A 2D parallel high-order sliding and deforming spectral difference method

Bin Zhang\textsuperscript{a}, Chunlei Liang\textsuperscript{a,1}, Jingjing Yang\textsuperscript{a}, Yongwu Rong\textsuperscript{b}

\textsuperscript{a} Department of Mechanical and Aerospace Engineering, The George Washington University, Washington, DC 20052, United States
\textsuperscript{b} Department of Mathematics, The George Washington University, Washington, DC 20052, United States

\begin{abstract}
In this paper, we present a novel parallel high-order sliding and deforming spectral difference (SD\textsuperscript{2}) method for solving the two-dimensional compressible Navier–Stokes equations on unstructured quadrilateral grids. In the SD\textsuperscript{2} method, an arbitrary Lagrangian–Eulerian (ALE) approach is combined with a novel sliding interface approach to separately handle rotational and translational motions on a dynamic grid. The SD\textsuperscript{2} method is shown to be capable of completely removing grid skewness caused by rotating boundaries to improve solution qualities. The SD\textsuperscript{2} method is verified as high-order accurate and is scalable for parallel computing on a distributed-memory computer. The SD\textsuperscript{2} method is also proven to be more robust and more accurate than deforming-only ALE approach for problems involving large-amplitude rotating motions.
\end{abstract}

1. Introduction

Objects that undergo combined rotational and translational motions in fluids are ubiquitous in the nature. For example, birds/insects twist and flap their wings to generate lift and thrust in the air [47], sea turtles rotate and sweep their flippers to migrate long distances in the ocean [11]. By mimicking these heaving and pitching motions for generating thrust, engineers have designed micro air vehicles that are of interest for military and civilian applications [40]. On the other hand, heaving and pitching motions of air/hydrofoils can also be utilized for energy harvesting [30]. A common feature of the flow fields around these objects is the existence of highly vortical flow structures, which are critical to the performance of the objects [3,7,36,44].

There are two major challenges to numerically study these flows. The first challenge is to accurately capture vortices that are sensitive to numerical dissipation. The second challenge is to incorporate the moving geometries into a computational framework both accurately and efficiently. High-order (third and above) numerical methods, such as spectral difference (SD) method [19,27,46] and discontinuous Galerkin method [6], are better candidates than low-order methods for overcoming the first challenge due to their substantially smaller numerical dissipation.

To tackle the second challenge, numerous approaches dealing with moving geometries were developed in the past decades. These approaches can be roughly divided into four categories: remeshing approach, Chimera/overset grid approach [42], immersed boundary method (IBM) [31], and the arbitrary Lagrangian–Eulerian (ALE) method [10]. Regenerating a mesh every time step, i.e. remeshing, seems to be a natural choice at first thought. However, generating a high quality mesh for a complex geometry is a time-consuming task, which makes frequent global remeshing not realistic in practice. For this reason, remeshing is usually done only locally. In Chimera/overset grid approach, multiple foreground grids that enclose moving objects are overlapped to a background Cartesian grid. These two sets of grids are coupled through an extensive number of searches and interpolations, which makes the overset grid approach expensive. Examples of conservative overset grid methods can be found in [4,20,45]. The IBM is applicable to a wide range of problems, however, it has difficulties representing physical boundaries exactly. Examples of high-order IBM can be found in [21,26]. In the ALE method, the interior grid points of a computational domain will move or deform to accommodate the boundary motions, and the flow equations are attached to moving frames of reference. The advantage of the ALE method is that it is conservative and can be applied to both low- and high-order methods. However, traditional ALE method also has its limitations. For example, when an object experiences very large movement especially very large rotational motion, the grid will become so distorted with unacceptable quality, which is detrimental to numerical simulations.

A feasible way to alleviate the aforementioned limitation on the ALE method is to decompose the computational domain and separately deal with the rotational and translational motions by...
introducing nonconforming sliding interfaces. Previously, Zhang and Liang [50] extended the straight fixed mortar approach [17,29] to a curved dynamic mortar approach for the spectral difference (SD) method in order to couple non-overlapping rotating and stationary meshes. The resulting sliding-mesh spectral difference (SSD) method is high-order accurate, highly efficient and very robust. In this work, we extend the SSD method to a more general ALE form capable of dealing with deforming and rotating meshes, namely the sliding and deforming spectral difference (SD²) method. Take the mesh for a plunging and pitching airfoil as shown in Fig. 3(c) for example, in the SD² method, we decompose the domain into two (or more) sub-domains: a small circular sub-domain that encloses the airfoil, and an outer one that takes the rest of the domain. Under the ALE framework, the inner sub-domain plumes and pitches as a rigid body following the airfoil, while the outer sub-domain deform to accommodate the plunging motion of the inner sub-domain. In this way, the rotational motion is completely removed from the outer sub-domain, forming a sliding interface in between the two sub-domains. Communication on the sliding interface is realized using the aforementioned curved dynamic mortar approach [50]. Some preliminary results of the present work were also reported in a previous conference paper [51].

The rest of this paper is organized as follows. In Section 2, the governing equations in both physical and computational forms are described. In Section 3, a brief review of the SD method, the mesh movement strategy, the sliding interface treatment and the parallelization are presented. In Section 4, a series of numerical tests are reported. Finally, Section 5 concludes this paper.

