Journal of Computational Physics 464 (2022) 111366

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Parallel-in-time simulation of biofluids

Weifan Liu [a], Minghao W. Rostami [b,*]

[a] Department of Mathematics, 215 Carnegie Building, Syracuse University, Syracuse, NY 13244, United States of America
[b] Department of Mathematics and BioInspired Syracuse, Syracuse University, Syracuse, NY 13244, United States of America

ARTICLE INFO

Article history:
Received 13 January 2022
Received in revised form 10 May 2022
Accepted 30 May 2022
Available online 2 June 2022

Keywords:
Parareal
Lagrangian extrapolation
Richardson extrapolation
Biofluid
Fluid-structure interaction
Method of regularized Stokeslets

ABSTRACT

We extend Parareal, a parallel-in-time method that alternates between a serial sweep and a parallel sweep, to simulate the fluid flow around bio-inspired, dynamic structures over a period of time. Our main contributions include demonstrating the applicability of Parareal to the simulation of biofluids and developing novel solvers for the serial sweeps of Parareal. We propose to construct non-intrusive solvers by extrapolating a parametrized family of existing solvers. Compared to the existing solvers, they either allow the use of larger time steps, have a higher order of accuracy in time, or both. They are also straightforward to implement and parallelize. Numerical results show that when the number of biological structures is small or the number of computer cores employed is sufficiently large, the proposed variant of Parareal can achieve a significantly higher parallel speedup than the more commonly used spatial parallelization.

1. Introduction

The goal of this paper is to develop a parallel-in-time method for simulating the fluid flow around bio-inspired, dynamic structures. As for many other problems in scientific computing, we observe that the parallel speedup "saturates" as the number of computer cores increases if spatial parallelization alone is used. We aim to take better advantage of the available computing resources by parallelizing the simulation in the time domain.

The first parallel-in-time method was proposed in [1] almost sixty years ago. In the past two decades, parallel-in-time methods have become increasingly more popular, as evidenced by a surge in the number of publications on them.[1] Several review papers on parallel-in-time methods are available [2–5]. In [4], the author divided parallel-in-time methods into four categories: the shooting type methods, the domain decomposition and waveform relaxation methods, the space-time multigrid methods, and the direct time parallel solvers. In their very recent review article [5], the authors summarized the applications of parallel-in-time methods as well as their speedup and efficiency reported in the literature. It should be noted that spatial parallelization and temporal parallelization are not mutually exclusive and they can both be utilized in solving a large-scale problem to achieve the highest speedup (see [6], for example).

Our approach is based on the Parareal algorithm [7], which has been shown in [8] to be a variant of both the multiple shooting method and the time-multigrid method developed earlier. It is an iterative method for solving a system of Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs) that "sweeps" the time domain multiple times. It alternates between a "serial sweep" and a "parallel sweep" and employs a coarse solver and a fine solver for the two

* Corresponding author.
E-mail addresses: wliu07@syr.edu (W. Liu), mwrostam@syr.edu (M.W. Rostami).

[1] See https://parallel-in-time.org/references/.

https://doi.org/10.1016/j.jcp.2022.111366
0021-9991/

processes, respectively. Compared to the fine solver, the coarse solver either employs numerical methods of lower orders of accuracy, uses coarser temporal and/or spatial discretization, or both. The convergence and stability of Parareal have been analyzed in [7,9–13]. In [14], the author studied the scheduling of tasks in Parareal and proposed ways to improve its speedup. A number of variants of Parareal have been developed over the years, including the Krylov-subspace-enhanced Parareal [15], the hybrid Parareal-Spectral Deferred Correction (SDC) method [16], the Parallel Full Approximation Scheme in Space and Time (PFASST) method [17], ParaExp (which uses an exponential integrator) [18], and the Multigrid Reduction In Time (MGRIT) method [19]. Parareal and its variants have been applied to a wide range of problems including molecular dynamics [20,21], American put [22] and options [23], fluid-structure interactions [9,24], Navier-Stokes equations [25–29], non-Newtonian flows [30], turbulent flows [31], N-body problems [6], and robotics [32].

In the simulation of a fluid flow around bio-inspired, dynamic structures, the structures are typically represented by a number of Lagrangian grid points whose dynamics can be described by a system of nonlinear ODEs. Applying an ODE solver to it entails evaluating the flow field at every time step using a solver for fluid-structure interactions, such as the Method of Regularized Stokeslets (MRS) [33,34], the immersed boundary method [35,36], the boundary integral equation method [37], and the Rotne-Prager-Yamakawa tensor [38,39]. In this work, we focus on swimmers in the regime of zero Reynolds number where the fluid dynamics can be described accurately by the incompressible Stokes equations. We therefore use the MRS, a Lagrangian method that does not require an Eulerian grid for the fluid domain, to resolve fluid-structure interactions in the ODE solver. While it is not uncommon to parallelize the calculation of fluid-structure interactions in the spatial domain (see [40–43], for example), to our knowledge, temporal parallelization of the simulation of fluid flows around bio-inspired, dynamic structures has not been considered yet.

The speedup of Parareal depends greatly on how the computational cost of the coarse solver compares to that of the fine solver and how many Parareal iterations are needed to produce an accurate enough solution. The cheaper the coarse solver is and the smaller the required number of Parareal iterations is, the higher the speedup of Parareal becomes. However, we encounter the following "bottleneck" when applying the Parareal equipped with an "off-the-shelf" coarse solver: in order to approximately maintain the size of the structures during the simulation, a very small time step must be taken by the solver, which limits the efficiency of the solver and hence the speedup of Parareal. It is well known that simulations of elastic structures interacting with a fluid often have these restrictions on time step. For example, the stiffness of the immersed boundary method and the numerical schemes to remedy such restrictions have been analyzed and developed [44–47]. We propose to construct novel coarse solvers by extrapolating members of a parametrized family of existing solvers. The new solvers either allow for the use of larger time steps, have a higher order of accuracy in time, or both compared to the existing ones. By construction, they are also straightforward to implement and parallelize. Using a larger time step makes the coarse solver computationally less costly and raising its order of accuracy reduces the required number of Parareal iterations, both of which increase the speedup of Parareal.

The rest of the paper is organized as follows. In Section 2, we briefly review the main "ingredients" of the simulation of bio-inspired, dynamic structures immersed in a viscous fluid. In Section 3, we briefly review the Parareal algorithm. In Section 4, we describe how to construct novel coarse solvers by extrapolating members of a parametrized family of existing solvers. We apply the Parareal equipped with the proposed coarse solver to simulate the dynamics of a rod-like swimmer and a spherical swimmer and present the numerical results, including the accuracy, speedup, scaling of Parareal as well as its comparison with spatial parallelization, in Section 5. A few concluding remarks are given in Section 6.

2. Review: numerical simulation of dynamic, elastic structures immersed in a viscous fluid

We introduce the system of ODEs arising from the simulation of a biofluid and briefly review how to solve it. In particular, the MRS, which is the method for calculating fluid-structure interactions in this work, is reviewed in Section 2.3.

2.1. Modeling and discretizing the structures

Each structure is represented by a number of Lagrangian grid points connected by a network of springs. The elastic energy in the springs causes the grid points to "push" the surrounding fluid, which in turn deforms and propels the structures. In Fig. 1, we show a rod-like swimmer inspired by biological structures such as bacterial flagella and tails of sperm and a spherical swimmer inspired by the membrane of a vesicle. The grid used to discretize each structure is shown as well.

2.2. The system of ODEs

Assume that the structures are discretized by N_σ grid points. For $i = 1,\ 2,\ \cdots,\ N_\sigma$, let $\mathbf{x}_i(t)$ and $\mathbf{u}_i(t)$ denote the location and velocity of the ith point at time t, respectively. To track $\{\mathbf{x}_i(t)\}_{i=1}^{N_\sigma}$ over the period of time $[0, T]$, we have to solve the following system of Initial Value Problems (IVPs):

$$\text{For } i = 1,\ 2,\ \cdots,\ N_\sigma\text{: } \frac{d\mathbf{x}_i}{dt} = \mathbf{u}_i(t) \text{ for } t \in (0, T] \text{ and } \mathbf{x}_i(0) = \mathbf{x}_{i,0}. \tag{1}$$

Let $\mathbf{u}(t, \mathbf{x})$ denote the fluid velocity at a location $\mathbf{x}$ and time t. We impose the non-slip condition so that $\mathbf{u}_i(t) = \mathbf{u}(t, \mathbf{x}_i)$.

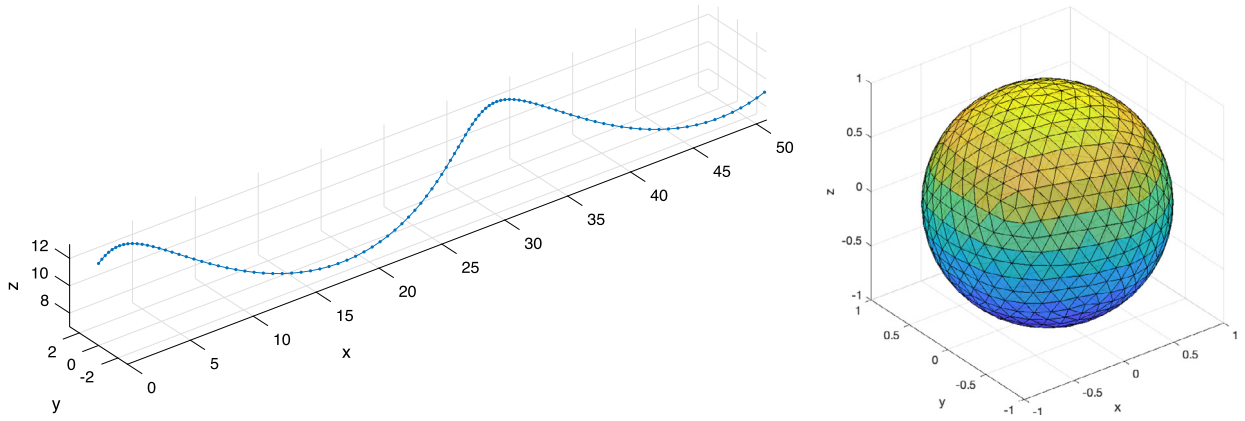

Fig. 1. Discretized rod-like swimmer (left) and spherical swimmer (right). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Eq. (1) can be solved by an ODE solver, such as forward Euler method, in which a solver for fluid-structure interactions needs to be invoked to evaluate $\{\mathbf{u}_i(t)\}_{i=1}^{N_\sigma}$ at every step. We partition the time domain $[0, T]$ uniformly into N_τ subintervals of length $\tau = T/N_\tau$ and let $t_\ell = \ell \cdot \tau$ for $\ell = 0,\ 1,\ 2,\ \cdots,\ N_\tau$. In addition, for $i = 1,\ 2,\ \cdots,\ N_\sigma$, let $\mathbf{f}_i(t)$ denote the force exerted by the ith grid point to the surrounding fluid at time t, and let $\mathbf{x}_{i,\ell}$, $\mathbf{f}_{i,\ell}$, $\mathbf{u}_{i,\ell}$ denote the approximations to $\mathbf{x}_i(t_\ell)$, $\mathbf{f}_i(t_\ell)$, $\mathbf{u}_i(t_\ell)$ respectively, which are calculated in the ODE solver. Here is an outline of the nth step of forward Euler method applied to Eq. (1).

1. Calculate the forces $\left\{\mathbf{f}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ that minimize the elastic energy at time $t_{\ell-1}$, which depends on the locations $\left\{\mathbf{x}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ of the grid points, the material properties of the structures, and their preferred shapes.
2. Calculate the fluid velocities $\left\{\mathbf{u}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ at $\left\{\mathbf{x}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ induced by $\left\{\mathbf{f}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$.
3. Calculate the location $\mathbf{x}_{i,\ell}$ as $\mathbf{x}_{i,\ell-1} + \tau \cdot \mathbf{u}_{i,\ell-1}$ for $i = 1,\ 2,\ \cdots,\ N_\sigma$.

Remark 1. Although the above three steps have to be performed in the order that they appear, each one of them can be parallelized since the calculation that needs to be performed for one grid point is independent of that for another. For example, once all of $\left\{\mathbf{f}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ are known, $\left\{\mathbf{u}_{i,\ell-1}\right\}_{i=1}^{N_\sigma}$ can be calculated in parallel. We refer to this fashion of parallelization as spatial parallelization and will compare the speedups of spatial parallelization and temporal parallelization in Section 5.

2.3. Modeling fluid-structure interactions

The fluid surrounding micro-swimmers such as bacterial flagella is in the regime of zero Reynolds number and can be accurately described by the incompressible Stokes equations:

$$\mathbf{0} = -\nabla p + \mu \Delta \mathbf{u} + \mathbf{f}^{\mathrm{s}} \tag{2a}$$

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

where μ is the viscosity of the fluid, $\mathbf{u} = \mathbf{u}(\mathbf{x}) \in \mathbb{R}^3$, $p = p(\mathbf{x}) \in \mathbb{R}$ denote the velocity, pressure at a point $\mathbf{x} \in \mathbb{R}^3$ respectively, and $\mathbf{f}^{\mathrm{s}} \in \mathbb{R}^3$ is a forcing term. When there is a single point force $\mathbf{f} \in \mathbb{R}^3$ at $\mathbf{y}$, $\mathbf{f}^{\mathrm{s}} = \mathbf{f}\delta(r)$ where δ is the Dirac Delta distribution centered at 0 and r denotes the distance $\|\mathbf{x} - \mathbf{y}\|$ between $\mathbf{x}$ and $\mathbf{y}$. The analytic solution to Eqs. (2a)-(2b), referred to as a Stokeslet, can be found as an expression in terms of $\mathbf{f}$, $\mathbf{x}$ and $\mathbf{y}$. However, this solution is singular at $\mathbf{y}$, where the point force is located, and thus does *not* allow for evaluating the velocities at the points on the structures.

In our simulation, we use the Method of Regularized Stokeslets (MRS) [33,34] to solve Eqs. (2a)-(2b) instead, which replaces the Dirac Delta distribution δ in the forcing term $\mathbf{f}^{\mathrm{s}}$ in Eq. (2a) by a radially symmetric, smooth approximation to it referred to as a blob function. A commonly used family of blob functions is

$$\phi_\epsilon(r) = \frac{15\epsilon^4}{8\pi\,(r^2 + \epsilon^2)^{7/2}} \tag{3}$$

where $\epsilon > 0$ is approximately the width of the blob. As ϵ approaches zero, ϕ_ϵ converges to δ in the distribution sense. With this modification, the analytic solution to Eqs. (2a)-(2b) is given by

$$\mathbf{u}(\mathbf{x}) = \frac{1}{\mu}\left(H_1(r)\mathbf{f} + H_2(r)\,(\mathbf{f}\cdot(\mathbf{x}-\mathbf{y}))\,(\mathbf{x}-\mathbf{y})\right), \tag{4}$$

where H_1 and H_2 are radially symmetric, smooth functions whose precise forms depend on the choice of ϕ_ϵ. Eq. (4) is referred to as a regularized Stokeslet and is defined at *any* $\mathbf{x} \in \mathbb{R}^3$, whether $\mathbf{x}$ is on the structures or not.

Recall that in the nth step of forward Euler method outlined in Section 2.1, we need to calculate the velocities $\{\mathbf{u}_{i,\ell-1}\}_{i=1}^{N_\sigma}$ of N_σ grid points located at $\{\mathbf{x}_{i,\ell-1}\}_{i=1}^{N_\sigma}$, which are induced by the forces $\{\mathbf{f}_{i,\ell-1}\}_{i=1}^{N_\sigma}$ that they exert to the fluid. Since all the quantities above correspond to the same time point, we drop the subscript $\ell-1$ from now on to simplify the notations. Therefore, in Eq. (2a), we let

$$\mathbf{f}^{\mathrm{s}} = \sum_{i=1}^{N_\sigma} \mathbf{f}_i \phi_\epsilon(r_i) \tag{5}$$

where $r_i = \|\mathbf{x} - \mathbf{x}_i\|$ is the distance between $\mathbf{x}$ and $\mathbf{x}_i$. Since Eqs. (2a)-(2b) are linear, when $\mathbf{f}^{\mathrm{s}}$ is given by Eq. (5), their analytic solution is simply the sum of the regularized Stokeslets corresponding to the individual forces, that is,

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{N_\sigma} \frac{1}{\mu} \left(H_1(r_i)\mathbf{f}_i + H_2(r_i)\left(\mathbf{f}_i \cdot (\mathbf{x} - \mathbf{x}_i)\right)(\mathbf{x} - \mathbf{x}_i) \right) \tag{6}$$

for any $\mathbf{x} \in \mathbb{R}^3$. Eqs. (4) and (6) are only valid when the fluid domain is $\mathbb{R}^3$. For a fluid bounded by an infinite, planar, and stationary wall, the regularized solution to Eqs. (2a)-(2b) has been derived in [48] using the method of images.

