Computers and Fluids 122 (2015) 165–183

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

Parallel iterative solution of the incompressible Navier–Stokes equations with application to rotating wings

Jakub Šístek[a], Fehmi Cirak[b,*]

[a] Institute of Mathematics of the Czech Academy of Sciences, Žitná 25, 115 67 Prague 1, Czech Republic
[b] Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK

ARTICLE INFO

Article history:
Received 19 May 2015
Revised 23 August 2015
Accepted 27 August 2015
Available online 3 September 2015

Keywords:
Navier–Stokes
Incompressible flow
Krylov subspace methods
Preconditioning
PETSc
Rotating insect wing

ABSTRACT

We discuss aspects of implementation and performance of parallel iterative solution techniques applied to low Reynolds number flows around fixed and moving rigid bodies. The incompressible Navier–Stokes equations are discretised with Taylor-Hood finite elements in combination with a semi-implicit pressure-correction method. The resulting sequence of convection–diffusion and Poisson equations are solved with preconditioned Krylov subspace methods. To achieve overall scalability we consider new auxiliary algorithms for mesh handling and assembly of the system matrices. We compute the flow around a translating plate and a rotating insect wing to establish the scaling properties of the developed solver. The largest meshes have up to 132×10^6 hexahedral finite elements leading to around 3.3×10^9 unknowns. For the scalability runs the maximum core count is around 65.5×10^3. We find that almost perfect scaling can be achieved with a suitable Krylov subspace iterative method, like conjugate gradients or GMRES, and a block Jacobi preconditioner with incomplete LU factorisation as a subdomain solver. In addition to parallel performance data, we provide new highly-resolved computations of flow around a rotating insect wing and examine its vortex structure and aerodynamic loading.

1. Introduction

The implicit computation of three-dimensional flow problems often requires parallel computing. In presence of highly resolved no-slip boundaries the discretisation of the incompressible Navier–Stokes equations can lead to linear systems of equations with several hundred millions to a few billion unknowns. In the course of a transient simulation these systems of equations have to be solved several thousand times. Hence, in order to achieve reasonable simulation turnaround times each system has to be solved within a few minutes. In combination with this computing time requirement, the large memory needs make it essential to use domain decomposition techniques and distributed-memory parallel computing platforms. As known, Krylov subspace iteration methods with efficient preconditioners are the only viable solvers on parallel computers with large processor counts [21,48,56]. In practice, efficient parallel algorithms for mesh handling and system matrix assembly are also relevant. The most efficient combination of iterative and preconditioning techniques usually depends on the specific application at hand. Finding a suitable combination can be greatly facilitated through the use of parallel linear algebraic solver libraries such as PETSc [4] or Trilinos [30]. In this work we make use of the PETSc library and compare techniques for the scalable solution of large-scale low Reynolds number flow problems with up to several billion unknowns.

For computing the flow around a moving rigid body, such as a rotating insect wing, the Navier–Stokes equations can be expressed either in a non-inertial body-fixed frame or in an inertial frame using the arbitrary Lagrangian–Eulerian (ALE) formulation [5,34]. In both approaches a fixed body-fitted finite element mesh is used and there is no need to update the mesh. In our computations we use the ALE formulation and relate the prescribed wing velocity to the ALE mesh velocity. For the considered range of problems the solution of the Navier–Stokes equations with pressure-correction methods can be very efficient. Such methods reduce the solution of the original discretised Navier–Stokes equations to the solution of several smaller subproblems that are solved instead of the original equations [9,28,36,45]. For instance, in the case of Taylor-Hood $Q2$–$Q1$ elements used in this work a mesh with n_e elements leads to a system size of approximately $(25n_e \times 25n_e)$. With the pressure-correction method, three systems of convection–diffusion type of size $(8n_e \times 8n_e)$, one system of Poisson type of size $(n_e \times n_e)$ and one L_2-projection of size $(n_e \times n_e)$ are solved. Moreover, the preconditioning of this smaller system matrices is more straightforward and easier to implement

* Corresponding author. Tel.: +44 1223 332716; fax: +44 (0)1223 339713.
E-mail addresses: sistek@math.cas.cz (J. Šístek), f.cirak@eng.cam.ac.uk (F. Cirak).

http://dx.doi.org/10.1016/j.compfluid.2015.08.026

than the preconditioning of the original indefinite system matrix. We solve each of the five systems with a suitable Krylov subspace method and investigate the performance of additive Schwarz and block Jacobi preconditioners with complete and incomplete LU factorisations as local solvers.

The driving application for the present work is the study of insect flight aerodynamics; see the textbooks [14,50] and review papers [15–19] for an introduction to flapping flight. The relevant Reynolds numbers range from about 100 for a fruit fly to about 10^5 for large insects, such as dragon flies. In order to create sufficient lift insects crucially rely on wings which flap with very high angles of attack (around 35°). This leads to separated flows with periodic vortex generation and shedding, which are exploited by insects to increase lift. The study of translating and rotating wings serves as a stepping-stone towards the understanding the more complex three-dimensional flapping flight. Both types of wing motions lead to the formation of a leading-edge, a trailing-edge and two tip vortices. However, this vortex structure is not stable for a translating wing, and it is periodically formed and shed, see [54] and references therein. Consequently, there are large fluctuations in the lift and drag coefficients of the wing. As first corroborated by the experiments of Ellington et al. [20] the leading-edge vortex for a rotating wing is stable and remains attached to the wing throughout the rotation. The low pressure zone at the vortex core immediately above the leading edge leads to a sustained large lift force. It is believed that the leading-edge vortex is stabilised by centrifugal and Coriolis accelerations, which create spanwise flow advecting vorticity from the leading-edge vortex. The exact mechanisms are however not yet well understood and there is an extensive amount of experimental studies [2,7,35,44] and some recent computational studies on the topic [24,25,29]. In this paper we present several new highly-resolved computations corroborating previous experimental and numerical findings.

The outline of this paper is as follows. Section 2 begins reviewing the incompressible Navier–Stokes equations in ALE form for computing the flow around moving rigid bodies. Subsequently, their solution with the incremental pressure-correction method and their finite element discretisation are introduced. Specifically, in Section 2.3 all the discretised subproblem sizes and types are given. In Section 3 the solution of the obtained discrete problems with parallel preconditioned iterative solvers is discussed. Efficient and scalable algorithms for partitioning of large meshes and assembly of large matrices are given. Section 4 is dedicated to numerical performance studies and presents several new highly resolved computations. First, in Section 4.1 the developed computational approach is validated and verified with the widely studied flow around an inclined flat plate. Subsequently, in Section 4.2 the flow around a rotating insect wing is used to investigate the mathematical and parallel scalability of various preconditioned iterative methods. Finally, the identified most efficient iterative methods are used to study the Reynolds number dependence of the vortex structure around a rotating wing.

2. Pressure-correction method for Navier–Stokes equations in ALE form

In this section we briefly review the ALE formulation of the incompressible Navier–Stokes equations and their finite element discretisation. The discussion is specialised to the simulation of flows around rotating rigid bodies, see Fig. 1. For time discretisation we use the implicit Euler scheme in combination with the semi-implicit pressure-correction technique. At each time step the solution of the Navier–Stokes equations is reduced to the solution of a sequence of convection–diffusion and Poisson problems.

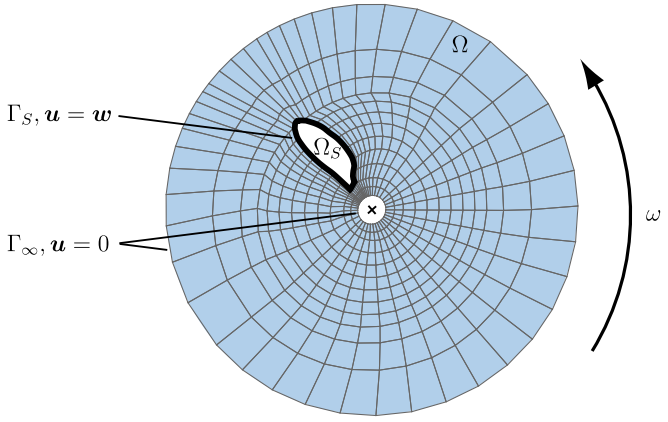


Fig. 1. Rotating insect wing (Ω_S) with a body fitted fluid mesh. In the used ALE formulation the mesh and the wing rotate with the prescribed angular velocity ω.

2.1. Governing equations

We consider the rotation of a rigid body with domain Ω_S and boundary Γ_S embedded in a fluid domain Ω. The boundary of the fluid domain Γ is comprised of two disjoint parts Γ_S and Γ_∞, $\Gamma = \Gamma_S \cup \Gamma_\infty$. The boundary Γ_S is the common interface between fluid and rigid body and Γ_∞ is the free-stream boundary. The rotation of the rigid body is prescribed with the angular velocity vector $\boldsymbol{\omega}$, the centre of rotation is $\boldsymbol{x}_0$ and the corresponding velocity is

$$\boldsymbol{w} = \boldsymbol{\omega} \times (\boldsymbol{x} - \boldsymbol{x}_0). \tag{1}$$

A computationally efficient approach for simulating the flow field generated by the rigid body is to consider the Navier–Stokes equations in a domain moving with velocity $\boldsymbol{w}$, i.e.,

$$\frac{D^{\mathcal{A}}\boldsymbol{u}}{Dt} + ((\boldsymbol{u} - \boldsymbol{w}) \cdot \nabla)\boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = \boldsymbol{0}, \tag{2a}$$

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

where $\boldsymbol{u}$ is the fluid velocity, ν is the kinematic viscosity and p is the normalised pressure. The time derivative in (2) is the ALE derivative

$$\frac{D^{\mathcal{A}}\boldsymbol{u}}{Dt} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{w} \cdot \nabla \boldsymbol{u}. \tag{3}$$

For further details on the ALE formulation of Navier–Stokes equations see, e.g., [8,13]. The Navier–Stokes equations are complemented by the following boundary conditions and the initial condition

$$\begin{aligned} \boldsymbol{u}(t, \boldsymbol{x}) &= \boldsymbol{0} \quad \text{on } \Gamma_\infty, \\ \boldsymbol{u}(t, \boldsymbol{x}) &= \boldsymbol{w} \quad \text{on } \Gamma_S, \\ \boldsymbol{u}(t = 0, \boldsymbol{x}) &= \boldsymbol{0} \quad \text{in } \Omega. \end{aligned} \tag{4}$$

2.2. Incremental pressure-correction method

For discretising the Navier–Stokes equations (2) in time, we use the backward Euler method with constant interval length Δt. In addition, we linearise the nonlinear convective term in (2a) with a semi-implicit approach leading to the discretised equations

$$\begin{aligned} &\frac{1}{\Delta t}\boldsymbol{u}^{n+1} + ((\boldsymbol{u}^n - \boldsymbol{w}) \cdot \nabla)\boldsymbol{u}^{n+1} - \nu \Delta \boldsymbol{u}^{n+1} + \nabla p^{n+1} = \frac{1}{\Delta t}\boldsymbol{u}^n, \\ &\nabla \cdot \boldsymbol{u}^{n+1} = 0, \end{aligned} \tag{5}$$

where the index n indicates the variables associated with the time step t^n.

With a view to parallelisation, the time-discretised semi-implicit Navier–Stokes system (5) can be efficiently solved with a pressure-correction, or a fractional-step, method [9,36]. A review and mathematical analysis of some of the prevalent pressure-correction approaches can be found, e.g., in [28,45]. The specific method we use

is the incremental pressure-correction method in rotational form as discussed in [47] and summarised below. To begin with, we define the pressure increment ψ^{n+1} in order to keep the derivations compact

$$\psi^{n+1} = p^{n+1} - p^n + \nu\nabla\cdot\boldsymbol{u}^{n+1}. \tag{6}$$

To compute the velocity and pressure fields $(\boldsymbol{u}^{n+1}, p^{n+1})$ at time t^{n+1} three successive subproblems are considered.

1. First, the velocity field $\boldsymbol{u}^{n+1}$ is obtained by solving the convection–diffusion problem

$$\begin{aligned}
&\frac{1}{\Delta t}\boldsymbol{u}^{n+1} + ((\boldsymbol{u}^n - \boldsymbol{w})\cdot\nabla)\boldsymbol{u}^{n+1} - \nu\Delta\boldsymbol{u}^{n+1}\\
&\quad = \frac{1}{\Delta t}\boldsymbol{u}^n - \nabla(p^n + \psi^n),\\
&\boldsymbol{u}^{n+1} = \boldsymbol{0} \quad \text{on } \Gamma_\infty,\\
&\boldsymbol{u}^{n+1} = \boldsymbol{w} \quad \text{on } \Gamma_S.
\end{aligned} \tag{7}$$

2. Next, the pressure increment ψ^{n+1} is obtained by solving the Poisson problem

$$\begin{aligned}
&\Delta\psi^{n+1} = \frac{1}{\Delta t}\nabla\cdot\boldsymbol{u}^{n+1},\\
&\frac{\partial\psi^{n+1}}{\partial\boldsymbol{n}} = 0 \quad \text{on } \Gamma_\infty\cup\Gamma_S.
\end{aligned} \tag{8}$$

3. Finally, the pressure field p^{n+1} is updated with

$$p^{n+1} = p^n + \psi^{n+1} - \nu\nabla\cdot\boldsymbol{u}^{n+1}. \tag{9}$$

Note that there is only the velocity field $\boldsymbol{u}$ at times t^{n+1} and t^n in these three equations. The intermediate and the end-of-step velocities familiar from conventional pressure-correction methods have been consolidated to one velocity field, see [28] for details. Furthermore, we do not apply any subiterations within each time step. As discussed in [47] it is possible to employ subiterations in order to improve the accuracy of the projection scheme.

The weak forms of the three subproblems (7)–(9) are needed for their finite element discretisation, see e.g. [21,27]. To this end, we introduce the function spaces

$$\begin{aligned}
\boldsymbol{V} &:= \{\boldsymbol{v}\in[H^1(\Omega)]^3, \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_\infty,\ \boldsymbol{v} = \boldsymbol{w} \text{ on } \Gamma_S\},\\
\boldsymbol{V}_0 &:= \{\boldsymbol{v}\in[H^1(\Omega)]^3, \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma\},
\end{aligned}$$

where $H^1(\Omega)$ is the usual Sobolev space.