2. The governing equations

The following two-dimensional Navier–Stokes equations in conservative form for ideal gas are considered,

\[
\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = 0,
\]

where \( \mathbf{Q} \) is the vector of conservative variables, \( \mathbf{F} \) and \( \mathbf{G} \) are the \( x \) and \( y \) flux vectors. These terms have the following expressions,

\[
\mathbf{Q} = [\rho \, \rho u \, \rho v \, E]^T,
\]

\[
\mathbf{F} = \mathbf{F}_\text{inv}(Q) + \mathbf{F}_\text{vis}(Q, \nabla Q),
\]

\[
\mathbf{G} = \mathbf{G}_\text{inv}(Q) + \mathbf{G}_\text{vis}(Q, \nabla Q),
\]

where \( \rho \) is fluid density, \( u \) and \( v \) are the \( x \) and \( y \) velocities, \( E = p/(\gamma - 1) + \frac{1}{2} \rho (u^2 + v^2) \) is the total energy per volume, \( p \) is pressure, \( \gamma \) is the ratio of specific heats and is set to 1.4.

In Eqs. (3) and (4), the fluxes have been split into inviscid and viscous components. The inviscid fluxes are only functions of the conservative variables, and have the following expressions,

\[
\mathbf{F}_\text{inv} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (E + p) u \end{bmatrix}, \quad \mathbf{G}_\text{inv} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ (E + p) v \end{bmatrix}.
\]

Whereas, the viscous fluxes are functions of both the conservative variables and their gradients, which can be expressed as,

\[
\mathbf{F}_\text{vis} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ u \tau_{xx} + v \tau_{yx} + kT \end{bmatrix}, \quad \mathbf{G}_\text{vis} = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u \tau_{xy} + v \tau_{yy} + kT \end{bmatrix}.
\]

where \( \tau_{ij} \) denotes the shear stress tensor that is related to velocity gradients as \( \tau_{ij} = \mu (u_{ij} + u_{ji}) + \lambda \delta_{ij} \delta_{kk} \), \( \mu \) is the dynamic viscosity, \( \lambda = -2/3 \mu \) based on Stokes’ hypothesis, \( \delta_{ij} \) is the Kronecker delta, \( k \) is the thermal conductivity, \( T \) is the temperature that is related to density and pressure through the ideal gas law \( p = \rho RT \). \( R \) is the gas constant.

As will be discussed in the next section, in the SD method, each physical grid cell is transformed to a standard square element in the computational space where the governing equations are solved. Assume that the physical time and space \((t,x,y)\) are related to the computational ones \((\tau,\xi,\eta)\) through a mapping: \( t = \tau, \, x = x(\tau, \xi, \eta), \, y = y(\tau, \xi, \eta) \). It can be shown that Eq. (1) will take the following conservative form in the computational space,

\[
\frac{\partial \mathbf{Q}}{\partial \tau} + \frac{\partial \mathbf{F}}{\partial \xi} + \frac{\partial \mathbf{G}}{\partial \eta} = 0. \tag{7}
\]

The computational solution and fluxes are related to the physical ones as,

\[
\begin{pmatrix} \mathbf{Q} \\ \mathbf{F} \\ \mathbf{G} \end{pmatrix} = \begin{pmatrix} \mathcal{J} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{Q} \\ \mathbf{F} \\ \mathbf{G} \end{pmatrix}, \tag{8}
\]

where \( |\mathcal{J}| \) and \( \mathcal{J}^{-1} \) are the determinant and inverse of the Jacobian matrix \( \mathcal{J} \), respectively. Since time is independent of space (i.e. \( \tau_r = 1, \, t_\xi = 0, \, t_\eta = 0 \)), \( |\mathcal{J}| \) and \( \mathcal{J}^{-1} \) have the following expressions,

\[
|\mathcal{J}| = \left| \frac{\partial (t,x,y)}{\partial (\tau,\xi,\eta)} \right| = \begin{vmatrix} x_t & x_\xi & x_\eta \\ y_t & y_\xi & y_\eta \end{vmatrix}, \tag{9}
\]

\[
\mathcal{J}^{-1} = \left| \frac{\partial (\tau,\xi,\eta)}{\partial (t,x,y)} \right| = \begin{vmatrix} 1 & 0 & 0 \\ \xi_t & \xi_\xi & \xi_\eta \\ \eta_t & \eta_\xi & \eta_\eta \end{vmatrix} = \begin{vmatrix} 1 / |\mathcal{J}| & 0 & 0 \\ \xi_t / \mathcal{J} & y_\eta & -x_\eta \\ \eta_t / \mathcal{J} & -y_\xi & x_\xi \end{vmatrix}, \tag{10}
\]

where \( A = -x_t y_\eta + y_t x_\eta, \, B = x_t y_\xi - y_t x_\xi \), and \( x_t \) and \( y_t \) can be interpreted as grid velocities. For all simulations in the present work, grid coordinates \((x,y)\) and velocities \((x_t,y_t)\) are prescribed analytically.