Eq. (6) implies that the computational cost of evaluating $\{\mathbf{u}_i\}_{i=1}^{N_\sigma}$ is $O\left(N_\sigma^2\right)$. Consequently, if an explicit ODE solver is applied to march forward in time and the MRS is applied to calculate fluid velocity, the computational cost of solving Eq. (1) is $O\left(N_\sigma^2 \cdot N_\tau\right)$.

Remark 2. Since the blob function ϕ_ϵ, roughly speaking, has the effect of "spreading" a point force over a sphere centered at the point with radius ϵ, the MRS can model structures with a non-zero thickness even when their numerical representations have no thickness. For example, a slender rod can be modeled by placing regularized point forces along its centerline. The value of ϵ can be chosen so that results of the MRS best match the theory [34] or experimental data [49]. Since different values of ϵ may be used in different applications, we consider a range of ϵ in the numerical experiments (see Section 5).

3. Review: the Parareal algorithm

The Parareal algorithm was introduced in [7] and has become a popular method for parallelizing the numerical solution of ODEs and PDEs in the time domain. For simplicity, consider the following IVP:

$$\frac{dx}{dt} = u(t,x) \ \text{ for } \ t \in (0,T] \ \text{ and } \ x(0) = x_0. \tag{7}$$

Assume that there are N_{c} computer cores at our disposal. We partition the time domain $[0,T]$ uniformly into N_{c} subintervals of length $\Delta T = T/N_{\mathrm{c}}$ and let $T_n = n\Delta T$ for $n = 0,\ 1,\ 2,\ \cdots,\ N_{\mathrm{c}}$. If $\{x(T_{n-1})\}_{n=1}^{N_{\mathrm{c}}}$ were readily available, Eq. (7) could be split into the following N_{c} independent IVPs:

$$\text{For } n = 1,\ 2,\ \cdots,\ N_{\mathrm{c}}: \ \frac{dx}{dt} = u(t,x) \ \text{ for } \ t \in (T_{n-1}, T_n] \ \text{ and } \ x(T_{n-1}) = x_{n-1}, \tag{8}$$

which could then be assigned to the N_{c} cores and solved in parallel. In practice, it is unlikely that $\{x(T_{n-1})\}_{n=1}^{N_{\mathrm{c}}}$ are available. Parareal is an iterative method that, loosely speaking, alternates between a parallel sweep for solving the IVPs in Eq. (8) concurrently where $\{x(T_{n-1})\}_{n=1}^{N_{\mathrm{c}}}$ have been estimated in the previous serial sweep, and a serial sweep for correcting the solutions obtained in the previous parallel sweep sequentially. The two processes employ a fine solver and a coarse solver respectively, the latter of which is less time-consuming albeit less accurate. For example, we can use forward Euler method as the coarse solver and a Runge-Kutta method with the same or a smaller time step as the fine solver. More details of Parareal are provided below.

Let $\mathcal{G}(t_{\mathrm{End}}, t_{\mathrm{Start}}, X_{\mathrm{Start}})$ and $\mathcal{F}(t_{\mathrm{End}}, t_{\mathrm{Start}}, X_{\mathrm{Start}})$ denote the approximations to $x(t_{\mathrm{End}})$ obtained by the coarse and fine solvers respectively subject to the initial condition $x(t_{\mathrm{Start}}) = X_{\mathrm{Start}}$, where $0 \le t_{\mathrm{Start}} < t_{\mathrm{End}} \le T$. Parareal computes a sequence of approximations $X_n^0,\ X_n^1,\ X_n^2,\ X_n^3,\ \cdots$ to $\mathcal{F}(T_n, 0, x_0)$ for $n = 1,\ 2,\ \cdots,\ N_{\mathrm{c}}$. It starts by computing the initial guess X_n^0 to $x(T_n)$ as

$$X_n^0 = \mathcal{G}(T_n, T_{n-1}, X_{n-1}^0) \tag{9}$$

for $n = 1,\ 2,\ \cdots,\ N_{\mathrm{c}}$, where $X_0^0 = x_0$ is given by Eq. (7). Since X_n^0 depends on X_{n-1}^0, these initial guesses have to be calculated sequentially. The kth iteration of Parareal for $k = 1,\ 2,\ 3, \cdots$ consists of the following two steps:

1. For $n = k,\ k+1,\ k+2,\ \cdots,\ N_c$, compute the approximate solution

$$X_n^{k'} = \mathcal{F}\left(T_n, T_{n-1}, X_{n-1}^{k-1}\right). \tag{10}$$

2. Let $X_n^k = X_n^{k-1}$ for $n = 1,\ 2,\ \cdots,\ k-1$ and $X_k^k = X_k^{k'}$. For $n = k+1,\ k+2,\ \cdots,\ N_c$, compute $\mathcal{G}\left(T_n, T_{n-1}, X_{n-1}^k\right)$ and obtain the kth iterate by correcting $X_n^{k'}$ as follows:

$$X_n^k = X_n^{k'} + \underbrace{\mathcal{G}\left(T_n, T_{n-1}, X_{n-1}^k\right) - \mathcal{G}\left(T_n, T_{n-1}, X_{n-1}^{k-1}\right)}_{\text{correction term}}. \tag{11}$$

Note that $\mathcal{G}\left(T_n, T_{n-1}, X_{n-1}^{k-1}\right)$ in Eq. (11) has been computed before the kth iteration and is thus readily available. In addition, by mathematical induction, one can show that $X_n^k = \mathcal{F}(T_n, 0, x_0)$ for $n = 1,\ 2, \cdots, k$. Since it is already as accurate as it can get when $\mathcal{F}$ is fixed, we only compute Eq. (11) for $n \geq k+1$.

The approximate solutions $\left\{X_n^{k'}\right\}_{n=k}^{N_c}$ can be calculated in parallel since they are independent of one another, whereas the actual iterates $\left\{X_n^k\right\}_{n=k+1}^{N_c}$ have to be calculated in serial as $\mathcal{G}\left(T_n, T_{n-1}, X_{n-1}^k\right)$ and hence X_n^k do depend on X_{n-1}^k. Let $\gamma_{\mathcal{G}}$ and $\gamma_{\mathcal{F}}$ denote the time that it takes to calculate $\mathcal{G}(T_n, T_{n-1}, X)$ and $\mathcal{F}(T_n, T_{n-1}, X)$, respectively. Let N_{it} be the number of Parareal iterations performed. Then the total runtime of Parareal is given by

$$\Upsilon_{pr} = \sum_{k=0}^{N_{it}} (N_c - k)\cdot\gamma_{\mathcal{G}} + N_{it}\cdot\gamma_{\mathcal{F}} + \Upsilon_{oh} = \frac{1}{2}(N_{it}+1)(2N_c - N_{it})\cdot\gamma_{\mathcal{G}} + N_{it}\cdot\gamma_{\mathcal{F}} + \Upsilon_{oh}, \tag{12}$$

where Υ_{oh} is the overhead associated with parallel computing. The runtime required to compute $\mathcal{F}(T, 0, x_0)$ without parallelization is

$$\Upsilon_{\mathcal{F}} = N_c\cdot\gamma_{\mathcal{F}}. \tag{13}$$

We define the parallel speedup of Parareal to be

$$\frac{\Upsilon_{\mathcal{F}}}{\Upsilon_{pr}} = \frac{1}{\frac{1}{2}(N_{it}+1)(2 - N_{it}/N_c)\cdot\gamma_{\mathcal{G}}/\gamma_{\mathcal{F}} + N_{it}/N_c + \Upsilon_{oh}/(N_c\cdot\gamma_{\mathcal{F}})}, \tag{14}$$

an upper bound of which is given by

$$\frac{2}{(N_{it}+1)(2 - N_{it}/N_c)}\cdot\frac{\gamma_{\mathcal{F}}}{\gamma_{\mathcal{G}}}. \tag{15}$$

This bound implies that when the number of cores N_c and fine solver $\mathcal{F}$ are fixed, we can improve the speedup of Parareal by reducing $\gamma_{\mathcal{G}}$ and/or N_{it}. In the next section, we propose one strategy for reducing $\gamma_{\mathcal{G}}$ and another for improving the accuracy of $\mathcal{G}$, the latter of which will in turn reduce N_{it}.

Remark 3. In the original Parareal described above, there is no parallel computing during a serial sweep, and a parallel sweep does not start until the previous serial sweep is finished. Therefore, at any point during a serial sweep, all but one core are idle. A pipelined Parareal was proposed in [16] that put the idle cores to work as well: a core starts to calculate $X_n^{k+1'}$ by Eq. (10) immediately after it has finished evaluating X_n^k by Eq. (9) or (11) instead of waiting for all of $\left\{X_n^k\right\}_{n=k+1}^{N_c}$ to be computed. The serial and parallel sweeps are therefore "interwoven" rather than two separate, alternating processes. In Section 4, we propose a completely different way of utilizing multiple cores during serial sweep. The comparison of the two is one of the future directions of this work.

4. Novel coarse solvers constructed via extrapolation

One major challenge in the numerical simulation of a biofluid is that in order to approximately maintain the sizes of biological structures, such as the length of the rod and the surface area of the vesicle shown in Fig. 1, a small time step must be taken when solving Eq. (1). This constraint limits how coarse the coarse solver for the serial sweep can be, which in turn limits the parallel speedup that can be achieved by Parareal. Building on a family of existing solvers, we develop new coarse solvers that are straightforward to implement and parallelize. Compared to the existing solvers, they either permit the use of larger time steps (Section 4.2), have a higher order of accuracy in time (Section 4.3), or both (Section 4.4). Although we focus on particle tracking in the simulation of a biofluid, the proposed approaches for constructing new solvers are applicable in a wide range of problems. An outline of Parareal equipped with a proposed coarse solver is given in Section 4.5.

4.1. Notations and assumptions

Assume that the fine solver, $\mathcal{F}$, for Eq. (1) is fixed. Let $\mathcal{G}_{\Theta,\tau}$ denote any member of a parameterized family of coarse solvers for Eq. (1), where $\Theta = \{\theta_i\}_{i=1}^{N_\theta}$ is a set of N_θ physical parameters and τ is the time step. The solvers in this family only differ in the values of τ and $\{\theta_i\}_{i=1}^{N_\theta}$. In the simulation of a biofluid, Θ may include fluid viscosity, parameters that characterize the material of the biological structures, and the regularization parameter ϵ used in the MRS which is a proxy for the thickness of the structures (see Section 2.3).

Let $\Theta^0 = \left\{\theta_i^0\right\}_{i=1}^{N_\theta}$ denote the set of correct parameter values determined by the application. On one hand, for the simulation to be realistic, we should choose $\Theta = \Theta^0$. On the other hand, if we disregard the physical meaning of these parameters and simply view them as parameters of a solver, we observe that a different choice of Θ may relax the constraint on τ and hence reduces the runtime of the solver. We assume that Θ^0 is always used in $\mathcal{F}$. For $i = 1,\ 2,\ \cdots,\ N_\theta$, let $\left\{\theta_i^j\right\}_{j=1}^{N_{\theta_i}}$ be a set of N_{θ_i} values of θ_i that satisfy the following:

1. $\theta_i^1 < \theta_i^2 < \cdots < \theta_i^{N_{\theta_i}}$,
2. either $\theta_i^0 < \theta_i^1$ or $\theta_i^{N_{\theta_i}} < \theta_i^0$, that is, θ_i^0 is outside of the interval $\left[\theta_i^1, \theta_i^{N_{\theta_i}}\right]$, and
3. using θ_i^j instead of θ_i^0 allows the solver to take larger time steps.

Eq. (1) is a system of $3N_\sigma$ ODEs since N_σ grid points are used to discretize the structures in three dimensions. For simplicity, in Sections 4.2 to 4.4, we describe our methods for a scalar ODE defined on the time domain $[0, T]$ instead. Let $S(\Theta)$ denote the exact solution to the ODE for a given set of parameter values. Thus, $S\left(\Theta^0\right)$ is the true solution that we seek. As in Section 3, let t_{Start} and t_{End} be two time points satisfying $0 \le t_{\text{Start}} < t_{\text{End}} \le T$. Let $\mathcal{F}(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}})$ and $\mathcal{G}_{\Theta,\tau}(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}})$ denote the approximate solutions to this ODE at $t = t_{\text{End}}$ calculated by $\mathcal{F}$ and $\mathcal{G}_{\Theta,\tau}$ respectively, subject to the initial condition that the solution at $t = t_{\text{Start}}$ is X_{Start}.

4.2. Extrapolation with respect to physical parameters

We could reduce the runtime of the coarse solver $\mathcal{G}_{\Theta,\tau}$ by choosing a value for θ_i that is different from its correct value θ_i^0 and allows the use of a larger time step τ. However, since θ_i is a physical parameter, varying it naïvely can cause the simulation to be non-physical; that is, we would be solving a wrong problem faster. We propose a coarse solver that is built on a sequence of $\mathcal{G}_{\Theta,\tau}$ using values of θ_i other than θ_i^0.

To illustrate the main idea, we first consider the case where only one physical parameter is allowed to vary in $\mathcal{G}_{\Theta,\tau}$, that is, $N_\theta = 1$, $\Theta = \theta_1$, $\mathcal{G}_{\Theta,\tau} = \mathcal{G}_{\theta_1,\tau}$, and $S(\Theta) = S(\theta_1)$. By the approximation properties of Lagrangian polynomial (see Theorem 5.2.3 on p. 108 of [50], for example), if S is a sufficiently smooth function of θ_1, then

$$S\left(\theta_1^0\right) - \sum_{j=1}^{N_{\theta_1}} w_1^j \cdot S\left(\theta_1^j\right) = \frac{S^{(N_{\theta_1})}(\xi)}{N_{\theta_1}!}\left(\theta_1^0 - \theta_1^1\right)\left(\theta_1^0 - \theta_1^2\right)\cdots\left(\theta_1^0 - \theta_1^{N_{\theta_1}}\right) \tag{16}$$

where $\xi \in \left[\theta_i^0, \theta_i^{N_{\theta_i}}\right]$, $S^{(N_{\theta_1})}$ is the N_{θ_1}th derivative of S, and $\left\{w_1^j\right\}_{j=1}^{N_{\theta_1}}$ are the weights in the Lagrangian polynomial of degree $N_{\theta_1} - 1$, that is,

$$w_1^j = \prod_{\substack{l=1 \\ l \ne j}}^{N_{\theta_1}} \frac{\theta_1^0 - \theta_1^l}{\theta_1^j - \theta_1^l}. \tag{17}$$

In other words, by choosing N_{θ_1} and $\theta_1^{N_{\theta_1}}$ properly, we can create the solution to the right problem, $S\left(\theta_1^0\right)$, using the solutions to a sequence of perturbed problems, $\left\{S\left(\theta_1^j\right)\right\}_{j=1}^{N_{\theta_1}}$. Intrigued by (16), we propose the following new solver:

$$\mathcal{G} = \sum_{j=1}^{N_{\theta_1}} w_1^j \cdot \mathcal{G}_{\theta_1^j,\tau} \tag{18}$$

It computes the solution

$$\mathcal{G}(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}) = \sum_{j=1}^{N_{\theta_1}} w_1^j \cdot \mathcal{G}_{\theta_1^j,\tau}(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}). \tag{19}$$

Although the sequence of "auxiliary" solvers

$$\left\{\mathcal{G}_{\theta_1^j,\tau}\right\}_{j=1}^{N_{\theta_1}} \tag{20}$$

that constitute $\mathcal{G}$ use values of θ_1 different from θ_1^0, when N_{θ_1} and $\theta_1^{N_{\theta_1}}$ are properly chosen, the solution (19) can still be a good estimate to the exact solution $S\left(\theta_1^0\right)$.

To apply $\mathcal{G}$, we need to apply the N_{θ_1} auxiliary solvers in (20). If the number of cores, N_c, is greater than or equal to N_{θ_1}, then they can be applied in parallel on different cores. Since using $\left\{\theta_1^j\right\}_{j=1}^{N_{\theta_1}}$ relaxes the constraint on time step, we can choose a time step τ in $\mathcal{G}$ that is permitted by $\left\{\theta_1^j\right\}_{j=1}^{N_{\theta_1}}$ but not allowed by the correct parameter value θ_1^0. Consequently, by choosing τ properly, applying the N_{θ_1} auxiliary solvers in parallel on N_{θ_1} cores can be faster than applying any single solver in the same family as $\mathcal{G}_{\theta_1,\tau}$ that uses θ_1^0.

