Gamma} = g$. To impose this BC, we can add to the quadratic program in Eq. (9) for the boundary GMLS nodes the following equality constraint:

$$\mathbf{n}_i \cdot \left[\frac{1}{2}(\nabla \mathbf{P}_i)^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_i^*\right] = g(\mathbf{x}_i) \ , \qquad \mathbf{x}_i \in \Gamma \ .$$

In addition, a zero-mean constraint is imposed for $p$ to ensure the uniqueness and physical consistency of the solution. Different from our previous work [1], we do not employ a Lagrangian multiplier for enforcing this zero-mean constraint in order to achieve scalability. Details are explained in Section 4.4.

### 2.2.4. Adaptive refinement

In order to reduce computational cost and to recover optimal convergence in the presence of singularities governing lubrication effects, the computational domain and boundaries are discretized with a hierarchy of different resolutions based on an adaptive refinement algorithm [1].

At each time step, we start from a uniform, coarse initial discretization resolution $\Delta x_i = \Delta x^0$, resulting in a set of GMLS nodes, $\Omega^0$. Then, more GMLS nodes are added adaptively with refined $\Delta x_i$, leading to new sets $\Omega^I$ of GMLS nodes. In each refinement iteration, a four-stage procedure is followed:

$$\textbf{SOLVE} \rightarrow \textbf{ESTIMATE} \rightarrow \textbf{MARK} \rightarrow \textbf{REFINE} \ . \tag{10}$$

In **SOLVE** stage, Eq. (2) is numerically solved using the GMLS discretization discussed in Section 2.2 and the linear solver introduced in Section 3; both the velocity and pressure fields are updated. In **ESTIMATE** stage, the recovery-based *a posteriori* error estimator $\eta_i^r$ is evaluated from Eq. (12) for all GMLS nodes. According to the value of the error estimator on each node, we sort all nodes in descending order into a new sequence $\mathbf{x}_{i'}$. In **MARK** stage, a fraction of GMLS nodes with largest errors are selected and marked for refining. This fraction of GMLS nodes contribute to $\alpha$ percentage of the total recovered error, where $\alpha$ is a preset parameter and $\alpha = 0.8$ in the present work. In **REFINE** stage, the marked GMLS nodes are refined. For solving 2D problems, generally any marked interior GMLS node $\mathbf{x}_i \in \Omega_f$ is split into four nodes with smaller spacing, and a marked boundary node $\mathbf{x}_i \in \Gamma$ is split into two nodes. For solving 3D problems, any marked interior GMLS node $\mathbf{x}_i \in \Omega_f$ is split into eight nodes with smaller spacing; a marked boundary node on the wall boundary $\mathbf{x}_i \in \Gamma_0$ is split into four nodes; and the details about how to refine any marked boundary node on the surface of a moving solid $\mathbf{x}_i \in \Gamma_n$ $(n = 1, 2, \ldots, N_s)$ are provided in Appendix B. Denote $\Delta x_i$ as the original spacing (or resolution) of the marked nodes, the nodes newly generated by splitting have the spacing $\Delta x_{i_{new}} = \frac{1}{2}\Delta x_i$. The updated set of GMLS nodes at the end of **REFINE** stage is denoted as $\Omega^{I+1}$. These four stages (10) are repeated iteratively at each time step until the total recovered error (Eq. (13)) is less than a preset tolerance, i.e.,

$$\eta^r \leq \varepsilon_{tol} \ . \tag{11}$$

The recovery-based *a posteriori* error estimator is defined based on velocity gradient. Due to the lack of exact solutions in practical applications, true errors cannot be evaluated. Thus, we use the *recovered* error to estimate the true error, which measures the difference between the direct and recovered velocity gradients and can be practically determined in any application problem. More specifically, it is defined as:

$$\eta_i^r = \frac{\sum_{j\in\mathcal{N}_{\epsilon_i}} \|\mathbf{R}[\nabla\mathbf{u}]_j - \nabla\mathbf{u}_{h,i\to j}\|^2 V_j}{\sum_{j\in\mathcal{N}_{\epsilon_i}} V_j} \ . \tag{12}$$

Here, the recovered velocity gradient can be evaluated locally on each GMLS node as:

$$\mathbf{R}[\nabla\mathbf{u}]_i = \frac{1}{N_i}\sum_{j\in\mathcal{N}_{\epsilon_i}} \nabla_h\mathbf{u}_{h,j\to i} \ ,$$

where $\nabla_h\mathbf{u}_{h,j\to i} = (\nabla\mathbf{P}_j^{\text{div}})^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_j^*$ is the velocity gradient reconstructed at $\mathbf{x}_j$ but evaluated at $\mathbf{x}_i$. In order to properly weigh the contributions from nodes with different discretization resolutions $\Delta\mathbf{x}_i$, a volumetric weight $V_i$ is assigned to each node at $\mathbf{x}_i$ and defined as $V_i = \Delta x_i^d$, where $d$ is the dimension of the problem's physical space. Then, the total recovered error over all GMLS nodes is:

$$\eta^r = \frac{\sum_{\mathbf{x}_i\in\Omega} \eta_i^r V_i}{\sum_{\mathbf{x}_i\in\Omega} \|\nabla\mathbf{u}_{h,i}\|^2 V_i} \ . \tag{13}$$

For applications of dilute suspensions of solids, the above adaptive refinement procedure can start from an uniform, coarse initial discretization. However, for concentrate suspensions of solids, or when gaps between some solid boundaries are rather narrow, even smaller than $\Delta x^0$, the initial coarse discretization can result in none GMLS nodes allocated within the gaps. In that case, adaptive refinement would not take place within those gaps. Thus, we invoke a preprocessing step to guarantee GMLS nodes allocated everywhere throughout the entire computational domain including narrow gaps, before we conduct adaptive refinement following the algorithm discussed above.

In this preprocessing step, we examine the $\epsilon$-neighborhood of each boundary GMLS node and check if the following requirement is satisfied:

$$\text{For } \mathbf{x}_i \in \Gamma_n \text{ and } \forall j \in \mathcal{N}_{\epsilon_i}, \quad \mathbf{x}_j \notin \Gamma_m \quad \text{with} \quad m \neq n \text{ and } m, n = 0, 1, \ldots, N_s \ . \tag{14}$$

This requirement ensures that each boundary GMLS node has enough interior GMLS nodes in its $\epsilon$-neighborhood. If it is not satisfied, the corresponding boundary node and all its neighbor nodes are refined. Note that in this preprocessing step, the neighbor nodes of a boundary node can be within solid bodies.

After preprocessing and any iteration of refinement, we perform a post-processing step to enforce quasi-uniform discretization within any $\epsilon$-neighborhood $\mathcal{N}_{\epsilon_i}$, because large difference in the discretization resolution within an $\epsilon$-neighborhood would result in ill-conditioned GMLS approximation. Thus, for any $\epsilon$-neighborhood that does not satisfy:

$$\frac{\max(\Delta x_j)}{\min(\Delta x_k)} \leq 2 \ , \quad \text{for } \forall\mathbf{x}_i \in \Omega_f \cup \Gamma \quad \text{and} \quad j, k \in \mathcal{N}_{\epsilon_i} \ , \tag{15}$$

we will mark and refine the nodes with the coarsest resolution in that $\epsilon$-neighborhood, until the requirement in Eq. (15) is satisfied.

### 2.2.5. Numerical quadrature

With the GMLS discretization, a composite quadrature rule is employed for approximating the integrals in Eq. (3) as:

$$\begin{aligned}\int_{\Gamma_n} \boldsymbol{\sigma} \cdot \mathrm{d}\boldsymbol{\mathcal{A}} &\approx \sum_{\mathbf{x}_i \in \Gamma_n} (-p_i \mathbf{I} + \nu[\nabla_h \mathbf{u}_{h,i} + (\nabla_h \mathbf{u}_{h,i})^{\mathsf{T}}]) \cdot (\Delta \mathcal{A}_i \mathbf{n}_i) \\ \int_{\Gamma_n} (\mathbf{x} - \mathbf{X}_n) \times (\boldsymbol{\sigma} \cdot \mathrm{d}\boldsymbol{\mathcal{A}}) &\approx \sum_{\mathbf{x}_i \in \Gamma_n} \left((\mathbf{x}_i - \mathbf{X}_n) \times (-p_i \mathbf{I} + \nu[\nabla_h \mathbf{u}_{h,i} + (\nabla_h \mathbf{u}_{h,i})^{\mathsf{T}}])\right) \cdot (\Delta \mathcal{A}_i \mathbf{n}_i) \ ,\end{aligned} \tag{16}$$

where $\nabla_h \mathbf{u}_{h,i} = (\nabla \mathbf{P}_i^{\mathrm{div}})^{\mathsf{T}}(\mathbf{x}_i)\mathbf{c}_i^*$; $\Delta \mathcal{A}_i = \Delta x_i$ for 2D problems, and how to determine $\Delta \mathcal{A}_i$ in 3D is provided in Appendix B. As shown in [1], this quadrature rule is sufficient for the entire numerical method to achieve high-order convergence.

## 3. Linear solver with multigrid preconditioner

Solving the linear systems resulting from discretization dominates the entire computational cost in simulations. Therefore, the main focus of this work is to design a scalable preconditioner for the Krylov method to solve the linear systems generated in the **SOLVE** stage. A simple preconditioner, such as the Gauss–Seidel preconditioner, cannot ensure convergence and is not effective in practice. In the previous work [1,4], a block-factorization-based preconditioner based on decoupling of the velocity and pressure fields was employed, and AMG methods were applied for inverting each block. For solving benchmark smaller-scale fluid–solid interaction problems [1,4], such a linear solver can work reasonably well. However, as the number of solid bodies increases, e.g., in the examples considered in the present work, the performance of such a block preconditioner deteriorates. One major reason is that the velocity block used in the block-factorization-based preconditioner consists of unknowns related to both the fluid and solid bodies. In particular, it combines the discretized curl–curl operator in Eq. (2) for the fluid with the discretized integrals in Eq. (16) for the solid bodies. In addition, the inclusion of many solid bodies results in a very irregular shape for the overall computational domain. All of these make it difficult for an AMG method to find a proper coarsening. Consequently, it fails to converge for the velocity block and strongly affects the overall performance of the block-factorization-based preconditioner.

This section introduces a monolithic GMG method to build a preconditioner for the Krylov linear solver such as GMRES to solve the linear systems resulted from GMLS discretization. The new multigrid method utilizes the hierarchical sets of GMLS nodes generated during the adaptive refinement. Therefore, no coarsening is needed, avoiding the major difficulty that the AMG method previously encountered. Using the geometric information of these hierarchical sets of the GMLS nodes, we can construct interpolation operators based on the GMLS approximation and restriction operators based on averaging. Furthermore, a two-stage smoother is developed based on the splitting between the fluid domain and the solid bodies. Finally, the interpolation/restriction operators and the smoother are used in a V-cycle fashion to define the overall monolithic GMG preconditioner.

We first discuss about the block structure of the resultant linear system in Section 3.1. Then, we briefly review the required operations and entire process of multigrid preconditioning in Section 3.2. In Section 3.3, we explain how to construct the interpolation and restriction operators. In Section 3.4, we present the smoothers designed based on physics splitting. The entire monolithic GMG method introduced in this work is finally summarized in Section 3.5.

### 3.1. Block structure of the linear system

After the governing Eqs. (2)–(3) are discretized by the GMLS discretization discussed in Section 2.2, the resulting linear system has the following block structure:

$$\mathbf{A}\boldsymbol{\chi} = \mathbf{y} \quad \text{with} \quad \mathbf{A} = \begin{bmatrix} \mathbf{K} & \mathbf{G} & \mathbf{C} \\ \mathbf{B} & \mathbf{L} & \\ \mathbf{D} & \mathbf{T} & \end{bmatrix}, \quad \boldsymbol{\chi} = \begin{bmatrix} \mathbf{u} \\ p \\ \dot{\mathbf{X}} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{f}_{tot} \\ g \\ \mathbf{f}_s \end{bmatrix}. \tag{17}$$

Here, $\dot{\mathbf{X}}$ includes all solids' both translational and rotational velocities, i.e., $\dot{\mathbf{X}} = \{\dot{\mathbf{X}}_n, \dot{\Theta}_n\}_{n=1,2,\ldots,N_s}$. $\mathbf{K}$ corresponds to the discretized curl–curl operator ($\nabla \times \nabla \times$) of velocity in Eq. (2). It is noted that the curl–curl operator for a divergence-free polynomial basis is equivalent to Laplacian. $\mathbf{L}$ corresponds to the discretized Laplacian operator ($\nabla^2$) of pressure in Eq. (2), which is obtained from the staggered discretization of div-grad operator discussed in Section 2.2.3. As such, the nonzero diagonal blocks in $\mathbf{A}$, i.e., $\mathbf{K}$ and $\mathbf{L}$, are all discretized Laplacian operators. While $\mathbf{B}$ denotes the contribution from the $\nu\nabla \times \nabla\times$ operator in the inhomogeneous Neumann BC in Eq. (2), $\mathbf{G}$ represents the discretized $\frac{1}{\rho}\nabla$ operator. For the discretized force-free and torque-free constraints as in Eq. (16), $\mathbf{D}$ denotes the contribution from viscous stress with velocity gradient; $\mathbf{T}$ denotes the contribution from pressure. $\mathbf{C}$ contains the velocity constraints (no-slip BCs) from each solid's kinematics on their boundaries in Eq. (2). $\mathbf{f}_{tot}$ combines the body force $\mathbf{f}$ exerted in fluid and the velocity $\mathbf{w}$ on the wall boundary. $g$ contains $\nabla \cdot \mathbf{f}$ and $\mathbf{n} \cdot \mathbf{f}$ in Eq. (2). Finally, $\mathbf{f}_s$ represents the external force and torque applied on the solid bodies, such that $\mathbf{f}_s = [-\mathbf{f}_{e,n} \quad -\boldsymbol{\tau}_{e,n}]^{\mathsf{T}}_{n=1,2,\ldots,N_s}$.

From its block structure, we see that $\mathbf{A}$ is neither symmetric nor positive definite. The strong coupling between the fluid and solid DOFs deteriorates the conditioning of $\mathbf{A}$. Thus, the linear system in Eq. (17) is challenging to solve in practice. Furthermore, since enough neighbor nodes are needed in GMLS discretization, the matrix $\mathbf{A}$ is dense. Therefore, designing a robust and efficient linear solver is crucial for achieving scalability in practical simulations.