The weak form of the convection–diffusion equation (7) reads: find $\boldsymbol{u}^{n+1}\in\boldsymbol{V}$ such that

$$\begin{aligned}
&\frac{1}{\Delta t}(\boldsymbol{u}^{n+1},\boldsymbol{v}) + c(\boldsymbol{u}^n,\boldsymbol{u}^{n+1},\boldsymbol{w},\boldsymbol{v}) + a(\boldsymbol{u}^{n+1},\boldsymbol{v})\\
&\quad = \frac{1}{\Delta t}(\boldsymbol{u}^n,\boldsymbol{v}) - (\nabla(p^n+\psi^n),\boldsymbol{v}) \quad \forall\boldsymbol{v}\in\boldsymbol{V}_0
\end{aligned} \tag{10}$$

with

$$\begin{aligned}
(\boldsymbol{u}^{n+1},\boldsymbol{v}) &= \int_\Omega \boldsymbol{u}^{n+1}\cdot\boldsymbol{v}\,d\boldsymbol{x},\\
a(\boldsymbol{u}^{n+1},\boldsymbol{v}) &= \nu\int_\Omega \nabla\boldsymbol{u}^{n+1} : \nabla\boldsymbol{v}\,d\boldsymbol{x},\\
c(\boldsymbol{u}^n,\boldsymbol{u}^{n+1},\boldsymbol{w},\boldsymbol{v}) &= \int_\Omega ((\boldsymbol{u}^n-\boldsymbol{w})\cdot\nabla)\boldsymbol{u}^{n+1}\cdot\boldsymbol{v}\,d\boldsymbol{x}.
\end{aligned}$$

Notice that the Cartesian components of the momentum equation (10) are decoupled, and (10) reduces to three independent scalar convection–diffusion equations. The weak form of the Poisson equation (8) for the pressure increment reads: find $\psi^{n+1}\in H^1(\Omega)$ such that

$$(\nabla\psi^{n+1},\nabla q) = -\frac{1}{\Delta t}(\nabla\cdot\boldsymbol{u}^{n+1}, q) \quad \forall\, q\in H^1(\Omega). \tag{11}$$

This is a pure Neumann problem and has a one-dimensional nullspace consisting of constant functions, which has implications for its numerical solution, see Section 3.1. Finally, for updating the new pressure field with (9) we use the L_2-projection: find $p^{n+1}\in H^1(\Omega)$ such that

$$(p^{n+1}, q) = (p^n + \psi^{n+1} - \nu\nabla\cdot\boldsymbol{u}^{n+1}, q) \quad \forall\, q\in H^1(\Omega). \tag{12}$$

This projection is only relevant in the finite element context because the divergence of the discrete velocity field $\nabla\cdot\boldsymbol{u}$ is in general discontinuous and the discrete pressure p and pressure increment fields ψ are continuous.

2.3. Finite element discretisation

The weak form of the incremental pressure-correction equations are discretised in space with hexahedral finite elements. Although we use the ALE description of the Navier–Stokes equations, there is no need to solve for mesh position and velocity since both are prescribed. As known, the basis functions for discretising the velocity and pressure fields have to be carefully chosen so that they satisfy the inf-sup, or Babuška–Brezzi stability, condition, see [21,27]. We use the Taylor-Hood $Q2$–$Q1$ elements discretising the velocity and pressure fields with tri-quadratic and tri-linear basis functions, respectively. In the resulting finite element mesh there are $n_{\boldsymbol{u}}$ velocity nodes and n_p pressure nodes with their ratio being $n_{\boldsymbol{u}}/n_p \approx 8$. Notably, in our computations we do not employ any convection stabilisation so that, in effect, performing a direct numerical simulation.

Let us now investigate the systems of linear equations resulting from the discretisation of the weak forms (10)–(12) closer. The approximation of the velocity and pressure fields with the Taylor-Hood elements reads

$$\boldsymbol{u}^h = \begin{pmatrix} u_1^h\\ u_2^h\\ u_3^h\end{pmatrix} = \sum_{i=1}^{n_{\boldsymbol{u}}}\phi_i\begin{pmatrix}u_{1,i}\\ u_{2,i}\\ u_{3,i}\end{pmatrix}, \quad \psi^h = \sum_{i=1}^{n_p}\xi_i\psi_i, \quad p^h = \sum_{i=1}^{n_p}\xi_i p_i. \tag{13}$$

Here ϕ_i and ξ_i are the tri-quadratic and tri-linear basis functions, respectively, associated to the finite element node with index i. Moreover, the nodal unknowns are assembled into the global arrays

$$\begin{aligned}
\boldsymbol{u}_d &= \left(u_{d,1},\ldots,u_{d,n_{\boldsymbol{u}}}\right)^{\mathsf{T}}\in\mathbb{R}^{n_{\boldsymbol{u}}},\\
\boldsymbol{\psi} &= \left(\psi_1,\ldots,\psi_{n_p}\right)^{\mathsf{T}}\in\mathbb{R}^{n_p},\\
\boldsymbol{p} &= \left(p_1,\ldots,p_{n_p}\right)^{\mathsf{T}}\in\mathbb{R}^{n_p}.
\end{aligned} \tag{14}$$

With these definitions at hand the weak forms (10)–(12) correspond, respectively, to the linear equation systems

$$\begin{aligned}
&\left(\frac{1}{\Delta t}\boldsymbol{M}_{\boldsymbol{u}} + \boldsymbol{N} + \nu\boldsymbol{A}_{\boldsymbol{u}}\right)\boldsymbol{u}_d^{n+1}\\
&\quad = \frac{1}{\Delta t}\boldsymbol{M}_{\boldsymbol{u}}\boldsymbol{u}_d^n - \boldsymbol{P}_d(\boldsymbol{p}^n + \boldsymbol{\psi}^n) \quad \text{with } d\in\{1,2,3\},
\end{aligned} \tag{15}$$

$$\boldsymbol{A}_p\boldsymbol{\psi}^{n+1} = -\frac{1}{\Delta t}\sum_{d=1}^{3}\boldsymbol{B}_d\boldsymbol{u}_d^{n+1}, \tag{16}$$

$$\boldsymbol{M}_p\boldsymbol{p}^{n+1} = \boldsymbol{M}_p(\boldsymbol{p}^n + \boldsymbol{\psi}^{n+1}) - \nu\sum_{d=1}^{3}\boldsymbol{B}_d\boldsymbol{u}_d^{n+1}, \tag{17}$$

with the matrices

$$\boldsymbol{M}_{\boldsymbol{u}} = \int_\Omega \phi_i\phi_j\,d\boldsymbol{x} \quad \text{with } i,j = 1,\ldots,n_{\boldsymbol{u}}, \tag{18a}$$

$$\boldsymbol{M}_p = \int_\Omega \xi_i\xi_j\,d\boldsymbol{x} \quad \text{with } i,j = 1,\ldots,n_p, \tag{18b}$$

$$\boldsymbol{N} = \int_\Omega \left((\boldsymbol{u}^n - \boldsymbol{w})\cdot\nabla\phi_j\right)\phi_i\,d\boldsymbol{x} \quad \text{with } i,j = 1,\ldots,n_{\boldsymbol{u}}, \tag{18c}$$

$$\boldsymbol{A}_{\boldsymbol{u}} = \int_\Omega \nabla\phi_i\cdot\nabla\phi_j\,d\boldsymbol{x} \quad \text{with } i,j = 1,\ldots,n_{\boldsymbol{u}}, \tag{18d}$$

Table 1
Summary of the properties of the five linear systems of equations solved in each time step.

		Velocities	Pressure increment	Pressure
Equation	Number	(10), (15)	(11), (16)	(12), (17)
	Type	Convection–diffusion	Poisson	L_2-projection
	Unknown(s)	u_1, u_2, u_3	ψ	p
Matrix	Size	$n_{\boldsymbol{u}} \times n_{\boldsymbol{u}}$	$n_p \times n_p$	$n_p \times n_p$
	Properties	Nonsymmetric	Sym. pos. semidefinite	Sym. pos. definite

$$\boldsymbol{A}_p = \int_\Omega \nabla\xi_i \cdot \nabla\xi_j \mathrm{d}\boldsymbol{x} \quad \text{with } i, j = 1, \ldots, n_p, \tag{18e}$$

$$\boldsymbol{P}_d = \int_\Omega \phi_i \frac{\partial \xi_j}{\partial x_d} \mathrm{d}\boldsymbol{x} \quad \text{with } i = 1, \ldots, n_{\boldsymbol{u}},\ j = 1, \ldots, n_p, \tag{18f}$$

$$\boldsymbol{B}_d = \int_\Omega \xi_i \frac{\partial \phi_j}{\partial x_d} \mathrm{d}\boldsymbol{x} \quad \text{with } i = 1, \ldots, n_p,\ j = 1, \ldots, n_{\boldsymbol{u}}. \tag{18g}$$

Note that for each velocity component one independent equation (15) is solved. Some properties of the linear equation systems (15)–(17) relevant to the selection of suitable iterative solution methods are summarised in Table 1.

3. Parallel iterative solvers and implementation

Next we introduce the solution of the linear systems of equations resulting from the finite element discretisation of the incremental pressure-correction method. The considered class of fluid problems have up to several billions unknowns and the target parallel architectures have up to hundred thousand processors. For such problems Krylov subspace methods with efficient preconditioners are the only suitable solution technique. In practice, the scalability of the overall finite element technique also depends on the efficiency of the data structures and algorithms for mesh decomposition and handling, and assembly of the systems matrices. In our finite element software openFTL we make extensive use of the C++ STL [10,32,53], METIS [33] and PETSc [4] libraries in order to achieve efficiency and scalability. Specifically, the use of PETSc enables us to perform a number of numerical experiments to identify the most suitable combinations of Krylov subspace methods and preconditioners.

3.1. Parallel preconditioned iterative solvers

We first provide a brief review of the parallel preconditioned Krylov subspace methods in order to fix terminology and notations. For details we refer to standard textbooks, e.g., [21,40,48]. Our discussion is restricted to iterative solvers and preconditioning techniques that are available in PETSc and which we use in our numerical computations.

The linear systems of equations introduced in Section 2.3 are of the generic form

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}. \tag{19}$$

The symmetry and the specific entries of the system matrix $\boldsymbol{A}$ and the right-hand side vector $\boldsymbol{f}$ depend on the considered problem. We use *GMRES* [49] or *BiCGstab* [58] for systems with a nonsymmetric matrix $\boldsymbol{A}$ and the conjugate gradient method [11] for systems with a symmetric matrix $\boldsymbol{A}$. Moreover, a preconditioning technique is necessary in order to improve the performance of the iterative solvers. To this end, we consider the (left-)preconditioned equation system

$$\boldsymbol{P}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{P}\boldsymbol{f}, \tag{20}$$

where $\boldsymbol{P}$ is a suitable preconditioning matrix that approximates $\boldsymbol{A}^{-1}$ (in some sense). The specific choices of preconditioners will be discussed in the following. For the subsequent discussions, it is also relevant to recall that for implementing preconditioned Krylov subspace methods only matrix-vector products with the system matrix $\boldsymbol{A}$ and the preconditioning matrix $\boldsymbol{P}$ are needed.

On a (distributed-memory) parallel computer the equation systems (19) and (20) are only available in a distributed format. The partitioning of both equation systems results from the partitioning of the computational domain Ω (and the corresponding triangulation) into n_d possibly overlapping subdomains Ω_i, with $i = 1, \ldots, n_d$. The overlap is a prescribed layer of elements between the subdomains. In our computations the number of subdomains n_d is equal to the number of available processors. The matrix-vector product with the distributed system matrix $\boldsymbol{A}$ is straightforward and can be assembled from subdomain contributions. The matrix-vector product with $\boldsymbol{P}$ depends on the specific form of the preconditioner.

In this work, we consider as parallel preconditioners the block Jacobi and the overlapping additive Schwarz methods available in PETSc, see, e.g., [46,48,51,57] for details. These one-level methods are not mathematically scalable for elliptic problems, such as the Poisson problem for the pressure increment [57]. It is necessary to use a two-level method in order to achieve mathematical scalability, i.e. convergence independent of the number of subdomains in a weak scaling test. Nevertheless, in our experience, the considered one-level methods perform reasonably well for the linear systems introduced in Section 2.3, with the most critical being the Poisson problem for the pressure increment. The state-of-the-art two-level methods include BDDC and FETI [12,22,23]. In these methods the challenge is the scalable solution of the coarse problem, which is an active area of research, see e.g. [3]. A possible solution is offered by the multilevel extension of BDDC [41,52]. Nevertheless, the multi-level methods should be avoided as long as a one-level method performs well, especially in a massively parallel environment.

In both the block Jacobi and the overlapping additive Schwarz methods, the preconditioner $\boldsymbol{P}$ is defined as the sum of local subdomain matrices $\boldsymbol{P}_i$, i.e.,

$$\boldsymbol{P} = \sum_{i=1}^{n_d} \boldsymbol{R}_i^{\mathsf{T}} \boldsymbol{P}_i \boldsymbol{R}_i, \tag{21}$$

where $\boldsymbol{R}_i^{\mathsf{T}}$ is a $1 - 0$ matrix which maps the local degrees of freedom in the interior of the subdomain Ω_i to the global degrees of freedom. The subdomain matrix $\boldsymbol{P}_i$ is an approximation to the inverse of the local system matrix $\boldsymbol{A}_i$ and is defined as

$$\boldsymbol{P}_i \approx \boldsymbol{A}_i^{-1} = \left(\boldsymbol{R}_i \boldsymbol{A} \boldsymbol{R}_i^{\mathsf{T}}\right)^{-1}. \tag{22}$$

Applying the inverse of the local system matrix $\boldsymbol{A}_i^{-1}$ represents solving a local Dirichlet problem because the matrix $\boldsymbol{R}_i$ does not include the degrees of freedom at subdomain boundaries. The multiplication of a vector $\boldsymbol{r}$ with the preconditioner $\boldsymbol{P}$ can now be established using

$$\boldsymbol{P}\boldsymbol{r} = \sum_{i=1}^{n_d} \boldsymbol{R}_i^{\mathsf{T}} \boldsymbol{P}_i \boldsymbol{R}_i \boldsymbol{r} = \sum_{i=1}^{n_d} \boldsymbol{R}_i^{\mathsf{T}} \boldsymbol{P}_i \boldsymbol{r}_i = \sum_{i=1}^{n_d} \boldsymbol{R}_i^{\mathsf{T}} \boldsymbol{w}_i, \tag{23}$$

with the local subdomain specific vectors $\boldsymbol{r}_i = \boldsymbol{R}_i \boldsymbol{r}$ and $\boldsymbol{w}_i = \boldsymbol{P}_i \boldsymbol{r}_i$. Bearing in mind $\boldsymbol{P}_i \approx \boldsymbol{A}_i^{-1}$, the local vector $\boldsymbol{w}_i$ is the (possibly approximate) solution of the local problem

$$\boldsymbol{A}_i \boldsymbol{w}_i = \boldsymbol{r}_i. \tag{24}$$

This problem is solved independently on each subdomain. The local solves are performed with a direct method, so that an LU (or LL^{T}) factorisation is performed during the set-up of the preconditioner, and only backsubstitutions are performed during the subsequent Krylov iterations. Performing a complete LU factorisation is called exact factorisation. Often one can save both time and memory by using an incomplete LU factorisation with a prescribed allowed fill-in, see, e.g., [48]. In this regard, ILU(0) is the basic approach which discards all entries of the factors which fall outside the sparsity pattern of the original matrix. While ILU(1) and ILU(2) improve the approximation of the inexact factorisation, they require new analysis of the sparsity structure of the factors and lead to longer times for both factorisation and back substitution.

A final remark concerns the solution of the pressure-corrector problem (16) which is a pure Neumann problem for the considered fluid flow problems with only Dirichlet boundary conditions. The corresponding symmetric matrix is singular with the nullspace spanned by constant vectors. In this case, the problem is solved only in the orthogonal complement of the nullspace. Namely, if we denote with $\boldsymbol{z} = (1, 1, \ldots, 1)^{\mathsf{T}}$ the basis vector of $\mathrm{null}(\boldsymbol{A})$, we can construct the orthogonal projection on its complement as $\boldsymbol{Q} = \boldsymbol{I} - \frac{1}{\boldsymbol{z}^{\mathsf{T}}\boldsymbol{z}} \boldsymbol{z}\boldsymbol{z}^{\mathsf{T}}$. If this matrix is applied after every multiplication with $\boldsymbol{A}$ and $\boldsymbol{P}$, the iterations are confined to the subspace orthogonal to $\mathrm{null}(\boldsymbol{A})$, and the following modified system is solved

$$\boldsymbol{Q}\boldsymbol{P}\boldsymbol{Q}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{P}\boldsymbol{Q}\boldsymbol{f}. \tag{25}$$

The preconditioned conjugate gradient method in this subspace is referred to as deflated PCG.