To further relax the constraint on time step, we can generalize the idea described above and allow more than one physical parameter to deviate from its correct value. For example, consider the case of three parameters, that is, $\Theta = \{\theta_1, \theta_2, \theta_3\}$. Similar to (18), we propose the following solver:

$$\mathcal{G} = \sum_{j_1=1}^{N_{\theta_1}} w_1^{j_1} \cdot \left(\sum_{j_2=1}^{N_{\theta_2}} w_2^{j_2} \cdot \left(\sum_{j_3=1}^{N_{\theta_3}} w_3^{j_3} \cdot \mathcal{G}_{\theta_1^{j_1},\theta_2^{j_2},\theta_3^{j_3},\tau} \right) \right) \tag{21}$$

where

$$w_i^j = \prod_{\substack{l=1 \\ l\neq j}}^{N_{\theta_i}} \frac{\theta_i^0 - \theta_i^l}{\theta_i^j - \theta_i^l}. \tag{22}$$

It computes the solution

$$\mathcal{G}\left(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}\right) = \sum_{j_1=1}^{N_{\theta_1}} w_1^{j_1} \cdot \left(\sum_{j_2=1}^{N_{\theta_2}} w_2^{j_2} \cdot \left(\sum_{j_3=1}^{N_{\theta_3}} w_3^{j_3} \cdot \mathcal{G}_{\theta_1^{j_1},\theta_2^{j_2},\theta_3^{j_3},\tau}\left(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}\right) \right) \right), \tag{23}$$

which can be a good estimate to the exact solution $S\left(\theta_1^0, \theta_2^0, \theta_3^0\right)$ if N_{θ_1}, N_{θ_2}, N_{θ_3} and $\theta_1^{N_{\theta_1}}$, $\theta_2^{N_{\theta_2}}$, $\theta_3^{N_{\theta_3}}$ are properly chosen. In addition, the proposed solver (21) is a linear combination of the sequence of $N_{\theta_1} \cdot N_{\theta_2} \cdot N_{\theta_3}$ auxiliary solvers

$$\left\{\mathcal{G}_{\theta_1^{j_1},\theta_2^{j_2},\theta_3^{j_3},\tau}\right\}_{j_1,j_2,j_3=1}^{N_{\theta_1},N_{\theta_2},N_{\theta_3}}, \tag{24}$$

which can again be applied in parallel if $N_c \geq N_{\theta_1} \cdot N_{\theta_2} \cdot N_{\theta_3}$. Extrapolating with respect to more than one parameters allows us to relax the time step τ even further and thus, the solver (21) can run faster than the solver (18) if sufficiently many cores are available.

To simply the notation, from now on, we rewrite (21) as

$$\mathcal{G} = \sum_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}} w_j \cdot \mathcal{G}_{\Theta^j,\tau} \tag{25}$$

where $w_j = w_1^{j_1} \cdot w_2^{j_2} \cdot w_3^{j_3}$ and $\Theta^j = \left\{\theta_1^{j_1}, \theta_2^{j_2}, \theta_3^{j_3}\right\}$ for some combination of j_1, j_2, and j_3.

In summary, we have developed a new solver inspired by the observation that certain physical parameters of a problem, if simply viewed as parameters of its solvers, can be tuned to allow the solvers to take larger times steps. It is straightforward to implement and non-intrusive: applying it entails applying an existing solver multiple times with varying values of said parameters. We can choose the parameter values in such a way that larger time steps can be used and the solution is still in good agreement with the solution obtained using the correct parameter values. The proposed solver is also trivial to parallelize: the runs of the existing solver using different parameter values are completely independent and can be performed in parallel.

4.3. Extrapolation with respect to time step

In Section 4.2, we have constructed a new coarse solver that is less restrictive about the size of time step by extrapolating a family of existing coarse solvers parametrized by a few physical parameters. Here, we construct another new coarse solver

Table 1
Values of the weights $\{v_q\}_{q=0}^{m}$ in Eq. (27) for $m = 0,\ 1,\ 2$ and fixed Θ, r, α.

m	$\{v_q\}_{q=0}^{m}$
0	$v_0 = 1$
1	$v_0 = r^{\alpha}/(r^{\alpha}-1)$ and $v_1 = -1/(r^{\alpha}-1)$
2	$v_0 = r^{\alpha+1}/(r^{\alpha+1}-1)\cdot r^{\alpha}/(r^{\alpha}-1)$, $v_1 = -r^{\alpha+1}/(r^{\alpha+1}-1)\cdot 1/(r^{\alpha}-1)$, and $v_2 = -1/(r^{\alpha+1}-1)$

by extrapolating a family of existing coarse solvers parametrized by the time step instead. Compared to the existing solvers, the new solver has a higher order of accuracy in time. In this section, the values of the physical parameters in Θ are fixed.

Let m be a non-negative integer, α be the order of accuracy of $\mathcal{G}_{\Theta,\tau}$ in time, and $0 < r < 1$. For fixed m, Θ, and r, let $R_m : (0, T] \longrightarrow \mathbb{R}$ be the following function:

$$R_m(\tau) = \begin{cases} \mathcal{G}_{\Theta,\tau}\left(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}\right), & \text{if } m = 0 \\ \dfrac{r^{\alpha+m-1}\cdot R_{m-1}(\tau) - R_{m-1}(r\cdot\tau)}{r^{\alpha+m-1}-1}, & \text{if } m \geq 1 \end{cases}. \tag{26}$$

We note that $R_m(\tau)$ is the Richardson extrapolation of $R_{m-1}(\tau)$. While Eq. (26) gives a recursive definition of $R_m(\tau)$, using mathematical induction, we can show that its closed form is a linear combination of $\left\{\mathcal{G}_{\Theta,r^q\cdot\tau}\left(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}\right)\right\}_{q=0}^{m}$. Since this is a well-known fact about recursive Richardson extrapolation (see [51], for example), we simply state it in Lemma 1 without giving a proof.

Lemma 1. *Let $R_m(\tau)$ be as defined in Eq. (26). Then*

$$R_m(\tau) = \sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta,r^q\cdot\tau}\left(t_{End}, t_{Start}, X_{Start}\right), \tag{27}$$

where $\{v_q\}_{q=0}^{m}$ are scalars that satisfy $\sum_{q=0}^{m} v_q = 1$.

For $m = 0,\ 1$, and 2, we list the values of $\{v_q\}_{q=0}^{m}$ in Table 1.

If $\mathcal{G}_{\Theta,\tau}\left(t_{\text{End}}, t_{\text{Start}}, X_{\text{Start}}\right)$ is αth-order accurate in time, by properties of the recursive Richardson extrapolation, $R_m(\tau)$ is $(\alpha+m)$th-order accurate in time. Therefore, we propose the following solver:

$$\mathcal{G} = \sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta,r^q\cdot\tau}, \tag{28}$$

where the weights $\{v_q\}_{q=0}^{m}$ in Eq. (28) are the same as in Lemma 1. It produces the solution $R_m(\tau)$. If there are at least $m+1$ cores available, the sequence of auxiliary solvers

$$\left\{\mathcal{G}_{\Theta,r^q\cdot\tau}\right\}_{q=0}^{m} \tag{29}$$

that constitute the proposed solver (28) can be applied in parallel. As a result, its runtime is about the same as that of the slowest solver among (29), that is, $\mathcal{G}_{\Theta,r^m\cdot\tau}$ since $0 < r < 1$. Like the solver proposed in Section 4.2, the solver (28) is also straightforward to implement, non-intrusive, and easy to parallelize.

Richardson extrapolation has been incorporated into parallel-in-time methods in various fashions in previous work (see [26,52–54], for example). In [52], it is applied to improve the accuracy per computational cost of the Multigrid Reduction in Time (MGRIT) method [19]. In [26] where Parareal is combined with the spectral element method to solve the Navier-Stokes equations, Richardson extrapolation is used to improve the accuracy of both the fine and coarse solvers. It appears that solvers using different time steps were run in serial in [26], whereas we are proposing to run them in parallel in the current work. In [54], a Parareal-Richardson algorithm is developed as a variant of the Parareal algorithm, where Richardson extrapolation is used to improve the formula (11) for serial correction. To the best of our knowledge, the current study marks the first time that Richardson extrapolation is proposed to improve the accuracy in time of the simulation of a biofluid, where the method of regularized Stokeslets is used to calculate fluid-structure interactions. In addition, as we will elaborate in Section 4.4, since the Lagrangian extrapolation with respect to physical parameters proposed in Section 4.2 allows for the use of larger time steps, by combining it and the Richardson extrapolation with respect to time step proposed here, we can construct a new solver that is more accurate and yet does not require more runtime. As far as we know, building such a solver on existing solvers in a non-intrusive manner is novel.

4.4. Extrapolation with respect to both physical parameters and time step

By using a combination of the extrapolation techniques proposed in Sections 4.2 and 4.3, we can also construct the new coarse solver

$$\mathcal{G} = \sum_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}} w_j \cdot \left(\sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta^j, r^q\cdot\tau} \right) \tag{30}$$

if the Lagrangian extrapolation is with respect to the three physical parameters θ_1, θ_2, and θ_3, as in Eqs. (21) and (25). The weights $\{w_j\}_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}}$ and $\{v_q\}_{q=0}^{m}$ are the same as the ones in Eq. (25) and Lemma 1, respectively; that is, we replace the solver $\mathcal{G}_{\Theta^j,\tau}$ in Eq. (25) with $\sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta^j, r^q\cdot\tau}$ whose accuracy in time is m orders higher.

If there are at least $N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}\cdot(m+1)$ cores available, then the sequence of auxiliary solvers

$$\left\{\mathcal{G}_{\Theta^j, r^q\cdot\tau}\right\}_{j=1,q=0}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}, m} \tag{31}$$

that constitute the proposed solver (30) can be applied in parallel. The runtime of (30) is therefore about the same as that of the slowest solver among the ones in (31): $\mathcal{G}_{\Theta^j, r^m\cdot\tau}$ for any j, since $0 < r < 1$. On one hand, the time step $r^m\cdot\tau$ is smaller than τ; and on the other hand, the parameter values in Θ^j allow for the use of a larger time step τ that is not permitted by the correct parameter values in Θ^0. As a result, by choosing N_{θ_1}, N_{θ_2}, N_{θ_3}, $\theta_1^{N_{\theta_1}}$, $\theta_2^{N_{\theta_2}}$, $\theta_3^{N_{\theta_3}}$ in the Lagrangian extrapolation and m, r in the Richardson extrapolation properly, the proposed solver (30) can be not only more accurate but also faster than any solver in the same family as $\mathcal{G}_{\Theta,\tau}$ that uses Θ^0. Increasing accuracy while decreasing runtime may seem counter-intuitive at first. We emphasize that this is made possible by the parallel implementation of the auxiliary solvers in (31).

The new solver that we propose here shares the following advantages with the solvers proposed in Sections 4.2 and 4.3: it is straightforward to implement, non-intrusive, and easy to parallelize.

4.5. An outline of Parareal equipped with a proposed coarse solver

We outline the Parareal algorithm equipped with the proposed coarse solver (30) for solving Eq. (1) in Algorithm 1. Both $\mathbf{X}_n^k$ and $\mathbf{X}_n^{k'}$ are $3\times N_\sigma$ matrices whose ith column corresponds to the ith grid point on the structures, and $\mathbf{X}_n^k$ denote the kth iterate of Parareal at time T_n. When the coarse solver proposed in Section 4.2 or 4.3 is employed instead, the algorithm reads very similarly. We note that parallel computing is used in both the serial sweep and parallel sweep. More specifically, for each time slice $[T_{n-1}, T_n]$ in the serial sweep, the sequence of auxiliary coarse solvers (31) will be run in parallel (steps 4 and 15), and then the solutions that they calculate will be combined to form the solution that (30) produces (steps 5 and 16).

Remark 4. In Algorithm 1, we assume that the same grid is used to discretize the structures in the fine solver $\mathcal{F}$ and coarse solver $\mathcal{G}_{\Theta,\tau}$. When a coarser grid is used in $\mathcal{G}_{\Theta,\tau}$ instead, such as in some of the numerical experiments in Section 5, we must be able to map a solution on one grid to a solution on the other grid. For example, in step 4 of Algorithm 1, we need to calculate

$$\mathcal{G}_{\Theta^j, r^q\cdot\tau}\left(T_n, T_{n-1}, \mathcal{I}_\sigma^{\sigma_c}\left(\mathbf{X}_{n-1}^0\right)\right) \tag{32}$$

instead, where $\mathcal{I}_\sigma^{\sigma_c}(\cdot)$ is a function that maps a given solution on the fine grid to a solution on the coarse grid. Similarly, step 6 of Algorithm 1 should be modified to read

$$\mathbf{X}_n^0 \leftarrow \mathcal{I}_{\sigma_c}^{\sigma}\left(\mathcal{G}\left(T_n, T_{n-1}, \mathcal{I}_\sigma^{\sigma_c}\left(\mathbf{X}_{n-1}^0\right)\right)\right), \tag{33}$$

where $\mathcal{I}_{\sigma_c}^{\sigma}(\cdot)$ is a function that interpolates a given solution on the coarse grid to produce a solution on the fine grid.

5. Numerical results

We present numerical results of Algorithm 1, the Parareal algorithm that employs one of the coarse solvers proposed in Section 4, applied to two model micro-swimmers. In Section 5.1, we apply this algorithm to simulate the swimming motion of slender, elastic rods, which can translate, bend, and twist in the fluid and represent biological structures such as tails of sperm and bacterial flagella. In Section 5.2, we apply Algorithm 1 to simulate the dynamics of an elastic sphere that resembles a vesicle placed in a shear flow.

All the computational results are obtained using the Parallel Computing Toolbox of MATLAB® version R2018b on Intel® Xeon® CPU E5-2699 v3.

Algorithm 1 Parareal equipped with the coarse solver (30) for Eq. (1)

Input: $\mathbf{X}_0^0 \in \mathbb{R}^{3\times N_\sigma}$ which contains the initial values of Eq. (1)
Number of cores N_c
Positive integers N_{θ_1}, N_{θ_2}, N_{θ_3} and parameter values $\left\{\theta_1^j\right\}_{j=0}^{N_{\theta_0}}$, $\left\{\theta_2^j\right\}_{j=0}^{N_{\theta_2}}$, $\left\{\theta_3^j\right\}_{j=1}^{N_{\theta_3}}$
Time step $\tau \in (0, T]$
Non-negative integer m and scalar $r \in (0, 1)$
(N_c, N_{θ_1}, N_{θ_2}, N_{θ_3}, and m satisfy $N_{\theta_1} \cdot N_{\theta_2} \cdot N_{\theta_3} \cdot (m+1) \le N_c$)
Output: $\left\{\mathbf{X}_n^k\right\}_{n=1}^{N_c}$ where $\mathbf{X}_n^k \in \mathbb{R}^{3\times N_\sigma}$ contains the estimated solution to Eq. (1) at time $T_n = n \cdot T/N_c$

Calculate the weights $\left\{w_j\right\}_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}}$ based on Eqs. (21), (22), and (25)
Calculate the weights $\left\{v_q\right\}_{q=0}^{m}$ based on Eq. (26) or Table 1
for $n = 1, 2\cdots, N_c$ **do** `% initial serial sweep`
  Calculate $\left\{\mathcal{G}_{\Theta^j, r^q\cdot\tau}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^0\right)\right\}_{j=1,q=0}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}, m}$ **in parallel**
  $\mathcal{G}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^0\right) \leftarrow \sum_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}} w_j \cdot \left(\sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta^j, r^q\cdot\tau}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^0\right)\right)$
  $\mathbf{X}_n^0 \leftarrow \mathcal{G}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^0\right)$
end for
for $k = 1, 2, 3, \cdots$ **do** `% the main for-loop`
  Calculate $\left\{\mathbf{X}_n^{k'} = \mathcal{F}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^{k-1}\right)\right\}_{n=k}^{N_c}$ **in parallel** `% parallel sweep`
  if $k > 1$ **then**
    $\mathbf{X}_n^k \leftarrow \mathbf{X}_n^{k-1}$ for $n = 1, 2, \cdots, k-1$
  end if
  $\mathbf{X}_k^k \leftarrow \mathbf{X}_k^{k'}$
  for $n = k+1, k+2, \cdots, N_c$ **do** `% serial sweep`
    Calculate $\left\{\mathcal{G}_{\Theta^j, r^q\cdot\tau}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^k\right)\right\}_{j=1,q=0}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}, m}$ **in parallel**
    $\mathcal{G}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^k\right) \leftarrow \sum_{j=1}^{N_{\theta_1}\cdot N_{\theta_2}\cdot N_{\theta_3}} w_j \cdot \left(\sum_{q=0}^{m} v_q \cdot \mathcal{G}_{\Theta^j, r^q\cdot\tau}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^k\right)\right)$
    $\mathbf{X}_n^k \leftarrow \mathbf{X}_n^{k'} + \left(\mathcal{G}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^k\right) - \mathcal{G}\left(T_n, T_{n-1}, \mathbf{X}_{n-1}^{k-1}\right)\right)$
  end for
end for

Throughout this section, to examine the accuracy of Algorithm 1, we define the true relative error and relative increment of the kth iteration of Algorithm 1 to be

$$\eta^k = \max_{i=1,\,2,\,\cdots,\,N_\sigma} \frac{\left\|\mathbf{x}_i^k - \mathbf{x}_i^{\mathcal{F}}\right\|_2}{\left\|\mathbf{x}_i^{\mathcal{F}}\right\|_2} \text{ and } \widetilde{\eta}^k = \max_{i=1,\,2,\,\cdots,\,N_\sigma} \frac{\left\|\mathbf{x}_i^k - \mathbf{x}_i^{k-1}\right\|_2}{\left\|\mathbf{x}_i^k\right\|_2} \text{ for } k = 1,\,2,\,\cdots \tag{34}$$

respectively, where N_σ is the number of grid points on the structures, $\mathbf{x}_i^k$ and $\mathbf{x}_i^{\mathcal{F}}$ denote the final positions of the ith grid point calculated by Algorithm 1 and the fine solver $\mathcal{F}$ respectively for $i = 1,\,2,\,\cdots,\,N_\sigma$. The relative error measures how close the solutions produced by Algorithm 1 and $\mathcal{F}$ are. However, in practice, it cannot be calculated since $\mathbf{x}_i^{\mathcal{F}}$ is not available. The relative increment measures how close consecutive iterates of Algorithm 1 are. It can be calculated easily and utilized in the stopping criterion of Algorithm 1.