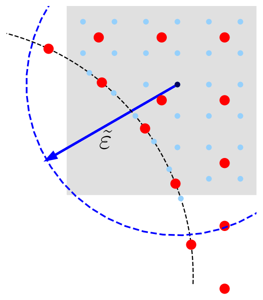
## 3.2. Multigrid methods

Most relaxation-type iterative solvers, like Gauss–Seidel (GS), converge slowly for linear systems with fine resolution discretized from partial differential equations, e.g., (2). The main reason is that the convergence rates of different error components can vary significantly, if we decompose the error using the eigenvectors of the linear system, e.g., $\mathbf{A}$ in Eq. (17). The components with the eigenvectors corresponding to large eigenvalues (in terms of magnitude) are usually referred to as high-frequency components of the error; the components with the eigenvectors corresponding to small eigenvalues are called low-frequency components of the error. In general, for relaxation-type iterative solvers, the high-frequency components converge to zero quickly independent of the discretization resolution. However, the low-frequency components converge slowly, getting worse when the resolutions are refined and hence slowing down the overall convergence. On the other hand, low-frequency components of the error on fine resolutions can be well approximated on coarse discretization and hence become high-frequency error components on coarse resolutions [31]. This fact motivates the idea of moving to a coarser resolution to eliminate the low-frequency error components and leads to a multilevel approach known as the multigrid method [18,31].

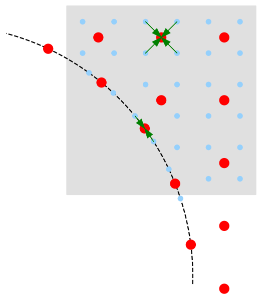
Multigrid methods exploit a discretization with different resolutions to obtain optimal convergence rate and hence are naturally compatible with adaptive $h$-refinement discussed in Section 2.2.4. The operations of going back and forth between coarse and fine resolutions are called the restriction and interpolation, respectively. To minimize the approximation errors across different resolution levels, a smoothing procedure is usually executed before the restriction operation, and another smoothing step is applied after the interpolation operation. The entire process as such is referred to as "V-cycle" [18] and is summarized in Algorithm 1. In the present work, we follow this V-cycle for building our multigrid preconditioner.

**Algorithm 1** V-cycle process of multigrid preconditioning: V-Cycle($\mathbf{A}^I, \mathbf{y}^I$)

```
    Input: Coefficient matrix $\mathbf{A}^I$ and right-hand side (RHS) vector $\mathbf{y}^I$
    Output: Approximate solution $\boldsymbol{\chi}^I$
 1: if $I == 0$ then
 2:     Solve: $\mathbf{A}^I\boldsymbol{\chi}^I = \mathbf{y}^I$
 3: else
 4:     Pre-smooth: $\boldsymbol{\chi}^I = \text{Smooth}(\mathbf{A}^I, \mathbf{y}^I)$
 5:     Compute residual: $\mathbf{r}^I = \mathbf{y}^I - \mathbf{A}^I\boldsymbol{\chi}^I$
 6:     Restrict: $\mathbf{y}^{I-1} = \boldsymbol{\mathcal{R}}^I\mathbf{r}^I$
 7:     Apply V-cycle recursively: $\mathbf{z}^{I-1} = \text{V-Cycle}(\mathbf{A}^{I-1}, \mathbf{y}^{I-1})$
 8:     Interpolate: $\boldsymbol{\chi}^I = \boldsymbol{\chi}^I + \boldsymbol{\mathcal{I}}^I\mathbf{z}^{I-1}$
 9:     Post-smooth: $\boldsymbol{\chi}^I = \boldsymbol{\chi}^I + \text{Smooth}(\mathbf{A}^I, \mathbf{y}^I - \mathbf{A}^I\boldsymbol{\chi}^I)$
10: end if
    Return: $\boldsymbol{\chi}^I$
```



(a) The interpolation operator at a fine-level node $\mathbf{x}_i^F$ (highlighted in dark blue) is constructed as the GMLS approximation from the coarse-level nodes (red) within the fine-level node's $\tilde{\epsilon}$-neighborhood.



(b) The restriction operator at a coarse-level node $\mathbf{x}_i^C$ (red) is constructed as averaging over its own child nodes (light blue with green arrows) generated in **REFINE** stage during an iteration of adaptive refinement.

**Fig. 1.** Schematic of the interpolation and restriction. Coarse-level nodes are displayed in red and fine-level nodes are in blue. The black dashed lines indicate part of a solid body's boundary. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In the monolithic setting, the interpolation operators $\mathcal{I}^I$, the restriction operators $\mathcal{R}^I$, and the smoothers (i.e., subroutine Smooth($\mathbf{A}$, $\mathbf{y}$)) need to be carefully designed to properly handle the coupling of fluid and solid-body DOFs. The adaptive refinement discussed in Section 2.2.4 yields a hierarchical series of node sets $\Omega^I$, $I = 0, 1, 2, \ldots$. We can utilize the hierarchical structures between node sets to construct the corresponding interpolation $\mathcal{I}^I$ and restriction $\mathcal{R}^I$ operators for each node set $\Omega^I$.

## 3.3. Interpolation and restriction operators

In mesh-based discretization methods, e.g. finite element method, the interpolation/restriction operators can be constructed from the nested function spaces across different levels of meshes. In this work, instead of using function spaces, we build the interpolation and restriction operators through local approximations, i.e., approximating any scalar or vector field locally from a set of neighbor nodes. More specifically, if we take the fine-level nodes $\Omega^F$ as the "target" nodes and the coarse-level nodes $\Omega^C$ as the "source", by constructing GMLS approximation (Section 2.2.1) on the target from the source, we define the interpolation operator. If the target are coarse-level nodes, and the source are fine-level nodes, then a local averaging approximation on the target from the source builds the restriction operator. An illustrative description about the interpolation and restriction is provided in Fig. 1.

Thus, the interpolation operator for the pressure field is constructed as:

$$p(\mathbf{x}_i^F) = \mathbf{P}_i^{\intercal}(\mathbf{x}_i^F)\mathbf{c}_i^* \quad \text{with} \quad \mathbf{c}_i^* = \mathbf{M}_i^{-1}\left(\sum_{j\in\mathcal{N}_{\tilde{\epsilon}_i}} \mathbf{P}_i(\mathbf{x}_j^C)p(\mathbf{x}_j^C)W_{ij}\right),$$
$$\mathbf{M}_i = \sum_{j\in\mathcal{N}_{\tilde{\epsilon}_i}} \mathbf{P}_i(\mathbf{x}_j^C)W_{ij}\mathbf{P}_i^{\intercal}(\mathbf{x}_j^C), \quad \mathbf{x}_i^F \in \Omega^F, \quad \mathbf{x}_j^C \in \Omega^C ,$$

where the polynomial basis $\mathbf{P}$ is the $\tilde{\epsilon}$-scaled Taylor monomials. For velocity, the interpolation operator is built by directly using the divergence-free reconstruction space as follows:

$$\mathbf{u}(\mathbf{x}_i^F) = (\mathbf{P}_i^{\text{div}})^{\intercal}(\mathbf{x}_i^F)\mathbf{c}_i^{\text{div}*} \quad \text{with} \quad \mathbf{c}_i^{\text{div}*} = {\mathbf{M}_i^{\text{div}}}^{-1}\left(\sum_{j\in\mathcal{N}_{\tilde{\epsilon}_i}} \mathbf{P}_i^{\text{div}}(\mathbf{x}_j^C)\mathbf{u}(\mathbf{x}_j^C)W_{ij}\right),$$
$$\mathbf{M}_i^{\text{div}} = \sum_{j\in\mathcal{N}_{\tilde{\epsilon}_i}} \mathbf{P}_i^{\text{div}}(\mathbf{x}_j^C)W_{ij}(\mathbf{P}_i^{\text{div}})^{\intercal}(\mathbf{x}_j^C), \quad \mathbf{x}_i^F \in \Omega^F, \quad \mathbf{x}_j^C \in \Omega^C .$$

As such, we ensure the divergence-free constraint for velocity is consistently preserved during interpolation across different fine/coarse levels of nodes. As is well-known [20], for solving the incompressible Stokes equations, interpolations that maintain the divergence-free property are crucial for developing an efficient GMG preconditioner. In the above GMLS approximations for constructing the interpolation operator, $\tilde{\epsilon}_i$ for each fine-level node needs to be large enough such that sufficient coarse-level nodes are included in each $\tilde{\epsilon}$-neighborhood to ensure unisolvency over the reconstruction space and a well-posed solution to the weighted least square optimization.

For the restriction operator, since the matrix $\mathbf{A}$ in the linear system (17) is non-symmetric, it is unnecessary to require the restriction operator to be the transpose of the interpolation operator. Hence, we construct the restriction operation based on $h$-refinement. A coarse-level node $\mathbf{x}_i^C \in \Omega^C$ marked in the **MARK** stage of the adaptive refinement algorithm is called a "parent" node. The newly generated nodes (within the fluid domain or on solid boundaries) in the **REFINE** stage are called the "child nodes" corresponding to their parent node $\mathbf{x}_i^C$. As a result, an interior parent node generally has four child nodes, and a boundary parent node has two child nodes in 2D, or eight child nodes for an interior parent node and four child nodes for a boundary parent node in 3D. The approximation at a parent node is given by averaging the field values from its child nodes, which provides the restriction operator. By such, the matrix corresponding to the restriction operator is much sparser than that of the interpolation operator, leading to cheaper cost for matrix multiplication during the restriction operations.

For the blocks in $\mathbf{A}$ related to solid DOFs, the interpolation and restriction operators are simply identity matrices. Assembled together, the interpolation and restriction operators can be summarized as:

$$\begin{bmatrix}\mathbf{u}^{I+1}\\ p^{I+1}\\ \dot{\mathbf{X}}\end{bmatrix} = \mathcal{I}^I \begin{bmatrix}\mathbf{u}^{I}\\ p^{I}\\ \dot{\mathbf{X}}\end{bmatrix} = \begin{bmatrix}\mathcal{I}_{\mathbf{u}} & & \\ & \mathcal{I}_p & \\ & & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{u}^{I}\\ p^{I}\\ \dot{\mathbf{X}}\end{bmatrix}, \qquad \begin{bmatrix}\mathbf{u}^{I}\\ p^{I}\\ \dot{\mathbf{X}}\end{bmatrix} = \mathcal{R}^I \begin{bmatrix}\mathbf{u}^{I+1}\\ p^{I+1}\\ \dot{\mathbf{X}}\end{bmatrix} = \begin{bmatrix}\mathcal{R}_{\mathbf{u}} & & \\ & \mathcal{R}_p & \\ & & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{u}^{I+1}\\ p^{I+1}\\ \dot{\mathbf{X}}\end{bmatrix}, \tag{18}$$

where the super index $I$ corresponds to the node set $\Omega^I$ resulted from the $I$th iteration of adaptive refinement; while $\mathcal{I}_{\mathbf{u}}$ denotes the interpolation for velocity, $\mathcal{I}_p$ represents the interpolation for the pressure field; $\mathbf{I}$ is an identity matrix; $\mathcal{R}_p$ is the restriction operator for a scalar valued function (e.g., pressure); $\mathcal{R}_{\mathbf{u}} = \mathrm{diag}(\mathcal{R}_p, \ldots, \mathcal{R}_p)$ contains $d$ copies of $\mathcal{R}_p$ on the diagonal and is the restriction operator for velocity (a $d$-dimensional vector field).

### 3.4. Smoother based on physics splitting

In this subsection, we introduce the design of the smoother, i.e., the subroutine Smooth($\mathbf{A}$, $\mathbf{y}$) used in Algorithm 1. As we discussed, it is a relaxation-type iterative method that can efficiently smooth the high-frequency components of the error. For solving a fluid–solid interaction problem, we need the smoother to handle the high-frequency components of the errors for the fluid part and solid part, respectively, as well as the strong coupling between them.

Eqs. (2)–(3) inherently state two types of physics: One is Stokesian flow in a confined space, and the other describes the dynamics of several rigid solid bodies undergoing external loads as well as drags exerted by surrounding Stokesian flow. The coupling of different types of physics inspires the design of our physics-based smoother, which contains two stages. The first stage takes care of the fluid DOFs, and the second handles the solid bodies as well as their neighboring fluid nodes. The details are presented as follows.

#### 3.4.1. Smoother for the fluid DOFs

The submatrix

$$\mathbf{F} = \mathcal{B}_F \mathbf{A} \mathcal{B}_F^{\mathsf{T}} = \begin{bmatrix}\mathbf{K} & \mathbf{G}\\ \mathbf{B} & \mathbf{L}\end{bmatrix} \tag{19}$$

is considered as the fluid part of the system. Here, $\mathcal{B}_F$ is a Boolean matrix, which picks the fluid field variables, $\mathbf{u}$ and $p$, out of the whole unknown vector. $\mathbf{F}$ contains all the field variables (velocity and pressure) directly related to the interior GMLS nodes. The diagonal part $\mathbf{K}$ and $\mathbf{L}$ denote the curl–curl operator onto the velocity field and the Laplacian operator onto the pressure field, respectively. As mentioned in Section 3.1, the curl–curl operator $\mathbf{K}$ is equivalent to the Laplacian operator on the divergence-free polynomial basis. Therefore, the diagonal blocks of the fluid part matrix $\mathbf{F}$ are all Laplacian-type operators. Further, as seen in the governing equation (2), there exists a strong coupling between the velocity and pressure fields. We suggest using a node-wise Gauss–Seidel (GS) smoother. In 2D, we organize all field values into a $3 \times 3$ sub-matrix for each GMLS node. In 3D, all field values are organized into a $4 \times 4$ sub-matrix for each GMLS node. Therefore, we would include its diagonal entities

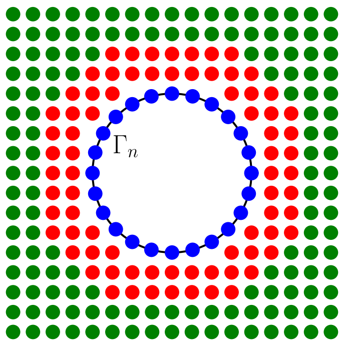


**Fig. 2.** Illustration of construction of $Q_n$. Blue nodes denote the boundary nodes on $\Gamma_n$. Red nodes represent the nodes near the boundary $\Gamma_n$ and contribute to the force and torque terms to the nth solid. The green nodes denote the normal interior GMLS nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

related to the two Laplacian operators and the discretized coupling terms at the node in the smoother. From our numerical experiments, the node-wise smoother outperforms the normal entry-wise one, and inverting small dense matrices does not affect the scalability of the smoother. This node-wise GS smoother is denoted as $\mathbf{S_F}$ in Algorithm 2.