3.2. Implementation details

3.2.1. Domain partitioning

As elucidated in the preceding Section 3.1, the parallel solution of the discretised finite element equations relies on the partitioning of the domain into subdomains and assigning them to different processors. In general, the number of subdomains is chosen equal to the number of available processors. In the computations presented in this paper the discretised domain is a block-structured hexahedral mesh and is generated with the GMSH mesh generator [26]. The subdomains are obtained by partitioning the mesh with METIS. The size and shape of each subdomain is chosen such that interprocessor communication is minimised and each processor is equally utilised.

Algorithm 1 Partitioning of the mesh into subdomains

1. Create the dual graph of the computational mesh.
2. Create a non-overlapping partitioning of elements into subdomains by partitioning the dual graph (using METIS).
3. Derive a partitioning of nodes such that all nodes present in a single subdomain are local to the processor and randomly assign shared nodes to subdomains.
4. Assign each node a unique global ID by looping over all subdomains and all nodes in each subdomain.
5. Build overlapping clusters of elements as a union of all elements contributing to local nodes.

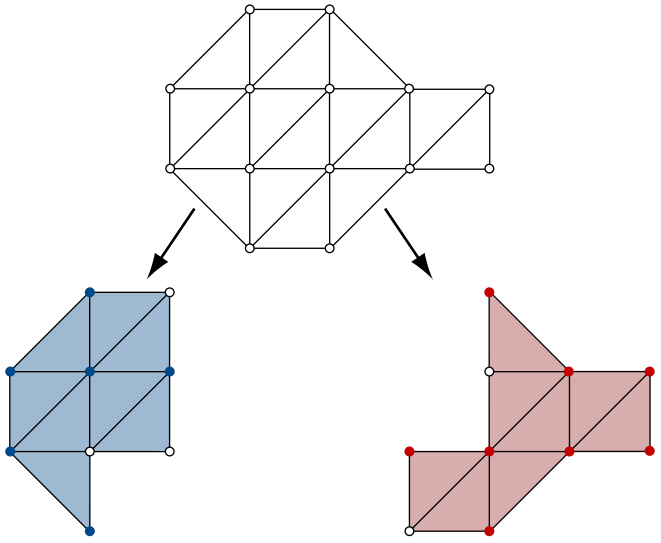


Fig. 2. Partitioning of a mesh into two nonoverlapping subdomain meshes. The nodes on the subdomain boundaries are uniquely assigned to one of the subdomains.

Our METIS-based mesh partitioning algorithm is shown in Algorithm 1, see also Fig. 2. As the first step we construct the dual graph of the finite element mesh. In the dual graph each finite element is a vertex and the edges of the graph represent two adjacent finite elements. Subsequently, we partition the dual graph with METIS. The partitioned graph gives a partitioning of the finite element mesh into nonoverlapping subdomains so that each finite element is uniquely assigned to a particular subdomain. Next the finite element nodes are assigned to subdomains. First, the nodes inside a subdomain are assigned to the respective subdomain. Subsequently, the nodes at subdomain boundaries are randomly assigned to the attached subdomains so that each has a similar number of nodes. In the last step we assign to each node a unique (global) ID by sequentially looping over the subdomains and consecutively numbering the nodes. Finally, for performance considerations during assembly it is necessary to form overlapping partitions so that the system matrices can be assembled without interprocessor communication.

3.2.2. Overlapping partitions for fast assembly

The partitioning of the finite elements and nodes into subdomains implies a partitioning of the system matrices and vectors into processors. Recall that each row of the global system matrix represents a node in the mesh, or more precisely one of its degrees of freedom. Moreover, the domain partitioning introduced in Section 3.2.1 ensures that the degrees of freedom associated to a domain lie all within a certain range. Hence, consecutive blocks of rows of the system matrix can be assigned to processors. In PETSc this is achieved with the `MPIAIJ` matrix format.

The rows of the global system matrix corresponding to finite element nodes at the subdomain boundaries receive contributions from several subdomains. During the assembly this requires frequent interprocessor communication. In practice, the associated overhead for assembly turns out to be excessively time consuming and presents a major performance bottleneck for large problems. In order to resolve this issue it is possible to eliminate any interprocessor communication during assembly. This can be achieved by providing each processor all the elements and nodes necessary for independently assembling its rows of the global system matrix. Therefore, we modify the partitioning introduced in Section 3.2.1 so that each subdomain stores in addition to its elements also elements of the neighbouring subdomains that contribute to local matrix rows, see Fig. 3. Evidently this leads to the notion of overlapping partitions. This can be accomplished using the partitioned dual graph provided by METIS and the

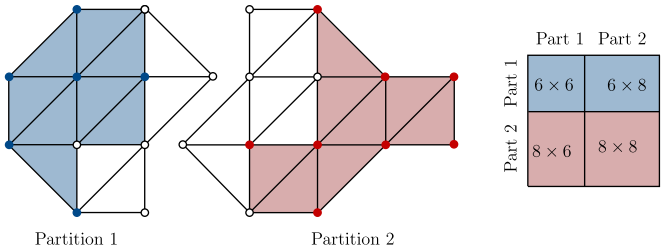


Fig. 3. Partitioning of a mesh with 14 nodes into two overlapping subdomain meshes and a sketch of the corresponding matrix. The first six rows of the system matrix are assigned to part one and the remaining eight to part two. The overlapping layer of elements ensures that the system matrix corresponding to each part can be assembled independently.

global transposed matrix of element connectivity, i.e. for each node, the list of elements surrounding it.

3.2.3. On-the-fly assembly and intermediary data container

In our finite element software openFTL, sparse matrices are directly assembled without determining the sparsity pattern of the matrix beforehand. The matrix is assembled on-the-fly while looping over the elements in the mesh and copying their contributions to an intermediary data container. In the implementation of the intermediary data container, we make extensive use of the C++ STL library, specifically of the `std::pair` object, `std::vector` container and `std::sort` algorithm. Similar to the coordinate sparse-matrix storage format we represent the sparse matrix as a vector of triplets (i,j,A_{ij}), where A_{ij} is an entry with the row index i and column index j. In C++ STL the type of each entry is chosen to be {`std::pair<std::pair<int,int>,double>`}. The key idea of the on-the-fly assembly is that the matrix entries are first one after the other appended to the end of the vector. The vector is subsequently sorted and triplets with the same row and column index (i and j) are combined to one triplet by summing the values A_{ij} of the matrix entries. Note that we could use instead of the `std::vector` container the sorted `std::multi_map` container. Although this would eliminate the sorting step, the insertion into an `std::multi_map` is substantially slower than into an `std::vector`. See also [42] for a discussion on the use of `std::vector` versus `std::multi_map`.

Algorithm 2 The in-place assembly of the matrix in the coordinate format

1. Loop over elements while appending each contribution to the global matrix as a new triplet (i,j,A_{ij}) at the end of the vector.
2. Sort the vector with the `std::sort` algorithm primarily according to the row index i and secondarily according to the column index j (with the standard `std::pair` comparison functor).
3. Loop over the vector, sum all entries with the same index pair (i, j) and store them at the end of the already assembled part of the vector.
4. Truncate the allocated memory to the actual size of the assembled vector.

A step-by-step description of the on-the-fly assembly algorithm is given in Algorithm 2. This algorithm is to be read in conjunction with Fig. 4. Step 1 of the algorithm is straightforward in the sense that the matrix entries are appended to the vector with a simple `push_back` operation. In step 2 we sort the triplets with the `std::sort` which must have $\mathcal{O}(n \log n)$ complexity in C++11, with n being the length of the vector [32,53]. Subsequently, in step 3 the entries are combined, i.e. assembled, in linear time, and they are stored at the end of the assembled part of the same vector, hence the assembly is performed *in-place*.

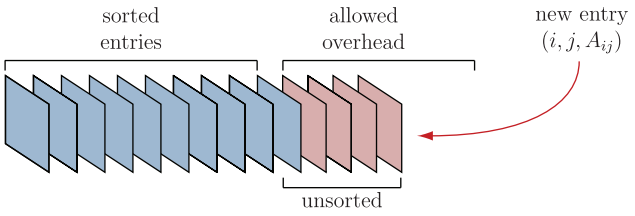


Fig. 4. Schematic illustration of an insertion into the sparse matrix format. A new entry is appended at the end of the vector until the limit of allowed overhead is reached, and a re-sorting is performed.

Due to memory restrictions, it is usually not possible to process all matrix contributions in one go. The vector is sorted and assembled in fixed prescribed intervals. In this way, we control both the memory overhead and number of sortings. The frequency of the intervals depends on a user prescribed allowed memory overhead. In Fig. 5a numerical timing study for the assembly of the system matrix of a 2D and 3D elasticity problem are reported. The study was performed on a single core of the Intel Core i7 CPU with frequency 2.7 GHz. We can observe that the time spent by sorting grows very slowly with increasing allowed memory overhead. In contrast, the number of sortings decreases linearly with increasing allowed memory overhead. Hence, the total time is clearly dictated by the number of sortings used during the assembly. Therefore, for achieving good performance the allowed memory overhead should be chosen as large as possible. Furthermore, it can be seen in Fig. 5 that the time for insertion of entries is mostly lower than the total sorting times and it is independent of the allowed overhead.

After all the finite element contributions are processed with Algorithm 2, we obtain a vector of the assembled matrix in the coordinate format, sorted primarily by rows and secondarily by columns. This allows us to quickly determine the structure of the PETSc matrix on each processor and to perform an exact pre-allocation of memory. Subsequently, all the vector entries are copied into the PETSc `MPIAIJ` matrix (using `MatSetValue` function with the `INSERT_VALUES` flag). Moreover, due to the overlapping partitions discussed in Section 3.2.2, there is no need to transfer stiffness matrix data between the processors. Therefore, the assembly of the PETSc matrix (by `MatAssemblyBegin` and `MatAssemblyEnd` functions) takes negligible time.

4. Numerical performance studies and results

In this section we first validate and verify our computational framework by analysing the flow around a low-aspect-ratio inclined flat plate. Subsequently, we consider the flow around a rotating insect wing to compare the performance of various Krylov solvers and their parallel scalability in combination with block Jacobi and additive Schwarz preconditioners. At the same time, we elucidate and compare the flow structures and aerodynamic forces for translating and rotating wings, especially the formation and persistence of leading-edge vortices. We generate all our finite element meshes using the GMSH [26] as block-structured boundary-conforming meshes. However, during the solution process the meshes are considered as unstructured. As finite elements we use the Taylor-Hood $Q2$–$Q1$ elements. In order to resolve the flow field without needing convection stabilisation sufficiently fine grids are used.

The considered flows and their numerical solution is strongly dependent on the Reynolds number

$$\mathrm{Re} = \frac{u_\infty L}{\nu}, \tag{26}$$

where u_∞ is the characteristic fluid speed (e.g., free-stream velocity), L is the characteristic length of the wing and ν is the kinematic viscosity. In our computations the Reynolds number is altered by modifying the kinematic viscosity.

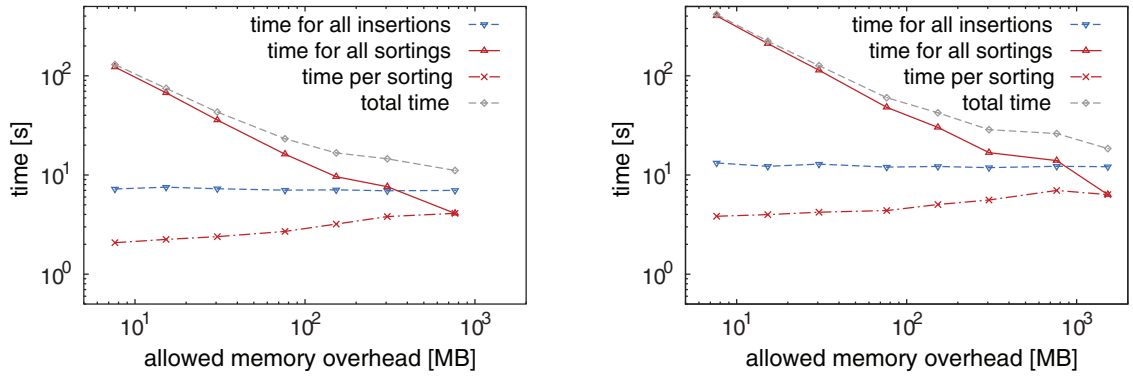


Fig. 5. The runtime of the on-the-fly assembly in dependence of the prescribed allowed memory overhead. (a) Bi-quadratic quadrilateral element system matrix of size 722×10^3 with 23.1×10^6 nonzeros (~445 MB) and 29.2×10^6 insertions. (b) Tri-quadratic hexahedral element system matrix of size 207×10^3 with 37.5×10^6 nonzeros (~801 MB) and 52.5×10^6 insertions.

As the result of our computations we provide plots of the flow fields in the form of isosurfaces of the Q-value. The Q-value is the second invariant of the velocity gradient tensor $\nabla \boldsymbol{u}$, and it is widely used to visualise vortices [31]. In incompressible flows the second invariant can also be expressed as

$$Q = \frac{1}{2}(\|\boldsymbol{\Omega}\|^2 - \|\boldsymbol{S}\|^2), \tag{27}$$

with the antisymmetric vorticity tensor $\boldsymbol{\Omega} = \frac{1}{2}(\nabla \boldsymbol{u} - (\nabla \boldsymbol{u})^{\mathsf{T}})$, the symmetric strain-rate tensor $\boldsymbol{S} = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\mathsf{T}})$ and $\|\cdot\|$ denoting the Frobenius norm. Informally, in flow regions with $Q > 0$ the vorticity is larger than the strain rate, which indicates the presence of vortices (i.e., regions with swirling-type motion). More in-depth discussion of applicability of the Q-criterion can be found in, e.g., [37–39].

We also report the aerodynamic forces acting on the wing in the form of non-dimensionalised force coefficients

$$C = \frac{2F}{\rho u_\infty^2 S}, \tag{28}$$

where F is a component of the force resultant vector $\boldsymbol{F}$, ρ is the density of the flow and S is the planform of the wing. In all our computations the flow density is $\rho = 1$. The coefficient C represents usually the drag C_D or lift C_L depending on the component of the considered force vector $\boldsymbol{F}$. The force resultant $\boldsymbol{F}$ is the integral of the boundary tractions, i.e.,

$$\boldsymbol{F} = \int_\Gamma \boldsymbol{\sigma}(\boldsymbol{u}, p) \cdot \boldsymbol{n} \, \mathrm{d}\Gamma, \tag{29}$$

which is equal to the sum of the reaction forces of all the finite element nodes located on the wing.

All performance and scalability studies are performed on the Cray XE6 supercomputer *HECToR*[1] (Phase 3). This computer is composed of 2816 XE6 compute nodes, each of which has 32GB memory. A compute node contains two AMD 2.3 GHz 16-core processors giving a total of 90,112 cores, with 65,536 cores being the maximum handled by the job scheduler. Cray Gemini chips are used for communication through a high-bandwidth network.