To examine the parallel speedup of Algorithm 1, we define the measured speedup of Algorithm 1 to be the ratio in Eq. (14). We also define the theoretical speedup of Algorithm 1 to be the same ratio in the ideal case where parallel computing does not incur any overhead, that is, $\Upsilon_{oh} = 0$ in Eq. (14). Comparing the two speedups allows us to examine the effect of the overhead of parallel computing on the performance of Algorithm 1. If not specified, "speedup" refers to the measured speedup that takes into account the overhead of parallel computing.

To compare temporal parallelization and spatial parallelization, we also define the (measured) speedup of the spatially parallelized $\mathcal{F}$ (see Remark 1 in Section 2.3) as

$$\frac{\Upsilon_{\mathcal{F}}}{\Upsilon_{sp}}, \tag{35}$$

where Υ_{sp} is the total runtime of the spatially parallelized $\mathcal{F}$.

5.1. A rod-like swimmer

In this section, each swimmer is represented by its centerline and modeled as the version of Kirchhoff rod considered in [40,55–59]. The parameter values that specify the physical properties of the rods are given in Table 2. We choose them within the ranges considered in [59], which were in turn determined based on properties of the tails of human sperm (see the references in [59]). The preferred shape of the rods is a planar, time-dependent sinusoidal wave. More precisely, the preferred strain twist vector along each rod is given by

Table 2
Parameter values that specify the properties of the rods and fluid.

Rod length, L (μm)	60
Amplitude, A (μm)	1
Frequency, f (Hz)	40π
Wavelength, λ (μm)	30
Bending modulus, $a_1 = a_2$ ($g \cdot \mu m^3 \cdot s^{-2}$)	1
Twist modulus, a_3 ($g \cdot \mu m^3 \cdot s^{-2}$)	1
Shear modulus, $b_1 = b_2$ ($g \cdot \mu m \cdot s^{-2}$)	0.6
Stretch modulus, b_3 ($g \cdot \mu m \cdot s^{-2}$)	0.6
Distance to wall, d (μm)	10
Fluid viscosity, μ ($g \cdot \mu m^{-1} \cdot s^{-1}$)	10^{-6}

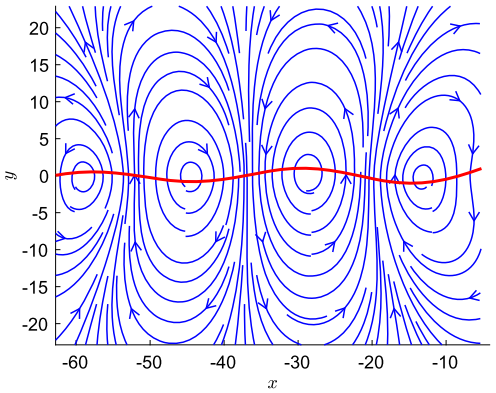

Fig. 2. Streamlines of the flow around a single swimming rod at time $t = 1$. They are plotted using only the x- and y-direction fluid velocities on the slice plane $z = 10$.

$$(\Omega_1, \Omega_2, \Omega_3) = \left(0, -k^2 A \sin(ks + ft), 0\right), \tag{36}$$

where s is the arclength, A is the amplitude, f is the frequency, and $k = 2\pi/\lambda$ for the wavelength λ. The values of A, f, and λ are given in Table 2. At any point in time, the forces and torques exerted to the fluid by the grid points on the rods are determined by the difference between the preferred and current shapes of the rods as well as their bending, twist, and shear moduli given in Table 2. The rods swim in the semi-infinite fluid domain $\{(x, y, z) \in \mathbb{R}^3 | z \geq 0\}$ bounded by an infinite, planar, and stationary wall located at $z = 0$, on which the flow vanishes. In addition, at time $t = 0$, the rods are initialized to be straight, parallel to, and a distance d away from the wall. In Fig. 2, we show the streamlines drawn based on the instantaneous fluid velocity around a single swimming rod.

In the fine solver $\mathcal{F}$, the grid spacing of the uniform grid used to discretize the centerline of each rod is $\sigma_f = 0.2$ μm (or equivalently, $N_\sigma = 301$ grid points are used to discretize the rod), and the ODE solver is the explicit two-stage Runge-Kutta method (RK2) with step size τ_f. In each auxiliary coarse solver $\mathcal{G}_{\Theta,\tau}$ that constitutes the coarse solvers proposed in Section 4, the grid spacing is $\sigma_c \geq \sigma_f$, and the ODE solver is forward Euler method with step size τ. Recall that the proposed solver (28) or (30) is a linear combination of auxiliary coarse solvers using various time steps; and when they can be applied in parallel, the runtime of (28) or (30) is about the same as the runtime of the slowest auxiliary solver, which uses the smallest time step $r^m \cdot \tau$. We therefore refer to $r^m \cdot \tau$ as the time step used by (28) or (30) and denote it by τ_c. The time steps τ_f and τ_c are chosen to ensure that the length of the rods does not fluctuate more than 1% throughout the simulation. In both $\mathcal{F}$ and $\mathcal{G}_{\Theta,\tau}$, the Method of Regularized Stokeslets (MRS) is applied to calculate fluid velocities. More specifically, the formulae in [58] are used, which were derived using the method of images and account for the point torques in addition to the point forces exerted by the grid points in the semi-infinite fluid domain.

The set of physical parameters, Θ, may include one or more of the following: the regularization parameter ϵ used in the MRS, the bending and twist moduli a, and the fluid viscosity μ. (We assume that the same value is always used for the two bending moduli a_1, a_2 and the twist modulus a_3.) The correct values of ϵ, a, and μ are denoted by ϵ^0, a^0, and μ^0, respectively. As shown in Table 2, $a^0 = 1$ and $\mu^0 = 10^{-6}$. Although each rod is represented by its centerline only, the thickness of the rod can be implicitly incorporated into the simulation by choosing ϵ properly (see Remark 2 in Section 2). To examine the performance of Algorithm 1 as the thickness of the rod varies, we consider multiple values of ϵ^0. In the fine solver $\mathcal{F}$, ϵ^0, a^0, and μ^0 are always used. Recall that in the proposed coarse solver (21) or (30), to relax the constraint on time step, we use a sequence of values that are different from the correct one for each physical parameter in Θ. Let $\{\epsilon^j\}_{j=1}^{N_\epsilon}$, $\{a^j\}_{j=1}^{N_a}$, and $\{\mu^j\}_{j=1}^{N_\mu}$ denote the values used for ϵ, a, and μ, respectively. We denote the number of auxiliary solvers that

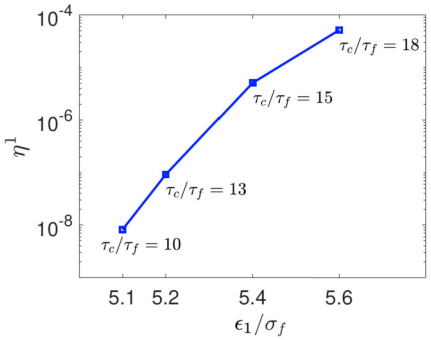

Fig. 3. The relative error of the first iteration of Algorithm 1 and τ_c/τ_f when the coarse solver is $\sum_{j=1}^{4}\mathcal{G}_{\epsilon^j,\tau_c}$. Four values of ϵ^1 are considered: $\epsilon^1/\sigma_f = 5.1,\ 5.2,\ 5.4,\ 5.6$. Given ϵ^1, $\epsilon^j = \epsilon^1 + (j-1)\cdot 0.05\sigma_f$ for $j = 2,\ 3,\ 4$.

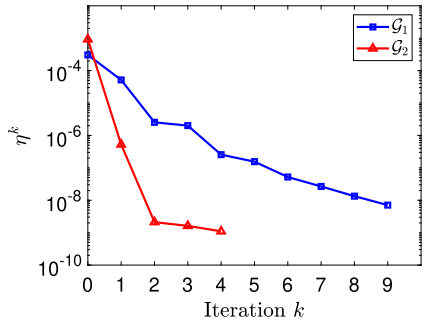

Fig. 4. Decay of the relative error for two versions of Algorithm 1 that use coarse solvers $\mathcal{G}^1$ and $\mathcal{G}^2$ specified in Table 3. $\tau_c = 18\tau_f$.

constitute the coarse solver $\mathcal{G}$ by $N_{\mathcal{G}}$ and always choose it to be no more than N_c (the number of cores) so that they can be run in parallel.

Unless otherwise specified, the time domain is $[0, T]$ where $T = 1$, $N_c = 40$, and there is only one swimmer in the fluid domain.

5.1.1. The effectiveness of Lagrangian extrapolation and Richardson extrapolation

In this section, we fix $\epsilon^0 = 5\sigma_f$, $\sigma_c = \sigma_f$, and $\tau_f = 10^{-6}$. Since ϵ^0 is roughly the cross-sectional radius of the rod and the length of the rod L is 60 (see Table 2), the ratio of the two is about 0.017 in this case.

We first consider coarse solvers constructed using only Lagrangian extrapolation with respect to a single physical parameter. We examine η^1, the relative error of the first iteration of Algorithm 1 (see Eq. (34)), when the coarse solver used in Algorithm 1 is $\sum_{j=1}^{4}\mathcal{G}_{\epsilon^j,\tau_c}$, that is, $\Theta = \{\epsilon\}$ and $N_\epsilon = 4$ in Lagrangian extrapolation. We consider four values of ϵ^1: $5.1\sigma_f$, $5.2\sigma_f$, $5.4\sigma_f$, and $5.6\sigma_f$. For each ϵ^1, we choose $\epsilon^j = \epsilon^1 + (j-1)\cdot 0.05\sigma_f$ for $j = 2,\ 3,\ 4$. In Fig. 3, we display η^1 along with τ_c/τ_f for each case, where τ_c is close to the largest time step permitted by ϵ^1. We observe that the constraint on τ_c can be relaxed by increasing ϵ. When ϵ^1 increases from $5.1\sigma_f$ to $5.6\sigma_f$, we can almost double the size of τ_c and hence reduce the runtime of the coarse solver by 50%. In the meantime, increasing ϵ_1 and τ_c increases both the extrapolation error and the discretization error in time, causing η^1 to grow from about 10^{-8} to about 10^{-4}.

Next, we continue to consider coarse solvers constructed using only Lagrangian extrapolation but allow extrapolation with respect to more than one physical parameter. In Fig. 4, we show the decay of the relative error as iteration count increases for two versions of Algorithm 1 that use two different coarse solvers $\mathcal{G}^1$ and $\mathcal{G}^2$. Lagrangian extrapolation with respect to only ϵ is used to construct $\mathcal{G}^1$, whereas Lagrangian extrapolation with respect to the bending/twist modulus a and fluid viscosity μ in addition to ϵ is used to construct $\mathcal{G}^2$. More details can be found in Table 3. As seen in Fig. 4, the relative error of Algorithm 1 decays much more rapidly if $\mathcal{G}^2$ is used. Since both solvers use the time step $\tau_c = 18\tau_f$, they are computationally as expensive as one another. As seen in Fig. 3, when Lagrangian extrapolation is performed with respect to ϵ only, we need to choose ϵ^1 to be as large as $5.6\sigma_f$ in order to relax the time step to $18\tau_f$. The coarse solver $\mathcal{G}^2$ can take the same time step even though ϵ^1 is only $5.15\sigma_f$ (see Table 3). That is, to relax the time step to the same extent, extrapolating with respect to more than one physical parameters allows us to choose their values to be closer to the correct ones and hence reduces extrapolation error.

We now consider coarse solvers constructed using both Lagrangian extrapolation and Richardson extrapolation. In Fig. 5, we plot the relative error η^1 of two versions of Algorithm 1 that use two different coarse solvers $\mathcal{G}^3$ and $\mathcal{G}^4$. $\mathcal{G}^3$ is constructed using only Lagrangian extrapolation with respect to ϵ, whereas $\mathcal{G}^4$ is constructed using both Lagrangian ex-

Table 3
Coarse solvers for the case $\epsilon^0 = 5\sigma_f$ considered in section 5.1.1.

Coarse solver	Specifications	$N_{\mathcal{G}}$
$\mathcal{G}^1$	$\tau_c = 18\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon\}$, $N_\epsilon = 4$ $\epsilon^j/\sigma_f = 5.60,\ 5.65,\ 5.70,\ 5.75$ for $j = 1,\ 2,\ 3,\ 4$	4
$\mathcal{G}^2$	$\tau_c = 18\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, a, \mu\}$, $N_\epsilon = 4$, $N_a = 3$, $N_\mu = 3$ $\epsilon^j/\sigma_f = 5.150,\ 5.225,\ 5.300,\ 5.375$ for $j = 1,\ 2,\ 3,\ 4$ $a^j/a^0 = 0.75,\ 0.80,\ 0.85$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.300,\ 1.375,\ 1.450$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^3$	$\tau_c = 10\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon\}$, $N_\epsilon = 2,\ 3,\ 4,\ 5,\ 6$ $\epsilon^1 = 5.1\sigma_f$ and $\epsilon^j = \epsilon^1 + (j-1)\cdot 0.05\sigma_f$ for $j = 2,\ \cdots,\ N_\epsilon$	2, 3, 4, 5, 6
$\mathcal{G}^4$	$\tau_c = 10\tau_f$, $\sigma_c = \sigma_f$ $m = 1$, $r = 0.83$, $\Theta = \{\epsilon\}$, $N_\epsilon = 2,\ 3,\ 4,\ 5,\ 6$ $\epsilon^1 = 5.1\sigma_f$ and $\epsilon^j = \epsilon^1 + (j-1)\cdot 0.05\sigma_f$ for $j = 2,\ \cdots,\ N_\epsilon$	4, 6, 8, 10, 12

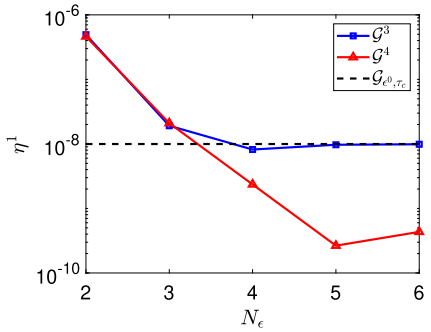

Fig. 5. The relative error of the first iteration for three versions of Algorithm 1 that use coarse solvers $\mathcal{G}_{\epsilon^0,\tau_c}$, $\mathcal{G}^3$, and $\mathcal{G}^4$. Details of $\mathcal{G}^3$ and $\mathcal{G}^4$ can be found in Table 3. $\tau_c = 10\tau_f$.

trapolation with respect to ϵ and Richardson extrapolation with respect to time step. We choose $m = 1$ in Richardson extrapolation, which increases the order of accuracy in time by one. Since the fine solver uses RK2 and the auxiliary coarse solvers use forward Euler method, $\mathcal{G}^4$ has the same order of accuracy in time as the fine solver. We vary N_ϵ in both $\mathcal{G}^3$ and $\mathcal{G}^4$ from 2 to 6. ($N_{\mathcal{G}} = N_\epsilon$ and $2N_\epsilon$ for $\mathcal{G}^3$ and $\mathcal{G}^4$ respectively.) The time step used by both solvers is $\tau_c = 10\tau_f$. More details of $\mathcal{G}^3$ and $\mathcal{G}^4$ can be found in Table 3. For reference, η^1 of the version of Algorithm 1 that simply employs $\mathcal{G}_{\epsilon^0,\tau_c}$ as the coarse solver is also included in Fig. 5. (Since no extrapolation is used to construct this solver, η^1 does not change with N_ϵ). We observe that as N_ϵ increases, extrapolation error becomes so small that Algorithm 1 equipped with $\mathcal{G}^3$ is as accurate as Algorithm 1 equipped with $\mathcal{G}_{\epsilon^0,\tau_c}$. For smaller values of N_ϵ, $\mathcal{G}^4$ is as accurate as $\mathcal{G}^3$, that is, Richardson extrapolation almost has no effect on the accuracy of Parareal. The reason is that in this regime, the discretization error in time is negligible compared to the error introduced by Lagrangian extrapolation. For larger values of N_ϵ, as extrapolation error becomes smaller, the advantage of Richardson extrapolation becomes evident. For example, it reduces η^1 from about 10^{-8} to about 10^{-10} when $N_\epsilon = 5$.