### 3.4.2. Smoother for the solid bodies

Now consider the sub-matrix directly related to each solid body and its neighbor fluid nodes. The submatrix

$$\mathbf{N}_n = \mathcal{B}_{N_n}\mathbf{A}\mathcal{B}_{N_n}^{\mathsf{T}} = \begin{bmatrix} \mathbf{K}_n & \mathbf{G}_n & \mathbf{C}_n \\ \mathbf{B}_n & \mathbf{L}_n & \\ \mathbf{D}_n & \mathbf{T}_n & \end{bmatrix} \tag{20}$$

corresponds to the nth solid body and its neighbor fluid nodes. It is a square matrix and contains discretized drag force and torque exerted by the fluid. Each $\mathbf{N}_n$ is constructed by the following procedure. First, for each solid body, we construct an index set $Q_n$ such that

$$Q_n = \{i, j \mid \mathbf{x}_i \in \Gamma_n,\ \ j \in \mathcal{N}_{\epsilon_i}\}\ , \quad n = 1, 2, \ldots, N_s\ .$$

Fig. 2 illustrates the construction of $Q_n$, where the green and red nodes are the interior GMLS (fluid) nodes; the blue nodes are the GMLS nodes on $\Gamma_n$ (the boundary of the nth solid body). Since the red nodes are within the $\epsilon$-neighborhood of the blue nodes, $Q_n$ includes the indices of the nodes rendered in blue and red in Fig. 2 but excludes the nodes rendered in green. Second, we form the Boolean matrix $\mathcal{B}_{N_n}$ from $Q_n$, which picks the variables related to the nth solid body and its neighbor fluid nodes out of the whole unknown vector. The submatrix $\mathbf{N}_n$ is then built according to Eq. (20). Note that there is no intersection between most $Q_n$ for $n = 1, 2, \ldots, N_s$, especially when after several iterations of adaptive refinement, there are plenty of neighbor fluid nodes between any two closely contacting solid bodies. Thus, the parallel scalability can still be ensured.

Once we construct all sub-matrices $\mathbf{N}_n$, we use a Schwarz-type relaxation method to build the smoother. The resulting additive Schwarz-type smoother can be defined as $\mathbf{S}_{\mathbf{N}}^{\text{exact}} := \sum_{n=1}^{N_s} \mathcal{B}_{N_n}^{\mathsf{T}}\mathbf{N}_n^{-1}\mathcal{B}_{N_n}$. Exactly inverting $\mathbf{N}_n$ could be expensive. Therefore, we use a Schur complement approach to invert it approximately, i.e.,

$$\widetilde{\mathbf{N}}_n^{-1} = \begin{bmatrix} \mathbf{I} & -\widetilde{\mathbf{F}}_n^{C^{-1}}\begin{bmatrix}\mathbf{C}_n \\ \mathbf{0}\end{bmatrix} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{F}}_n^{C^{-1}} & \mathbf{0} \\ \mathbf{0} & \widetilde{\boldsymbol{\Psi}}_n^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\begin{bmatrix}\mathbf{D}_n & \mathbf{T}_n\end{bmatrix}\widetilde{\mathbf{F}}_n^{C^{-1}} & \mathbf{I} \end{bmatrix}, \tag{21}$$

where $\widetilde{\mathbf{F}}_n^{C^{-1}}$ approximates the inverse of the submatrix

$$\mathbf{F}_n^C = \begin{bmatrix} \mathbf{K}_n & \mathbf{G}_n \\ \mathbf{B}_n & \mathbf{L}_n \end{bmatrix}.$$

In our numerical experiments, we use one iteration of the node-wise GS smoother to define $\widetilde{\mathbf{F}}_n^{C^{-1}}$. In addition, the approximated Schur complement $\widetilde{\boldsymbol{\Psi}}_n$ is given as:

$$\widetilde{\boldsymbol{\Psi}}_n = \begin{bmatrix}\mathbf{D}_n & \mathbf{T}_n\end{bmatrix} \text{block-diag}^{-1}(\mathbf{F}_n^C) \begin{bmatrix}\mathbf{C}_n \\ \mathbf{0}\end{bmatrix} .$$

Here, block-diag$^{-1}(\cdot)$ denotes the diagonal block inversion of a matrix, which takes a $3 \times 3$ or $4 \times 4$ sub-matrix for each GMLS node in $\mathbf{F}_n^C$ and inverts those sub-matrices. Since the size of matrix $\widetilde{\boldsymbol{\Psi}}_n$ is quite small, only $3 \times 3$ in 2D and $6 \times 6$ in 3D, a direct solver is applied whenever the inversion of $\widetilde{\boldsymbol{\Psi}}_n$ is needed. Finally, for the solid bodies, the overall additive Schwarz-type smoother using the approximate Schur complement approach is given by:

$$\mathbf{S}_\mathbf{N} := \sum_{n=1}^{N_s} \mathcal{B}_{N_n}^\mathsf{T} \widetilde{\mathbf{N}}_n^{-1} \mathcal{B}_{N_n} . \tag{22}$$

Given the two smoothers $\mathbf{S}_\mathbf{F}$ and $\mathbf{S}_\mathbf{N}$, we connect them through an overlapping multiplicative Schwarz approach and hence establish the proposed two-stage smoothing scheme. Algorithm 2 summarizes the established overall smoother.

**Algorithm 2** Smoother: Smooth($\mathbf{A}$, $\mathbf{y}$)

```
    Input: Coefficient matrix $\mathbf{A}$ and RHS vector $\mathbf{y}$
    Output: Relaxed solution $\chi$
1:  Initialize $\chi = \mathbf{0}$
2:  Perform $k$ steps of node-wise GS smoother on the fluid DOFs:
3:  for $i = 1, \cdots, k$ do
4:      $\chi \leftarrow \chi + \mathcal{B}_F^\mathsf{T} \mathbf{S}_\mathbf{F} \mathcal{B}_F (\mathbf{y} - \mathbf{A}\chi)$
5:  end for
6:  Apply the additive Schwarz-type smoother as in Eq. (22) for the solid bodies:
        $\chi \leftarrow \chi + \mathbf{S}_\mathbf{N}(\mathbf{y} - \mathbf{A}\chi), \quad \text{where } \mathbf{S}_\mathbf{N} = \sum_{n=1}^{N_s} \mathcal{B}_{N_n}^\mathsf{T} \widetilde{\mathbf{N}}_n^{-1} \mathcal{B}_{N_n}$
7:  Return: $\chi$
```

### 3.5. Linear solver summary

Given the constructed interpolation/restriction operators (Eq. (18)) and smoothers (Algorithm 2), we follow the V-cycle process in Algorithm 1 to build the entire monolithic GMG preconditioner. Since the coefficient matrix $\mathbf{A}$ is non-symmetric, we employ the GMRES method as the Krylov iterative solver for solving Eq. (17). In each GMRES iteration, we call once the multigrid preconditioning, i.e., Algorithm 1. With the proposed monolithic GMG preconditioner, we aim to ensure the convergence of the linear solver and to optimize the scaling of the number of GMRES iterations required with respect to the numbers of solid bodies and total DOFs.

## 4. Parallel implementation

To solve large-scale fluid–solid interaction problems, the parallel implementation of the proposed monolithic GMG preconditioner is developed. We aim to achieve the parallel scalability of our numerical solver.

### 4.1. Domain decomposition and neighbor search

All the GMLS nodes, $\forall \mathbf{x}_i \in \Omega^I$, yield after each iteration $I$ of adaptive refinement, are evenly distributed into $N_c$ sets, where $N_c$ is the number of CPU cores invoked in the simulation. Each set of nodes, denoted as $\Omega_k^I$ ($k = 1, 2, \ldots, N_c$), is then allocated into one core, following the Recursive Coordinate Bisection (RCB) method [32]. In each core, `Compadre` [33] is used to generate the coefficients resulted from the GMLS discretization for the node-set $\Omega_k^I$. This way of domain decomposition guarantees that the nodes in the same set $\Omega^I$ are evenly distributed among

CPU cores and spatially clustered on each core, and hence balances the workload and minimizes the communication required between cores for solving the linear system in Eq. (17).

After domain decomposition, a ghost node set $\mathcal{G}^I_{k_\xi}$ is determined according to the order $(m)$ of GMLS discretization and the spatial locations of the GMLS nodes in the node set $\Omega^I_k$, as:

$$\mathcal{G}^I_{k_\xi} = \{\mathbf{x}_j \mid \|\mathbf{x}_j - \mathbf{x}_i\| < \xi \Delta x_i,\ \forall \mathbf{x}_i \in \Omega^I_k,\ \mathbf{x}_j \in \Omega^I\}\ , \quad k = 1, 2, \ldots, N_c\ , \quad \xi = \xi(m)\ ,$$

where $\xi$ is a function of $m$ and we use $\xi = 4m$ in our numerical tests; $\Delta x_i$ denotes the discretization resolution of node $\mathbf{x}_i$. Collecting this ghost node set only calls one communication between all cores. Once the collection is done, neighbor search for each node in $\Omega^I_k$ is performed by calling `nanoflann` [34], which is a function building KD-trees [35] and finds all nodes in the $\epsilon$-neighborhood of any node $\mathbf{x}_i \in \Omega^I_k$ from the ghost node set $\mathcal{G}^I_{k_\xi}$, i.e.,

$$\mathcal{N}_{\epsilon_i} = \{j \mid \|\mathbf{x}_i - \mathbf{x}_j\| < \epsilon_i,\ \forall \mathbf{x}_i \in \Omega^I_k,\ \mathbf{x}_j \in \mathcal{G}^I_{k_\xi}\}\ .$$

$\mathcal{N}_{\epsilon_i}$ hence provides the neighbor list needed in the GMLS discretization that leads to the coefficient matrix $\mathbf{A}$ of the linear system in Eq. (17).

Note that when building the interpolation operator in Section 3.3, GMLS approximation is needed, for which we also need to build the neighbor list. Different from the above, the neighbor nodes of $\mathbf{x}_i$ in this GMLS approximation are not in the same node set as $\mathbf{x}_i$. Thus, for that purpose the ghost node set is determined as:

$$\widetilde{\mathcal{G}}^I_{k_\xi} = \{\mathbf{x}_j \mid \|\mathbf{x}_j - \mathbf{x}_i\| < \xi \Delta x_i,\ \forall \mathbf{x}_i \in \Omega^{I+1}_k,\ \mathbf{x}_j \in \Omega^I\}\ , \quad k = 1, 2, \ldots, N_c\ ,$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the node sets yield from different iterations of adaptive refinement, respectively. And then the neighbor list is built. This time, `nanoflann` finds all nodes in the $\epsilon$-neighborhood of any node $\mathbf{x}_i \in \Omega^{I+1}_k$ in the ghost node set $\widetilde{\mathcal{G}}^I_{k_\xi}$. Thus, we have:

$$\widetilde{\mathcal{N}}_{\epsilon_i} = \{j \mid \|\mathbf{x}_i - \mathbf{x}_j\| < \epsilon_i,\ \forall \mathbf{x}_i \in \Omega^{I+1}_k,\ \mathbf{x}_j \in \widetilde{\mathcal{G}}^I_{k_\xi}\}\ ,$$

which provides the neighbor list for the GMLS approximation needed to construct the interpolation operator.

## 4.2. Data storage

To reduce the data storage, all DOFs related to the solid bodies can be stored in a single CPU core (e.g., the last core). Thus, following the convention of PETSc, the sub-matrices $\mathbf{D}$ and $\mathbf{T}$ reside in the last rows of the matrix $\mathbf{A}$. However, the number of nonzero entities in $\mathbf{D}$ and $\mathbf{T}$ would increase with the inclusion of more solid bodies. As a result, the parallel scalability deteriorates as the number of solid bodies increases. Therefore, we spread the whole storage of data and workload related to $\mathbf{D}$ and $\mathbf{T}$ among all cores as follows. When assembling $\mathbf{D}$ and $\mathbf{T}$, each core would go through the GMLS nodes stored in it to generate its local $\mathbf{D}_i$ and $\mathbf{T}_i$. Accordingly, any matrix–vector multiplication operation for $\mathbf{D}$ and $\mathbf{T}$ would be split among all cores. That is, the matrix–vector multiplication is done locally within each core, followed by a parallel summation over all cores. For example, the matrix–vector multiplication for $\mathbf{D}$ is given by:

$$\mathbf{D}\mathbf{u} = \sum_{i=0}^{N_c-1} \mathbf{D}_i \mathbf{u}\ , \tag{23}$$

where $N_c$ is the total number of CPU cores invoked in a simulation; $\mathbf{D}_i\mathbf{u}$ is computed in each core parallelly and then summed up over all cores. By such, the parallel scalability is recovered as the number of solid bodies increases.

## 4.3. Linear algebra operations

The parallel implementation of our linear solver is achieved by interfacing with PETSc package [28]. Once all coefficients resulting from the GMLS discretization are generated from each core, the entire coefficient matrix $\mathbf{A}$ is assembled in parallel through PETSc. And all linear algebra operations associated with the proposed monolithic GMG preconditioner, as well as the Krylov iterations, are performed by calling the corresponding PETSc functions, including the matrix–vector multiplication, matrix–matrix multiplication, and matrix inversion by a direct solver, as well as the node-wise GS iteration and the GMRES iterative solver.

### *4.4. Neumann BC for the pressure*

As stated in Eq. (2), an inhomogeneous Neumann BC is imposed for pressure. To ensure the uniqueness and physical consistency of the solution, we additionally enforce a zero-mean constraint for the pressure field, i.e., requiring $\Xi^\top \mathbf{p} = 0$, where $\Xi$ is a constant vector (e.g. $\Xi = \mathbf{1}$); $\mathbf{p}$ denotes the vector of discretized pressure with entities $p(\mathbf{x}_i)$, $\mathbf{x}_i \in \Omega$. If we use a Lagrange multiplier to impose this zero-mean constraint, it would break the block structure of $\mathbf{F}$ in Eq. (19), and in the meanwhile introduce a dense row and column, i.e., $\Xi^\top$ and $\Xi$, respectively, into $\mathbf{F}$, which in turn would deteriorate the parallel scalability of our preconditioner. Thus, we instead solve the following problem:

$$\left(\mathbf{I} - \frac{\Xi\Xi^\top}{\Xi^\top\Xi}\right)\mathbf{L}\mathbf{p} = \left(\mathbf{I} - \frac{\Xi\Xi^\top}{\Xi^\top\Xi}\right)\mathbf{g}\,, \tag{24}$$

which is equivalent to using the Lagrangian multiplier. Here, $\mathbf{g}$ denotes the vector with entities $g(\mathbf{x}_i)$, $\mathbf{x}_i \in \Gamma$. In practice, the matrix $\left(\mathbf{I} - \frac{\Xi\Xi^\top}{\Xi^\top\Xi}\right)$ does not need to be explicitly assembled. Noting that $\frac{\Xi^\top\mathbf{p}}{\Xi^\top\Xi}$ actually calculates the mean of the entities in $\mathbf{p}$, the application of $\left(\mathbf{I} - \frac{\Xi\Xi^\top}{\Xi^\top\Xi}\right)$ can be implemented as: first calculating the mean of all entities of the vector $\mathbf{p}$ in parallel and then subtracting the mean from each entity of $\mathbf{p}$. By such, the block structure of $\mathbf{F}$ can be preserved, and the parallel scalability is unaffected.