4.1. Flow around an inclined flat plate

4.1.1. Problem definition and discretisation

The inclined flat plate represents some of the flow features typical for animal locomotion in air and water and has been experimentally and numerically studied by a number of authors, including Taira and Colonius [54] and Wang and Zhang [59]. As in these two references we consider a rectangular thin plate with the chord-length $c = 1$ and the span $2c$ resulting in the aspect ratio $\mathcal{AR} = 2$, see

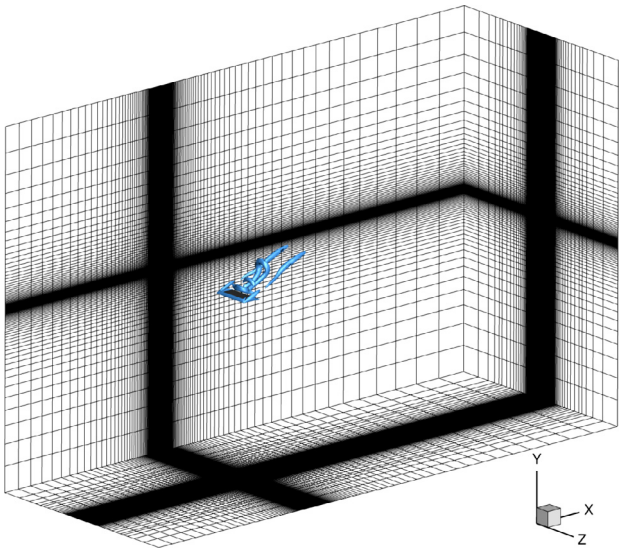


Fig. 6. Flow around an inclined plate. Computational mesh with the wing and its wake at Re = 300 and an angle of attack of 30°. All the mesh lines and the flat plate are aligned with the coordinate system.

Fig. 6. The thickness of the plate is $0.01875c$. The bounding box of the computational flow domain is a rectangular box with dimensions $[-10, 21] \times [-10, 10.01875] \times [-5, 7]$. From this rectangular box, a smaller axis-aligned cuboid representing the thin plate is subtracted in order to obtain the computational fluid domain. The position of the flat plate in the fluid domain is $[0, 1] \times [0, 0.01875] \times [0, 2]$.

The outer rectangular box is discretised by $210 \times 110 \times 120$ elements along its length, height and width, respectively. Each of the cuboidal finite elements in the domain are axis-aligned with the domain boundaries and become progressively smaller close to the flat plate. The inner fluid boundary representing the wing is discretised by $100 \times 10 \times 80$ elements along its chord, thickness and span directions, respectively. This discretisation leads to approximately 2.5 million elements and 20.6 million nodes.

The boundary condition at the plate surface is set to no-slip, $\boldsymbol{u}(t, \boldsymbol{x}) = \boldsymbol{0}$, and at the outer surface of the box to free-stream

[1] http://www.hector.ac.uk

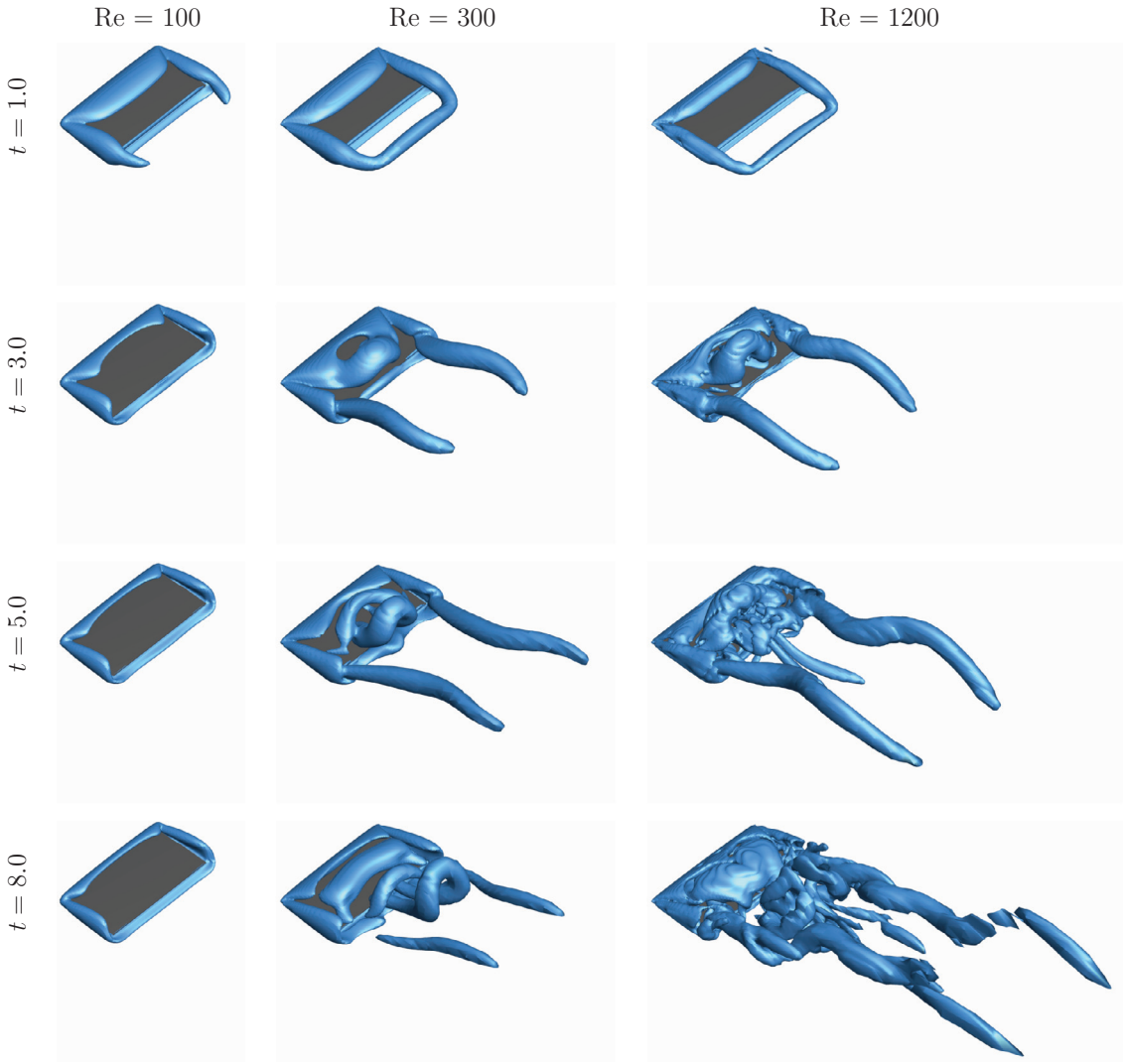


Fig. 7. Flow around an inclined plate. Isosurfaces of $Q = 2$ at different time instants, $\alpha = 30^\circ$.

velocity $\boldsymbol{u}(t, \boldsymbol{x}) = (u_\infty \cos\alpha, u_\infty \sin\alpha, 0)^\mathsf{T}$. The free-stream velocity magnitude is $u_\infty = 1$ and the angle of attack is α. Starting impulsively at $t = 0$, the time dependent problem is solved for $t \in [0, 18]$. The time-step size is chosen with $\Delta t = 0.02$ resulting in 901 time steps in a typical run. The Courant number based on the free stream velocity is $u_\infty \Delta t / h_x \approx 1.64$, where h_x is the smallest element size along the chord. Moreover, the change of the angle of attack is achieved through changing the direction of the flow rather than changing the mesh in the computations.

4.1.2. Flow characteristics and forces

Before proceeding to validation and verification, we present the results of our computations for (chord-length based) Re = 100, 300 and 1200 and the angle of attack of 30°. Our aim is to illustrate the Reynolds number dependence of the flow characteristics and forces, which in turn determine the spatial and temporal resolution needs of the discretisation. In Fig. 7 the isosurfaces of the Q-value are shown. At time $t = 1.0$, in all plots a leading-edge and two tip vortices can be identified. Furthermore, for Re = 300 and 1200 also a convected trailing-edge vortex (starting vortex) is visible, which is for Re = 100 not strong enough to be shown by the $Q = 2$ isosurface. For later times, as a general trend the complexity of the observed vortex structures becomes more pronounced with increasing Reynolds number due to the decrease in the diffusivity of the flow. At time $t = 3.0$ for Re = 300 and 1200, there are two columnar tip vortices and an already pinching off leading-edge vortex is visible. This process continues with consecutive formation and shedding of leading-edge vortices as visible for time $t = 5.0$ and $t = 8.0$ for Re = 300. In order to be conclusive about the vortex structures observed for Re = 1200 at $t = 5.0$ and $t = 8.0$ computations with finer meshes are needed.

It is instructive to consider the Q-value plots in Fig. 7 in conjunction with the history of drag and lift coefficients in Fig. 8. As an artefact of the impulsive start, the coefficients have a large peak in the immediate vicinity of $t = 0$ which is not physically relevant. The subsequent sustained increase in the lift coefficient occurs while the leading-edge vortex is formed and the trailing-edge vortex is advected. This is due to the low pressure zone created by the leading-edge vortex above the wing. The obtained maximum lift coefficient becomes larger with increasing Reynolds number. The difference in the drag coefficients corresponding to the maximum lift coefficients is far less pronounced. As a result the aerodynamic efficiency (lift coefficient divided by the drag coefficient) is proportional to the Reynolds number. After the leading-edge vortex detaches for Re = 100 the lift and drag coefficients reach a steady state. In contrast, for Re = 300 and Re = 1200 both coefficients continue oscillating in line with shedding of vortices. For a more in-depth discussion of the relevant flow characteristics we refer to [54].

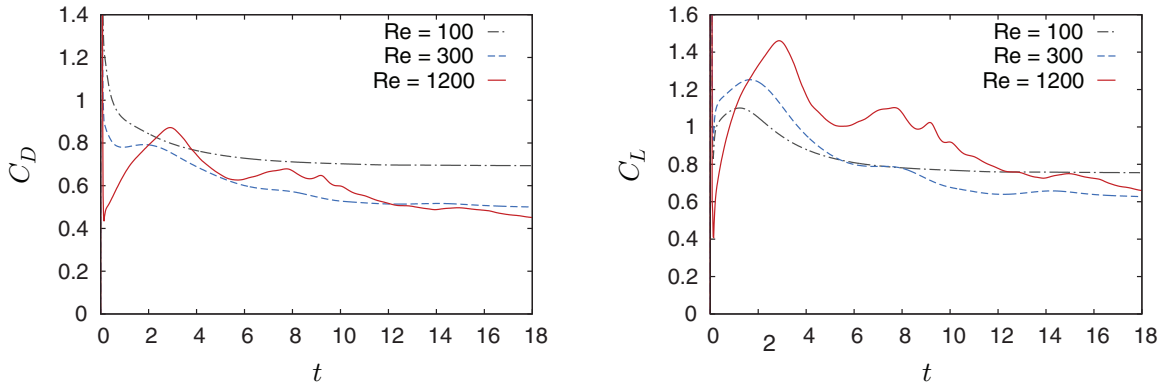


Fig. 8. Flow around an inclined flat plate. Comparison of drag (left) and lift (right) coefficients for an angle of attack of 30°, Re = 100, 300, and 1200.

4.1.3. Validation and verification

We compare our results with the experimental and computational results of Taira and Colonius [54]. They report for Re = 300 only computational and for Re = 100 computational and experimental drag and lift coefficients for various angles of attack. The same set-up is also computationally investigated by Wang and Zhang [59] for Re = 100. In [54] and [59] the discretisation is based on the immersed boundary method and the plate is assumed to have zero thickness. Since for Re = 100 the flow quickly reaches a steady state for all considered angles of attack, it is meaningful to compare the steady state coefficients. In Fig. 9 our steady state drag and lift coefficients at time $t = 13$ and angles of attack $\alpha = 10°$, 30°, 50°, and 70° are compared with the ones presented in [54] and [59].

One can observe that for $\alpha = 10°$ and 30° our values are in excellent agreement with the experimental data and other computational results. The difference in C_D can be attributed to the thickness of the plate, which is about a half of that used in the experiment, while it is ignored in other computations. The agreement is slightly worse for $\alpha = 50$ and 70, where the computation seems to be affected by the interaction of the wake with coarser mesh above the plate. As mentioned, the change of angle of attack is achieved through changing the direction of the flow. At last, in Table 2 the maximal lift coefficients and the times at which they are attained are given for Re = 300 and Re = 1200. Our results are in good agreement with the computational results in [54] for Re = 300.

Table 2
Flow around an inclined flat plate. Maximum lift coefficients and corresponding times for two different Reynolds numbers and angles of attack. For Re = 300 results are compared to [54].

α (deg)	Reference	Re = 300		Re = 1200	
		Max. C_L	t	Max. C_L	t
10	[54]	0.46	1.63	–	–
	Our result	0.43	1.4	0.48	2.68
30	[54]	1.29	1.68	–	–
	Our result	1.25	1.66	1.46	2.88

4.2. Flow around a rotating insect wing

4.2.1. Problem definition and discretisation

It is well-known that a rotating wing generates a leading-edge vortex which remains attached to the wing. The attendant sustained lift forces are believed to be crucial for the success and efficiency of flapping flight in nature. In order to study the vortex formation and forces generated by rotating wings, we consider a fruit fly (Drosophila melanogaster) wing, as experimentally studied in [43], at an angle of attack of 40° rotating around a vertical axis near its root, see Fig. 10. The length of the wing is R; its aspect ratio is $\mathcal{AR} = R^2/S = 3.1$, where S is the planform area; and its thickness is $0.01R$.

As shown in Fig. 11, the computational fluid domain consists of a cylinder with radius $2.9R$ and an axial hole with radius $0.016R$. The height of the cylinder is $4.7R$. The block structured finite element mesh is generated with GMSH [26] with the block topology depicted in Fig. 12, although it is handled as unstructured in the solver. The mesh is refined towards the wing, resulting in $30 \times 5 \times 30$ elements along the chord, thickness, and span of the wing, respectively, see Fig. 11. The whole fluid domain contains $266 \times 102 \times 77$ Taylor-Hood elements along circumference, height, and radius of the cylinder, resulting in approximately 2.1×10^6 elements and 16.8×10^6 nodes.