5.1.2. The accuracy and parallel speedup of Algorithm 1

In this section, we present the numerical results of Algorithm 1 applied to simulate swimming rods of different thickness. Recall that ϵ in the MRS is a proxy for the thickness. We consider $\epsilon^0/\sigma_f = 2,\ 3,\ 5,\ 7$. Accordingly, the ratios of the cross-sectional radius of the rod to its length are about 0.007, 0.010, 0.017, and 0.023, respectively.

For the case $\epsilon^0 = 5\sigma_f$, we again choose $\sigma_c = \sigma_f$ and $\tau_f = 10^{-6}$ as in section 5.1.1. In Fig. 6a, we show the decay of the relative error as iteration count increases for four versions of Algorithm 1, where the 0th iteration corresponds to the initial serial sweep (steps 3 to 7 of Algorithm 1). The four versions employ four different coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$, and $\mathcal{G}^4$, the details of which can be found in Table 4. $\mathcal{G}^1$ and $\mathcal{G}^2$ are constructed using Lagrangian extrapolation only, $\mathcal{G}^3$ is constructed using both Lagrangian extrapolation and Richardson extrapolation, and $\mathcal{G}^4$ is simply $\mathcal{G}_{\epsilon^0,\tau_c}$ (no extrapolation is used). Consequently, $\mathcal{G}^3$ is second-order accurate in time, whereas the other three solvers are only first-order accurate in time. As shown in Fig. 6a, when $\mathcal{G}^1$ is used as the coarse solver, the error of Algorithm 1 decreases the slowest. This is because $\mathcal{G}^1$ introduces the largest extrapolation error and discretization error in time. As seen in Table 4, the values of $\{\epsilon^j\}_{j=1}^{N_\epsilon}$, $\{a^j\}_{j=1}^{N_a}$, and $\{\mu^j\}_{j=1}^{N_\mu}$ used to construct $\mathcal{G}^1$ deviate the farthest away from their correct values ϵ^0, a^0, and μ^0. These values also allow $\mathcal{G}^1$ to take the largest time step. After one iteration, the version of Algorithm 1 using $\mathcal{G}^3$ as the

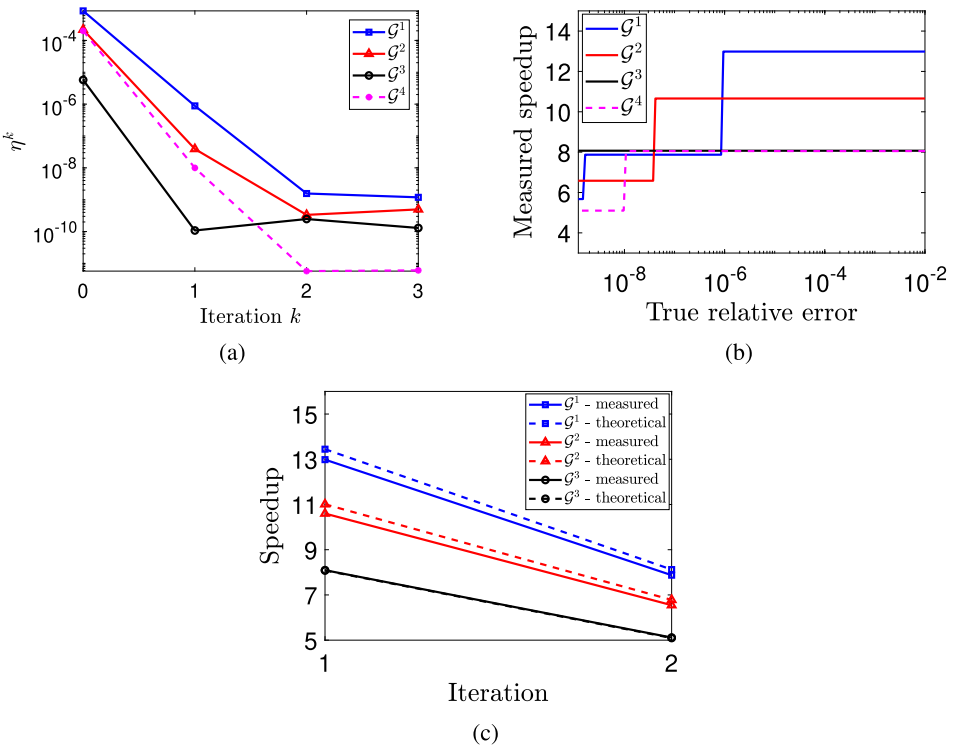

Fig. 6. The accuracy and parallel speedup of four versions of Algorithm 1 in the case $\epsilon^0 = 5\sigma_f$. The four versions use four different coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$, and $\mathcal{G}^4$, which are specified in Table 4. The number of cores used is 40. (a) Decay of the relative error as iteration count increases. (b) Increase of the parallel speedup as the desired error increases. (c) Measured and theoretical speedup after one and two iterations.

Table 4
Coarse solvers for the case $\epsilon^0 = 5\sigma_f$ considered in section 5.1.2.

Coarse solver	Specifications	$N_{\mathcal{G}}$
$\mathcal{G}^1$	$\tau_c = 20\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, a, \mu\}$, $N_\epsilon = 4$, $N_a = 3$, $N_\mu = 3$ $\epsilon^j/\sigma_f = 5.20,\ 5.25,\ 5.30,\ 5.35$ for $j = 1,\ 2,\ 3,\ 4$ $a^j/a^0 = 0.70,\ 0.75,\ 0.80$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.30,\ 1.45,\ 1.60$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^2$	$\tau_c = 15\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, a, \mu\}$, $N_\epsilon = 4$, $N_a = 3$, $N_\mu = 3$ $\epsilon^j/\sigma_f = 5.15,\ 5.20,\ 5.25,\ 5.30$ for $j = 1,\ 2,\ 3,\ 4$ $a^j/a^0 = 0.85,\ 0.90,\ 0.95$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.10,\ 1.15,\ 1.20$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^3$	$\tau_c = 10\tau_f$, $\sigma_c = \sigma_f$ $m = 1$, $r = 0.83$, $\Theta = \{\epsilon, a\}$, $N_\epsilon = 5$, $N_a = 3$ $\epsilon^j/\sigma_f = 5.02,\ 5.04,\ 5.06,\ 5.08,\ 5.10$ for $j = 1,\ 2,\ 3,\ 4,\ 5$ $a^j/a^0 = 0.94,\ 0.96,\ 0.98$ for $j = 1,\ 2,\ 3$	30
$\mathcal{G}^4$	$\tau_c = 10\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \emptyset$ (no Lagrangian extrapolation)	1

coarse solver is the most accurate since $\mathcal{G}^3$ introduces the smallest extrapolation error among $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$ and has the highest order of accuracy in time among all four solvers. Since $\mathcal{G}^4$ does not introduce any extrapolation error, the version of Algorithm 1 using $\mathcal{G}^4$ eventually calculates the most accurate solution although it initially converges more slowly than the version of Algorithm 1 using $\mathcal{G}^3$.

Next, we examine the parallel speedup of the four versions of Algorithm 1. As Eq. (14) indicates, the speedup of Parareal depends on the number of iterations, N_{it}, that it entails, which in turn depends on the desired accuracy of its solution. For each version of Algorithm 1, we display its measured speedup as a function of the desired relative error in Fig. 6b. (We assume that at least one iteration of Algorithm 1 has to be run.) The speedup decreases as the desired error decreases, which is what we expect; and its graph looks like a series of steps because N_{it} is an integer. For example, in the case of $\mathcal{G}^1$, as the desired error drops below 10^{-6}, the speedup goes down by a step because N_{it} has to increase from 1 to 2 (see

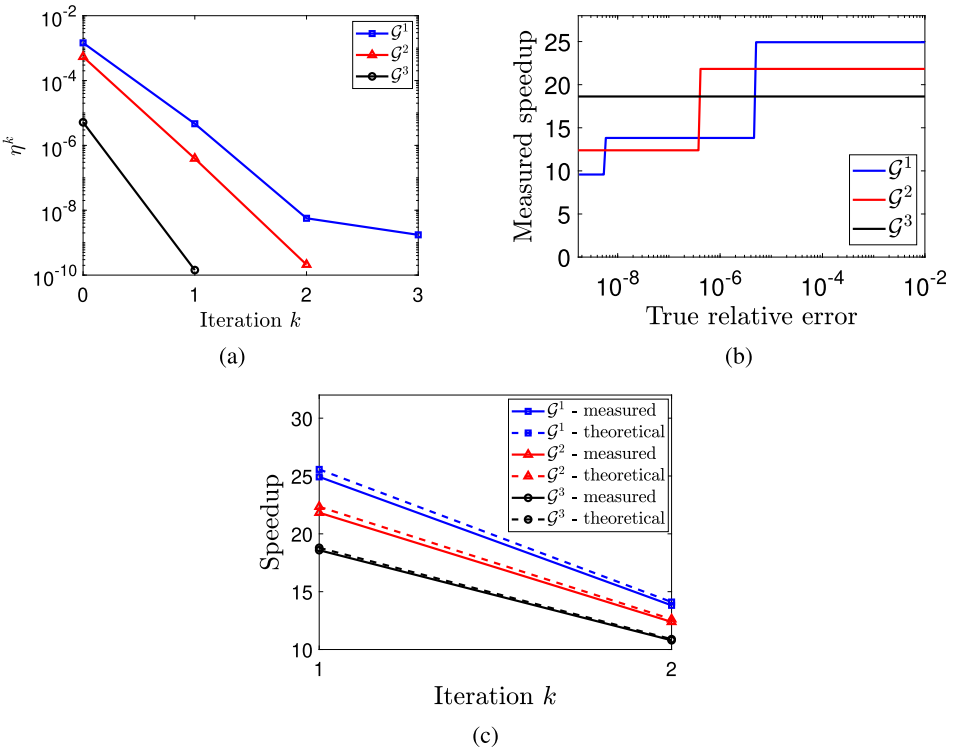

Fig. 7. The accuracy and parallel speedup of three versions of Algorithm 1 in the case $\epsilon^0 = 7\sigma_f$. The three versions use three different coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, and $\mathcal{G}^3$, which are specified in Table 5. The number of cores used is 40. (a) Decay of the relative error as iteration count increases. (b) Increase of the parallel speedup as the desired error increases. (c) Measured and theoretical speedup after one and two iterations.

Fig. 6a). In the case of $\mathcal{G}^3$, the speedup remains unchanged as the desired error decreases from 10^{-2} to 10^{-10} since it only takes Algorithm 1 one iteration to reach the error 10^{-10} (see Fig. 6a). In addition, the coarse solver that leads to the highest speedup changes with the desired accuracy: it is $\mathcal{G}^1$ in the regime $\left[10^{-6}, 10^{-2}\right]$, $\mathcal{G}^2$ in the regime $\left[10^{-7}, 10^{-6}\right]$, and $\mathcal{G}^3$ if we want the error to be smaller than 10^{-8}. $\mathcal{G}^1$ is superior when the requirement on accuracy is less stringent because it is constructed to take larger time steps at the expense of larger extrapolation error and discretization error in time compared to $\mathcal{G}^2$ and $\mathcal{G}^3$ (see Table 4). We also observe from Figs. 6a and 6b that as long as the desired error is no less than 10^{-10}, using one of the proposed coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, and $\mathcal{G}^3$ instead of the generic coarse solver $\mathcal{G}^4$ increases, rather substantially in some cases, the speedup of Parareal.

We also examine the effect of the overhead incurred by our specific parallel computing environment on the speedup of Algorithm 1. In Fig. 6c, we show the theoretical speedup and measured speedup when $N_{\text{it}} = 1$ or 2 for the three versions of Algorithm 1 equipped with the coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, and $\mathcal{G}^3$. Recall that the theoretical speedup is evaluated under the assumption that the overhead, Υ_{oh}, in Eq. (14) is zero. As seen in Fig. 6c the measured speedup is less than 10% lower than the theoretical speedup when the coarse solver used is $\mathcal{G}^1$ or $\mathcal{G}^2$, and the two speedups are almost identical when $\mathcal{G}^3$ is used. The overhead is the least prominent in the case of $\mathcal{G}^3$ mainly for the following reason. As seen in Table 4, compared to $\mathcal{G}^1$ and $\mathcal{G}^2$, $\mathcal{G}^3$ uses a considerably smaller time step, which, for fixed N_{c} and N_{it}, causes the time devoted to serial sweeps (the first term on the right-hand side of Eq. (12)) to be much higher. Since the three versions use the same $\mathcal{F}$, the time that they devote to parallel sweeps (the second term on the right-hand side of Eq. (12)) is the same when N_{c} and N_{it} are fixed. Consequently, when $\mathcal{G}^3$ is used, the sum of these two times is the most dominant, rendering the overhead the least noticeable.

We conduct a similar set of experiments for the case $\epsilon^0 = 7\sigma_f$ and display the results in Fig. 7. As in the case $\epsilon^0 = 5\sigma_f$, we choose $\sigma_c = \sigma_f$ and $\tau_f = 10^{-6}$. Three versions of Algorithm 1 that use three different coarse solvers, specified in Table 5, are considered. Comparing Figs. 6 and 7, we note that to obtain a solution of the same accuracy, Algorithm 1 can achieve a considerably higher speedup when $\epsilon^0 = 7\sigma_f$. This is mainly because a larger ϵ^0 allows for the use of larger time steps. As shown in Tables 4 and 5, $\tau_c/\tau_f = 70,\ 50,\ 35$ for the three coarse solvers in the case $\epsilon^0 = 7\sigma_f$, whereas they are only $20,\ 15,\ 10$ in the case $\epsilon^0 = 5\sigma_f$.

Finally, we consider the cases $\epsilon^0 = 2\sigma_f$ and $\epsilon^0 = 3\sigma_f$. They correspond to very slender rods. For example, when $\epsilon^0 = 3\sigma_f$, the radius of the circular cross section of the rod is only 1% of the length of the rod. These cases turn out to be rather challenging for two reasons. Firstly, the fine solver may need to take a smaller time step. In previous cases where $\epsilon^0 = 5\sigma_f$ or $7\sigma_f$, we choose $\tau_f = 10^{-6}$. While we continue to use $\tau_f = 10^{-6}$ in the case $\epsilon^0 = 3\sigma_f$, $\tau_f = 10^{-7}$ has to be used in the

Table 5
Coarse solvers for the case $\epsilon^0 = 7\sigma_f$ considered in section 5.1.2.

Coarse solver	Specifications	$N_{\mathcal{G}}$
$\mathcal{G}^1$	$\tau_c = 70\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, a, \mu\}$, $N_\epsilon = 4$, $N_a = 3$, $N_\mu = 3$ $\epsilon^j/\sigma_f = 7.20,\ 7.25,\ 7.30,\ 7.35$ for $j = 1,\ 2,\ 3,\ 4$ $a^j/a^0 = 0.70,\ 0.75,\ 0.80$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.30,\ 1.45,\ 1.60$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^2$	$\tau_c = 50\tau_f$, $\sigma_c = \sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, a, \mu\}$, $N_\epsilon = 4$, $N_a = 3$, $N_\mu = 3$ $\epsilon^j/\sigma_f = 7.10,\ 7.15,\ 7.20,\ 7.25$ for $j = 1,\ 2,\ 3,\ 4$ $a^j/a^0 = 0.85,\ 0.90,\ 0.95$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.10,\ 1.15,\ 1.20$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^3$	$\tau_c = 35\tau_f$, $\sigma_c = \sigma_f$ $m = 1$, $r = 0.875$, $\Theta = \{\epsilon, a\}$, $N_\epsilon = 5$, $N_a = 3$ $\epsilon^j/\sigma_f = 7.02,\ 7.04,\ 7.06,\ 7.08,\ 7.10$ for $j = 1,\ 2,\ 3,\ 4,\ 5$ $a^j/a^0 = 0.94,\ 0.96,\ 0.98$ for $j = 1,\ 2,\ 3$	30

Table 6
Coarse solvers for the cases $\epsilon^0 = 2\sigma_f$ and $\epsilon^0 = 3\sigma_f$ considered in section 5.1.2. ($\tau_f = 10^{-7}$ and 10^{-6} in the two cases, respectively.)