## **5. Numerical examples**

In this section, we systematically assess the effectiveness and scalability of our proposed monolithic GMG preconditioner through several numerical examples, including pure fluid flows and fluid–solid interaction problems.

### *5.1. Pure fluid flows*

We first verify our GMLS discretization, GMG preconditioner, and parallel implementation by solving problems of pure fluid flows. We start with a simple Taylor–Green vortex flow followed by a more complicated Stokes flow in an artificial vascular network.

#### *5.1.1. Taylor–Green vortex*

The fluid domain is set as $\Omega_f = [-1, 1] \times [-1, 1]$. Given the source term in Eq. (2) as:

$$\mathbf{f} = \begin{bmatrix} 2\pi^2 \cos(\pi x)\sin(\pi y) + 2\pi \sin(2\pi x) \\ -2\pi^2 \sin(\pi x)\cos(\pi y) + 2\pi \sin(2\pi y) \end{bmatrix}, \quad \forall\, \mathbf{x} = (x, y) \in \Omega_f\,, \tag{25}$$

and the no-slip BC for velocity as

$$\begin{cases} u = \cos(\pi x)\sin(\pi y) \\ v = -\sin(\pi x)\cos(\pi y) \end{cases}, \quad \forall\, \mathbf{x} = (x, y) \in \Gamma_0\,, \tag{26}$$

where $\Gamma_0 = \partial\Omega_f$ denotes the outer boundary of the fluid domain, the analytical solution of Eq. (2) is the following:

$$\begin{cases} u = \cos(\pi x)\sin(\pi y) \\ v = -\sin(\pi x)\cos(\pi y) \\ p = -\cos(2\pi x) - \cos(2\pi y) \end{cases}, \quad \forall\, \mathbf{x} = (x, y) \in \Omega_f. \tag{27}$$

Our numerical solutions of this problem obtained with the 2nd-order or 4th-order GMLS discretization, respectively, are compared with the analytical solution. The root mean square (RMS) errors are computed for both velocity and pressure. As shown in Fig. 3, the numerical results exhibit the consistent 2nd-order or 4th-order convergence, as theoretically expected; the velocity and pressure fields achieve equal-order optimal convergence.

After verifying the accuracy and convergence of the numerical solutions, we next examine the scalability of the proposed GMG preconditioner, as well as the parallel scalability of our implementation. To this end, the numbers of CPU cores $N_c$ are varied from 25 to 289 in the tests. For such a pure fluid flow problem without singularities,

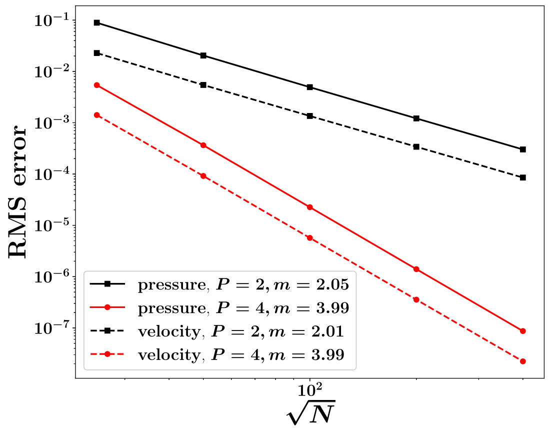
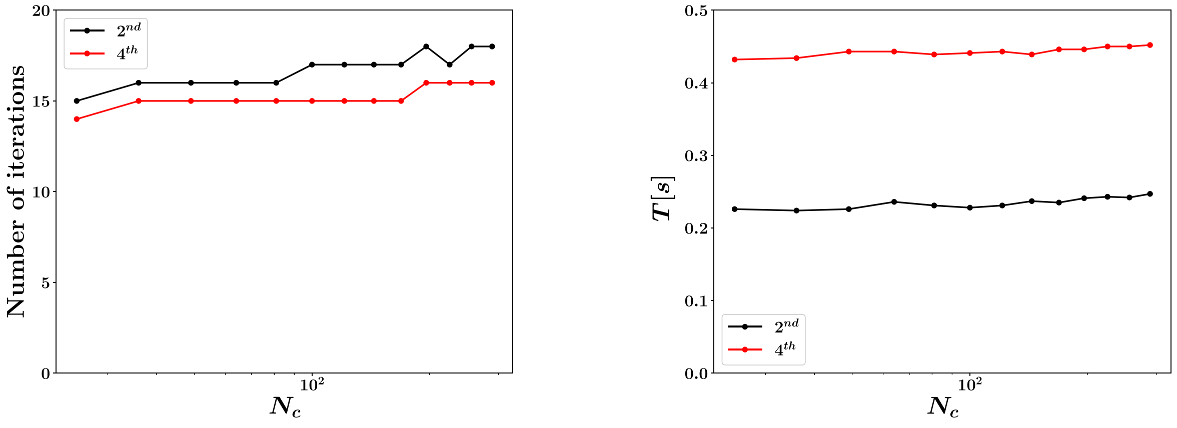


**Fig. 3.** Pure fluid flow-Taylor–Green vortex: RMS errors and convergence for the numerical solutions of the velocity (dashed line) and pressure (solid line). Here, $N$ denotes the total number of GMLS nodes; $P$ denotes the order of polynomial basis used in the GMLS discretization; $m$ is the slope of each line.



(a) Number of iterations required for the linear solver to converge. (b) Computer time (in seconds) spent for a single step of preconditioning.

**Fig. 4.** Pure fluid flow-Taylor–Green vortex: Scalability of the proposed GMG preconditioner and the weak scalability of our parallel implementation of the preconditioner, tested for different order of GMLS discretization (black for the 2nd order and red for the 4th order). Here, $N_c$ denotes the number of CPU cores used in each test. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

adaptive refinement is not needed, and hence only uniform refinement is conducted. To maintain the same number of GMLS nodes distributed in each CPU core, we start from $1 \times 1$ GMLS nodes in each core, and then execute the same uniform refinement for the GMLS nodes within each core. After seven iterations of uniform refinement, there are $128 \times 128$ GMLS nodes in each CPU core. To verify the scalability of the proposed GMG preconditioning method, we examine the number of iterations required for the linear solver to converge. To evaluate the parallel scalability of our implementation, we record the computer time spent for a single step of preconditioning. Our tests are particularly for the linear system generated from the last iteration ($I = 7$) of (uniform) refinement, i.e., with the most DOFs. In Fig. 4(a), we show the variation of the number of iterations with respect to different numbers of CPU cores used and find that it only varies slightly from 15 to 18 for the 2nd order GMLS discretization and 14 to 16 for the 4th order GMLS as the number of CPU cores increases from 25 to 289. The computer time spent on a single step of preconditioning stays almost constant with increasing CPU cores, independent of the order of GMLS discretization, as depicted in Fig. 4(b). By these results, we demonstrate the scalability of the proposed GMG preconditioner and the weak scalability of our parallel implementation of the preconditioner.

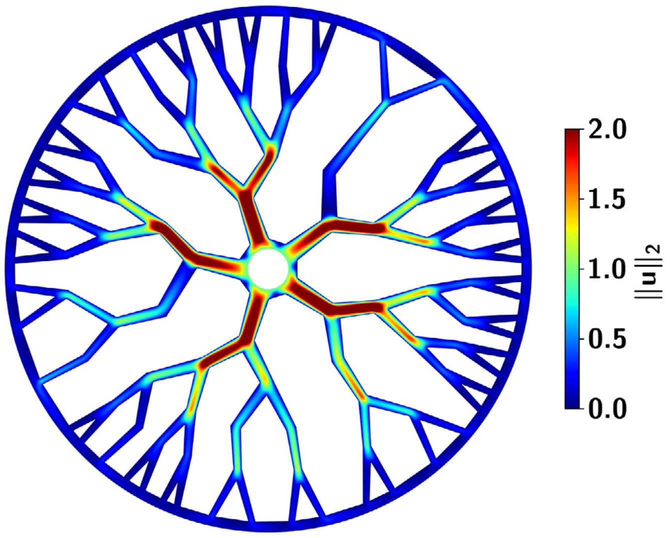


**Fig. 5.** Pure fluid flow-Artificial vascular network: Computed velocity field, where the color bar indicates the magnitude of velocity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

### *5.1.2. Artificial vascular network*

We next solve a more complicated pure fluid flow problem, which is in an artificial vascular network, mimicking the structure of a zebrafish's eye [36]. The artificial vascular network is built as in Fig. 5. A constant inflow flux drives the flow in this network at the inner circular boundary in the center, and a constant outflow flux is imposed at the outermost circular boundary. The overall volume of the fluid in the network is conserved. All the other boundaries in the network are imposed no-slip BCs for the velocity. Due to the irregular computational domain with corner singularities, adaptive refinement is required such that the numerical solution can achieve optimal convergence. Starting from a coarse resolution at $\Delta x^0 = 0.02$ and setting $\alpha = 0.8$ (marking percentage) in **MARK** stage, we perform 8 iterations of adaptive refinements. The velocity field computed after the 8th iteration of adaptive refinement is shown in Fig. 5, where there are in total 930,240 GMLS nodes, and the 2nd-order GMLS discretization is employed.

Due to a lack of the true solution, we calculate the total recovered errors (Eq. (13)), instead of true errors, during iterations of adaptive refinement to evaluate the accuracy and convergence of the numerical solutions, as shown in Fig. 6. We note that the convergence rate does not reach the theoretical 2nd order. This arises from the presence of significant portions of nonconvex boundaries in the computational domain. It is well-known that nonconvexity can lead to low regularity of the solutions to elliptic PDEs, which in turn affects the performance and liability of recovery-based *a posteriori* error estimator, resulting in deteriorated convergence in adaptive refinement [37,38]. Therefore, we design a numerical test to elucidate this issue. In the test, we let the computational domain be constrained by two boundaries, an exterior boundary $\Gamma_{ext}$ and an interior boundary $\Gamma_{int}$. $\Gamma_{ext}$ is set the same as that in Section 5.1.1 with the same BCs (Eq. (26)). $\Gamma_{int}$ is stationary but with different shapes, including square, hexagon, triangle, and parallelogram. Taking $\Gamma_{int}$ of a square as an example, the computational domain $\Omega_f$ is depicted in Fig. 7(a). For each shape of $\Gamma_{int}$, the problem is solved with 10 iterations of adaptive refinement, and the total recovered errors calculated during adaptive refinement is summarized in Fig. 7(b). As expected, the convergence rate decreases from 2 to 1 when $\Omega_{int}$ changes from hexagon to triangle, with increasing nonconvexity. This confirms our sub-optimal convergence results observed in Fig. 6.

After examining the convergence of our numerical solutions, we further assess the scalability of our proposed GMG preconditioner and parallel implementation. Due to the complexity of the computational domain, we cannot guarantee that the total number of GMLS nodes resulted in each adaptive refinement iteration increases proportionally, thus, the weak scaling test is not appropriate for this problem. Instead, we perform a strong scaling test to demonstrate the parallel scalability of the proposed GMG preconditioning method. In the strong scaling test, the number of cores invoked changes from 40 to 240 cores in the simulation. The statistics of the test is collected in

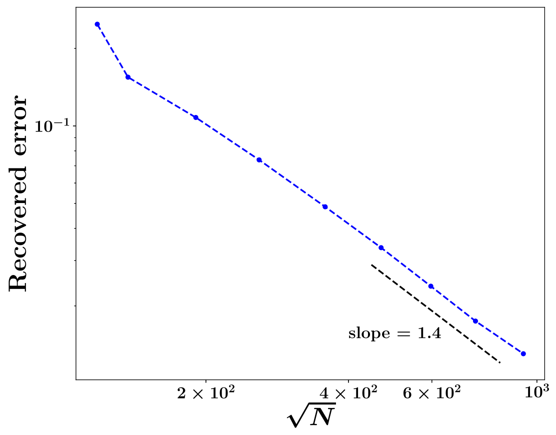
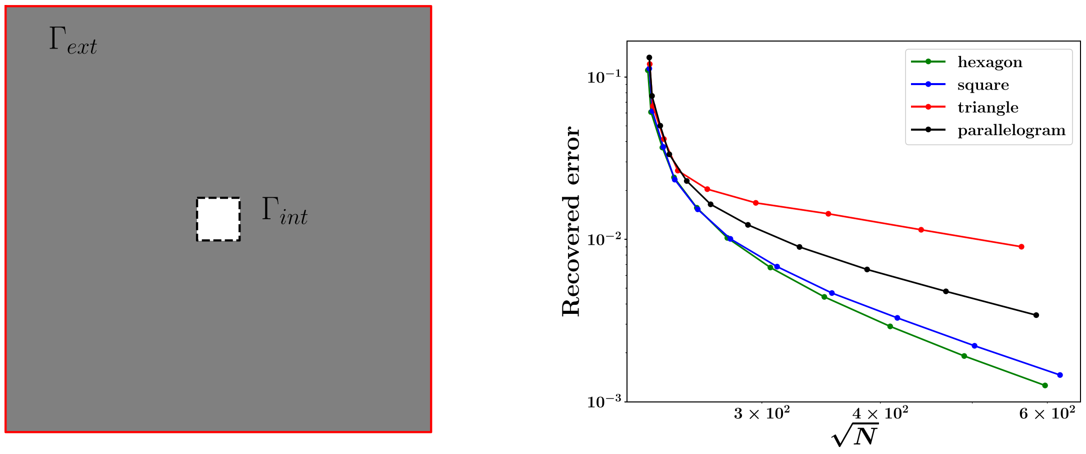


**Fig. 6.** Pure fluid flow-Artificial vascular network: Convergence of the total recovered error. Here, the slope is regressed from the last 4 points; $N$ denotes the total number of GMLS nodes.



(a) Illustration of the computational domain: Fluid (gray) is confined in $\Omega_f$ by $\Gamma_{ext}$ and $\Gamma_{int}$. $\Gamma_{ext}$ (red solid line) is a square with side length of 1. $\Gamma_{int}$ (black dashed line) is at the center of domain and varied from a square to a hexagon, triangle, or parallelogram, all with an equal side length of 0.2.