For deforming grids, besides the above equations, the following geometric conservation law (GCL) [43] has to be considered to ensure global conservation of cell volumes and free-stream preservation,

\[
\frac{\partial}{\partial \tau} (|\mathcal{J}| \xi_t) + \frac{\partial}{\partial \eta} (|\mathcal{J}| \eta_t) = 0, \quad \frac{\partial}{\partial \tau} (|\mathcal{J}| \xi_\xi) + \frac{\partial}{\partial \eta} (|\mathcal{J}| \eta_\eta) = 0,
\]

\[
\frac{\partial}{\partial \tau} (|\mathcal{J}| \xi_\xi) + \frac{\partial}{\partial \eta} (|\mathcal{J}| \eta_\eta) = 0. \tag{11}
\]

Since grid motion has been prescribed, the first two GCL equations are satisfied automatically. From the third equation, \( |\mathcal{J}| \mathcal{J}^{-1} / \partial \tau \) is computed and added to the governing equations as a source term.

3. Numerical methods

3.1. The SD method

The SD method was first proposed by Kopriva and Koliás [19] as a staggered-grid Chebyshev multidomain method. Liu et al. [27] and Wang et al. [46] extended this method to unstructured triangular grids for solving the Euler equations. Liang et al. [22] further extended the SD method to Navier–Stokes equations
on unstructured grids with mixed elements. The stability of the SD method for 1D wave equations was proved by Jameson [13] for all orders of accuracy.

In SD method for quadrilateral grids, each physical cell is mapped to a standard square element \((0 \leq \xi \leq 1, 0 \leq \eta \leq 1)\) where staggered solution points (SPs, denoted as \(X_i\)) and flux points (FPs, denoted as \(X_{i+1/2}\)) are defined. Fig. 1 shows a schematic of distributions of SPs and FPs for a third order SD scheme. Generally, for an \(N\)th order scheme, \(N\) SPs and \((N+1)\) FPs are employed in each coordinate direction within an element. In the present study, the SPs are chosen as \(N\) Chebyshev–Gauss points, and FPs are \((N−1)\) Legendre–Gauss points plus two end points.

To construct solution and flux polynomials, the following Lagrange interpolation bases \(h_i\) and \(l_{i+1/2}\) are defined at the SPs and FPs, respectively,

\[
h_i(X) = \sum_{j=1}^{N} \frac{X - X_j}{X_i - X_j},
\]

\[
l_{i+1/2}(X) = \sum_{j=0}^{N} \frac{X - X_{i+1/2}}{X_{i+1/2} - X_j}.
\]

After that, the solution and fluxes are constructed from tensor products of the Lagrange bases,

\[
\tilde{Q}(\xi, \eta) = \sum_{j=1}^{N} \sum_{i=1}^{N} \tilde{Q}_{i,j} h_i(\xi) \cdot h_j(\eta),
\]

\[
\tilde{F}(\xi, \eta) = \sum_{j=0}^{N} \tilde{F}_{i+1/2,j} l_{i+1/2}(\xi) \cdot h_j(\eta),
\]

\[
\tilde{G}(\xi, \eta) = \sum_{j=0}^{N} \sum_{i=0}^{N} \tilde{G}_{i,j+1/2} h_i(\xi) \cdot l_{j+1/2}(\eta),
\]

where \(\tilde{Q}_{i,j}, \tilde{F}_{i+1/2,j}\) and \(\tilde{G}_{i,j+1/2}\) are the discrete solution and fluxes at the corresponding SPs and FPs.

The above constructed solution and fluxes are only element-wise continuous, but discontinuous across cell interfaces. For inviscid fluxes, a Riemann solver, such as [37, 38], has to be employed to compute common inviscid fluxes at cell boundaries to ensure conservation and stability. In the current implementation, the Rusanov solver [38] has been used for this purpose. The common viscous fluxes are computed from the common solutions and common gradients, details can be found in [22]. Finally, the residual is computed by directly taking derivatives of the flux polynomials, and a time marching scheme is used to update the solution.

The SD method is one type of compact differential formulations [12], and has been thoroughly tested by many researchers. For example, Liang et al. [23] performed large eddy simulation (LES) of channel flows using the SD method, Castonguay et al. [5] simulated transitional flow over airfoils using the SD method, Mohammad et al. [32] carried out LES of flow past a cylinder, Parsani et al. [34] studied flow induced noise, Lodato et al. [28] did wall modeled LES, Ou et al. [33] and Yu et al. [49] applied the SD method to moving grids, to name just a few. As pointed out by Bassi and Rebay [1], for high-order methods involving curved boundaries, the use of high-order elements is crucial for achieving smooth solution and for improving robustness, examples for the SD method can be found in [22, 24]. Through numerical tests, we also notice that high-order representation is important on the sliding interface. Therefore, throughout this paper, high-order cubic elements represented by 12 nodes are used for all curved boundaries including the sliding interfaces.

3.2. Mesh movement strategy

In the ALE method, two categories of approaches are most widely used to update grid coordinates and motions. One is to solve a partial differential equation with given boundary motions, for example, by solving a biharmonic equation [9]. The other is to use an algebraic function to control mesh movements, examples are [8, 35]. The first kind of methods usually give better mesh qualities, but are generally much more expensive than the second kind. For problems with small to moderate mesh deformation, the second kind is usually the preferred choice as it is efficient with acceptable mesh qualities. In the present work, we combine an approach from the second category together with sliding interface technique to control mesh movement.