ϵ^0/σ_f	Specifications	$N_{\mathcal{G}}$
2	$\tau_c = 70\tau_f$, $\sigma_c = 6\sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon\}$, $N_\epsilon = 2$ $\epsilon^j/\sigma_f = 4.0,\ 4.1$ for $j = 1,\ 2$	2
3	$\tau_c = 7\tau_f$, $\sigma_c = 6\sigma_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon\}$, $N_\epsilon = 2$ $\epsilon^j/\sigma_f = 4.0,\ 4.1$ for $j = 1,\ 2$	2

case $\epsilon^0 = 2\sigma_f$. Secondly, we find that Lagrangian extrapolation alone is unable to relax the time step of the coarse solver enough for Algorithm 1 to achieve a decent speedup. We propose to use a coarser grid in the coarse solver in conjunction with Lagrangian extrapolation. More precisely, we choose $\sigma_c = 6\sigma_f$, that is, we use 51 grid points instead of the 301 grid points used in the fine solver to discretize each rod. The specifics of the coarse solvers that Algorithm 1 uses for the cases $\epsilon^0 = 2\sigma_f$ and $\epsilon^0 = 3\sigma_f$ can be found in Table 6. The advantages of using a coarser grid are twofold. Firstly, it reduces the computational cost of evaluating the velocities of the grid points since this cost is proportional to the number of grid points squared (see Eq. (6) and the paragraph that follows). Secondly, it allows the use of considerably larger values of physical parameters in Lagrangian extrapolation, which in turn allow the use of larger time steps. For example, as seen in Table 6, the values of $\{\epsilon^j\}_{j=1}^{N_\epsilon}$ are about twice as large as ϵ^0 in the case $\epsilon^0 = 2\sigma_f$. We also note that N_ϵ is only 2 and Richardson extrapolation is not applied. This is another consequence of using a coarser grid in the coarse solver: the discretization error in space is so dominant that using a higher-order Lagrangian extrapolation or increasing the accuracy in time offers little improvement to the overall accuracy. In addition, as we now use a fine grid in the fine solver and a coarse grid in the coarse solver, we need to map the solution on one grid to a solution on the other grid (see Remark 4 in Section 4.5). We use cubic splines in terms of the arclength of the rod for both. The convergence and speedup of Algorithm 1 are illustrated in Fig. 8 for the cases $\epsilon^0 = 2\sigma_f$ and $\epsilon^0 = 3\sigma_f$. The speedup is higher in the case $\epsilon^0 = 2\sigma_f$ because τ_c/τ_f is significantly larger, as shown in Table 6.

5.1.3. Strong and weak scaling

In this section, we examine the strong and weak scaling of Algorithm 1 in the case $\epsilon^0 = 3\sigma_f$. As in Section 5.1.2, we choose $\tau_f = 10^{-6}$ and the coarse solver described in Table 6. In Sections 5.1.1 and 5.1.2, $T = 1$ and $N_c = 40$ are fixed. Here, they will be varied as appropriate.

To investigate strong scaling, we fix $T = 1$ and calculate the speedup of Algorithm 1 when $N_c = 5,\ 10,\ 20,\ 40$. As $N_{\mathcal{G}} = 2$ for the coarse solver of choice (see Table 6), $N_{\mathcal{G}} < N_c$ in all four cases and thus, the auxiliary coarse solvers can always be run in parallel as desired. Decay of the relative error of Algorithm 1 as iteration count increases from 1 to 3 for each N_c is displayed in Fig. 9a. The four error curves are close to one another, indicating that the accuracy of Algorithm 1 is insensitive to N_c. In Fig. 9b, we display proportions of the runtime spent on serial sweeps and parallel sweeps when $N_{it} = 1$ for each N_c. Since $N_{it} = 1$, two serial sweeps and one parallel sweep are performed. We observe that as N_c increases, the runtime becomes less dominated by the time spent on the parallel sweep. This can be explained as follows. We first note that Eq. (12) can be rewritten as

$$\Upsilon_{pr} \approx (N_{it} + 1) \cdot N_c \cdot \gamma_{\mathcal{G}} + N_{it} \cdot \gamma_{\mathcal{F}} + \Upsilon_{oh} \tag{37}$$

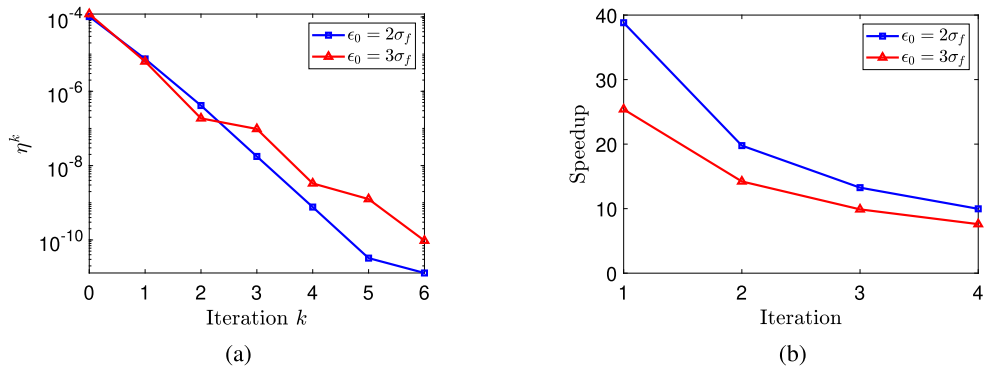

Fig. 8. The accuracy and parallel speedup of Algorithm 1 in the cases $\epsilon^0 = 2\sigma_f$ and $\epsilon^0 = 3\sigma_f$. The coarse solvers are specified in Table 6. The number of cores used is 40. Decay of the relative error (a) and measured speedup (b) as iteration count increases.

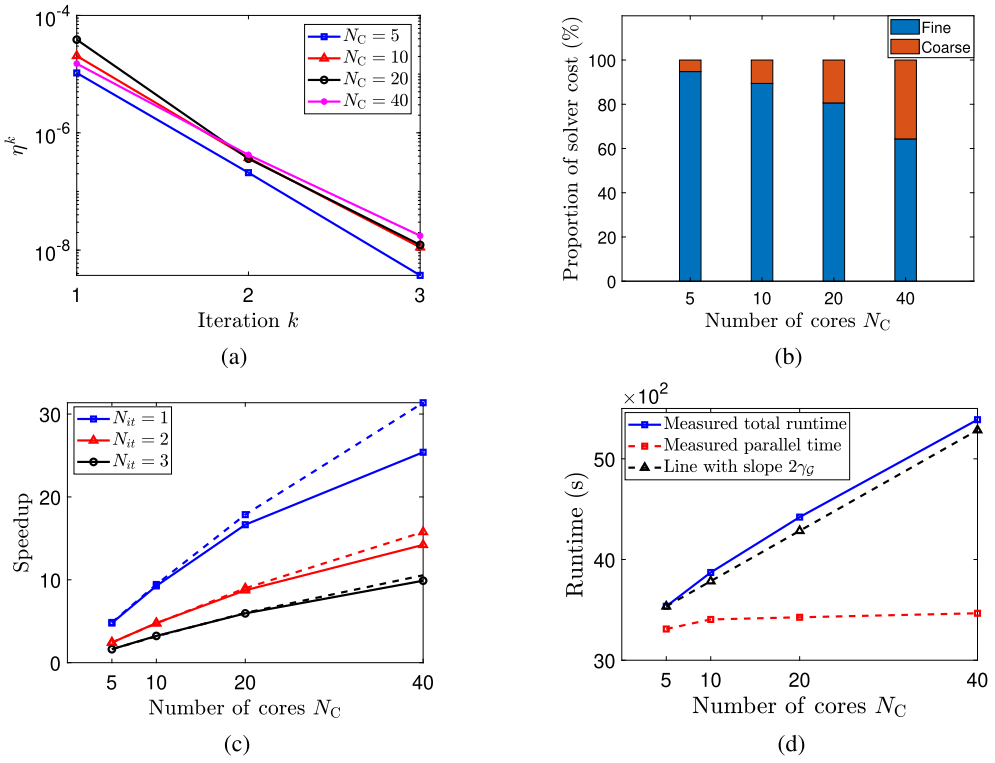

Fig. 9. The strong scaling and weak scaling of Algorithm 1 in the case $\epsilon^0 = 3\sigma_f$. The coarse solver is the one specified in Table 6. In (a), (b), and (c), $T = 1$ is fixed. In (d), $T = 0.125,\ 0.250,\ 0.500,\ 1.000$ when $N_c = 5,\ 10,\ 20,\ 40$. (a) Decay of the relative error of Algorithm 1 as iteration count increases. (b) Proportions of the time spent on the parallel sweep and the time spent on serial sweeps when $N_{it} = 1$. (c) The measured speedup (solid line) and theoretical speedup (dashed line) when $N_{it} = 1,\ 2,\ 3$. (d) Runtime of Algorithm 1 and runtime of the parallel sweep when $N_{it} = 1$.

given $N_{it} \ll 2N_c$. When T is fixed, $\gamma_{\mathcal{G}}$ and $\gamma_{\mathcal{F}}$, which respectively denote the times entailed by the coarse solver and the fine solver to sweep each time slice, decrease like $1/N_c$ as N_c increases. Thus, if N_{it} is fixed as well, as N_c increases, the time spent on serial sweeps (the first term on the right-hand side of Eq. (37)) stays about the same, whereas the time spent on parallel sweeps (the second term on the right-hand side of Eq. (37)) decreases like $1/N_c$. Since the total runtime decreases and the time spent on serial sweeps stays the same, the proportion of the runtime spent on serial sweeps increases with N_c, as shown in Fig. 9b. The measured speedup and theoretical speedup as N_c increases are illustrated in Fig. 9c for $N_{it} = 1,\ 2,\ 3$. As N_c increases from 5 to 40, the increase in the measured speedup is about fivefold for each choice of N_{it}. We also observe that as N_c increases while N_{it} is fixed or as N_{it} decreases while N_c is fixed, the effect of the overhead of parallel computing on speedup becomes more pronounced. This can again be explained using Eq. (37). As we increase N_c while fixing N_{it}, the sum of the first two times in Eq. (37) decreases, making the overhead (the third term on the right-hand side of Eq. (37)) more prominent. If we decrease N_{it} while fixing N_c, a similar argument can be made.

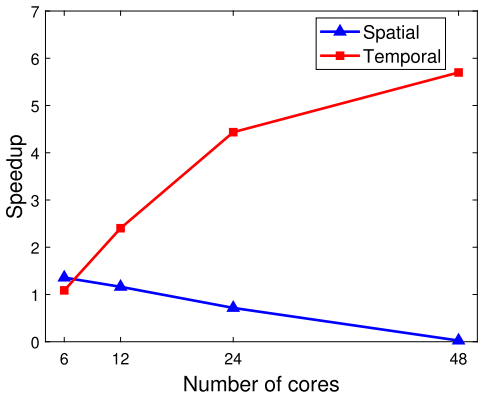

Fig. 10. Comparison of the measured speedups of the spatially parallelized fine solver and Algorithm 1 in the case where the number of rods is five and $\epsilon^0 = 3\sigma_f$. The coarse solver is the one specified in Table 6. The stopping criterion of Algorithm 1 is $\widetilde{\eta}^k < 10^{-5}$. $T = 1$ is fixed.

To examine weak scaling, we fix $N_{\text{it}} = 1$ and choose $T = 0.125,\ 0.250,\ 0.500,\ 1.000$ when $N_{\text{c}} = 5,\ 10,\ 20,\ 40$, respectively. (As N_{c} increases, we also increase the temporal instead of the spatial scale of the problem because Algorithm 1 is a parallel-in-time method.) The runtime of Algorithm 1 in the four cases are shown in Fig. 9d. Since T/N_{c} remains fixed, $\gamma_{\mathcal{G}}$ and $\gamma_{\mathcal{F}}$ stay the same as well. Since $N_{\text{it}} = 1$ is also fixed, as N_{c} increases, the time spent on the parallel sweep should be approximately the same, as shown in Fig. 9d. By Eq. (37), the total runtime of Algorithm 1 is about $2\gamma_{\mathcal{G}} \cdot N_{\text{c}} + \gamma_{\mathcal{F}}$ where $\gamma_{\mathcal{G}}$ and $\gamma_{\mathcal{F}}$ are constants. Thus, its graph is approximately a straight line. This is again verified by Fig. 9d, where a line with slope $2\gamma_{\mathcal{G}}$ is plotted next to the graph of total runtime for reference.

5.1.4. Comparison of spatial and temporal parallelization

In Sections 5.1.1 to 5.1.3, we have applied Algorithm 1 to simulate the swimming motion of a single rod. We observe that in this case, due to the small number of grid points ($N_\sigma = 301$), the overhead of parallel computing is too high for the spatially parallelized fine solver $\mathcal{F}$ to achieve a speedup greater than 1 regardless of how many cores are used.[2] Therefore, Algorithm 1 is superior when the number of rods is one.

Here, we consider five interacting rod-like swimmers instead to further compare spatial parallelization and temporal parallelization. Accordingly, the total number of grid points in $\mathcal{F}$ is $N_\sigma = 1505$. We compare the measured speedups of the spatially parallelized $\mathcal{F}$ and Algorithm 1 while increasing N_{c} in the case $\epsilon^0 = 3\sigma_f$. The coarse solver used in Algorithm 1 is again the one specified in Table 6 and $T = 1$ is fixed. For all values of N_{c}, the stopping criterion of Algorithm 1 is $\widetilde{\eta}^k < 10^{-5}$, where $\widetilde{\eta}^k$ is the relative increment defined in Eq. (34). As seen in Fig. 10, when $N_{\text{c}} = 6$, the speedups of the two algorithms are similar and both greater than 1. As N_{c} increases, the advantage of Algorithm 1 becomes more evident. In particular, its speedup increases with N_{c}, whereas the speedup of the spatially parallelized $\mathcal{F}$ decreases as N_{c} increases and goes below 1 for N_{c} greater than 12, indicating that the spatially parallelized $\mathcal{F}$ actually becomes slower than the serial $\mathcal{F}$.

This example shows that for a fixed problem, as the number of cores employed increases, even though spatial parallelization may outperform temporal parallelization at first, the speedup of temporal parallelization can continue to grow after the speedup of spatial parallelization has stagnated.

5.2. A spherical swimmer

Models of capsules and vesicles with an elastic membrane immersed in viscous flow have been used to study the biomembrane mechanics of red blood cell, artificial capsules in drug delivery, and liquid droplets [60]. We apply Algorithm 1 to simulate the dynamics of an elastic spherical surface in a simple shear flow in an unbounded domain. The surface can be modeled by a network of springs subject to Hooke's Law if it only undergoes small deformations. We assume that the springs are in their relaxed states at time $t = 0$. As time goes on, the interplay between the shear flow and spring forces causes the surface to deform: while the shear flow would "stretch" the springs, the displacements would in turn generate spring forces to "pull back" the springs.

The spherical surface is $x^2 + y^2 + z^2 = 1$ and placed in the shear flow $\mathbf{u}_\infty(x, y, z) = (0.1z, 0, 0)^T$. We triangulate it using $N_\sigma = 642$ grid points that form 1280 almost uniform triangles. The length of the longest edge of them, denoted by $l_{\max}$, is about 0.165. Each edge is modeled as a Hookean spring whose stiffness is characterized by a positive constant κ. The force exerted to each grid point is the sum of the spring forces exerted by all the edges incident with that point; and the grid

[2] We emphasize that this is observed in the specific parallel computing environment we use. In [40] where the computing environment is very different from ours, for example, it was reported that the speedup of spatial parallelization is greater than 1 when there is also only one rod and the number of grid points is even smaller than 301.

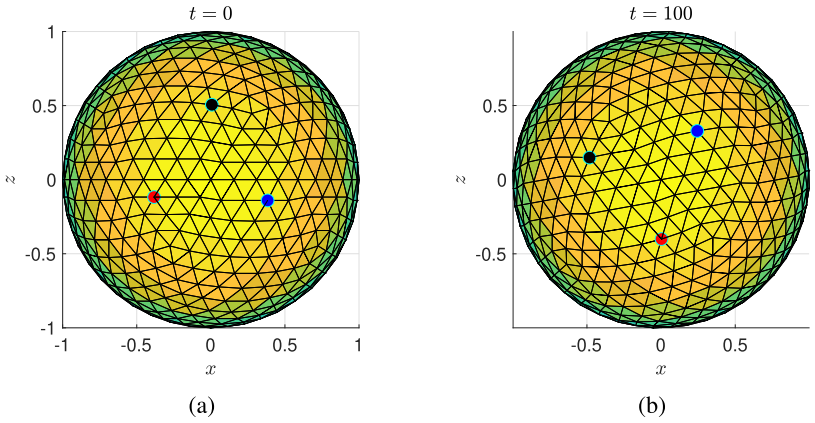

Fig. 11. The projections of the surface onto the xz plane at $t = 0$ (a) and $t = 100$ (b). Three grid points are tracked to show rotation.

point exerts the opposite force to the surrounding fluid. Similar force models have been considered in [61–63] to simulate the motion of spherical surfaces in a viscous shear flow. The fluid velocity at any point $\mathbf{x}$ is given by $\mathbf{u}_\infty(\mathbf{x}) + \mathbf{u}^s(\mathbf{x})$, where $\mathbf{u}^s(\mathbf{x})$ is induced by the spring forces and calculated using Eq. (6).