(b) Convergence of the total recovered error. $N$ denotes the total number of GMLS nodes. The convergence rate is estimated as the slope regressed from the last 4 data points: slop≈2.0 for hexagon and square; slop≈1.6 for parallelogram; slop≈1.0 for triangle.

**Fig. 7.** Convergence test with respect to the nonconvexity of computational domain.

Table 1. The average DOFs per core after the 8th adaptive refinement iteration is listed in the column *DOFs/Core*, which shows the scale of the workload to this problem. As expected, the workload decreases when the number of cores increases, since we are doing a strong scalability test and the total DOFs are fixed. The average number of GMRES iterations required in each adaptive refinement iteration step is listed in the column *Average Iterations*. It stays around 66, independent of different numbers of cores invoked, which indicates that our parallel implementation does not affect the convergence of the proposed GMG preconditioning method. To evaluate the parallel scalability of our implementation, we record the CPU wall time spent for solving the linear systems in each adaptive refinement iteration and sum them up, which is denoted as $T_s$ in Table 1; we also track the total CPU wall time spent for the entire simulation, denoted as $T$. By comparing $T_s$ and $T$, we note that the time spent for solving the linear systems indeed dominates the overall computing time, by more than 70%, regardless of how many cores used. The GMLS discretization can be trivially parallelized, and hence the overall parallel scalability is dictated by the proposed preconditioning method. By invoking different numbers of cores, we can compare the overall execution time and thereby evaluate the speed up factor (denoted as $S$) using the least number of cores (i.e., 40 cores) as the base. We hence use Amdahl's law [39] to assess the performance of our parallel implementation, which is given

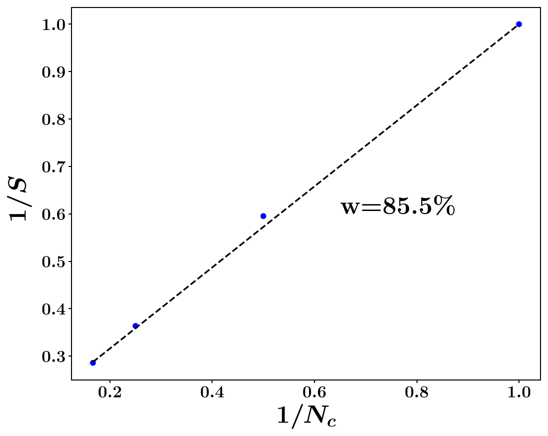


**Fig. 8.** Pure fluid flow-Artificial vascular network: Parallel portion $w$ determined from Amdahl's law in Eq. (28).

**Table 1**
Pure fluid flow–Artificial vascular network: Strong scaling test.

| $N_c$ | DOFs/Core | Average iterations | Time for (17) $T_s$ [s] | Overall time $T$ [s] | Overall speedup $S$ |
|---|---|---|---|---|---|
| 40 | 69,768 | 65.5 | 90.52 | 128.47 | 1.00 |
| 80 | 34,884 | 67.1 | 52.43 | 76.50 | 1.68 |
| 160 | 17,442 | 66.5 | 33.71 | 46.73 | 2.75 |
| 240 | 11,628 | 66.3 | 28.29 | 36.68 | 3.50 |

by:

$$S = \frac{1}{(1-w) + \dfrac{w}{N_c}} \,,$$

where $w$ denotes the parallel portion of a numerical solver. To estimate the value of $w$ for our solver, we take the reciprocal of Amdahl's law as:

$$\frac{1}{S} = (1-w) + \frac{w}{N_c} \,. \tag{28}$$

Using the data of $S$ in Table 1, we can determine $w$ by linear regression. In particular, we employ only the data of $N_c = 40$ and $N_c = 80$ to determine $w$, and then use the data of $N_c = 160$ and $N_c = 240$ for testing. As depicted in Fig. 8, the last two data points fall very close to the line fitted from the first two data points, and the estimated parallel portion $w = 85.5\%$. We hence demonstrate the consistency and scalability of our parallel implementation of the proposed monolithic GMG preconditioner.

## 5.2. Fluid–solid interactions

After pure fluid flows, we next address fluid–solid interaction problems with the inclusion of multiple solid bodies of different shapes.

### 5.2.1. Duplicate cells with cylinders

First, for an accurate assessment of the scalability of our proposed monolithic GMG preconditioner, we design a numerical example that allows both the solid bodies and the GMLS nodes to be evenly distributed among computer cores. To this end, the entire computational domain is partitioned into $N_c$ square cells, each of which includes four cylinders, as shown in Fig. 9. All DOFs associated in each square cell are allocated to a single CPU core. Thus, there are in total $N_s = 4N_c$ solid bodies, where $N_c$ is the number of CPU cores for each test. We intentionally place the cylinders in close contacts to demonstrate that our spatially adaptive GMLS method can resolve the singularities

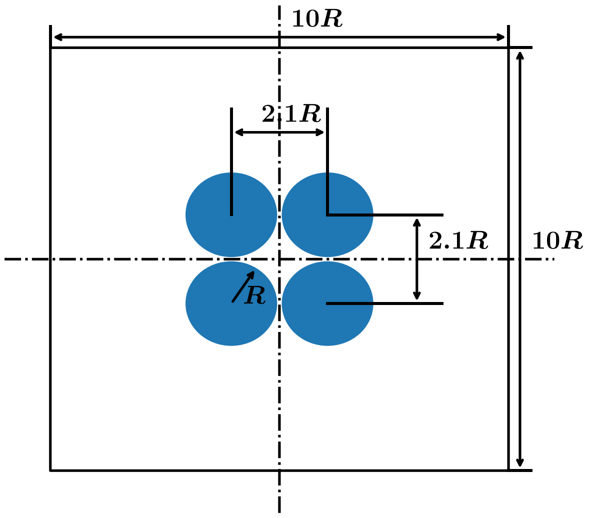


**Fig. 9.** Fluid–solid interactions-Duplicate cells with cylinders: Schematic of a square cell with four cylinders.

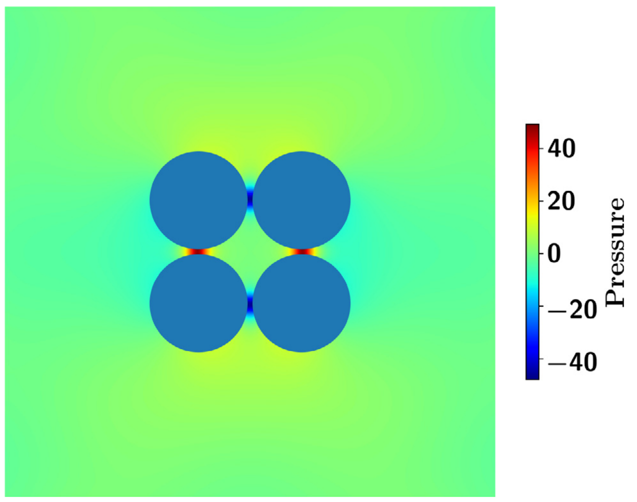
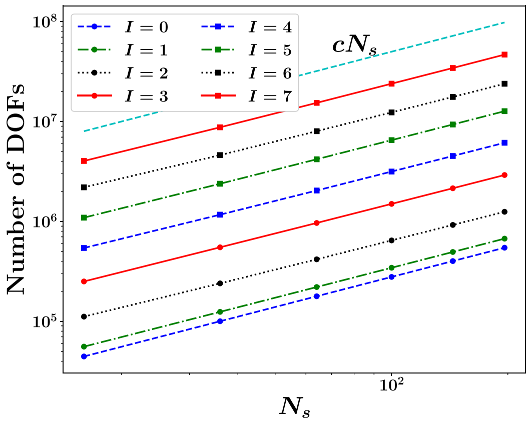


**Fig. 10.** Fluid–solid interactions-Duplicate cells with cylinders: The pressure field computed in each cell. The color is correlated to the magnitude of pressure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

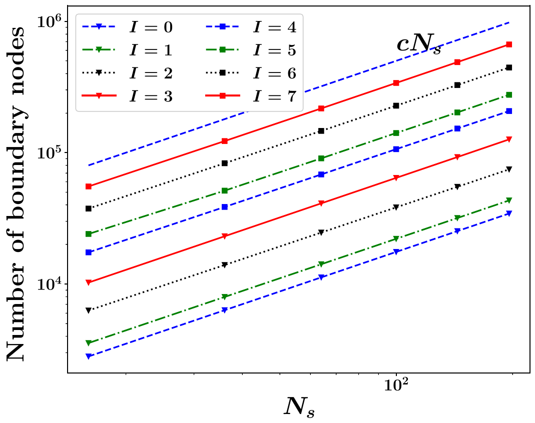
governing the lubrication effects. To maintain the same total DOFs after adaptive refinement in each cell, the four cylinders are placed near the center of each cell such that the distances between the solid bodies in different cells are much larger than the distances between the solid bodies within one cell.

The source term in Eq. (2) and the BC at the outer boundary of the entire computational domain are set the same as those in Section 5.1.1, i.e., Eqs. (25)–(26). The 2nd-order GMLS discretization is employed for solving this problem. Seven iterations of adaptive refinement with $\alpha = 0.8$ are conducted until the total recovered error reaches the preset error tolerance $\varepsilon_{tol} = 10^{-3}$ in Eq. (11). The stopping criterion for the GMRES iteration is set as $10^{-6}$. The resultant pressure field in a single cell is shown in Fig. 10. We can see that all the singular pressures within the narrow gaps between cylinders are correctly captured.

Before we assess the parallel scalability of our preconditioner, we examine how the total DOFs consisting of fluid (GMLS) nodes and boundary (GMLS) nodes grow with adaptive refinement and the inclusion of more solid bodies. In Fig. 11(a), we can see that the total DOFs increases linearly with respect to the number of solids included in the domain, regardless of after any iteration of adaptive refinement; in Fig. 11(b), we find that the total boundary nodes



(a) Number of total DOFs.



(b) Number of the boundary GMLS nodes.

**Fig. 11.** Fluid–solid interactions-Duplicate cells with cylinders: The growths of total DOFs and the boundary (GMLS) nodes with respect to the number of solids included in the domain and different iterations of adaptive refinement.

also increases linearly with respect to the number of solids, after each iteration of adaptive refinement. From these examinations, we confirm that the workload related to the node-wise GS smoother $\mathbf{S_F}$, the additive Schwarz-type smoother $\mathbf{S_N}$, and the MG preconditioner can be evenly distributed among all cores. Only with that ensured, we can make an accurate assessment of scalability.

We first check the number of iterations required for the GMRES iterative solver to converge, i.e., to reach the stopping criterion. In Fig. 12, we show the number of GMRES iterations required in the **SOLVE** stage of each iteration of adaptive refinement. As can be seen, for a fixed number of solids, the number of GMRES iterations required generally stays constant or decreases slightly. Noting that the total DOFs continuously increase during iterations of adaptive refinement, we hence demonstrate the weak scalability of our preconditioner with respect to the number of DOFs for fixed number of solids. With inclusion of more solids, we expect that the number of GMRES iterations required would increase. However, the scaling of this increase is critical for the sake of scalability. Fig. 13(a) shows that the number of GMRES iterations scales with $O(\sqrt{N_s})$, and the scaling is generally consistent for different iteration step of adaptive refinement. We next check the computer time spent finishing all seven adaptive refinement iterations. Although it includes the time spent on the GMLS discretization and the time for solving the linear system and executing other stages of adaptive refinement, solving the linear system dominates the computer time. Fig. 13(b) depicts how the computer time varies with an increasing number of solids. Overall, it exhibits a scaling of $\mathcal{O}(\sqrt{N_s}\log N_s)$, for which the factor $\sqrt{N_s}$ arises from the scaling of the number of GMRES iterations and the factor $\log N_s$ is mainly contributed by the additional operation introduced in Section 4.2 such as Eq. (23). Note that in this numerical example, the number of CPU cores $N_c$ is proportional to the number of solids $N_s$, in fact $N_s = 4N_c$. Thus, the above scaling behaviors are shown in Fig. 13 also hold with respect to the number of cores $N_c$.

### 5.2.2. *Particulate suspensions in 2D*

Finally, we perform long-time simulations to examine the performance of the proposed monolithic GMG preconditioning method. We first simulate 2D suspension flows of freely moving particles of different shapes. The flow is driven by the source term in Eq. (2) and the BC at the outer boundary of fluid domain ($[-1, 1] \times [-1, 1]$) as in Eqs. (25)–(26). One hundred solid particles are suspended in the flow, subject to bidirectional hydrodynamic couplings. Initially, all particles are evenly distributed throughout the domain. Due to hydrodynamic couplings, particles are freely moving with the flow. The physical time of the entire simulation is $T = 5$. The 2nd-order GMLS is employed for spatial discretization. The 5th-order Runge–Kutta integrator with adaptive time stepping is used for temporal integration to update the particles' translational and angular positions. An initial time step $\Delta t = 0.2$ is applied. In each time step, adaptive $h$-refinement with $\alpha = 0.8$ are conducted until the total recovered error reaches the preset error tolerance $\varepsilon_{tol} = 10^{-3}$ in Eq. (11). For the linear solver, the stopping criterion for the GMRES iteration is set as $10^{-6}$.

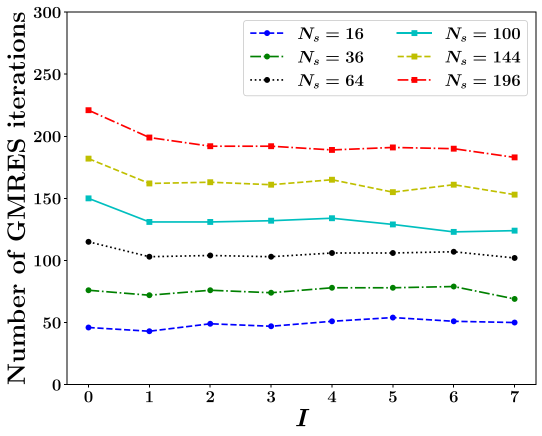
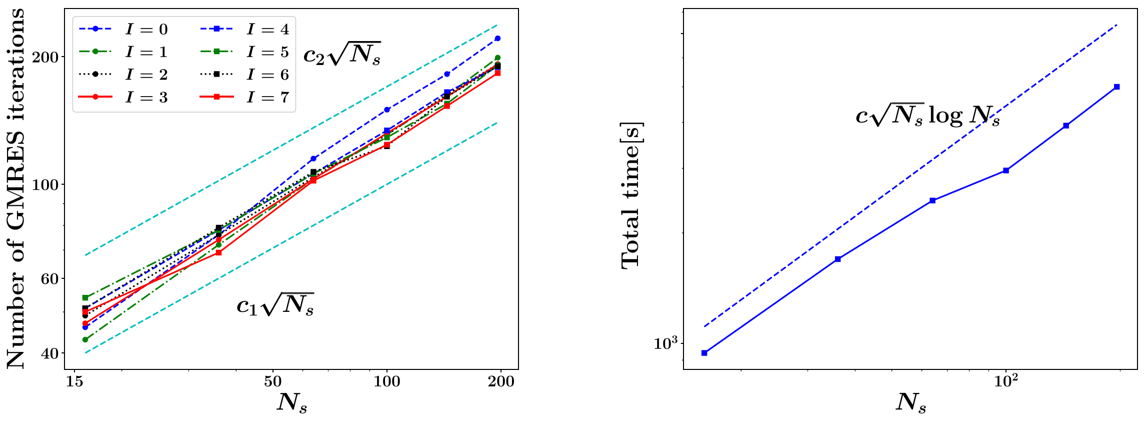


**Fig. 12.** Fluid–solid interactions-Duplicate cells with cylinders: The number of GMRES iterations required in the **SOLVE** stage of each adaptive refinement iteration, for fixed numbers of solids.