To make the explanation intuitive, we take a schematic computational domain for a plunging and pitching airfoil as an example. As shown in Fig. 2, the domain has been split into two parts: an inner circular sub-domain \(\Omega_1\) with a radius \(R_1\), and an outer sub-domain \(\Omega_2\) that takes the rest of the domain. The outer sub-domain is further split into three virtual regions (i.e. \(\Omega_{2, 1}, \Omega_{2, 2}\) and \(\Omega_{2, 3}\)) by two virtual circular boundaries \(R_2\) and \(R_3\). The initial stationary mesh is taken as a reference, and we denote the coordinates of it as \(x_0\) and \(y_0\), radius as \(r_0\). Meanwhile, we denote coordinates of a moving mesh as \(x\) and \(y\). Suppose that, up to a
time instant, the airfoil has translated $(\Delta x_x, \Delta y_x)$ and rotated $\theta_c$ with respect to its initial position. We let the inner sub-domain translate and rotate following the airfoil as a rigid body. Thus, the coordinates of the moving mesh in this region can be expressed as,

$$\begin{aligned}
\begin{cases}
  x = x_0 + \Delta x_x + r_0 \cdot \cos(\theta_c) & \text{in } \Omega_1, \\
  y = y_0 + \Delta y_x + r_0 \cdot \sin(\theta_c)
\end{cases}
\end{aligned}$$

(17)

At the same time, we let the outer sub-domain translate and deform in order to accommodate the translational motion of the inner sub-domain. The deformation is controlled by using the polynomial blending function reported in [35]. To accomplish our goal, we first define an intermediate variable $s = (r_0 - R_0)/(R_0 - R_2)$ on the initial stationary reference mesh. After that, a blending function is defined on the reference mesh as,

$$b(s) = \begin{cases}
  1, & \text{if } s \leq 0 \quad (\text{i.e. in } \Omega_{2,1}) \\
  1 - 10s^3 + 15s^4 - 6s^5, & \text{if } 0 < s < 1 \quad (\text{i.e. in } \Omega_{2,2}) \\
  0, & \text{if } 1 \leq s \quad (\text{i.e. in } \Omega_{2,3})
\end{cases}$$

(18)

where $b(s) = 1$ means that mesh in that region undergoes a rigid body motion without any deformation, $b(s) = 0$ means that mesh in that region is stationary, $b(s)$ with a value between 0 and 1 indicates that mesh there is deformed. This blending function has derivatives of zero on the deforming boundaries and can thus generate smooth deformation on the mesh. Coordinates of moving mesh in domain $\Omega_2$ can be expressed as,

$$\begin{aligned}
\begin{cases}
  x = x_0 + b(s) \cdot \Delta x_x & \text{in } \Omega_2 = \Omega_{2,1} \cup \Omega_{2,2} \cup \Omega_{2,3}, \\
  y = y_0 + b(s) \cdot \Delta y_x
\end{cases}
\end{aligned}$$

(19)

Since velocities are simply time derivatives of $x$ and $y$, if grid displacement is given analytically, then grid velocities can be achieved by differentiating $x$ and $y$ with respect to time directly.

The advantage of splitting the domain using sliding interface is that, the rotating motion will be completely removed from the outer sub-domain. Therefore, mesh skewness can be greatly reduced for problems with large rotational motions. For example, Fig. 3 shows the meshes for the computational domain in Fig. 2. The airfoil has a chord length of $c = 1$. The size of the domain is $7 \times 7$. The motion of the airfoil is prescribed as: $h(t) = 0.5 \sin(2\pi ft)$ and $\theta(t) = 70^\circ \sin(2\pi ft)$, where $h$ is the vertical displacement, $\theta$ is the rotating angle. Fig. 3(a) shows the initial mesh that has not been deformed. Fig. 3(b) shows a deforming-only mesh at a time instant, the deforming region is in between $R_0 = 0.6$ and $R_3 = 3.4$. Fig. 3(c) is a sliding-deforming mesh at the same time as in Fig. 3(b), the sliding and deforming boundaries are $R_4 = 1$, $R_0 = 1.1$ and $R_0 = 3.4$. It is obvious that by introducing a sliding interface, the mesh quality in Fig. 3(c) is much better than the one in Fig. 3(b).

3.3. Mesh partitioning

The first step in parallelization is to partition and distribute the mesh to all processors. Since each sub-domain has a separate mesh in the SD2 method, more thoughts are required on the partitioning. Let us assume that we have $N_p$ processors and $N_M$ separate meshes. One possible approach is that we can partition each of the $N_M$ meshes separately into $N_p$ partitions, and each processor receives $N_M$ mesh partitions. However, this approach has several disadvantages. It will create a total number of $N_M \times N_p$ partitions, which are likely to cause heavy inter-processor communications. It may also lead to load-imbalance, because achieving load-balance on many small pieces of meshes is more difficult than on a big mesh. This approach may even fail in the extreme situation when the number of processors is larger than the number of grid cells in a sub-domain.