After a smooth acceleration to the final angular velocity during the time $t \in [0, 1]$, the wing rotates with a constant angular velocity until

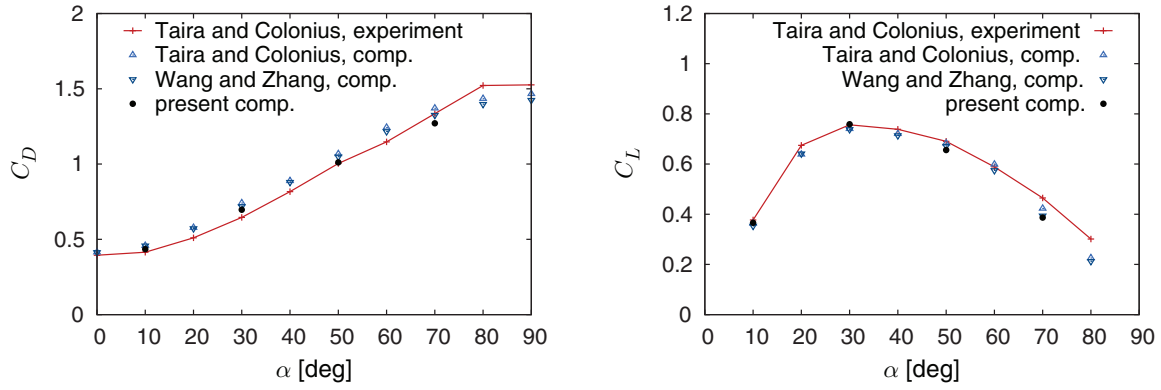
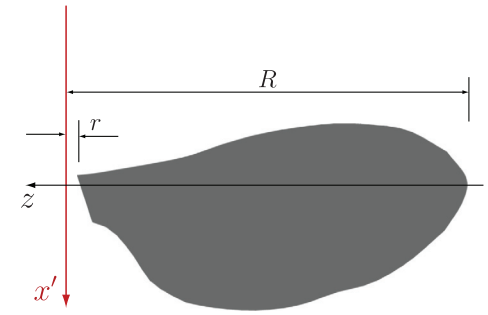
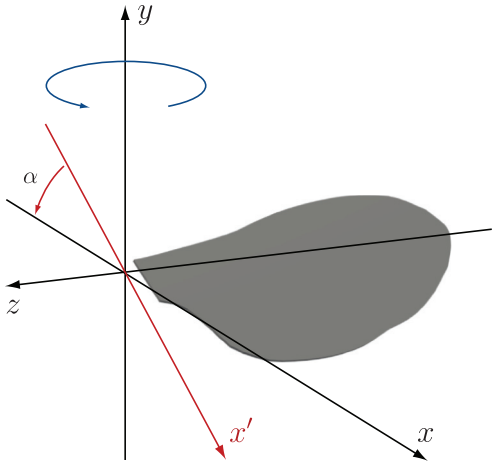


Fig. 9. Comparison of steady drag C_D (left) and lift C_L (right) coefficients at Re = 100 for different angles of attack α with experimental and computational results by Taira and Colonius [54], and computational results by Wang and Zhang [59].



(a) Planform



(b) Isometric view

Fig. 10. Flow around a rotating insect wing. Geometry and prescribed motion of the Drosophila wing (with $r = 0.0625\,R$ and $\alpha = 40^\circ$).

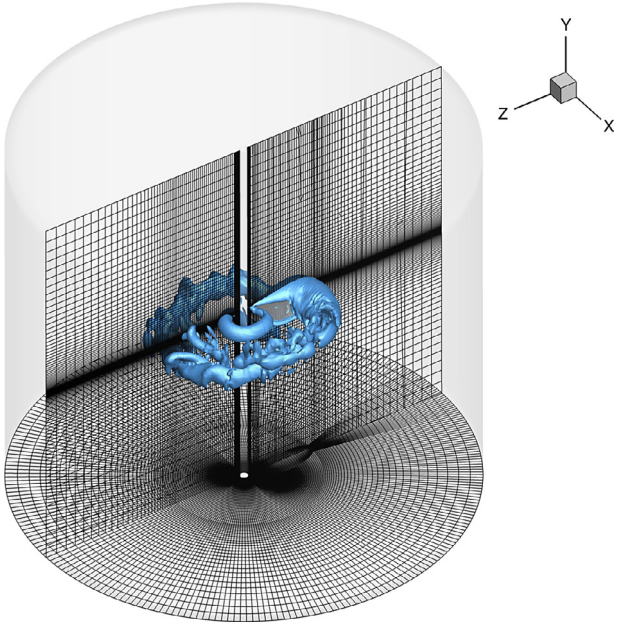


Fig. 11. Flow around a rotating insect wing. Computational mesh with wing and its wake at Re = 526 for angle of attack of 40° with approximately 2.1×10^6 finite elements and 16.8×10^6 nodes.

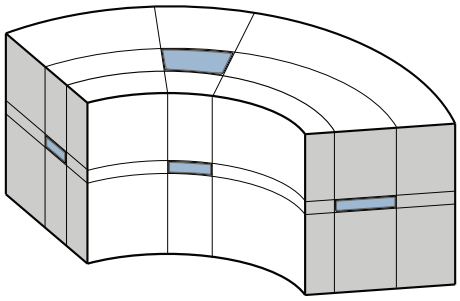


Fig. 12. Flow around a rotating insect wing. Topology of the 27 blocks of the structured mesh used for meshing the computational domain. The Drosophila wing geometry is mapped to the inner block (at the intersection of the dark shaded faces). The mesh is connected into a ring at the lightly shaded faces.

$t = 7.7$. At $t = 7.7$ one full revolution is completed. With the uniform time-step size $\Delta t = 0.002$ the whole computation requires 3850 time steps. The velocity of the tip of the wing is prescribed with $|\boldsymbol{u}_{tip}| = 1$, resulting in $|\boldsymbol{u}_{tip}|\Delta t/h_{tip} \approx 0.1$, with h_{tip} denoting the smallest element size near the tip of the wing. The angular velocity is $\omega = |\boldsymbol{u}_{tip}|/R$. An important length is the radius of gyration $R_g = \sqrt{S_2/S}$, where S_2 is the second moment of the wing with respect to the axis of rotation [16,29]. For the considered wing geometry the radius of gyration is $R_g = 0.5255R$.

The reported Reynolds numbers are based on the velocity at the radius of gyration $\boldsymbol{u}_{rg}$, i.e.,

$$\mathrm{Re} = \frac{R\,|\boldsymbol{u}_{rg}|}{\mathcal{R}\,\nu} = \frac{R\,R_g\,\omega}{\mathcal{R}\,\nu}, \tag{30}$$

which is altered by choosing a suitable kinematic viscosity ν. In order to study the flow in laminar and transient regimes we consider four different Reynolds numbers $\mathrm{Re} \in \{105, 263, 526, 1051\}$[2].

4.2.2. Performance comparison of the iterative solvers

To begin with, we aim to identify the most efficient combination of Krylov subspace methods and preconditioners suitable for solving the systems of equations resulting from the discretisation of the Navier–Stokes equations. As summarised in Table 1, in the incremental pressure-correction approach five linear systems of equations are solved at every time step. Namely three equations for the update of the velocity components, one equation for the update of the pressure increment and one equation for the pressure update. We only consider iterative solvers and preconditioners that are available in PETSc (version 3.2).

For all numerical studies in this section we use the block-structured mesh described in the foregoing section and shown in Fig. 11 (with 2.1×10^6 elements and 16.8×10^6 nodes), unless stated otherwise. The number of subdomains and utilised processors is chosen to be 2048 and the Reynolds number for the flow is Re = 1051. The reported iteration counts and times are averaged over the initial 400 time steps, 200 of which are in the acceleration stage.

First, we consider the velocity update which involves three independent discrete convection–diffusion type equations (15). Each of the three equations has the same system matrix. Since these are non-symmetric we use the GMRES [49] and BiCGstab [58] methods. In Table 3 the performance of both methods with no preconditioner and with block Jacobi preconditioner are compared. For solving the sub-problems of the diagonal blocks in the block Jacobi preconditioner (see Section 3.1), we use ILU preconditioner with no fill-in ILU(0),

[2] The selected values of Reynolds numbers correspond to the tip velocity based Reynolds numbers $\mathrm{Re}_{tip} = \frac{R\,|\boldsymbol{u}_{tip}|}{\mathcal{R}\,\nu} \in \{200, 500, 1000, 2000\}$.

Table 3
Flow around a rotating insect wing. Iteration counts and timings for velocity update using two different Krylov solvers and a block Jacobi preconditioner with different subdomain solvers. All computations are using the mesh shown in Fig. 11 and 2048 subdomains. For each case the reported numbers are average values over the initial 400 time steps (of which 200 are in acceleration phase). The minimal time is emphasised in boldface.

Method	Prec.	Local sol.	Number of iter. Min–max (*avg.*)			Avg. sol. time (s)		
			u_1	u_2	u_3	u_1	u_2	u_3
GMRES	No	–	1–313(*259.0*)	0–10k(*9.8k*)	2–1.1k(*0.9k*)	1.49	55.67	5.26
	bl. Jacobi	ILU(0)	7–14(*12.6*)	0–13(*11.9*)	7–15(*13.7*)	**0.20**	**0.11**	**0.12**
		ILU(1)	7–14(*12.6*)	0–13(*11.7*)	7–14(*13.4*)	3.10	0.32	0.36
		ILU(2)	7–14(*12.0*)	0–13(*11.0*)	7–14(*12.7*)	29.67	0.72	0.82
		LU	7–14(*12.6*)	0–13(*11.7*)	7–14(*13.4*)	1.01	0.27	0.30
BiCGstab	No	–	1–183(*151.4*)	0–3.5k(*2.5k*)	1–440(*342.5*)	1.34	21.49	2.99
	bl. Jacobi	ILU(0)	4–9(*7.5*)	0–7(*6.9*)	4–9(*8.0*)	0.22	0.12	0.14
		ILU(1)	4–8(*7.3*)	0–7(*6.9*)	4–9(*8.6*)	3.12	0.37	0.45
		ILU(2)	4–8(*7.2*)	0–7(*6.8*)	4–9(*8.1*)	29.86	0.87	1.03
		LU	4–8(*7.6*)	0–7(*6.9*)	4–9(*8.6*)	1.06	0.31	0.38

Table 4
Flow around a rotating insect wing. Iteration counts and timings for velocity update using GMRES and different preconditioners and subdomain solvers. The considered preconditioners and their abbreviations are: block Jacobi (bl. Jacobi) and additive Schwarz method with algebraic overlap 1 (ASM-1) and 2 (ASM-2). All computations are using the mesh shown in Fig. 11 and 2048 subdomains. For each case the reported numbers are average values over the initial 400 time steps (of which 200 are in acceleration phase). The minimal time is emphasised in boldface.

Method	Prec.	Local sol.	Num. iter. Min–max (*avg.*)			Avg. sol. time (s)		
			u_1	u_2	u_3	u_1	u_2	u_3
GMRES	bl. Jacobi	ILU(0)	7–14(*12.6*)	0–13(*11.9*)	7–15(*13.7*)	**0.20**	0.11	0.12
GMRES	ASM – 1	ILU(0)	5–8(*7.7*)	0–8(*7.4*)	5–9(*8.6*)	1.80	**0.10**	**0.11**
		ILU(1)	3–4(*3.7*)	0–4(*3.7*)	3–4(*3.9*)	5.54	0.19	0.20
		ILU(2)	3–3(*3.0*)	0–3(*2.9*)	3–3(*3.0*)	57.01	0.40	0.41
		LU	3–3(*3.0*)	0–4(*3.1*)	3–4(*3.5*)	13.05	0.16	0.18
GMRES	ASM – 2	ILU(0)	5–8(*7.7*)	0–8(*7.4*)	5–9(*8.6*)	1.15	0.15	0.17
		ILU(1)	3–4(*3.6*)	0–4(*3.6*)	3–4(*3.6*)	8.75	0.31	0.31
		ILU(2)	2–2(*2.0*)	0–2(*2.0*)	2–2(*2.0*)	74.76	0.48	0.49
		LU	2–3(*2.5*)	0–3(*2.1*)	2–3(*2.0*)	5.08	0.25	0.25

with prescribed additional fill-in ILU(1) and ILU(2), and a complete sparse LU factorisation. In the last case, the MUMPS direct solver [1] is used. In all computations, the preconditioner is set up in every time step once and used for all three velocity components. The time for the set-up is included in the computational time for the velocity component u_1.

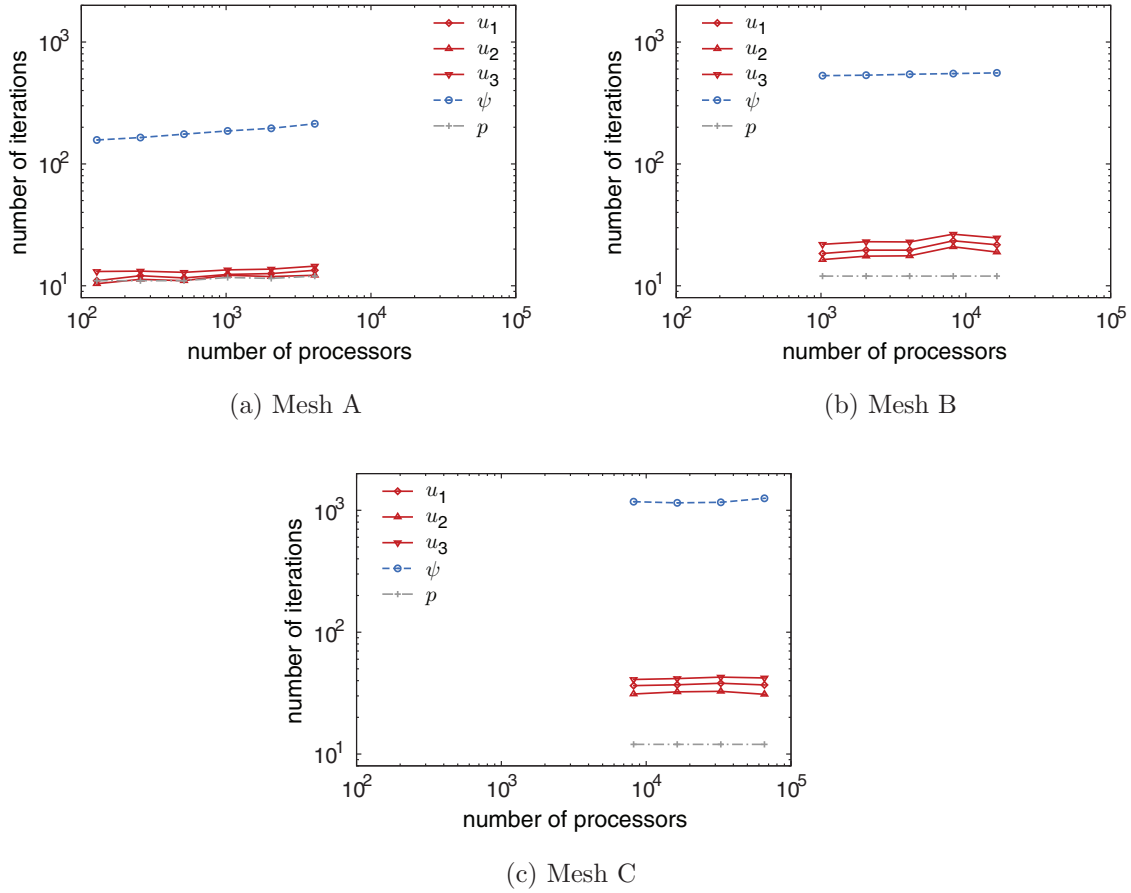
In Table 3 we can see that GMRES and BiCGstab perform similarly well, the former being marginally faster. In terms of number of iterations, we recall that two actions of the system matrix as well as of the preconditioner are performed within each iteration of BiCGstab. Hence, the number of iterations for BiCGstab should be about one half of those by GMRES for comparable accuracy. Moreover, it is evident from Table 3 that a preconditioner is crucial. It is interesting that the convergence of the preconditioned methods does not significantly improve with better approximation of the incomplete factors, moving from ILU(0) to ILU(2), while it increases the cost of the solve for the first velocity component drastically. Surprisingly the full LU factorisation of the diagonal blocks with MUMPS is faster than ILU(1) and ILU(2), but still more expensive than ILU(0). As a conclusion, the ILU(0) appears to be the best local solver in combination with the block Jacobi preconditioner for the velocity components. Our most recent studies indicate that even a simple diagonal Jacobi preconditioner appears to be competitive in terms of computing time.