The fine solver $\mathcal{F}$ and coarse solver $\mathcal{G}$ use the same grid specified above. As in Section 5.1, the ODE solver used in $\mathcal{G}$ is forward Euler method with step size τ_c. The ODE solver used in $\mathcal{F}$ is RK2 or RK4 with step size τ_f. The time steps are chosen such that throughout the simulation, the surface area does not fluctuate more than 0.01%.

The set of physical parameters, Θ, may include one or more of the following: the regularization parameter ϵ used in the MRS, the spring constant κ, and the fluid viscosity μ. The correct values of ϵ, κ, and μ are denoted by ϵ^0, κ^0, and μ^0, respectively and always used in $\mathcal{F}$. We fix $\mu^0 = 0.001\ g \cdot \mu m^{-1} \cdot s^{-1}$ and consider a few values of κ^0 and ϵ^0. Although the surface has no thickness, the biological structures that it represents do. Like for the rod-like swimmers considered in Section 5.1, their thickness is modeled by ϵ in the MRS. Let $\{\epsilon^j\}_{j=1}^{N_\epsilon}$, $\{\kappa^j\}_{j=1}^{N_\kappa}$, and $\{\mu^j\}_{j=1}^{N_\mu}$ denote the values used for ϵ, κ, and μ in Lagrangian extrapolation, respectively. As in Section 5.1, $N_\mathcal{G} \le N_c$ so that the auxiliary solvers that constitute $\mathcal{G}$ can always be run in parallel.

The time domain is $[0, 100]$, and the number of cores used is $N_c = 40$.

We perform two sets of experiments that are similar to the ones in Section 5.1.2. In the first set, $\epsilon^0 = 0.3 l_{\max}$, $\kappa^0 = 0.1\ g/s^2$, and the ODE solver used in $\mathcal{F}$ is RK2 with $\tau_f = 0.001$. In Figs. 11a and 11b, we show the projections of the surface onto the xz plane at time $t = 0$ and $t = 100$, respectively. The surface is rather stiff in this case, as evidenced by the lack of deformation in Fig. 11b. The movement of the three grid points highlighted in Figs. 11a and 11b also indicates that the sphere undergoes rotation. We examine the performance of four versions of Algorithm 1 that employ the four different coarse solvers specified in Table 7. Note that $\mathcal{G}^4$ is simply $\mathcal{G}_{\Theta^0,\tau_c}$ (no extrapolation). For each version of Algorithm 1, the decay of the relative error as iteration count increases is displayed in Fig. 12a, and the increase of speedup as the tolerance of error relaxes is shown in Fig. 12b. As observed in Section 5.1.2 for the rod-like swimmer, the best coarse solver to use depends on the desired accuracy; and for the range of error considered in Fig. 12b, one of $\mathcal{G}^1$, $\mathcal{G}^2$, and $\mathcal{G}^3$, which is constructed using Lagrangian extrapolation alone or both Lagrangian extrapolation and Richardson extrapolation, always leads to higher speedup than $\mathcal{G}^4$.

In the second set of experiments, $\epsilon^0 = 0.5 l_{\max}$, $\kappa^0 = 0.05\ g/s^2$, and the ODE solver used in $\mathcal{F}$ is RK4 with $\tau_f = 0.002$. We examine the performance of three versions of Algorithm 1 that employ the three different coarse solvers specified in Table 8. The coarse solvers are constructed using Lagrangian extrapolation alone or both Lagrangian extrapolation and Richardson extrapolation. The accuracy and speedup of each version are illustrated in Fig. 13. Comparing Figs. 12 and 13, we notice that the speedup in this case is considerably higher than the speedup in the previous case. This is mainly because τ_c/τ_f is larger than before (see Tables 7 and 8) and a computationally more expensive ODE solver, RK4, is used in the fine solver as well.

Comparing Figs. 12c, 13c with Figs. 6c, 7c, we also observe that the effect of overhead on speedup is more prominent in the case of a spherical swimmer. This is mainly because the runtime of Algorithm 1 applied to the two examples above is only about 10% of the runtime of Algorithm 1 applied to the case $\epsilon^0/\sigma_f = 5$ or 7 in Section 5.1.2.

6. Conclusion

In the simulation of a fluid flow around dynamic biological structures, the speedup of spatial parallelization can saturate as the number of computer cores employed increases. The main contributions of this paper include demonstrating the applicability of Parareal, a popular parallel-in-time method, to such a simulation and building novel, non-intrusive coarse solvers on existing ones to improve the speedup of Parareal. Our numerical results show that the proposed variant of Parareal is a highly competitive alternative to spatial parallelization at expediting the simulation of a biofluid. We also

Table 7
Coarse solvers for the case $\epsilon^0 = 0.3l_{\max}$ considered in section 5.2. ($\tau_f = 0.001$.)

Coarse solver	Specifications	$N_{\mathcal{G}}$
$\mathcal{G}^1$	$\tau_c = 20\tau_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, \kappa, \mu\}$, $N_\epsilon = 4$, $N_k = 3$, $N_\mu = 3$ $\epsilon^j/\epsilon^0 = 1.6,\ 1.9,\ 2.2,\ 2.5$ for $j = 1,\ 2,\ 3,\ 4$ $\kappa^j/\kappa^0 = 0.6,\ 0.7,\ 0.8$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 2.0,\ 2.5,\ 3.0$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^2$	$\tau_c = 10\tau_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, \kappa, \mu\}$, $N_\epsilon = 4$, $N_k = 3$, $N_\mu = 3$ $\epsilon^j/\epsilon^0 = 1.350,\ 1.525,\ 1.700,\ 1.875$ for $j = 1,\ 2,\ 3,\ 4$ $\kappa^j/\kappa^0 = 0.7,\ 0.8,\ 0.9$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.40,\ 1.65,\ 1.90$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^3$	$\tau_c = 5\tau_f$ $m = 1$, $r = 0.83$, $\Theta = \{\epsilon\}$, $N_\epsilon = 4$ $\epsilon^j/\epsilon^0 = 1.3,\ 1.45,\ 1.6,\ 1.75$ for $j = 1,\ 2,\ 3,\ 4$	8
$\mathcal{G}^4$	$\tau_c = 3\tau_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \emptyset$ (no Lagrangian extrapolation)	1

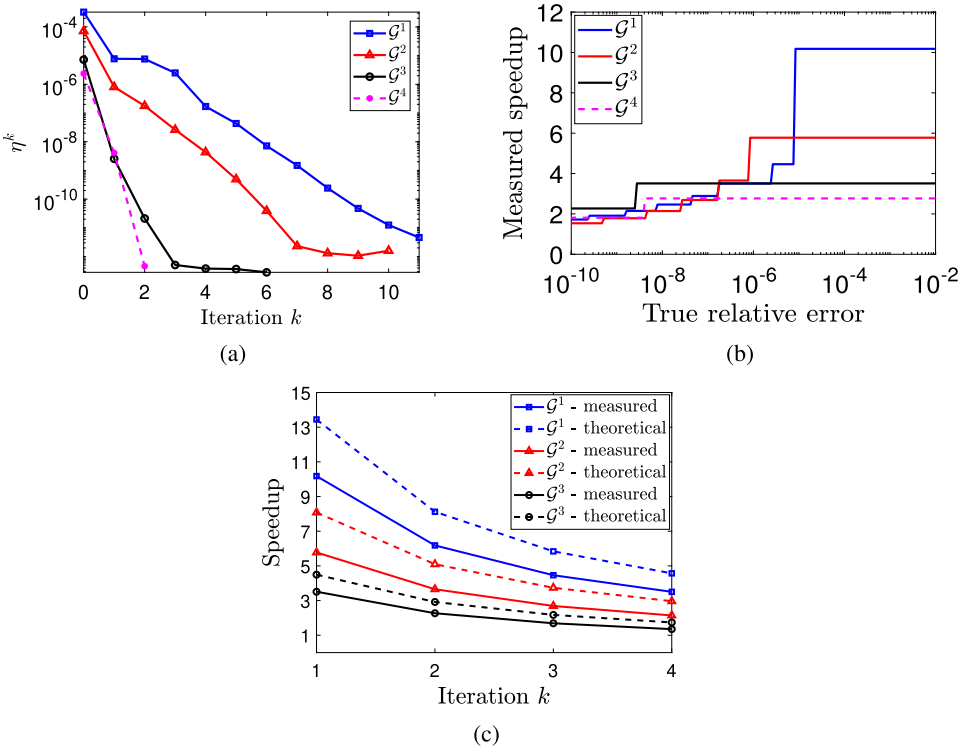

Fig. 12. The accuracy and parallel speedup of four versions of Algorithm 1 in the case $\epsilon^0 = 0.3l_{\max}$. The four versions use four different coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$, and $\mathcal{G}^4$, which are specified in Table 7. The number of cores used is 40. The ODE solver used in the fine solver is RK2. (a) Decay of the relative error as iteration count increases. (b) Increase of the parallel speedup as the desired error increases. (c) Measured and theoretical speedup after one to four iterations.

observe that for different regimes of the regularization parameter used in the Method of Regularized Stokeslets (MRS), the solver for fluid-structure interactions in this work, different strategies for constructing the coarse solver may need to be adopted in order for Parareal to achieve the highest speedup.

To be more precise, we propose to construct the coarse solver for Parareal via extrapolation: either the Lagrangian extrapolation with respect to a set of physical parameters, or the Richardson extrapolation with respect to the time step used by the ODE solver, or a combination of the two. The physical parameters can include the regularization parameter in the MRS, fluid viscosity, and parameters that characterize the material properties of the biological structures. Lagrangian extrapolation can relax the restriction on time step commonly encountered in the simulation of a fluid around elastic structures, and Richardson extrapolation increases the order of accuracy of the ODE solver. The resulting solver is a linear combination of a sequence of existing solvers that use various parameter values and/or time steps; and applying it simply entails ap-

Table 8
Coarse solvers for the case $\epsilon^0 = 0.5 l_{\max}$ considered in section 5.2. ($\tau_f = 0.002$.)

Coarse solver	Specifications	$N_{\mathcal{G}}$
$\mathcal{G}^1$	$\tau_c = 35\tau_f$ $m = 0$ (no Richardson extrapolation), $\Theta = \{\epsilon, \kappa, \mu\}$, $N_\epsilon = 4$, $N_\kappa = 3$, $N_\mu = 3$ $\epsilon^j/\epsilon^0 = 1.50,\ 1.75,\ 2.00,\ 2.25$ for $j = 1,\ 2,\ 3,\ 4$ $\kappa^j/\kappa^0 = 0.6,\ 0.7,\ 0.8$ for $j = 1,\ 2,\ 3$ $\mu^j/\mu^0 = 1.6,\ 1.9,\ 2.2$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^2$	$\tau_c = 22\tau_f$ $m = 1$, $r = 0.88$, $\Theta = \{\epsilon, \kappa, \mu\}$, $N_\epsilon = 3$, $N_\kappa = 2$, $N_\mu = 3$ $\epsilon^j/\epsilon^0 = 1.350,\ 1.525,\ 1.700$ for $j = 1,\ 2,\ 3$ $\kappa^j/\kappa^0 = 0.70,\ 0.85$ for $j = 1,\ 2$ $\mu^j/\mu^0 = 1.350,\ 1.525,\ 1.700$ for $j = 1,\ 2,\ 3$	36
$\mathcal{G}^3$	$\tau_c = 14\tau_f$ $m = 1$, $r = 0.93$, $\Theta = \{\epsilon, \kappa\}$, $N_\epsilon = 4$, $N_\kappa = 3$ $\epsilon^j/\epsilon^0 = 1.2,\ 1.3,\ 1.4,\ 1.5$ for $j = 1,\ 2,\ 3,\ 4$ $\kappa^j/\kappa^0 = 0.80,\ 0.85,\ 0.90$ for $j = 1,\ 2,\ 3$	24

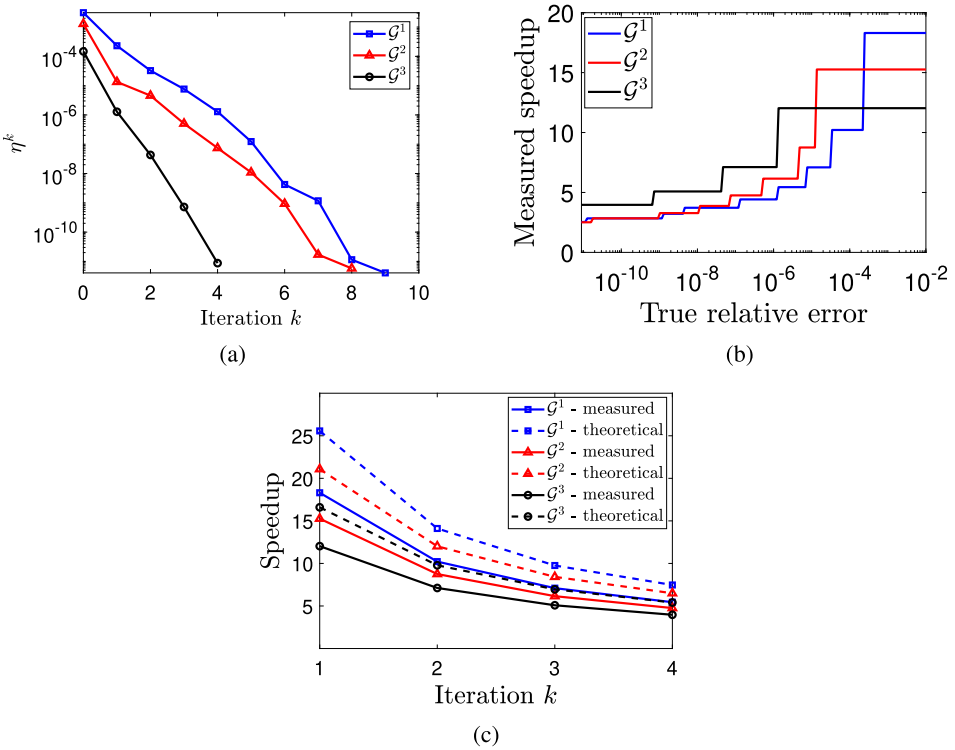

Fig. 13. The accuracy and parallel speedup of three versions of Algorithm 1 in the case $\epsilon^0 = 0.5 l_{\max}$. The three versions use three different coarse solvers $\mathcal{G}^1$, $\mathcal{G}^2$, and $\mathcal{G}^3$, which are specified in Table 8. The number of cores used is 40. The ODE solver used in the fine solver is RK4. (a) Decay of the relative error as iteration count increases. (b) Increase of the parallel speedup as the desired error increases. (c) Measured and theoretical speedup after one to four iterations.

plying the sequence of solvers, which is non-intrusive and straightforward to parallelize. If we choose the parameter values and time steps appropriately, the new solver can achieve higher accuracy without requiring additional runtime. We note that neither extrapolation technique is limited to the MRS or to the simulation of biofluids and they are both applicable to a wide range of problems.

We plan to explore the following three directions in future work. In the original Parareal, only one core is utilized in the serial stage. In this paper, we propose to utilize multiple cores in the serial stage to parallelize the new coarse solvers. The pipelined Parareal [16] also utilizes multiple cores in the serial stage (see Remark 3 in Section 3). A comparison of the two will be quite interesting. In this work, we simply use a Runge-Kutta (RK) method as the fine solver. In the hybrid Parareal Spectral Deferred Corrections (SDC) method [16], instead of a high-order RK method, the fine solver is one first-order SDC "sweep" [64], which is considerably more efficient. The numerical experiments in [16] demonstrate that this replacement does not significantly affect the convergence of Parareal. Replacing the RK method by a first-order SDC sweep in the fine solver may further improve the speedup of the variant of Parareal proposed here. After the biological structures

immersed in the fluid have been discretized by a Lagrangian grid, evaluating the velocities at all N_σ grid points using Eq. (6) is an N_σ-body problem whose asymptotic computational complexity is $O(N_\sigma^2)$. When the number of structures is large, this can be prohibitively expensive. In previous work [43,65], we have extended the Kernel-Independent Fast Multipole Method (KIFMM) [66,67], a fast summation method, to expedite the evaluation of velocities in the MRS, which reduces this complexity to $O(N_\sigma)$. Another future direction is to consider spatially large-scale problems in which there are hundreds or more structures and solve them by coupling KIFMM and Parareal.