In the present implementation, we combine the separate sub-domain meshes into a single big mesh, and then partition this single but discontinuous mesh. The steps are summarized as below:

1. The root processor reads in $N_M$ separate sub-domain meshes and temporarily store them.
2. The root processor converts the numbering of cells, vertices and boundaries, etc. of each sub-domain mesh to global numbering by adding an offset. The offset is the sum of the corresponding numbers from previous sub-domain meshes. For example, a problem of three sub-domains is assumed to have 100 cells for each sub-domain. The offset is set to 0, 100 and 200, respectively, for sub-domain 1, 2 and 3.
3. The root processor decomposes the single combined mesh into $N_p$ partitions by calling subroutines of the Metis library [15].
4. The root processor broadcasts the partition information to other processors and frees its memory.
5. All processors read in their own part of mesh from the separate mesh files according to the partition information.

The advantage of this approach is that the actual number of partitions is usually minimized, which reduces interprocessor communications. In addition, load-balance can be achieved globally, i.e., if load-balance can not be achieved in one sub-domain mesh, that processor then “borrows” cells from other sub-domain meshes to achieve load-balance. For example, Fig. 4 shows a partition for 8 processors for the mesh in Fig. 3(c). It is seen that the partition for processor P3 is not continuous: one half of the partition is within the sliding sub-domain, while the other half is at the right
of solution/fluxes face cal mortar time face of on curved grids which well mean that for most cases, this approach can achieve load-balance as well as minimum number of partitions.

3.4. Sliding-mesh interface treatment

The mortar method was introduced by Mavriplis [29] to solve the incompressible flow equations on stationary nonconforming grids in the context of finite element methods. Kopriva [17, 18] applied this approach to the spectral Chebyshev method for solving the compressible flow equations on stationary nonconforming grids, and proved the conservation property of this approach. Zhang and Liang [50] extended this approach to a dynamic curved mortar approach, and combined it with a sliding-mesh technique to study flows around rotating objects. They showed that the curved dynamic mortar approach provides a simple, efficient and high-order accurate way to couple rotating and stationary meshes with sliding interfaces.

The key idea of the mortar method is to project solution/fluxes from cell face to mortars, then compute common solution/fluxes on the mortars, and finally project the common values back to cell faces to ensure conservation. On the left of Fig. 5 is a schematic of the distribution of mortars between two relatively rotating sub-domains for a serial solver. As we can see, for a serial solver, at each time instant a cell face is connected to two mortars, and each mortar is associated with two cell faces, i.e. one on the left (inner sub-domain) and one on the right (outer sub-domain). This cell face and mortar connectivity needs to be updated every stage of a time marching scheme. The detailed steps of the curved dynamic mortar approach for serial solver can be found in our previous paper [50]. However, for a parallel solver, the situation will be a bit different. On the right of Fig. 5 is a schematic of the mortar distribution on 6 processors (partitions). Now, each processor has its own local mortars, and each mortar is associated with only one cell face. For example, a mortar on processor P4 is only connected to a cell face on its left, and only the left solution/fluxes are locally available, while the right solution/fluxes need to be sent from its pairing mortar on processor P3 or P4. A local mortar finds its pairing mortar through a local-mortar-to-global-mortar connectivity: two local mortars that share the same global mortar will form a pair. A global mortar is similar to the mortar in a serial solver, and is represented by a thick gray line segment in Fig. 5. The local-mortar-to-global-mortar connectivity needs to be updated every sub-time step to ensure correct inter-processor communication. In practice, global mortars can be fixed to either the inner or the outer sub-domain. For simplicity, let us assume that global mortars are fixed on the inner sub-domain. Thus, the aforementioned connectivity only needs to be updated for local mortars on the outer sub-domain.

Let $\Omega$ denote a cell face, $\Xi$ denote a mortar, and assume that cell faces on the sliding interface have been ordered to be counterclockwise during pre-processing. The curved dynamic mortar approach for a parallel solver then includes the following steps:

1. On the outer sliding interface, find the cell face that the starting point of the first global mortar has traveled to; compute mortar scalings, offsets and projection matrices.
2. Using the above matched cell face to update local-mortar-to-global-mortar connectivity on the outer sliding interface.
3. Pair mortars using on the above connectivity.
4. Map curved mortars to straight ones with unit length and compute normals of mortars.
5. Project solutions from cell faces to local mortars, then send/receive solutions to/from pairing mortar. For example, to project solution from a face $\Omega$ to the left side of mortar $\Xi$, we require that
   \[
   \int_{\Omega} (Q_{\Xi}^2 \cdot h_{\Omega}(z) \, dz = 0), \quad j = 1, 2, \ldots, N, \quad \text{where} \quad z \text{ represents coordinate in mortar space,} \quad Q_{\Xi}.h \text{ is the solution on the left side of } \Xi, \quad h_j \text{ is the Lagrange polynomial basis.}
   \]
6. On a mortar $\Xi$, compute the common solution by averaging the left and right solutions, i.e.
   \[
   Q_{\Xi} = \frac{1}{j} (Q_{\Xi}^{L E} + Q_{\Xi}^{R E}).
   \]
   Compute the common inviscid fluxes $F_{\Xi_{mir}}$ and $G_{\Xi_{mir}}$ from the left and right solutions using a Riemann solver, e.g. Rusanov [38] or Roe [37] solver.
7. Project the common inviscid fluxes from mortars back to cell faces in a least-squares fashion, i.e.
   \[
   \int_{\Omega} (F_{\Xi_{mir}}(\xi) - F_{\Xi_{mir}}(\xi_{j}) h_{\Omega}(\xi) \, d\xi + \int_{\Xi} (G_{\Xi_{mir}}(\xi) - G_{\Xi_{mir}}(\xi_{j}) h_{\Omega}(\xi) \, d\xi = 0, \quad \text{where} \quad \xi_j \text{ is the offset of the second mortar,} \quad F_{\Xi_{mir}} \text{ and } G_{\Xi_{mir}} \text{ are the common inviscid fluxes on the first and the second mortar connected to face } \Omega, \text{ and } C_{\Xi_{mir}} \text{ is the expected inviscid flux on the cell face. The common solution } Q_{\Xi} \text{ is projected back to cell faces in the same way.}
   \]
8. On cell faces, compute viscous fluxes based on the updated solutions from step (5), and project them to mortars in the same way as in step (3) for solutions.
9. Compute common viscous fluxes on mortars by averaging left and right viscous fluxes, and project back to cell faces in the same way as in step (5) for inviscid fluxes.