(a) Scaling of the number of GMRES iterations required at different iteration step of adaptive refinement.

(b) Scaling of the total computer time spent for all seven adaptive refinement iterations.

**Fig. 13.** Fluid–solid interactions-Duplicate cells with cylinders: Scalability results. Here, $N_s$ denotes the number of solids, and $N_s = 4N_c$ with $N_c$ the number of CPU cores.

*100 similar particles*

In the first simulation, the 100 particles are all circular with the radius $R = 0.04$. The snapshot of the particles' configuration at the terminal time $T = 5$ is shown in Fig. 14. The zoom-in pressure distributions at several locations are also shown in Fig. 14, for which we intentionally choose to show where the particles are in close contact either with each other or with the outer wall. We can see that even though it is challenging in a dynamic simulation to resolve all point singularities governing lubrication effects, our numerical solver can stably predict the pressure fields without invoking any artificial subgrid-scale lubrication models. The convergence of the total recovered error as defined in Eq. (13) for the last time step is shown in Fig. 15(a). We find that the convergence rate reaches the theoretically expected 2nd order.

To thoroughly examine the performance of our proposed preconditioner, we track the number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step, as depicted in Fig. 15(b). Several iterations of adaptive $h$-refinement are needed at each step to reach the preset error tolerance. Hence, the number of GMRES iterations required in each iteration of adaptive $h$-refinement at different time steps is rendered as the height of the bar with different colors and stacked together for various $h$-refinement iterations. For example, the lowest black bar represents the number of GMRES iterations required in the first iteration ($I = 1$) of adaptive $h$-refinement at different time steps; the upper gray bar depicts the number of GMRES iterations needed for the second iteration

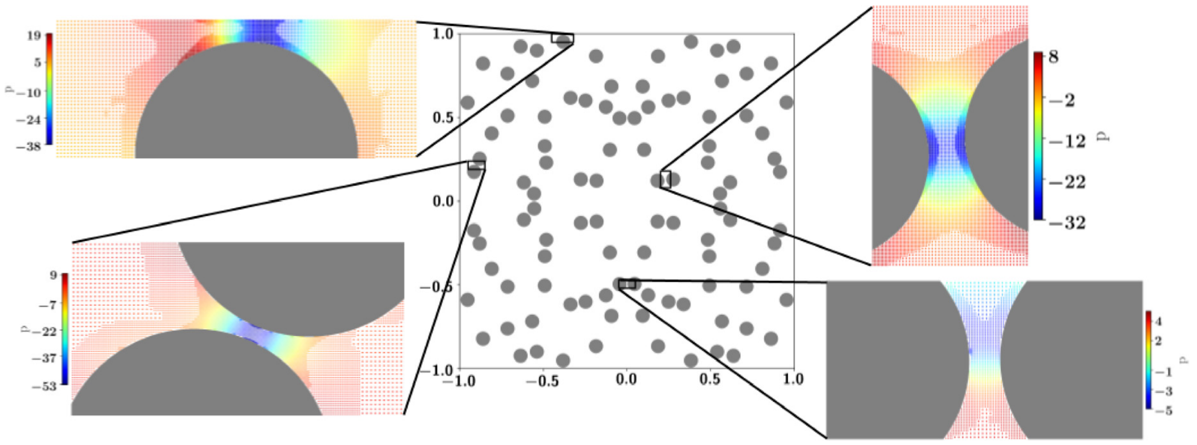
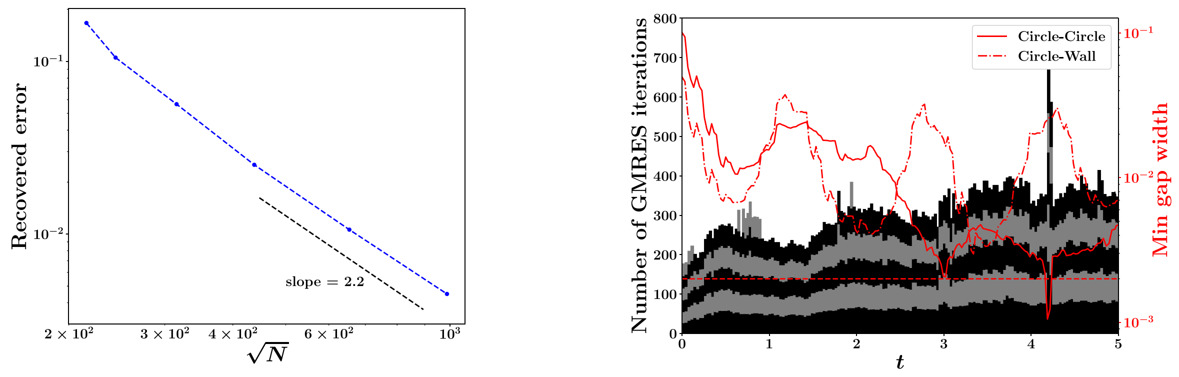


**Fig. 14.** Fluid–solid interactions-Particulate suspensions in 2D: Configuration of 100 freely moving circular particles in a Taylor–Green vortex flow at the terminal time. The zoom-in images are the computed pressure distributions at selected locations where the particles are either in close contact with each other or with the outer wall. Here, the color is correlated to the magnitude of pressure, and the point clouds are the GMLS nodes with adaptive refinement. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Convergence of the total recovered error (Eq. (13)) at the last time step. The slope measured by the last four data points is 2.2. $N$ denotes the total number of GMLS nodes.

(b) Number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step during the entire simulation.

**Fig. 15.** Fluid–solid interactions-Particulate suspensions in 2D-100 similar particles: Convergence of the recovered error and the required number of GMRES iterations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

($I = 2$) of adaptive $h$-refinement at different time steps; and so forth. By comparing the height of each bar at a fixed time step, we can see that the number of GMRES iterations generally stays constant across different iterations of adaptive $h$-refinement, indicating the scalability of the proposed monolithic GMG preconditioner in terms of increasing total DOFs. By comparing the heights of a bar at different time steps, we note that the number of GMRES iterations is highly correlated with the minimum gap width between solid boundaries (particle–particle or particle–wall). To elaborate on that, we add two lines (red) in Fig. 15(b) showing the minimum gap widths between solid boundaries at different time instances. Note that the minimum gap width can be as small as $0.03R$ with $R = 0.04$ the particles' radius. Generally, when the minimum gap widths between solid boundaries are small, more GMRES iterations are required for the linear solver to converge. That is because finer GMLS nodes are needed to resolve narrower gaps between solid boundaries, resulting in more ill-conditioned linear systems to solve. The numbers of required GMRES iterations are comparable when the minimum gap widths are larger than 0.002 (indicated by the horizontal red dash line in Fig. 15(b)), which is equivalent to $0.05R$.

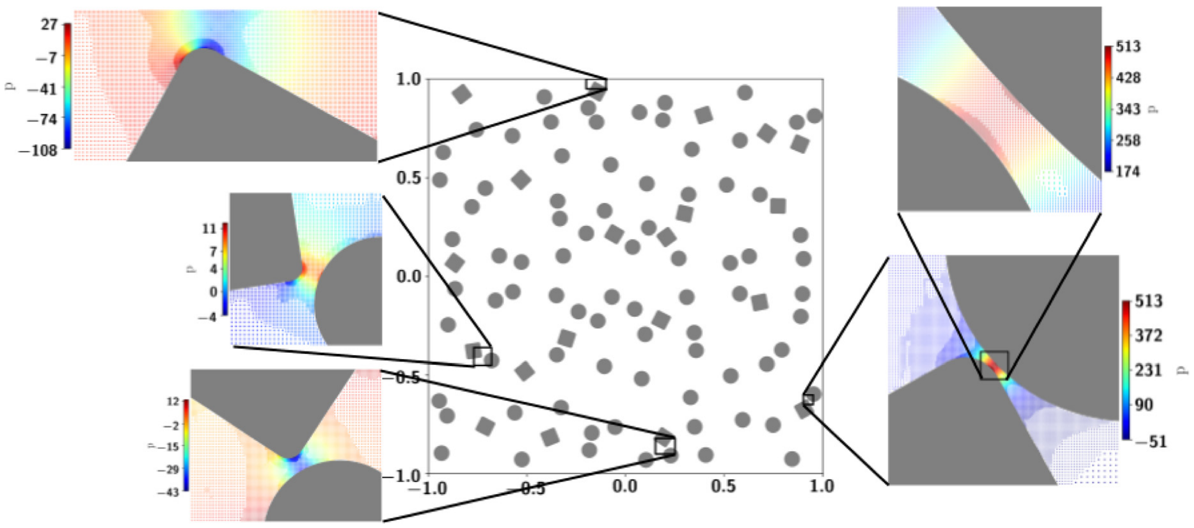
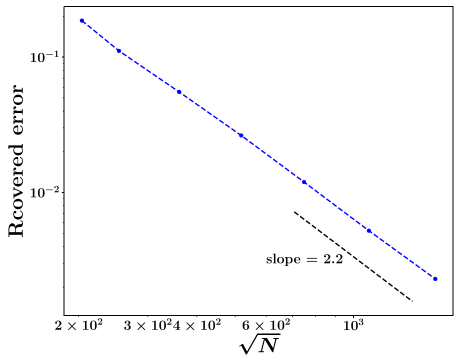
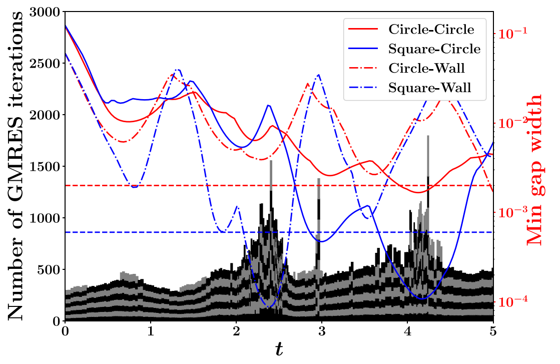


**Fig. 16.** Fluid–solid interactions-Particulate suspensions in 2D: Configuration of 100 freely moving dissimilar particles in a Taylor–Green vortex flow at the terminal time. The zoom-in images are the computed pressure distributions at selected locations where the particles are either in close contact with each other or with the outer wall. Here, the color is correlated to the magnitude of pressure, and the point clouds are the GMLS nodes with adaptive refinement. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Convergence of the total recovered error (Eq. (13)) at the last time step. The slope measured by the last four data points is 2.2. $N$ denotes the total number of GMLS nodes.



(b) Number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step during the entire simulation.

**Fig. 17.** Fluid–solid interactions-Particulate suspensions in 2D-100 dissimilar particles: Convergence of the recovered error and the required number of GMRES iterations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

## *100 dissimilar particles*

In the second simulation, the suspended particles are a mixture of eighty circles and twenty squares. The radius of circular particles is still $R = 0.04$; the side length of square particles is $L = 2R = 0.08$. To mimic the particles in real applications of particulate suspensions, the squares are rounded at the corners with a rounding radius $R' = 0.1L = 0.008$. The snapshot of the particles' configuration at the terminal time $T = 5$ is shown in Fig. 16, where the zoom-in pressure distributions, particularly around square particles, are also shown at selected locations. Regardless of particle shapes, our numerical solver can stably solve the problem even when the particles are in close contact with each other or with the outer wall boundary. No any artificial subgrid-scale lubrication model is employed during the entire simulation. The convergence of the total recovered error as defined in Eq. (13) for the last time step is shown in Fig. 17(a). With the inclusion of different shapes of solids, we still see the convergence rate reaching the theoretically expected 2nd order.

By tracking the number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step, we assess the performance of the proposed preconditioning method. Similarly, we compare the number of GMRES iterations, as depicted in Fig. 17(b). For this case with mixed shapes of solids, more iterations of adaptive $h$-refinement are needed to reach the preset error tolerance at each time step. By comparing the height of each bar at a fixed time step, we also see that the number of GMRES iterations generally stays constant across different iterations of adaptive $h$-refinement, implying the scalability of the proposed monolithic GMG preconditioner in terms of increasing total DOFs. As in the case of 100 similar particles, the number of GMRES iterations is correlated with the minimum gap width between solid boundaries (particle–particle or particle–wall). In addition, the inclusion of square-shaped particles can worsen the problem's conditioning in the continuous limit. To reflect the combined effects of the minimum gap width between solid boundaries and the shape of particles, in Fig. 17(b) we add lines to track the minimum gap widths associated with particles of different shapes (circular or square). We find that the square particles make the dominant contributions to the required GMRES iterations. When the minimum gap widths associated with square particles are larger than 0.0006 (indicated by the horizontal blue dash line in Fig. 17(b)), which is equivalent to $0.075R'$ with $R' = 0.008$ the square particles' corner rounding radius, the number of GMRES iterations required at a given $h$-refinement iteration is generally comparable across different time instances; when they are smaller than 0.0006, significantly more GMRES iterations are required for the linear solver to converge, because the corresponding linear system can become severely ill-conditioned.