Continuing with the velocity update, we also investigate the algebraic versions of the additive Schwarz method (ASM) with one or two elements of overlap. In the `PCASM` preconditioner of PETSc the clusters of overlapping elements are reconstructed from the graph of the local matrices without overlaps. The ASM simplifies to the block Jacobi preconditioner when no overlap is used. The corresponding results are presented in Table 4. It can be observed that an overlap is capable of improving the preconditioner (cf. results for block Jacobi preconditioner in Table 4) by significantly reducing the number of iterations. However, in terms of computational time, the time spent on the set-up of the preconditioner is too high to be amortised by the slightly faster solution times of the ASM method with one element overlap.

Table 5
Flow around a rotating insect wing. Iteration counts and timings for pressure increment ψ update using deflated PCG with block Jacobi preconditioner and different subdomain solvers. All computations are using the mesh shown in Fig. 11 and 2048 subdomains. For each case the reported numbers are average values over the initial 400 time steps (of which 200 are in acceleration phase). The minimal time is emphasised in boldface.

Method	Prec.	Local sol.	Num. iter. Min–max (*avg.*)	Avg. sol. time (s)
Defl. PCG	bl. Jacobi	ILU(0)	144–249(*195.9*)	**0.13**
		ILU(1)	127–216(*172.9*)	0.14
		ILU(2)	151–207(*188.6*)	0.18
		LU	122–194(*163.4*)	0.31

Next, we consider the update of the incremental pressure field ψ by solving the Poisson problem (16) with pure Neumann boundary conditions, see Table 5 for results. Our study is restricted to the deflated preconditioned conjugate gradient (PCG) method using the block Jacobi preconditioner and different local solvers. As Table 5 suggests, the solution times are comparable to those necessary for solving for one component of velocity. However, the equation system for velocity component is around eight times larger than the one for the



(a) Mesh A

(b) Mesh B

(c) Mesh C

Fig. 13. Flow around a rotating insect wing. Iteration counts for meshes A, B and C (see Table 8) and different number of subdomains. Problems for velocity components solved with GMRES, problems for pressure increment and for pressure with CG, all with block Jacobi preconditioner and ILU(0) as the local solver. See Table A.10 in the Appendix for numerical values.

Table 6
Flow around a rotating insect wing. Iteration counts and timings for the L_2-projection of the pressure p using PCG with block Jacobi preconditioner and different subdomain solvers. All computations are using the mesh shown in Fig. 11 and 2048 subdomains. For each case the reported numbers are average values over the initial 400 time steps (of which 200 are in acceleration phase). The minimal time is emphasised in boldface.

Method	Prec.	Local sol.	Num. iter. Min–max (*avg.*)	Avg. sol. time (s)
PCG	bl. Jacobi	ILU(0)	11–12(*11.5*)	**0.009**
		ILU(1)	11–12(*11.0*)	0.010
		ILU(2)	11–12(*11.1*)	0.014
		LU	11–12(*11.1*)	0.022

Table 7
Flow around a rotating insect wing. The identified most efficient combination of Krylov solver, preconditioner and subdomain solver.

Problem	Velocity components u_1, u_2, u_3	Pressure increment ψ	Pressure p
Krylov method	GMRES	Deflated PCG	PCG
Preconditioner	block Jacobi	block Jacobi	block Jacobi
Subdomain solver	ILU(0)	ILU(0)	ILU(0)

incremental pressure update, cf. Section 2.3. Overall, similar to velocity problems, the lowered number of iterations by better (or even exact) LU factorisation is, for this case, not sufficient to save computational time. The simplest ILU(0) factorisation of the local problems remains the most efficient method.

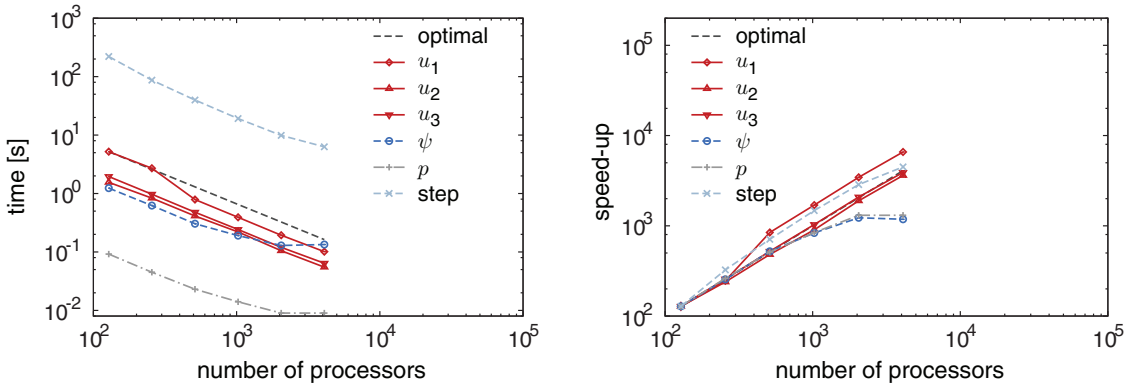
For the sake of completeness, we also perform a similar study for the L_2-projection of pressure p, cf. (17), see Table 6 for results. It should be stressed that this problem is by an order of magnitude faster to solve than the velocity update (15) and the auxiliary pressure update (16). Consequently, savings for this problem do not lead to any significant gain in the overall algorithm. The fast convergence of all considered methods is reported in Table 6.

4.2.3. Parallel scalability of the iterative solvers

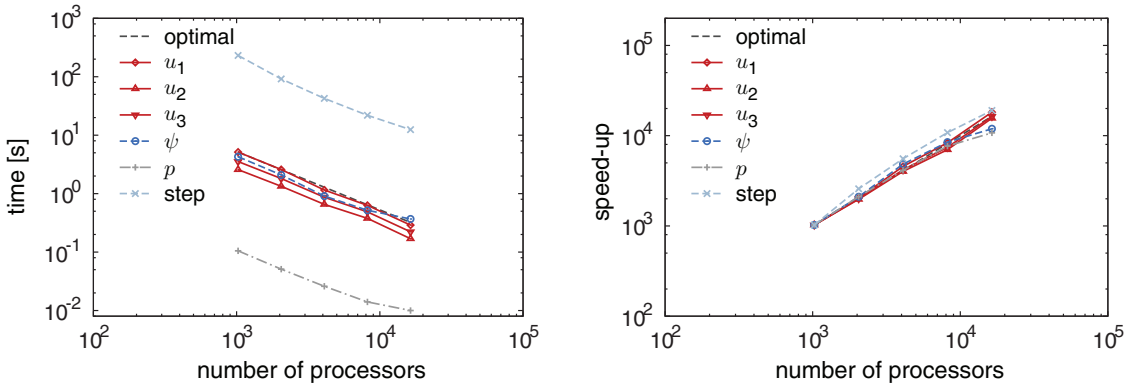
We now investigate the parallel scalability of the preconditioned iterative solvers identified as most efficient in the foregoing section, see Table 7. The velocity components u_1, u_2 and u_3 are updated with GMRES, the pressure increment ψ is updated with deflated PCG, and PCG is used for L_2-projection of the pressure p. In each case the block Jacobi preconditioner with ILU(0) local solver is used. The Reynolds number of the flow is $\mathrm{Re} = 1051$ for all computations. We study both the weak and strong scalability of the iterative solvers. During the weak scaling runs the problem size grows with the number of processors while keeping the load on each processor approximately constant. In contrast, during the strong scaling runs the problem size is fixed and only the number of processors is increased. An algorithm is optimally scalable when the solution time is constant during a weak scaling test and when the solution time is halved each time the number of processors is doubled during a strong scaling test.

For the weak scaling runs, we use two additional meshes generated by octasecting the hexahedral elements of the computational mesh described in Section 4.2.2, see also Fig. 11. During each refinement the problem size increases approximately by factor eight. The sizes of the considered three meshes are given in Table 8.

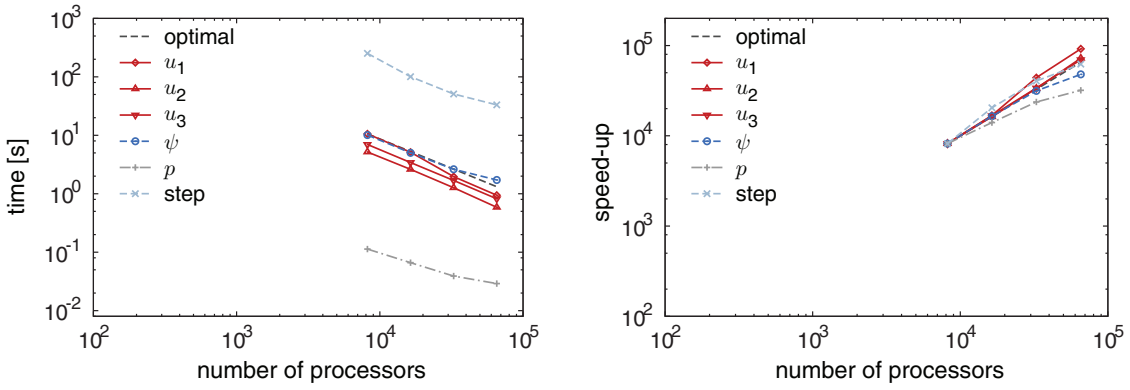
Figs. 13 and 14 show the average number of iterations and average solution times per time step in dependence of number of utilised



(a) Mesh A



(b) Mesh B



(c) Mesh C

Fig. 14. Flow around a rotating insect wing. Timings (left) and speed-ups (right) for meshes A, B and C (see Table 8) and different number of subdomains. Problems for velocity components solved with GMRES, problems for pressure increment and for pressure with CG, all with block Jacobi preconditioner and ILU(0) as the local solver. See Table A.11 in the Appendix for numerical values. The line 'step' presents the average total time for a time step.

processors, respectively. The reported numbers are averaged over 400 time steps, 200 of which are in the initial acceleration phase. In addition, in Fig. 14 we also report the parallel speed-up

$$s_{n_p} = \frac{n_{p_{\text{ref}}}\, t_{\text{ref}}}{t_{n_p}}, \tag{31}$$

where $n_{p_{\text{ref}}}$ is the lowest number of utilised processors, t_{ref} is the corresponding time, and t_{n_p} is the time on n_p processors.

In Fig. 13 one can see that for each given mesh the number of iterations is almost independent of the number of processors, only for the pressure increment ψ the number of iterations increases slightly with increasing processor numbers. Moreover, the number of iterations for the pressure increment are significantly higher than the ones for the other problems. Therefore, in Fig. 14 the pressure increment update requires on average a comparable time like the update of the velocity components, despite having considerably less degrees of freedom. The worsening strong scalability of the pressure

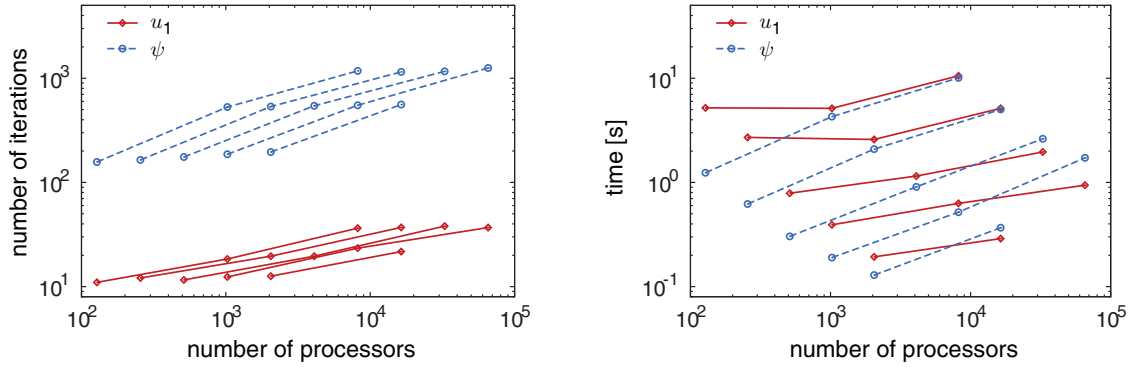


Fig. 15. Flow around a rotating insect wing. Weak scaling plots. Iteration counts (left) and timings (right) for meshes A, B and C (see Table 8) and different numbers of subdomains. Lines from the lower right to the upper left corner correspond to approximately 1000, 2000, 4000, 8000 and 16,000 finite elements per subdomain. Problems for velocity component u_1 solved with GMRES, problems for pressure increment with CG, all with block Jacobi preconditioner and as the ILU(0) local solver. See Table A.10 in the Appendix for numerical values.

Table 8
Flow around a rotating insect wing. Description of the computational meshes for the scaling runs.

Reference	Description	# elements	# velocity nodes	# pressure nodes
Mesh A	Mesh described in Sec. 4.2.2	2.1×10^6	16.8×10^6	2.1×10^6
Mesh B	Uniform refinement of Mesh A	16.6×10^6	134×10^6	16.8×10^6
Mesh C	Uniform refinement of Mesh B	133×10^6	1.07×10^9	134×10^6

increment update suggests that the local work on subdomains is too small to balance the cost of communication. The scalability begins to deviate slightly from optimal going from 2000 elements to 1000 elements per core and is lost when computing with 500 elements per core. A similar effect is seen for the L_2-projection of the pressure p, although we again emphasise, that this problem is by an order of magnitude quicker to solve than the others. In Fig. 14, we also present the average total time for one time step, including the time for computing and assembling the matrices, computing aerodynamic forces and output of results. Although these operations are embarrassingly parallel, as the plots indicate they can take significant amount of time. Hence, optimisation of these parts of the computation should be performed next.

Finally, in Fig. 15 the number of iterations and computational times for Meshes A, B and C are combined in order to illustrate weak scalability. It can be inferred from Fig. 15 that with increasing mesh size the number of iterations approximately doubles for the velocity problem. The increase is even higher for the pressure increment. This behaviour is common to all one-level domain decomposition techniques including the block Jacobi method. Therefore, weak scalability cannot be expected, although the solution times remain acceptable. Looking at the computational times in Fig. 15 we can see that the time required for the update of the pressure increment ψ is now comparable to that of the update of the first velocity component u_1. For 32,768 and 65,536 subdomains, the pressure corrector problem already dominates the solution process, and a better two-level preconditioner may become beneficial.

4.2.4. Reynolds number dependence of the iterative solvers

We compute one entire revolution of the Drosophila wing for four different Reynolds numbers (based on the velocity at the radius of gyration) Re $\in \{105, 263, 526, 1051\}$. The time step increment is chosen constant such that one entire revolution requires 3000 time steps. In accordance with Table 7, the velocity components u_1, u_2 and u_3 are updated with GMRES, the pressure increment ψ is updated with deflated PCG, and PCG is used for L_2-projection of the pressure p. In each case the block Jacobi preconditioner with ILU(0) as the local solver is used. For all computations we use Mesh A, see Table 8, and the number of utilised processors is 2048.

In Table 9 the minimum, maximum and average numbers of iterations over an entire revolution are given. The prescribed tolerance of the relative residual is 10^{-6}. We can see that the number of iterations for the update of velocity components decreases significantly with increasing Reynolds number. We recall here that the velocities are updated by solving a convection–diffusion equation and it is known that the ILU preconditioner performs best for convection-dominated problems, see [21,51]. Interestingly, the number of iterations for obtaining the pressure increment ψ also becomes smaller with higher Reynolds number. The number of iterations for the L_2-projection of pressure is very low and independent of the Reynolds number.