4. Numerical tests

In this section, we report a series of numerical tests using the SD2 solver. For all test cases, a five-stage fourth-order explicit Runge–Kutta scheme [39,41] is employed for the temporal discretization. All simulations were performed on a cluster configured with 2.6GHz Intel Xeon E5-2670 CPUs, Mellanox FDR Infiniband (56Gbps) interconnect and distributed memories.

4.1. Euler vortex flow on rotating and deforming grid

In the Euler vortex flow problem, an isentropic vortex is superimposed to and convected by an uniform mean flow. The flow field in an infinite domain at a time instant $t$ can be analytically
expressed as,
\[ u = U_\infty \left\{ \cos \theta - \frac{\epsilon y_r}{r_c} \exp \left( \frac{1 - x_r^2 - y_r^2}{2r_c^2} \right) \right\}, \]  
(20)
\[ v = U_\infty \left\{ \sin \theta + \frac{\epsilon x_r}{r_c} \exp \left( \frac{1 - x_r^2 - y_r^2}{2r_c^2} \right) \right\}, \]  
(21)
\[ \rho = \rho_\infty \left\{ 1 - \frac{(y - 1)(\epsilon M_\infty)^2}{2} \exp \left( \frac{1 - x_r^2 - y_r^2}{r_c^2} \right) \right\} \frac{1}{t^4}, \]  
(22)
\[ p = p_\infty \left\{ 1 - \frac{(y - 1)(\epsilon M_\infty)^2}{2} \exp \left( \frac{1 - x_r^2 - y_r^2}{r_c^2} \right) \right\} \frac{1}{t^4}, \]  
(23)
where \( U_\infty, \rho_\infty, \rho_\infty, M_\infty, \) and \( \theta \) are the speed, density, pressure, Mach number, and direction, respectively, of the mean flow. \( \epsilon \) and \( r_c \) can be interpreted as the vortex strength and size. The relative coordinates \((x_r, y_r)\) are defined as \( x_r = x - x_0 - \hat{u}t \), and \( y_r = y - y_0 - \hat{v} \), where \( \hat{u} = U_\infty \cos \theta \) and \( \hat{v} = U_\infty \sin \theta \) are the \( x \) and \( y \) components of the mean velocity. \((x_0, y_0)\) is the initial position of the vortex. The Euler vortex flow within a square domain \( (0 \leq x, y \leq L) \) with periodic boundary conditions can also be obtained analytically, details can be found in a previous paper [50].

In the present test, the uniform mean flow is chosen as \((U_\infty, \rho_\infty, p_\infty) = (1, 1, 1)\) with a Mach number of \( M_\infty = 0.3 \). The flow direction is set to \( \theta = \arctan(1/2) \). A vortex with \( \epsilon = 1 \) and \( r_c = 1 \), is superimposed to the mean flow. The domain size is \( 0 \leq x, y \leq 10 \) (i.e. \( L = 10 \)), and the vortex is initially placed at the domain center \((x_0, y_0) = (5, 5)\). Periodic boundary conditions are applied in both \( x \) and \( y \) directions. The computational domain is divided into two sub-domains: an inner one with a radius of \( R_i = 2 \); an outer one that takes the rest of the domain. The inner circular sub-domain undergoes a vertical heaving motion plus a rotating motion around its center. The rotating speed is \( \omega = \pi \), and the heaving motion is prescribed as \( h = 0.5 \sin(\omega t) \). The outer sub-domain is deformed to accommodate the heaving motion of the inner sub-domain. As defined in the previous section, the deformation region for this case is bounded by \( R_0 = 2.3 \) and \( R_0 = 4.7 \). Three meshes with total numbers of cells of 180, 700 and 2731 are used for accuracy tests.

Fig. 6 (a) and (b) show the density contours on top of the corresponding mesh with 700 cells at an upstroke and a downstroke time instant computed using 8 processors. We see that, in both figures, mesh cells in either the top or the bottom region have been squeezed while minor or no deformation elsewhere. From the density contours, it is seen that grid motion and the sliding interface do not cause any alteration to the shape of the vortex.

To evaluate the spatial order of accuracy, the \( L_1 \) and \( L_2 \) errors of density are computed at \( t = 2 \), i.e. when the vortex center travels right onto the sliding interface. The errors have been plotted against the number of cells in logarithm scale in Fig. 7. It is obvious that the errors decrease at the correct orders as the number of grid cells increases for both third- and fourth-order schemes.