### *5.2.3. Particulate suspensions in 3D*

We next perform a long-time simulation to demonstrate that the proposed monolithic GMG preconditioning method can be extended to 3D, with the scalability and robustness maintained. In particular, a 3D suspension flow of 27 freely moving spherical particles is considered. The source term that drives a Taylor–Green vortex flow in 3D is given by:

$$\mathbf{f} = \begin{bmatrix} 3\pi^2\cos(\pi x)\sin(\pi y)\sin(\pi z) + 2\pi\sin(2\pi x) \\ -6\pi^2\sin(\pi x)\cos(\pi y)\sin(\pi z) + 2\pi\sin(2\pi y) \\ 3\pi^2\sin(\pi x)\sin(\pi y)\cos(\pi z) + 2\pi\sin(2\pi z) \end{bmatrix}, \quad \forall \mathbf{x} = (x, y, z) \in \Omega_f \ , \tag{29}$$

and the no-slip BC for the velocity imposed at the outer boundary of the 3D fluid domain ($[-1, 1]\times[-1, 1]\times[-1, 1]$) is specified as:

$$\begin{cases} u = \cos(\pi x)\sin(\pi y)\sin(\pi z) \\ v = -2\sin(\pi x)\cos(\pi y)\sin(\pi z) \ , \quad \forall \mathbf{x} = (x, y, z) \in \Gamma_0 \ . \\ w = \sin(\pi x)\sin(\pi y)\cos(\pi y) \end{cases} \tag{30}$$

The suspended 27 spherical particles with the radius $R = 0.1$ are subject to bidirectional hydrodynamic couplings and freely moving in the flow. Initially, all particles are evenly distributed throughout the domain. The 2nd-order GMLS is employed for spatial discretization with initial discretization resolution at $\Delta x^0 = 0.04$. The 5th-order Runge–Kutta integrator with adaptive time stepping is used for temporal integration to update the particles' positions and orientations, with the initial time step set as $\Delta t = 0.1$. The physical time of the entire simulation is $T = 1$, and the configuration of particles at the last time step is shown in Fig. 18. At each time step, the adaptive $h$-refinement is conducted according to the recovered error estimator (Eq. (12)), with $\alpha = 0.9$, and for three times until the total recovered error close to the preset tolerance $10^{-2}$. The convergence of the total recovered error as defined in Eq. (13) for the last time step is presented in Fig. 19(a). We find that the convergence rate reaches the theoretically expected 2nd order.

For the linear solver, the stopping criterion for the GMRES iteration is set as $10^{-6}$. The performance of the proposed preconditioner is again assessed by tracking the number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step during the entire simulation. Without any modification on the preconditioner, the proposed preconditioning method preserves the attributes seen in the 2D counterparts. As shown in Fig. 19(b), by inspecting the heights of the bars with gray/black colors at each time step, we can see that the number of GMRES iterations almost stay constant across different iterations of adaptive $h$-refinement. This indicates that, with the increasing total DOFs, the scalability attribute of the proposed monolithic GMG preconditioner is preserved when applying it to solving a 3D problem. By examining the variation of the height of each bar with respect to time, we find that the numbers of GMRES iterations are generally comparable across different time steps.

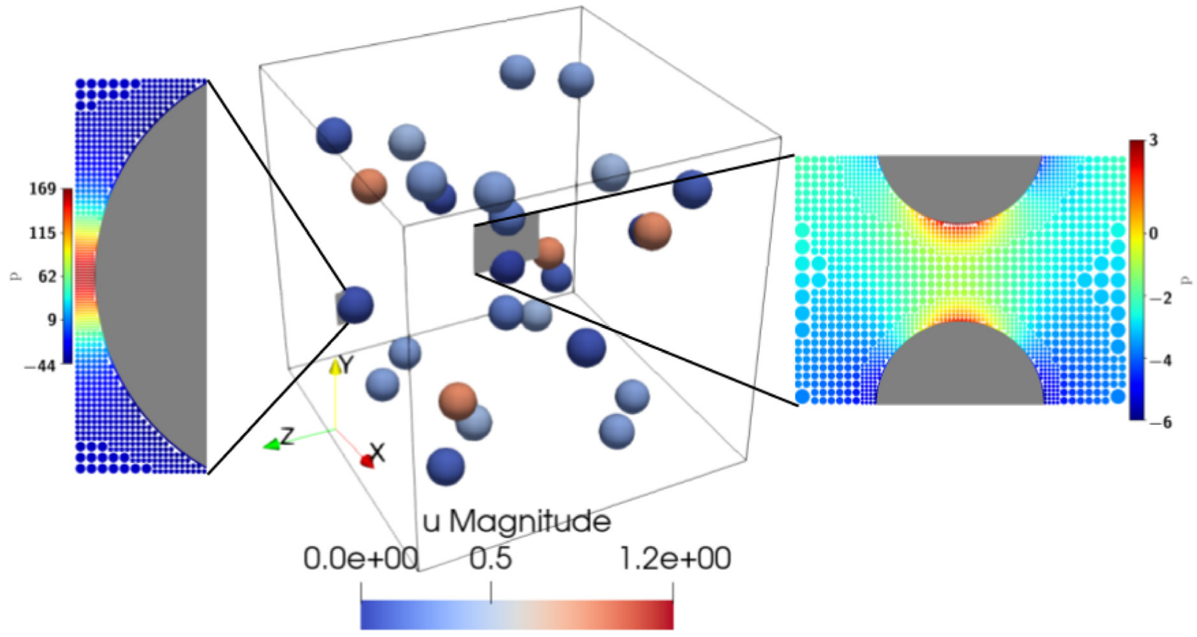
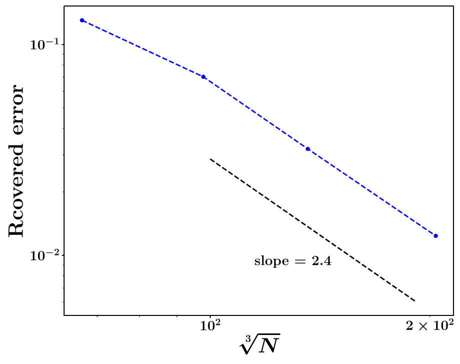
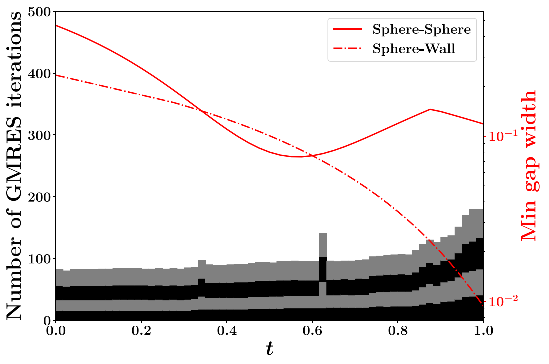


**Fig. 18.** Fluid–solid interactions-Particulate suspensions in 3D: Configuration of 27 freely moving spherical particles in a Taylor–Green vortex flow at the terminal time. The color on each particle is correlated to the magnitude of its velocity. The zoom-in images show the flow-field pressure distributions on selected planes where the particles are either in close contact with each other or with the outer wall. Here, the color is correlated to the magnitude of pressure, and the point clouds are the GMLS nodes with adaptive refinement. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Convergence of the total recovered error (Eq. (13)) at the last time step. The slope measured by the last three data points is 2.4. $N$ denotes the total number of GMLS nodes.



(b) Number of GMRES iterations required in each adaptive $h$-refinement iteration and at each time step during the entire simulation.

**Fig. 19.** Fluid–solid interactions-Particulate suspensions in 3D: Convergence of the recovered error and the required number of GMRES iterations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The slight increase can be correlated to the decreased minimum gap width between solid boundaries (sphere-sphere or sphere-wall). The decreased gaps between solid boundaries require finer GMLS nodes to resolve and hence lead to more ill-conditioned linear system to solve, which in turn calls for more GMRES iterations to reach convergence.

## 6. Conclusions

We have presented a monolithic GMG preconditioner for solving fluid–solid interaction problems in Stokes limit. The linear systems of equations are generated from the spatially adaptive GMLS discretization, which was developed in our previous work [1]. The GMLS discretization is meshless and can handle large displacements and rotations of solid bodies without the expensive cost of generating and managing meshes. It guarantees the same order of accuracy for both the velocity and pressure fields, and high-order accuracy is achievable by its polynomial reconstruction property and choosing appropriate polynomial bases. A staggered discretization approximates the div-grad operator to ensure stable solutions of Stokes equations. With adaptive $h$-refinement directed by an *a posteriori* recovery-based error estimator, it can resolve the singularities governing the lubrication effects between solid bodies without invoking any artificial subgrid-scale lubrication models. With the proposed monolithic GMG preconditioner in this work, we are able to scale up the spatially adaptive GMLS discretization for solving larger-scale fluid–solid interaction problems and achieve scalability with increasing numbers of solid bodies and total DOFs, while preserving accuracy and stability.

The preconditioner is composed of two main ingredients: the interpolation/restriction operators and the smoothers. While the interpolation/restriction operators transfer the velocity and pressure values between different resolution levels, the smoothers damp the high-frequency error components on each resolution level. These two ingredients and their interplay determine the performance and scalability of the preconditioner and, thereby, the linear solver. The hierarchical structure of the adaptively refined GMLS nodes has provided us with an appropriate geometric setting for constructing the interpolation/restriction operators. In particular, to construct the interpolation operator, we employ local GMLS approximations from the coarse-level nodes to the fine-level nodes. To build the restriction operator, we let the value of a variable on a parent GMLS node equal the average value of its child nodes generated in a new iteration of adaptive refinement. During the interpolation and restriction processes, the divergence-free property of velocity is preserved, which plays an essential role in designing efficient MG preconditioning methods for solving Stokes equations. For the smoothers, we build a smoother for the fluid domain and a smoother for the solid bodies and then integrate them following a multiplicative overlapping Schwarz method. We have used a node-wise block Gauss–Seidel smoother for the fluid domain to address the coupling between the two field variables: velocity and pressure. The smoother for the solid bodies handles each solid body separately: for each solid body, a submatrix is first assembled corresponding to the solid body and its neighboring interior GMLS nodes; a Schur complement approach is then employed to approximately invert the submatrix based on the block splitting between the fluid and solid DOFs.

Such a developed preconditioner is integrated with the Krylov iterative solver, GMRES, for solving the linear systems of equations generated from the GMLS discretization. We have leveraged PETSc [28] for the parallel implementation of the proposed monolithic GMG preconditioner. In addition to all associated linear algebra operations handled by PETSc, we have carefully taken care of domain decomposition, neighbor search, data storage, and imposing inhomogeneous Neumann BC for pressure in parallel implementation in order to warrant the parallel scalability of our numerical solver. Through a series of numerical tests, including simulating pure fluid flows and fluid–solid interactions with the inclusion of different numbers and shapes of solid bodies, we have demonstrated the performance and parallel scalability of our proposed preconditioner. Specifically, as the number of solid bodies and total DOFs increases, the convergence of the Krylov iterative solver can be ensured. For a fixed number of solid bodies, our preconditioner scales nearly linearly with respect to the total DOFs. When the number of solid bodies increases, our preconditioner exhibits sublinear optimality with respect to the number of solid bodies. In addition, our parallel implementation has achieved weak scalability for both pure fluid flows and fluid–solid interactions. For the flow in an artificial vascular network, the numerical result shows consistency with the prediction of Amdahl's law, indicating strong scalability and efficiency of our parallel implementation.

The proposed monolithic GMG preconditioning method is generally applicable to 2D or 3D fluid–solid interaction problems. And we have demonstrated its scalability and robustness in both 2D and 3D examples. For solving large-scale 3D problems, we expect that a more efficient parallel implementation is necessary. That would include a better data storage strategy, more advanced domain decomposition and neighbor searching techniques, and parallelization of the monolithic GMG preconditioner accelerated by a hybrid GPU-CPU implementation.

Furthermore, we have noted that the performance of the recovery-based *a posteriori* error estimator in adaptive refinement can be affected by the PDE solutions' low regularity caused by the computational domain's nonconvexity, as discussed in the case of an artificial vascular network. As a result, it can lead to highly ill-conditioned linear systems. Therefore, developing a more theoretically guaranteed and robust *a posteriori* error estimator would be another subject in our future work.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

Z. Y. was supported by the University of Wisconsin - Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation, United States; W. P. was supported by the National Science Foundation, United States under Grant No. CMMI-1761068; X. H. was partially supported by the National Science Foundation under grant CCF-1934553.

## Appendix A. Divergence-free vector polynomial basis

When $d = 2$ and $m = 2$, the divergence-free vector polynomial basis $\mathbf{P}^{\text{div}}(\mathbf{x})$ at particle $i$ is given by:

$$\mathbf{P}_i^{\text{div}}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 & \frac{y-y_i}{\epsilon_i} & \frac{x-x_i}{\epsilon_i} & 0 & (\frac{y-y_i}{\epsilon_i})^2 & (\frac{x-x_i}{\epsilon_i})^2 & -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} \\ 0 & 1 & \frac{x-x_i}{\epsilon_i} & 0 & -\frac{y-y_i}{\epsilon_i} & (\frac{x-x_i}{\epsilon_i})^2 & 0 & -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} & (\frac{y-y_i}{\epsilon_i})^2 \end{bmatrix}^{\mathsf{T}}; \tag{A.1}$$

when $d = 2$ and $m = 4$, it is given by:

$$\begin{aligned}\mathbf{P}_i^{\text{div}}(\mathbf{x}) = \Bigg[ & \begin{matrix} 1 & 0 & 0 & \frac{y-y_i}{\epsilon_i} & \frac{x-x_i}{\epsilon_i} & 0 & (\frac{y-y_i}{\epsilon_i})^2 & (\frac{x-x_i}{\epsilon_i})^2 & -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} \\ 0 & 1 & \frac{x-x_i}{\epsilon_i} & 0 & -\frac{y-y_i}{\epsilon_i} & (\frac{x-x_i}{\epsilon_i})^2 & 0 & -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} & (\frac{y-y_i}{\epsilon_i})^2 \end{matrix} \\ & \begin{matrix} 0 & (\frac{y-y_i}{\epsilon_i})^3 & (\frac{x-x_i}{\epsilon_i})^3 & -3\frac{x-x_i}{\epsilon_i}(\frac{y-y_i}{\epsilon_i})^2 & (\frac{x-x_i}{\epsilon_i})^2\frac{y-y_i}{\epsilon_i} \\ (\frac{x-x_i}{\epsilon_i})^3 & 0 & -3(\frac{x-x_i}{\epsilon_i})^2\frac{y-y_i}{\epsilon_i} & (\frac{y-y_i}{\epsilon_i})^3 & -\frac{x-x_i}{\epsilon_i}(\frac{y-y_i}{\epsilon_i})^2 \end{matrix} \\ & \begin{matrix} 0 & (\frac{y-y_i}{\epsilon_i})^4 & (\frac{x-x_i}{\epsilon_i})^4 & -4\frac{x-x_i}{\epsilon_i}(\frac{y-y_i}{\epsilon_i})^3 & (\frac{x-x_i}{\epsilon_i})^3\frac{y-y_i}{\epsilon_i} & (\frac{x-x_i}{\epsilon_i})^2(\frac{y-y_i}{\epsilon_i})^2 \\ (\frac{x-x_i}{\epsilon_i})^4 & 0 & -4(\frac{x-x_i}{\epsilon_i})^3\frac{y-y_i}{\epsilon_i} & (\frac{y-y_i}{\epsilon_i})^4 & -\frac{x-x_i}{\epsilon_i}(\frac{y-y_i}{\epsilon_i})^3 & -(\frac{x-x_i}{\epsilon_i})^2(\frac{y-y_i}{\epsilon_i})^2 \end{matrix} \Bigg]^{\mathsf{T}} \end{aligned}; \tag{A.2}$$