4.2.5. Flow characteristics and forces

The vortex structures exhibited by a rotating wing and the corresponding aerodynamic forces are very different from the ones for a translating wing. At high angles of attack both trajectories generate a

Table 9
Flow around a rotating insect wing. Number of iterations in dependence on the Reynolds number for solving for velocity components (u_1, u_2, u_3), pressure increment ψ and pressure p. For each Reynolds number the number of iterations over an entire revolution of the wing (3000 time steps) is reported. The format of the entries is 'minimum–maximum (*average*)' number of iterations. All computations are using the mesh shown in Fig. 11 and 2048 subdomains.

Re	u_1	u_2	u_3	ψ	p
105	23–29(*26.6*)	21–27(*25.8*)	24–29(*27.2*)	100–242(*168.6*)	7–7(*7.0*)
263	18–22(*19.9*)	15–21(*19.8*)	18–22(*20.3*)	93–229(*150.4*)	7–7(*7.0*)
526	14–17(*16.0*)	12–17(*16.6*)	14–18(*16.3*)	61–181(*106.5*)	6–8(*7.0*)
1051	11–15(*13.9*)	10–15(*14.8*)	13–17(*14.3*)	33–178(*73.9*)	6–8(*6.8*)

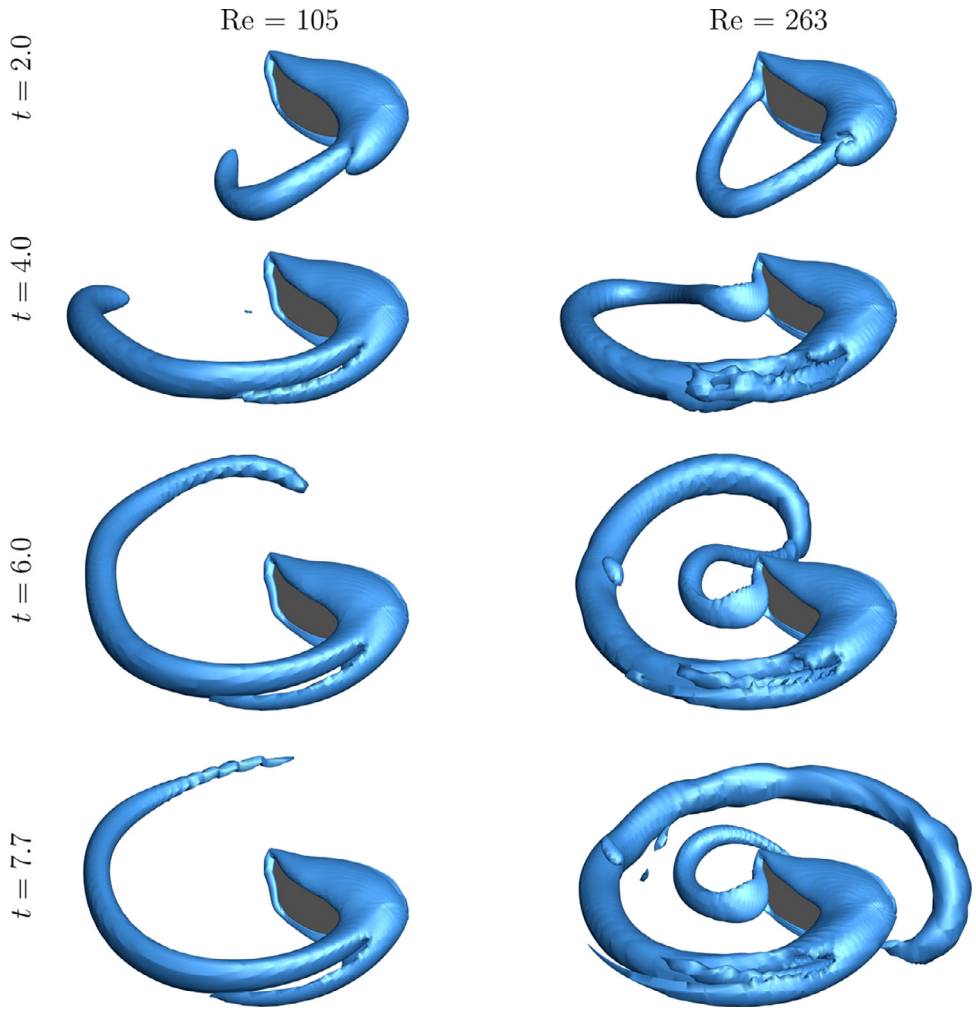


Fig. 16. Flow around a rotating insect wing. Isosurfaces of $Q = 3$ at different time instants, $\alpha = 40^\circ$, Re = 105 (left) and Re = 263 (right).

large leading-edge vortex. As it is known, whereas the leading-edge vortex of a translating wing is periodically shed (cf. Section 4.1.2), it remains attached for a rotating wing (under insect-like flight conditions). However, for higher aspect ratios and larger Reynolds numbers the leading-edge vortex may not be so well defined and breaks down near the tip [24,29]. The presence of the attached leading-edge vortex and the associated low pressure zone leads to a sustained lift force. Although this has been extensively discussed in the animal flight and engineering literature, its computational study became feasible only recently. Due to the truly three-dimensional nature of the flow, the size of the discretised problem becomes very large, especially when a direct numerical simulation is performed.

In Figs. 16 and 17 the Q-criterion isosurfaces with $Q = 3$ are used to visualise the flow. All snapshots show a vortex loop consisting of a leading-edge, trailing-edge and two tip vortices, see also the experimental results in [35,44]. The segments of the loop not connected to the wing have a downward velocity. Importantly, throughout the rotation the leading-edge vortex remains attached to the wing. For all the considered Reynolds numbers, the leading-edge vortex is compact close to the wing root and becomes larger and then lifts up nearer to the wing tip. Moreover, with increasing Reynolds number it is slightly tighter in the vicinity of the wing root. As to be expected the complexity of the observed vortex structures becomes more pronounced with increasing Reynolds number due to the decrease in the diffusivity of the flow. In particular, for Re = 526 and Re = 1051 we can see finger-like subvortices emanating from the wing tip and spiralling around the leading-edge vortex. As also observed in [25], the orientational sense of this spiralling appears to be opposite to the rotation of the vortex. Although not shown in Figs. 16 and 17 there is a significant spanwise flow, with its maximum comparable to the radial velocity, from the root to the tip of the wing. As initially postulated in [20] the spanwise flow and the corresponding vorticity flux limits an unbounded growth of the leading-edge vortex and prevents its shedding.

Finally, we present the history of aerodynamic coefficients in Fig. 18. The lift, drag, and spanwise force coefficients are computed according to (28) with the planform $S = R^2/\mathit{AR}$ and $u_\infty = R_g\omega$, the final velocity at the radius of gyration. It is helpful to recall that the wing is initially accelerated until $t = 1$ and then rotates with constant angular velocity ω until one full revolution is completed at $t = 7.7$. After a slight dip immediately after the acceleration phase, the coefficients are steady throughout the revolution even for the highest Reynolds number. Moreover, towards the end of the simulation when a full rotation is completed, the coefficients become slightly smaller. This is most likely due to the interaction of the tip vortices with the previously generated vortices. As can be seen in Fig. 18a and b both drag and lift coefficients become larger with increasing Reynolds number. However, the lift coefficient is much more sensitive to the Reynolds number than the drag coefficient. Especially, for an increase from Re = 105 to Re = 263 there is a sudden jump in the lift forces. The obtained drag and lift coefficients show similar trends as recently reported by Harbig et al. [29]. The lift-to-drag ratio (or,

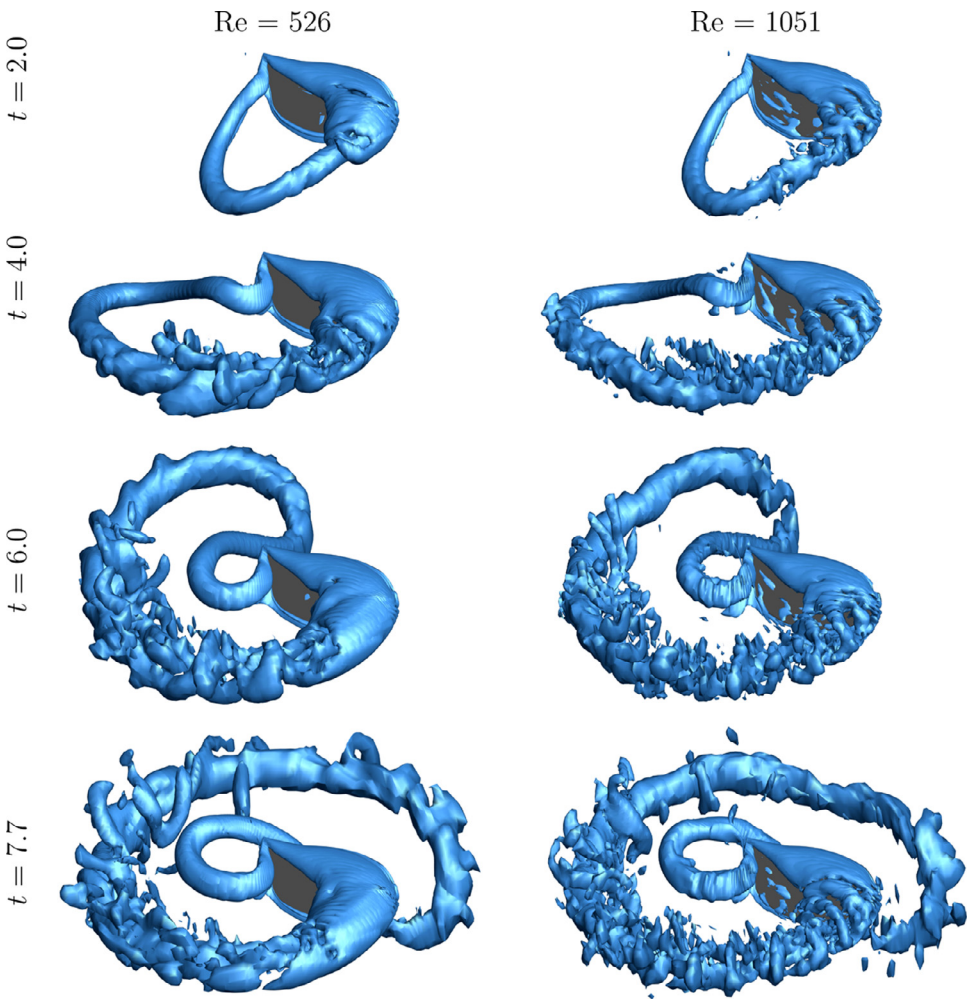


Fig. 17. Flow around a rotating insect wing. Isosurfaces of $Q = 3$ at different time instants, $\alpha = 40^\circ$, Re = 526 (left) and Re = 1051 (right).

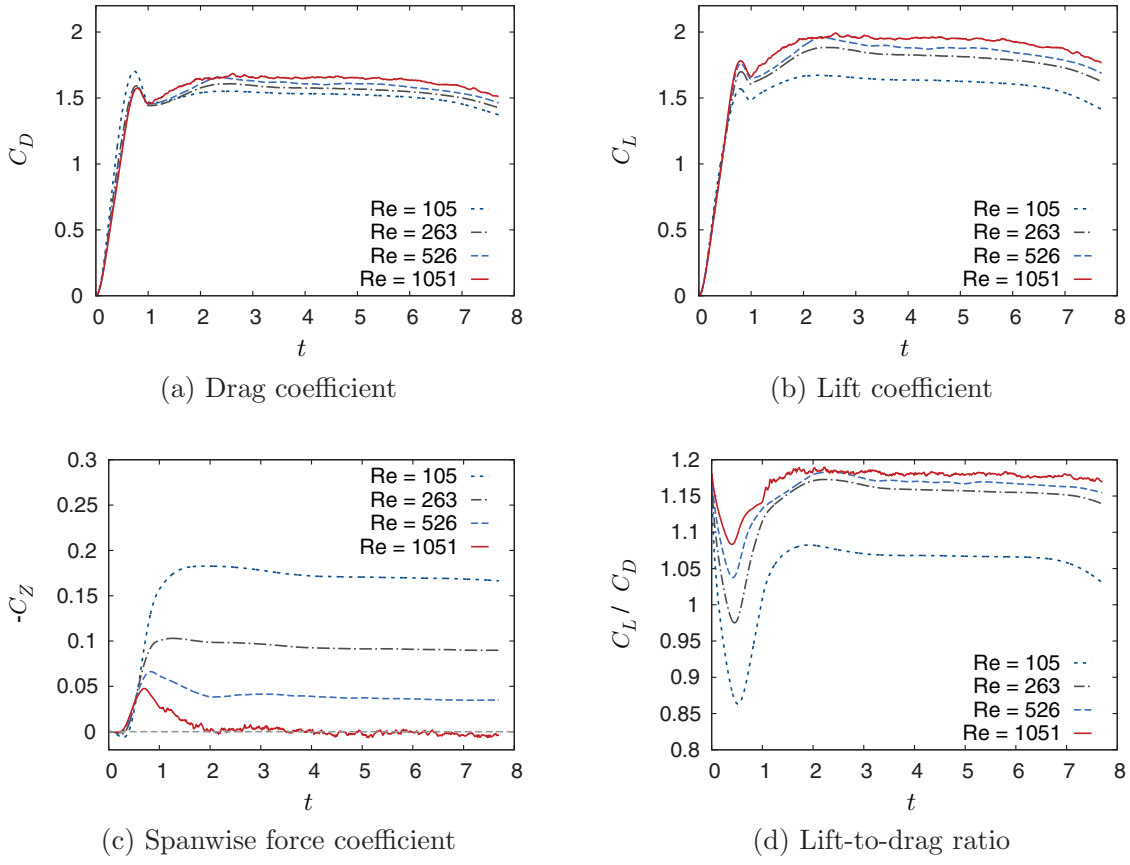
aerodynamic efficiency) in Fig. 18d indicates also a sudden increase going from Re = 105 to Re = 263. The force in the spanwise (root-to-tip) direction plotted in Fig. 18c is by an order of magnitude lower than drag and lift, and it becomes lower with increasing Reynolds number.

5. Conclusions

We considered the implicit parallel computation of three-dimensional low Reynolds number flows on stationary and rotating domains. The scalability of the overall finite element technique depends on the suitable formulation of the physical problem, the data structures, the algorithms for data handling and the preconditioned iterative solver. To this end, we introduced efficient and scalable techniques for domain partitioning, system matrix assembly and preconditioned iterative solution. In our implementation we make extensive use of open source libraries METIS, C++ STL and PETSc. The resulting software is very efficient and it is able to solve systems with $\approx 3.34 \times 10^9$ unknowns (of the corresponding coupled Navier–Stokes system) in around thirty seconds on 65,536 processors.