To verify the parallelization, we have summarized in Table 1 the computation time, speedup, and \( L_2 \) errors for a run of 1000 time steps with the mesh of 700 cells using a fourth-order scheme and various numbers of processors. As we can see, the parallelization shows an almost linear speedup for this test case. Meanwhile, the differences between the \( L_2 \) errors from different numbers of processors are within machine precision.

### 4.2. Taylor–Couette flow
Taylor–Couette flow is formed between two concentric circular walls, with one or both of them rotating. When Reynolds number is small, Taylor–Couette flow will reach a steady state where viscous effects balances inertia. The steady-state circumferential velocity of this flow has the following analytical expression,
\[ v_\theta = \omega_1 r_i (r_0/r - r/r_0) + \omega_0 r_0 r/r_0 - r/r_0, \]  
(24)
on both walls. The Reynolds number based on inner wall radius and speed is $Re = 10$. Reference Mach number on the inner wall is set to $Ma_0 = 0.1$. Three meshes with 200, 800 and 3200 cells are used for spatial accuracy tests.

Fig. 8 shows the contours of the $u$ velocity and the Mach number on top of the corresponding mesh with 200 cells computed using 8 processors. We see that the steady state $u$ velocity contours have a symmetric shape with values of opposite signs. The Mach number contours form a series of concentric circles. These results are consistent with the analytical solutions.

The $L_1$ and $L_2$ errors of the $u$ velocity (i.e. the $x$-component of $v_0$) are adopted to compute the orders of accuracy. Fig. 9 shows the errors against the numbers of cells for third- and fourth-order schemes. The SD$^2$ solver is found to give the optimal orders of accuracy on this viscous flow as well.

Again, we monitored the speedup and $L_2$ errors on various numbers of processors. The results are shown in Table 2. The simulation was started from a uniform flow field and ran for 1000 time steps on the mesh with 800 cells using a fourth-order scheme.

Similarly, we have observed a linear speedup and the differences in $L_2$ errors are within machine precision.

4.3. Flow over a plunging and pitching airfoil

In this test case, we simulate flow over a plunging and pitching airfoil and compare results from deforming-only mesh and sliding-deforming mesh. The airfoil is chosen as a NACA0012 airfoil with a chord length of $c = 1$. It undergoes a plunging motion $h(t) = h_0 \cos(2\pi ft)$, and a pitching motion $\theta(t) = \theta_0 \sin(2\pi ft)$ around one-third chord. We fix the frequency and plunging amplitude as $f = 0.4$ and $h_0 = 0.25$. Three pitching amplitudes $\theta_0 = 30^\circ$, $45^\circ$ and $60^\circ$ have been tested. The free-stream flow has a Mach number of $Ma_{\infty} = 0.2$. The Reynolds number based on the free-stream velocity, viscosity, and the airfoil chord length is $Re = 1000$.

Fig. 10 shows a global and a local view of the mesh for this simulation. The deforming-only mesh is exactly the same as the sliding-deforming mesh but without a sliding interface (i.e., is conforming). The computational domain has a size of $100 \times 100$. The airfoil is located 30 chord lengths downstream from the inlet. The domain is discretized into 9659 mesh cells. As can be seen, cells are highly clustered to the center of the domain (around the airfoil and in the wake region). The smallest grid cell has a size of 0.005 which is found on the surface of the airfoil. The mesh is stretched out quickly, and the largest cells with sizes of about 10 chord lengths lie on the outer boundaries of the domain. The sliding-deforming mesh has a sliding interface at $R_s = 2.0$ measured from the pitching center of the airfoil. The deforming region for both meshes are defined by $R_a = 2.5$ and $R_b = 15.5$. A Dirichlet boundary condition is employed at the inlet. A no-slip isothermal wall boundary condition is applied on the airfoil surface. All other boundaries are set to characteristic non-reflecting boundaries that avoid acoustic waves being reflected back to the domain.

The $\theta_0 = 30^\circ$ case was previously studied using a flux reconstruction solver on deforming-only mesh [23]. We compare the lift
and drag coefficients from the present sliding-deforming approach with the previous results in Fig. 11. As can be seen, the present results are in good agreement with results on deforming-only mesh computed using a different solver. This good agreement is mostly due to the fact that the pitching amplitude is small. When the pitching amplitude gets larger, the deforming-only approach will very likely become unreliable.

To see the advantage of sliding-deforming approach over deforming-only approach, we compare the instantaneous flow fields at the three pitching amplitudes in Fig. 12. It is evident that, for \( \theta_0 = 30^\circ \) there is no obvious difference between the flow fields from the two approaches, and both flow fields are very smooth showing no visible mesh effects. For \( \theta_0 = 45^\circ \), which we consider as a moderate pitching amplitude, vortices from the deforming-only mesh already start showing slightly distorted shapes comparing to the sliding-deforming case. The situation gets much worse for the deforming-only approach for \( \theta_0 = 60^\circ \), vortices have been severely distorted due to large mesh skewness. In contrast, no obvious mesh effects have been observed for the sliding-deforming case at this large pitching amplitude.