CRediT authorship contribution statement

Weifan Liu: Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Minghao W. Rostami:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing.

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.

Acknowledgements

Liu and Rostami were supported, in part, by the National Science Foundation under award DMS-1818833 (PI: Rostami). Rostami was also supported, in part, by the Simons Foundation under award 527247 and by the Oak Ridge Associated Universities under a Ralph E. Powe Junior Faculty Enhancement Award. Access to a supercomputer was provided by the Extreme Science and Engineering Discovery Environment under startup allocation DMS-200009 (PI: Liu, co-PI: Rostami).

References

[1] J. Nievergelt, Parallel methods for integrating ordinary differential equations, Commun. ACM 7 (12) (1964) 731–733, https://doi.org/10.1145/355588.365137.
[2] C.W. Gear, Parallel methods for ordinary differential equations, Calcolo 25 (1–2) (1988) 1–20, https://doi.org/10.1007/BF02575744.
[3] K. Burrage, Parallel and Sequential Methods for Ordinary Differential Equations, Clarendon Press, New York, NY, 1995.
[4] M.J. Gander, 50 years of time parallel time integration, in: T. Carraro, M. Geiger, S. Körkel, R. Rannacher (Eds.), Multiple Shooting and Time Domain Decomposition Methods, in: Contributions in Mathematical and Computational Sciences, vol. 9, Springer, Cham, 2015, pp. 69–113.
[5] B.W. Ong, J.B. Schroder, Applications of time parallelization, Comput. Vis. Sci. 23 (2020) 11, https://doi.org/10.1007/s00791-020-00331-4.
[6] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, P. Gibbon, A massively space-time parallel N-body solver, in: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–11.
[7] J.-L. Lions, Y. Maday, G. Turinici, A "parareal" in time discretization of PDE's, C. R. Acad. Sci. Paris, Ser. I 332 (7) (2001) 661–668, https://doi.org/10.1016/S0764-4442(00)01793-6.
[8] M.J. Gander, S. Vandewalle, Analysis of the parareal time-parallel time-integration method, SIAM J. Sci. Comput. 29 (2) (2007) 556–578, https://doi.org/10.1137/05064607X.
[9] C. Farhat, M. Chandesris, Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications, Int. J. Numer. Methods Eng. 58 (9) (2003) 1397–1434, https://doi.org/10.1002/nme.860.
[10] G. Bal, On the convergence and the stability of the parareal algorithm to solve partial differential equations, in: T.J. Barth, M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, J. Xu (Eds.), Domain Decomposition Methods in Science and Engineering, in: Lecture Notes in Computational Science and Engineering, vol. 40, Springer, Berlin, Heidelberg, 2005, pp. 425–432.
[11] G.A. Staff, E.M. Rønquist, Stability of the Parareal algorithm, in: T.J. Barth, M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, J. Xu (Eds.), Domain Decomposition Methods in Science and Engineering, in: Lecture Notes in Computational Science and Engineering, vol. 40, Springer, Berlin, Heidelberg, 2005, pp. 449–456.
[12] M.J. Gander, S. Vandewalle, On the superlinear and linear convergence of the parareal algorithm, in: O.B. Widlund, D.E. Keyes (Eds.), Domain Decomposition Methods in Science and Engineering XVI, in: Lecture Notes in Computational Science and Engineering, vol. 55, Springer, Berlin, Heidelberg, 2007, pp. 291–298.
[13] M.J. Gander, E. Hairer, Nonlinear convergence analysis for the parareal algorithm, in: U. Langer, M. Discacciati, D.E. Keyes, O.B. Widlund, W. Zulehner (Eds.), Domain Decomposition Methods in Science and Engineering XVII, in: Lecture Notes in Computational Science and Engineering, vol. 60, Springer, Berlin, Heidelberg, 2008, pp. 45–56.
[14] E. Aubanel, Scheduling of tasks in the parareal algorithm, Parallel Comput. 37 (3) (2011) 172–182, https://doi.org/10.1016/j.parco.2010.10.004.
[15] M.J. Gander, M. Petcu, Analysis of a Krylov subspace enhanced parareal algorithm for linear problems, in: E. Cancès, S. Faure, B. Graille (Eds.), Proc. of Paris-Sud Working Group on Modelling and Scientific Computing 2007-2008, in: ESAIM: Proceedings, EDP Sciences, vol. 25, 2008, pp. 114–129.
[16] M.L. Minion, A hybrid parareal spectral deferred corrections method, Commun. Appl. Math. Comput. Sci. 5 (2) (2010) 265–301, https://doi.org/10.2140/camcos.2010.5.265.
[17] M. Emmett, M.L. Minion, Toward an efficient parallel in time method for partial differential equations, Commun. Appl. Math. Comput. Sci. 7 (1) (2012) 105–132, https://doi.org/10.2140/camcos.2012.7.105.
[18] M.J. Gander, S. Güttel, PARAEXP: a parallel integrator for linear initial-value problems, SIAM J. Sci. Comput. 35 (2) (2013) C123–C142, https://doi.org/10.1137/110856137.
[19] R.D. Falgout, S. Friedho, T.V. Kolev, S.P. MacLachlan, J.B. Schroder, Parallel time integration with multigrid, SIAM J. Sci. Comput. 36 (6) (2014) C635–C661, https://doi.org/10.1137/130944230.
[20] L. Baffico, S. Bernard, Y. Maday, G. Turinici, G. Zérah, Parallel-in-time molecular-dynamics simulations, Phys. Rev. E 66 (5) (2002) 057701, https://doi.org/10.1103/PhysRevE.66.057701.

[21] C. Audouze, M. Massot, S. Volzc, Symplectic multi-time step parareal algorithms applied to molecular dynamics, hal-00358459, available from: https://hal.archives-ouvertes.fr/file/index/docid/358459/filename/CAMMSV_moleculardynamics.pdf, 2009.
[22] G. Bal, Y. Maday, A"parareal" time discretization for non-linear PDE's with application to the pricing of an American put, in: L.F. Pavarino, A. Toselli (Eds.), Recent Developments in Domain Decomposition Methods, in: Lecture Notes in Computational Science and Engineering, vol. 23, Springer, Berlin, Heidelberg, 2002, pp. 189–202.
[23] G. Pagès, O. Pironneau, G. Sall, The parareal algorithm for American options, C. R. Acad. Sci. Paris, Ser. I 354 (11) (2016) 1132–1138, https://doi.org/10.1016/j.crma.2016.09.010.
[24] N. Margenberg, T. Richter, Parallel time-stepping for fluid–structure interactions, Math. Model. Nat. Phenom. 16 (2021) 20.
[25] J.M.F. Trindade, J.C.F. Pereira, Parallel-in-time simulation of the unsteady Navier-Stokes equations for incompressible flow, Int. J. Numer. Methods Fluids 45 (10) (2004) 1123–1136, https://doi.org/10.1002/fld.732.
[26] P.F. Fischer, F. Hecht, Y. Maday, A Parareal in time semi-implicit approximation of the Navier-Stokes equations, in: T.J. Barth, M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick, R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, J. Xu (Eds.), Domain Decomposition Methods in Science and Engineering, in: Lecture Notes in Computational Science and Engineering, vol. 40, Springer, Berlin, Heidelberg, 2005, pp. 433–440.
[27] J.M.F. Trindade, J.C.F. Pereira, Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows, Numer. Heat Transf., Part B, Fundam. 50 (1) (2006) 25–40, https://doi.org/10.1080/10407790500459379.
[28] R. Croce, D. Ruprecht, R. Krause, Parallel-in-space-and-time simulation of the three-dimensional, unsteady Navier-Stokes equations for incompressible flow, in: H. Bock, X. Hoang, R. Rannacher, J. Schlöder (Eds.), Modeling, Simulation and Optimization of Complex Processes - HPSC 2012, Springer, Cham, 2014, pp. 13–23.
[29] J. Steiner, D. Ruprecht, R. Speck, R. Krause, Convergence of Parareal for the Navier-Stokes equations depending on the Reynolds number, in: A. Abdulle, S. Deparis, D. Kressner, F. Nobile, M. Picasso (Eds.), Numerical Mathematics and Advanced Applications - ENUMATH 2013, in: Lecture Notes in Computational Science and Engineering, vol. 103, Springer, Cham, 2015, pp. 195–202.
[30] E. Celledoni, T. Kvamsdal, Parallelization in time for thermo-viscoplastic problems in extrusion of aluminium, Int. J. Numer. Methods Eng. 79 (5) (2009) 576–598, https://doi.org/10.1002/nme.2585.
[31] Q. Wang, S.A. Gomez, P.J. Blonigan, A.L. Gregory, E.Y. Qian, Towards scalable parallel-in-time turbulent flow simulations, Phys. Fluids 25 (11) (2013) 110818, https://doi.org/10.1063/1.4819390.
[32] W. Agboh, O. Grainger, D. Ruprecht, M. Dogar, Parareal with a learned coarse model for robotic manipulation, Comput. Vis. Sci. 23 (2020) 8, https://doi.org/10.1007/s00791-020-00327-0.
[33] R. Cortez, The method of regularized Stokeslets, SIAM J. Sci. Comput. 23 (4) (2001) 1204–1225, https://doi.org/10.1137/S106482750038146X.
[34] R. Cortez, L. Fauci, A. Medovikov, The method of regularized Stokeslets in three dimensions: analysis, validation, and application to helical swimming, Phys. Fluids 17 (3) (2005) 031504, https://doi.org/10.1063/1.1830486.
[35] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 459–517, https://doi.org/10.1017/S0962492902000077.
[36] B.E. Griffith, R.D. Hornung, D.M. McQueen, C.S. Peskin, An adaptive, formally second order accurate version of the immersed boundary method, J. Comput. Phys. 223 (1) (2002) 10–49, https://doi.org/10.1016/j.jcp.2006.08.019.
[37] C. Pozrikidis, Boundary Integral and Singularity Methods for Linearized Viscous Flow, Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, UK, 1992.
[38] J. Rotne, S. Prager, Variational treatment of hydrodynamic interaction in polymers, J. Chem. Phys. 50 (11) (1969) 4831–4837, https://doi.org/10.1063/1.1670977.
[39] H. Yamakawa, Transport properties of polymer chains in dilute solution: hydrodynamic interaction, J. Chem. Phys. 53 (1) (1970) 436–443, https://doi.org/10.1063/1.1673799.
[40] S.D. Olson, S. Lim, R. Cortez, Modeling the dynamics of an elastic rod with intrinsic curvature and twist using a regularized Stokes formulation, J. Comput. Phys. 238 (2013) 169–187, https://doi.org/10.1016/j.jcp.2012.12.026.
[41] M.T. Gallagher, D.J. Smith, Passively parallel regularized stokeslets, Philos. Trans. R. Soc. A 378 (2179) (2020) 20190528.
[42] L. Wang, S. Tlupova, R. Krasny, A treecode algorithm for 3D Stokeslets and stresslets, Adv. Appl. Math. Mech. 11 (4) (2019) 737–756, https://doi.org/10.4208/aamm.OA-2018-0187.
[43] M.W. Rostami, S.D. Olson, Kernel independent fast multipole method within the framework of regularized Stokeslets, J. Fluids Struct. 67 (2016) 60–84, https://doi.org/10.1016/j.jfluidstructs.2016.07.006.
[44] T.Y. Hou, Z. Shi, Removing the stiffness of elastic force from the immersed boundary method for the 2D Stokes equations, J. Comput. Phys. 227 (21) (2008) 9138–9169, https://doi.org/10.1016/j.jcp.2008.03.002.
[45] D.-V. Le, J. White, J. Peraire, K.M. Lim, B. Khoo, An implicit immersed boundary method for three-dimensional fluid–membrane interactions, J. Comput. Phys. 228 (22) (2009) 8427–8445, https://doi.org/10.1016/j.jcp.2009.08.018.
[46] J.M. Stockie, B.T. Wetton, Stability analysis for the immersed fiber problem, SIAM J. Appl. Math. 55 (6) (1995) 1577–1591, https://doi.org/10.1137/S0036139994267018.
[47] J.M. Stockie, B.R. Wetton, Analysis of stiffness in the immersed boundary method and implications for time-stepping schemes, J. Comput. Phys. 154 (1) (1999) 41–64, https://doi.org/10.3390/fluids6110387.
[48] J. Ainley, S. Durkin, R. Embid, P. Boindala, R. Cortez, The method of images for regularized Stokeslets, J. Comput. Phys. 227 (9) (2008) 4600–4616, https://doi.org/10.1016/j.jcp.2008.01.032.
[49] O. Shindell, H. Nguyen, N. Coltharp, F. Healy, B. Rodenborn, Using experimentally calibrated regularized Stokeslets to assess bacterial flagellar motility near a surface, Fluids 6 (11) (2021) 387, https://doi.org/10.3390/fluids6110387.
[50] L. Eldén, L. Wittmeyer-Loch, Numerical Analysis: An Introduction, Academic Press, Inc., San Diego, CA, 1990.
[51] A. Sidi, Practical Extrapolation Methods: Theory and Applications, vol. 10, Cambridge University Press, 2003.
[52] R.D. Falgout, T.A. Manteuffel, B. O'Neill, J.B. Schroder, Multigrid reduction in time with Richardson extrapolation, Electron. Trans. Numer. Anal. 54 (2021) 210–233, https://doi.org/10.1553/etna_vol54s210.
[53] P.J. van der Houwen, Parallel step-by-step methods, Appl. Numer. Math. 11 (1) (1993) 69–81, https://doi.org/10.1016/0168-9274(93)90040-X.
[54] S. Wu, B. Shi, C. Huang, Parareal-Richardson algorithm for solving nonlinear ODEs and PDEs, Commun. Comput. Phys. 6 (4) (2009) 883–904.
[55] S. Lim, A. Ferent, X.S. Wang, C.S. Peskin, Dynamics of a closed rod with twist and bend in fluid, SIAM J. Sci. Comput. 31 (1) (2008) 273–302, https://doi.org/10.1137/070699780.
[56] S. Lim, Dynamics of an open elastic rod with intrinsic curvature and twist in a viscous fluid, Phys. Fluids 22 (2) (2010) 024104, https://doi.org/10.1063/1.3326075.
[57] S.D. Olson, Motion of filaments with planar and helical bending waves in a viscous fluid, in: A.T. Layton, S.D. Olson (Eds.), Biological Fluid Dynamics: Modeling, Computations, and Applications, in: Contemporary Mathematics, vol. 628, AMS, 2014, pp. 109–127.
[58] J. Huang, L. Carichino, S.D. Olson, Hydrodynamic interactions of actuated elastic filaments near a planar wall with applications to sperm motility, J. Coupled Syst. Multiscale Dyn. 6 (3) (2018) 163–175, https://doi.org/10.1166/jcsmd.2018.1166.
[59] L. Carichino, S.D. Olson, Emergent three-dimensional sperm motility: coupling calcium dynamics and preferred curvature in a Kirchhoff rod model, Math. Med. Biol. 36 (4) (2019) 439–469, https://doi.org/10.1093/imammb/dqy015.

[60] S.K. Veerapaneni, A. Rahimian, G. Biros, D. Zorin, A fast algorithm for simulating vesicle flows in three dimensions, J. Comput. Phys. 230 (14) (2011) 5610–5634, https://doi.org/10.1016/j.jcp.2011.03.045.
[61] E.L. Bouzarth, M.L. Minion, A multirate time integrator for regularized Stokeslets, J. Comput. Phys. 229 (11) (2010) 4208–4224, https://doi.org/10.1016/j.jcp.2010.02.006.
[62] T. Jackson, A. Radunskaya, Applications of Dynamical Systems in Biology and Medicine, vol. 158, Springer, 2015.
[63] Y. Navot, Elastic membranes in viscous shear flow, Phys. Fluids 10 (8) (1998) 1819–1833, https://doi.org/10.1063/1.869702.
[64] A. Dutt, L. Greengard, V. Rokhlin, Spectral deferred correction methods for ordinary differential equations, BIT 40 (2) (2000) 241–266, https://doi.org/10.1023/A:1022338906936.
[65] M.W. Rostami, S.D. Olson, Fast algorithms for large dense matrices with applications to biofluids, J. Comput. Phys. 394 (2019) 364–384, https://doi.org/10.1016/j.jcp.2019.05.042.
[66] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, J. Comput. Phys. 196 (2004) 591–626, https://doi.org/10.1016/j.jcp.2003.11.021.
[67] L. Ying, A kernel independent fast multipole algorithm for radial basis functions, J. Comput. Phys. 213 (2006) 451–457, https://doi.org/10.1016/j.jcp.2005.09.010.