and when $d = 3$ and $m = 2$, it is given by:

$$\begin{aligned}\mathbf{P}_i^{\text{div}}(\mathbf{x}) = \Bigg[ & \begin{matrix} 1 & 0 & 0 & 0 & 0 & \frac{y-y_i}{\epsilon_i} & 0 & \frac{z-z_i}{\epsilon_i} & 0 \\ 0 & 1 & 0 & \frac{x-x_i}{\epsilon_i} & 0 & 0 & 0 & 0 & \frac{z-z_i}{\epsilon_i} \\ 0 & 0 & 1 & 0 & \frac{x-x_i}{\epsilon_i} & 0 & \frac{y-y_i}{\epsilon_i} & 0 & 0 \end{matrix} \\ & \begin{matrix} \frac{x-x_i}{\epsilon_i} & \frac{x-x_i}{\epsilon_i} & 0 & 0 & (\frac{y-y_i}{\epsilon_i})^2 & 0 & (\frac{z-z_i}{\epsilon_i})^2 & 0 \\ -\frac{y-y_i}{\epsilon_i} & 0 & (\frac{x-x_i}{\epsilon_i})^2 & 0 & 0 & 0 & 0 & (\frac{z-z_i}{\epsilon_i})^2 \\ 0 & -\frac{z-z_i}{\epsilon_i} & 0 & (\frac{x-x_i}{\epsilon_i})^2 & 0 & (\frac{y-y_i}{\epsilon_i})^2 & 0 & 0 \end{matrix} \\ & \begin{matrix} (\frac{x-x_i}{\epsilon_i})^2 & (\frac{x-x_i}{\epsilon_i})^2 & -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} & 0 & -2\frac{x-x_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} & 0 \\ -2\frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} & 0 & (\frac{y-y_i}{\epsilon_i})^2 & (\frac{y-y_i}{\epsilon_i})^2 & 0 & -2\frac{y-y_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} \\ 0 & -2\frac{x-x_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} & 0 & -2\frac{y-y_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} & (\frac{z-z_i}{\epsilon_i})^2 & (\frac{z-z_i}{\epsilon_i})^2 \end{matrix} \\ & \begin{matrix} \frac{y-y_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} & 0 & 0 \\ 0 & \frac{x-x_i}{\epsilon_i}\frac{z-z_i}{\epsilon_i} & 0 \\ 0 & 0 & \frac{x-x_i}{\epsilon_i}\frac{y-y_i}{\epsilon_i} \end{matrix} \Bigg]^{\mathsf{T}} \end{aligned}. \tag{A.3}$$

## Appendix B. Discretization and adaptive refinement in 3D for the surfaces of moving solids

The boundaries of solid bodies in fluid–solid interactions can generally be curved surfaces or manifolds, so generating and refining point clouds on them need additional care. First of all, a tool to generate initial, uniform point clouds in the curved surfaces (e.g., spherical surfaces) needs to be invoked. In this work, we adopt *DistMesh* [40], which can generate a quasi-uniform point cloud with a given discretization resolution via triangulation. The point cloud generated by *DistMesh* with the initial resolution $\Delta x^0$ on the surface of each solid, $\Gamma_n$ $(n = 1, 2, \dots, N_s)$, is denoted as $\mathcal{P}_n^0 = \{\mathcal{V}_n^0, \mathcal{E}_n^0, \mathcal{T}_n^0\}$, where $\mathcal{V}_n^0 = \{\mathbf{x}_i^0 \in \Gamma_n\}$ denotes the set of boundary GMLS nodes on $\Gamma_n$; $\mathcal{E}_n^0 = \{e_{ij}^0 = (\mathbf{x}_i^0, \mathbf{x}_j^0) \mid \mathbf{x}_i^0, \mathbf{x}_j^0 \in \mathcal{V}_n^0\}$ denotes the set of edges determined by *DistMesh* connecting two adjacent nodes; and $\mathcal{T}_n^0 = \{t_{ijk}^0 = (\mathbf{x}_i^0, \mathbf{x}_j^0, \mathbf{x}_k^0) \mid \mathbf{x}_i^0, \mathbf{x}_j^0, \mathbf{x}_k^0 \in \mathcal{V}_n^0; e_{ij}^0, e_{ik}^0, e_{jk}^0 \in \mathcal{E}_n^0\}$ represents the set of triangles formed from three adjacent nodes by *DistMesh*. Starting from $\mathcal{P}_n^0$, the nodes marked for refinement are refined. $\mathcal{P}_n^I$ denotes the point cloud on $\Gamma_n$ resulted from the $I$th iteration of adaptive refinement on $\mathcal{P}_n^{I-1}$.

Specifically, for a node $\mathbf{x}_i^{I-1} \in \mathcal{V}_n^{I-1}$ marked for refinement, a new resolution $\Delta x_{i_{new}} = \frac{1}{2}\Delta x_i$ is assigned to this node, and the midpoints on all edges connected with $\mathbf{x}_i^{I-1}$ are newly generated nodes to be added into $\mathcal{V}_n^I$. For an edge $e_{ij}^{I-1} \in \mathcal{E}_n^{I-1}$, its midpoint $\mathbf{x}_{ij}^{I-1}$ is determined as the intersection between $\Gamma_n$ and the line connecting $\frac{\mathbf{x}_i^{I-1}+\mathbf{x}_j^{I-1}}{2}$ and the origin of the coordinate system defining the surface manifold of the solid body. In the end of refinement, such determined midpoints along with $\mathcal{V}_n^{I-1}$ make up $\mathcal{V}_n^I$. The edge between the midpoint $\mathbf{x}_{ij}^{I-1}$ and the marked node $\mathbf{x}_i^{I-1}$ is regarded as a new edge to be added into $\mathcal{E}_n^I$. If the node $\mathbf{x}_j^{I-1}$ is also marked for refinement, the edge between the midpoint $\mathbf{x}_{ij}^{I-1}$ and $\mathbf{x}_j^{I-1}$ is also added into $\mathcal{E}_n^I$, but the original edge $e_{ij}^{I-1}$ is removed from $\mathcal{E}_n^I$. As for the triangles, when all the three nodes in $t_{ijk}^{I-1}$ are marked for refinement, they along with the midpoints on the three edges form four new triangles to be added into $\mathcal{T}_n^I$, but $t_{ijk}^{I-1}$ is removed from $\mathcal{T}_n^I$. As such, a clear derivation relation between $\mathcal{P}_n^I$ and $\mathcal{P}_n^{I-1}$ is still retained and can be utilized in the interpolation and restriction operations.

For each node $\mathbf{x}_i^I \in \mathcal{V}_n^I$, $\Delta\mathcal{A}_i$ in Eq. (16) is evaluated from the areas of the triangles $t_{ijk}^I \in \mathcal{T}_n^I$ connected to the node. The area for each such triangle is denoted as $\mathcal{A}_{ijk}^I$. $\Delta\mathcal{A}_i$ is then calculated as:

$$\Delta\mathcal{A}_i = \frac{1}{3}\sum_{\{j,k|t_{ijk}^I\}} \mathcal{A}_{ijk}^I \tag{B.1}$$

## References

[1] W. Hu, N. Trask, X. Hu, W. Pan, A spatially adaptive high-order meshless method for fluid–structure interactions, Comput. Methods Appl. Mech. Engrg. 355 (2019) 67–93.
[2] H. Wendland, Scattered Data Approximation, Vol. 17, Cambridge University Press, 2004.
[3] D. Mirzaei, R. Schaback, M. Dehghan, On generalized moving least squares and diffuse derivatives, IMA J. Numer. Anal. 32 (3) (2012) 983–1000.
[4] N. Trask, M. Maxey, X. Hu, A compatible high-order meshless method for the Stokes equations with applications to suspension flows, J. Comput. Phys. 355 (2018) 310–326.
[5] N. Trask, M. Perego, P. Bochev, A high-order staggered meshless method for elliptic problems, SIAM J. Sci. Comput. 39 (2) (2017) A479–A502.
[6] W.K. Liu, S. Jun, S. Li, J. Adee, T. Belytschko, Reproducing kernel particle methods for structural dynamics, Internat. J. Numer. Methods Engrg. 38 (10) (1995) 1655–1679.
[7] W.-K. Liu, S. Li, T. Belytschko, Moving least-square reproducing kernel methods (I) methodology and convergence, Comput. Methods Appl. Mech. Engrg. 143 (1–2) (1997) 113–154.
[8] S. Li, W.K. Liu, Reproducing kernel hierarchical partition of unity, part I—formulation and theory, Internat. J. Numer. Methods Engrg. 45 (3) (1999) 251–288.
[9] S. Li, W.K. Liu, Reproducing kernel hierarchical partition of unity, part II—applications, Internat. J. Numer. Methods Engrg. 45 (3) (1999) 289–317.
[10] T. Belytschko, Y.Y. Lu, L. Gu, Element-free Galerkin methods, Internat. J. Numer. Methods Engrg. 37 (2) (1994) 229–256.
[11] D. Silvester, A. Wathen, Fast iterative solution of stabilised Stokes systems Part II: Using general block preconditioners, SIAM J. Numer. Anal. 31 (5) (1994) 1352–1367.
[12] D. Loghin, A.J. Wathen, Analysis of preconditioners for saddle-point problems, SIAM J. Sci. Comput. 25 (6) (2004) 2029–2049.
[13] C. Siefert, E. de Sturler, Preconditioners for generalized saddle-point problems, SIAM J. Numer. Anal. 44 (3) (2006) 1275–1296.
[14] K.-A. Mardal, R. Winther, Preconditioning discretizations of systems of partial differential equations, Numer. Linear Algebra Appl. 18 (1) (2011) 1–40.
[15] Y. Ma, K. Hu, X. Hu, J. Xu, Robust preconditioners for incompressible MHD models, J. Comput. Phys. 316 (2016) 721–746.
[16] J.H. Adler, F.J. Gaspar, X. Hu, C. Rodrigo, L.T. Zikatanov, Robust block preconditioners for Biot's model, in: International Conference on Domain Decomposition Methods, Springer, 2017, pp. 3–16.
[17] A. Brandt, N. Dinar, Multigrid solutions to elliptic flow problems, in: Numerical Methods for Partial Differential Equations, Elsevier, 1979, pp. 53–147.
[18] A. Brandt, O.E. Livne, Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition, SIAM, 2011.

[19] D. Braess, R. Sarazin, An efficient smoother for the Stokes problem, Appl. Numer. Math. 23 (1) (1997) 3–19.
[20] J. Schöberl, Robust multigrid preconditioning for parameter-dependent problems I: The Stokes-type case, in: Multigrid Methods V, Springer, 1998, pp. 260–275.
[21] C.W. Oosterlee, F.J.G. Lorenz, Multigrid methods for the Stokes system, Comput. Sci. Eng. 8 (6) (2006) 34–43.
[22] A. Janka, Smoothed aggregation multigrid for a Stokes problem, Comput. Vis. Sci. 11 (3) (2008) 169–180.
[23] M. Wang, L. Chen, Multigrid methods for the Stokes equations using distributive Gauss–Seidel relaxations based on the least squares commutator, J. Sci. Comput. 56 (2) (2013) 409–431.
[24] L. Chen, X. Hu, M. Wang, J. Xu, A multigrid solver based on distributive smoother and residual overweighting for Oseen problems, Numer. Math.: Theory Methods Appl. 8 (2) (2015) 237–252.
[25] J. Hron, S. Turek, A monolithic FEM/Multigrid solver for an ALE formulation of fluid-structure interaction with applications in biomechanics, in: Fluid-Structure Interaction, Springer, 2006, pp. 146–170.
[26] S. Turek, J. Hron, M. Madlik, M. Razzaq, H. Wobker, J.F. Acker, Numerical simulation and benchmarking of a monolithic multigrid solver for fluid-structure interaction problems with application to hemodynamics, in: Fluid Structure Interaction II, Springer, 2011, pp. 193–220.
[27] M.W. Gee, U. Küttler, W.A. Wall, Truly monolithic algebraic multigrid for fluid-structure interaction, Internat. J. Numer. Methods Engrg. 85 (8) (2011) 987–1016.
[28] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W.D. Gropp, D. Karpeyev, D. Kaushik, M.G. Knepley, D.A. May, L.C. McInnes, R.T. Mills, T. Munson, K. Rupp, P. Sanan, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Users Manual, Tech. Rep. ANL-95/11 - Revision 3.13, Argonne National Laboratory, 2020.
[29] J.R. Dormand, P.J. Prince, A family of embedded Runge-Kutta formulae, J. Comput. Appl. Math. 6 (1) (1980) 19–26.
[30] L.F. Shampine, M.W. Reichelt, The MATLAB ODE suite, SIAM J. Sci. Comput. 18 (1) (1997) 1–22.
[31] A. Brandt, Multilevel adaptive computations in fluid dynamics, AIAA J. 18 (10) (1980) 1165–1172.
[32] Zoltan2 Project Team, The Zoltan2 Project Website.
[33] P. Kuberry, P. Bosler, N. Trask, Compadre toolkit, 2019.
[34] J.L. Blanco, P.K. Rai, Nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees, 2014.
[35] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517.
[36] Y. Alvarez, M.L. Cederlund, D.C. Cottell, B.R. Bill, S.C. Ekker, J. Torres-Vazquez, B.M. Weinstein, D.R. Hyde, T.S. Vihtelic, B.N. Kennedy, Genetic determinants of hyaloid and retinal vasculature in zebrafish, BMC Dev. Biol. 7 (1) (2007) 1–17.
[37] J. Xu, Z. Zhang, Analysis of recovery type a posteriori error estimators for mildly structured grids, Math. Comp. 73 (247) (2004) 1139–1152.
[38] S.C. Brenner, L.R. Scott, The Mathematical Theory of Finite Element Methods, Vol. 3, Springer., 2008.
[39] R.E. Bryant, O. David Richard, O. David Richard, Computer Systems: A Programmer's Perspective, Vol. 2, Prentice Hall, Upper Saddle River, 2003.
[40] P.-O. Persson, G. Strang, A simple mesh generator in MATLAB, SIAM Rev. 46 (2) (2004) 329–345.