The discretisation of the Navier–Stokes equations with the pressure-correction method leads to one convection–diffusion problem with three different right-hand sides, one Poisson problem and one L_2-projection. We find that it is most efficient to precondition each subproblem with the block Jacobi preconditioner using ILU(0) on subdomains and to solve each with a suitable Krylov subspace iterative method, namely GMRES for nonsymmetric and PCG for symmetric problems. This gives an overall solution technique with almost perfect scaling even for very large problems and processor counts. The block Jacobi preconditioner with ILU(0) exhibits perfect scalability in case of the convection–diffusion problems and the considered discretisations with up to 1.07×10^9 velocity nodes and 65,536 processors. Moreover, it was observed that the number of iterations is inversely proportional to the Reynolds number of the flow. Although for larger processor counts the solution of the Poisson problem for the pressure increment becomes a bottleneck, it has only limited influence on overall solution time. The size of the discretised Poisson problem is only about 1/8 of the size of the discretised convection–diffusion problem. The discretised L_2-projection problem is of the same size as the discretised Poisson problem. However, it can be solved with much fewer iterations due to its better conditioning. In order to compute on larger processor counts, especially with a view to peta-scale computing platforms, it appears to be crucial to improve the scalability of the solution of the Poisson problem. To this end, the most promising techniques appear to be two-level or multilevel BDDC and FETI domain decomposition methods, see e.g. [12,22,23,52].

In order to make the developed approach truly useful for the study of flapping-flight aerodynamics, it is necessary to consider flexible wings. The used ALE approach with rigidly rotating meshes is however only suitable for rigid bodies and not applicable to flexible wings. In case of flexible wings it is in principle possible to use either non-boundary-fitting immersed grids [47], boundary-fitted meshes [55] or a combination of both [6]. When a partitioned (or, block



(a) Drag coefficient (b) Lift coefficient

(c) Spanwise force coefficient (d) Lift-to-drag ratio

Fig. 18. Flow around a rotating insect wing. Aerodynamic coefficients during one revolution for different Reynolds numbers.

Gauss–Seidel) approach is used for solving the resulting fluid-structure interaction problem, the approach developed here can be used for solving the fluid problem.

Acknowledgement

We are grateful to Charles P. Ellington and Holger Babinsky for valuable discussions on aerodynamics of insect flight. This research was supported by the Engineering and Physical Sciences Research Council (EPSRC) through Grant # EP/G008531/1. Additional support was provided by the Czech Science Foundation through Grant 14-02067S, and by the Czech Academy of Sciences through RVO:67985840. The presented computations were performed on HECToR at the Edinburgh Parallel Computing Centre through PRACE-2IP (FP7 RI-283493).

Appendix A

The following two tables give the data for the Figs. 13 and 14 introduced in Section 4.2.3.

Table A.10
Flow around a rotating insect wing. Iteration counts for meshes A, B and C (see Table 8) and different number of subdomains. Problems for velocity components u_1, u_2 and u_3 solved with GMRES, problems for pressure increment ψ and for pressure p with CG, all with block Jacobi preconditioner and ILU(0) as the local solver.

	Num. processors	Number of iterations				
		u_1	u_2	u_3	ψ	p
Mesh A	128	7–12(*11.0*)	0–11(*10.4*)	7–14(*13.1*)	119–198(*157.2*)	11–11(*11.0*)
	256	7–13(*12.1*)	0–13(*11.3*)	7–15(*13.2*)	125–205(*164.6*)	11–11(*11.0*)
	512	7–13(*11.6*)	0–13(*11.0*)	7–14(*12.9*)	130–217(*175.4*)	11–11(*11.0*)
	1024	7–13(*12.4*)	0–13(*12.2*)	8–14(*13.5*)	138–234(*186.5*)	11–12(*11.7*)
	2048	7–14(*12.6*)	0–13(*11.9*)	7–15(*13.7*)	144–249(*195.9*)	11–12(*11.5*)
	4096	8–14(*13.4*)	0–13(*12.2*)	8–16(*14.5*)	157–269(*213.5*)	12–12(*12.0*)
Mesh B	1024	11–20(*18.4*)	0–18(*16.4*)	11–24(*21.9*)	369–680(*529.7*)	12–12(*12.0*)
	2048	10–22(*19.6*)	0–19(*17.5*)	11–25(*23.0*)	373–688(*534.4*)	12–12(*12.0*)
	4096	10–22(*19.6*)	0–20(*17.6*)	10–25(*22.9*)	382–696(*543.9*)	12–12(*12.0*)
	8192	11–27(*23.4*)	0–24(*20.9*)	11–30(*26.5*)	385–704(*550.5*)	12–12(*12.0*)
	16 384	10–24(*21.7*)	0–22(*18.9*)	11–28(*24.7*)	394–712(*558.4*)	12–12(*12.0*)
Mesh C	8192	18–43(*36.4*)	0–38(*31.1*)	19–48(*40.9*)	820–1425(*1175.9*)	12–12(*12.0*)
	16 384	19–42(*37.0*)	0–38(*32.4*)	20–47(*41.6*)	832–1437(*1150.6*)	12–12(*12.0*)
	32 768	19–44(*38.1*)	0–39(*32.7*)	21–49(*42.8*)	836–1452(*1163.2*)	12–12(*12.0*)
	65 536	19–44(*36.9*)	0–38(*30.9*)	20–49(*42.1*)	855–1472(*1254.6*)	12–12(*12.0*)

Table A.11
Flow around a rotating insect wing. Average times for solving the linear systems and the total time per step for meshes A, B and C (see Table 8) with different number of subdomains. Problems for velocity components u_1, u_2 and u_3 solved with GMRES, problems for pressure increment ψ and for pressure p with CG, all with block Jacobi preconditioner and ILU(0) as the local solver. The number of processes (the same as cores) is denoted with 'num. proc.' and the average number of finite elements per subdomain with 'loc. size' .

	Num. proc.	Loc. size ($\times 10^3$)	Solution time (s)					Time per step (s)
			u_1	u_2	u_3	ψ	p	
Mesh A	128	16	5.198	1.552	1.952	1.238	0.092	220.640
	256	8	2.703	0.832	0.971	0.621	0.045	86.915
	512	4	0.789	0.413	0.479	0.304	0.023	39.801
	1024	2	0.392	0.220	0.242	0.190	0.014	19.127
	2048	1	0.193	0.105	0.120	0.129	0.009	9.876
	4096	0.5	0.101	0.055	0.064	0.134	0.009	6.286
Mesh B	1024	16	5.139	2.577	3.538	4.286	0.105	231.137
	2048	8	2.585	1.337	1.810	2.087	0.051	91.739
	4096	4	1.152	0.656	0.867	0.907	0.026	42.604
	8192	2	0.630	0.377	0.487	0.518	0.014	21.857
	16 384	1	0.289	0.169	0.222	0.368	0.010	12.439
Mesh C	8192	16	10.559	5.188	6.972	10.085	0.113	251.728
	16 384	8	5.139	2.614	3.423	5.013	0.066	100.716
	32 768	4	1.961	1.268	1.688	2.625	0.039	50.968
	65 536	2	0.941	0.587	0.820	1.721	0.029	33.112

References

[1] Amestoy PR, Duff IS, L'Excellent J-Y. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput Methods Appl Mech Eng 2000;184:501–20.
[2] Ansari SA, Phillips N, Stabler G, Wilkins PC, Zbikowski R, Knowles K. Experimental investigation of some aspects of insect-like flapping flight aerodynamics for application to micro air vehicles. Exp Fluids 2009;46(5):777–98.
[3] Badia S, Martín A, Principe J. A highly scalable parallel implementation of Balancing Domain Decomposition by Constraints. SIAM J Sci Comput 2014;36(2):C190–218.
[4] Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, et al. PETSc Web page. http://www.mcs.anl.gov/petsc; 2014.
[5] Bazilevs Y, Hughes TJR. NURBS-based isogeometric analysis for the computation of flows about rotating components. Comput Mech 2008;43(1):143–50.
[6] Bazilevs Y, Korobenko A, Deng X, Yan J. Novel structural modeling and mesh moving techniques for advanced fluid–structure interaction simulation of wind turbines. Int J Numer Methods Eng 2015;102(3–4):766–83.
[7] Birch J, Dickinson M. Spanwise flow and the attachment of the leading-edge vortex on insect wings. Nature 2001;412:729–33.
[8] Česenek J, Feistauer M, Horáček J, Kučera V, Prokopová J. Simulation of compressible viscous flow in time-dependent domains. Appl Math Comput 2013;219(13):7139–50.
[9] Chorin AJ. Numerical solution of the Navier-Stokes equations. Math Comput 1968;22:745–62.
[10] Cirak F, Cummings J. Generic programming techniques for parallelizing and extending procedural finite element programs. Eng Comput 2008;24:1–16.
[11] Concus P, Golub GH, O'Leary DP. A generalized conjugate gradient method for the numerical solution of elliptic PDE. In: Bunch JR, Rose DJ, editors. Sparse matrix computations. New York: Academic Press; 1976. p. 309–32.
[12] Dohrmann CR. A preconditioner for substructuring based on constrained energy minimization. SIAM J Sci Comput 2003;25(1):246–58.
[13] Donea J, Huerta A. Finite element methods for flow problems. John Wiley & Sons; 2003.
[14] Dudley R. The biomechanics of insect flight: form, function, evolution. Princeton, New Jersey: Princeton University Press; 2000.
[15] Ellington C. The aerodynamics of hovering insect flight. I. The quasi-steady analysis. Philos Trans R Soc Lond Ser B, Biol Sci 1984;305:1–15.
[16] Ellington C. The aerodynamics of hovering insect flight. II. Morphological parameters. Philos Trans R Soc Lond Ser B, Biol Sci 1984;305:17–40.
[17] Ellington CP. The aerodynamics of hovering insect flight. III. Kinematics. Philos Trans R Soc Lond Ser B, Biol Sci 1984;305:41–78.
[18] Ellington CP. The aerodynamics of hovering insect flight. IV. Aerodynamic mechanisms. Philos Trans R Soc Lond Ser B, Biol Sci 1984;305:79–113.
[19] Ellington CP. The aerodynamics of hovering insect flight. V. A vortex theory. Philos Trans R Soc Lond Ser B, Biol Sci 1984;305:115–44.
[20] Ellington CP, van den Berg C, Willmott AP, Thomas ALR. Leading-edge vortices in insect flight. Nature 1996;384:626–30.
[21] Elman HC, Silvester DJ, Wathen AJ. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. In: Numerical mathematics and scientific computation. New York: Oxford University Press; 2005. 978-0-19-852868-5; 0-19-852868-X.
[22] Farhat C, Lesoinne M, Le Tallec P, Pierson K, Rixen D. FETI-DP: a dual-primal unified FETI method. I. A faster alternative to the two-level FETI method. Int J Numer Methods Eng 2001;50(7):1523–44.
[23] Farhat C, Roux F-X. A method of finite element tearing and interconnecting and its parallel solution algorithm. Int J Numer Methods Eng 1991;32(6):1205–27.
[24] Garmann D, Visbal M. Dynamics of revolving wings for various aspect ratios. J Fluid Mech 2014;748:932–56.
[25] Garmann DJ, Visbal MR, Orkwis PD. Three-dimensional flow structure and aerodynamic loading on a revolving wing. Phys Fluids (1994-present) 2013;25(3):034101.
[26] Geuzaine C, Remacle J-F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 2009;79(11):1309–31.
[27] Girault V, Raviart P-A. Finite element methods for Navier-Stokes equations. Berlin: Springer-Verlag; 1986.
[28] Guermond JL, Minev P, Shen J. An overview of projection methods for incompressible flow. Comput Methods Appl Mech Eng 2006;195:6011–45.
[29] Harbig R, Sheridan J, Thompson M. Reynolds number and aspect ratio effects on the leading-edge vortex for rotating insect wing planforms. J Fluid Mech 2013;717:166–92.
[30] Heroux MA, Bartlett RA, Howle VE, Hoekstra RJ, Hu JJ, Kolda TG, et al. An overview of the Trilinos project. ACM Trans Math Softw 2005;31(3):397–423.
[31] Hunt JCR, Wray AA, Moin P. Eddies, stream, and convergence zones in turbulent flows. technical report. Center for Turbulence Research; 1988.
[32] Josuttis NM. The C++ standard library: a tutorial and reference. 2nd ed. Addison-Wesley; 2012.
[33] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 1998;20(1):359–92.
[34] Kim D, Choi H. Immersed boundary method for flow around an arbitrarily moving body. J Comput Phys 2006;212(2):662–80.
[35] Kim D, Gharib M. Experimental study of three-dimensional vortex structures in translating and rotating plates. Exp Fluids 2010;49(1):329–39.
[36] Kim J, Moin P. Application of a fractional-step method to incompressible Navier-Stokes equations. J Comput Phys 1985;59(2):308–23.
[37] Kolář V. Compressibility effect in vortex identification. AIAA J 2009;47(2):473–5.
[38] Kolář V, Šístek J. Corotational and compressibility aspects leading to a modification of the vortex-identification Q-criterion. AIAA J 2015;53(8):2406–10.
[39] Kolář V, Šístek J, Cirak F, Moses P. Average corotation of line segments near a point and vortex identification. AIAA J 2013;51(11):2678–94.
[40] Liesen J, Strakoš Z. Krylov subspace methods. Oxford University Press; 2013.
[41] Mandel J, Sousedík B, Dohrmann CR. Multispace and multilevel BDDC. Computing 2008;83(2–3):55–85.
[42] Meyers S. Effective STL: 50 specific ways to improve your use of the standard template library. Pearson Education; 2001.
[43] Nolan GR. Aerodynamics of vortex lift in insect flight. Department of Zoology, University of Cambridge; 2004.
[44] Ozen CA, Rockwell D. Three-dimensional vortex structure on a rotating wing. J Fluid Mech 2012;707:541–50.
[45] Quarteroni A, Saleri F, Veneziani A. Factorization methods for the numerical appproximation of Navier-Stokes equations. Comput Methods Appl Mech Eng 2000;188:505–26.
[46] Quarteroni A, Valli A. Domain decomposition methods for partial differential equations. Oxford Science Publications; 1999.

[47] Rüberg T, Cirak F. A fixed-grid b-spline finite element technique for fluid–structure interaction. Int J Numer Methods Fluids 2014;74(9):623–60.
[48] Saad Y. Iterative methods for sparse linear systems,. 2nd ed. Philadelpha, PA: SIAM; 2003.
[49] Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput 1986;7(3):856–69.
[50] Shyy W, Lian Y, Tang J, Viieru D, Liu H. Aerodynamics of low Reynolds number flyers. Cambridge University Press; 2008.
[51] Smith BF, Bjørstad PE, Gropp WD. Domain decomposition: parallel multilevel methods for elliptic partial differential equations. Cambridge: Cambridge University Press; 1996. ISBN 0-521-49589-X.
[52] Sousedík B, Šístek J, Mandel J. Adaptive-multilevel BDDC and its parallel implementation. Computing 2013;95(12):1087–119.
[53] Stroustrup B. The C++ programming language. 4th ed. Addison-Wesley; 2013.
[54] Taira K, Colonius T. Three-dimensional flows around low-aspect-ratio flat-plate wings at low Reynolds numbers. J Fluid Mech 2009;623:187–207.
[55] Takizawa K, Henicke B, Puntel A, Kostov N, Tezduyar TE. Space–time techniques for computational aerodynamics modeling of flapping wings of an actual locust. Comput Mech 2012;50(6):743–60.
[56] Tezduyar TE, Sameh A. Parallel finite element computations in fluid mechanics. Comput Methods Appl Mech Eng 2006;195(13):1872–84.
[57] Toselli A, Widlund OB. Domain decomposition methods—algorithms and theory. Springer Series in Computational Mathematics, 34. Berlin: Springer-Verlag; 2005.
[58] van der Vorst HA. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J Sci Stat Comput 1992;13(2):631–44.
[59] Wang S, Zhang X. An immersed boundary method based on discrete stream function formulation for two- and three-dimensional incompressible flows. J Comput Phys 2011;230:3479–99.