The speedup of parallelization has also been tested for this single airfoil case using up to 256 processors. The scalability results are shown in Fig. 13. Linear speedup is well achieved with up to 128 processors. As the number of processors further increases, the number of cells on each processor decreases and interprocessor communication takes a larger portion of the overall time on each processor and thus deteriorates the speedup. It is interesting to notice that the scalability improves as the order of \( S^D \) method (denoted by \( N \)) increases. This is consistent with the fact that the computation cost on each processor is \( O(N^2) \) while the communication cost is \( O(N) \).

4.4. Flow over two tandem oscillating wings

Oscillating wings can be used to extract energy from fluid flows, for example [14,16,30,48]. In this test case, we simulate flow over
two tandem oscillating wings and study the energy extraction performance of this system. Both wings are chosen as NACA0012 airfoil and have the same plunging and pitching motions that can be expressed as,

\[ h(t) = h_0 \cos(2\pi ft), \quad \theta(t) = \theta_0 \sin(2\pi ft), \]

where the plunging amplitude is \( h_0 = 1 \), the pitching amplitude is \( \theta_0 = 75^\circ \), the pitching center is at one-third chord, the reduced frequency is set to \( f = 1/7 \). The chord length of both wings is \( c = 1 \). The incoming free-stream flow has a Mach number of \( Ma_\infty = 0.1 \). The Reynolds number is \( Re = 1000 \) based on the free-stream velocity, viscosity, and the wing chord length.
To measure the power extraction of the wings, several parameters need to be defined. The instantaneous power extraction rate is defined as,
\[ P(t) = P_x(t) + P_y(t), \]
where \( P_x(t) = YH(t) \) is the heaving component, \( P_y(t) = MU(t) \) is the pitching component, \( Y \) is the vertical force, \( U \) is the torque about the pitching center. The instantaneous power extraction coefficient is,
\[ C_P(t) = P(t) \left( \frac{1}{\pi} \rho \infty U^3 \ell \right), \]
where \( \rho \infty \) and \( U \infty \) are the free-stream density and speed, respectively, \( \ell \) is the chord length. The mean power extraction coefficient is defined as,
\[ \bar{C}_P = \frac{1}{T} \int_0^T C_P(t) dt, \]
where \( T = 1/f \) is the wing oscillating period. Finally, the power extraction efficiency can be computed as,
\[ \eta = \frac{\bar{C}_P}{\bar{C}_P_0} \]
where \( d \) is the airfoil’s swept area, which is usually larger than \( 2h_0 \) due to larger trailing or leading edge displacement. For the present set up, we have \( d \approx 2.55 \). The physical meaning of \( \eta \) is the ratio of the mean power extraction rate and the available kinetic energy rate in the wing’s swept area.

Phase averaging is performed on \( C_P \) and its components for 20 oscillating periods after the flow is fully developed. The results have been plotted in Fig. 16. From the curves, it is obvious that \( C_P \) is periodic and has a period of \( T/2 \) for both wings. The value of this period is due to the fact that a wing has the same energy extraction performance in a downstroke process as in an upstroke process. The \( C_P \) for the first airfoil (i.e. \( C_P1 \)) takes positive values most of the time, which indicates that the wing is extracting energy from the flow most of the time. The second wing also extracts energy, however, the performance is not as good. The poor performance of the down-stream wing is partially due to the fact that the flow there contains less energy as the leading airfoil has already extracted some from it. The complex flow environment also plays an important role on the second wing’s performance. It is interesting to notice that, heaving plays a more important role than pitching does. For the first airfoil, pitching does not seem to have an obvious overall contribution. The situation is even worse for the second airfoil, where pitching causes obvious energy loss and degrades the performance of that wing.

The instantaneous flow fields corresponding to the peaks and troughs on the \( C_{P1} \) curve are plotted in Fig. 17, where (a)-(d) are
Fig. 15. An instantaneous flow field for two tandem oscillating wings, visualized by vorticity contours.

Fig. 16. Phase averaged power-extraction coefficient and its components for two tandem oscillating wings.

Fig. 17. Local views of flow fields of two tandem oscillating wings within an oscillating period.
in a downstroke process, (e)–(h) are in an upstroke process. In Fig. 17(a), the airfoils are slowly moving downward and rapidly rotating counterclockwise, a negative and positive vortex pair is being shed off from the trailing edge. The small heaving speed leads to a small \( C_{p1-h} \), while the rapid rotation results in a loss of kinetic energy. The overall effect is a trough on both the \( C_p \) and \( C_{p2} \) curves at this time instant. In Fig. 17(b), the airfoils are quickly moving downward and slowly rotating counterclockwise, which results in a large \( C_{p1-h} \) and a small \( C_{p1-\theta} \), and thus a peak in the \( C_{p1} \) curve. The pair of vortices that shed off from the first airfoil have arrived at the trailing edge of the second airfoil at this moment, and have caused a dipole on the \( C_{p2} \) curve between \( t = 0.177 \) and \( t = 0.227 \). Fig. 17(c) corresponds to a dipole on the \( C_{p1} \) curve, where the airfoil is rotating clockwise while the downward heaving motion is slowing down. The large vortex on the lower surface of the airfoil causes low pressure in that region, and in turn a clockwise torque on the airfoil and positive energy extraction from pitching. From Fig. 17(d)–(d), the rotating speed increases and the low pressure vortex travels further towards the trailing edge along the lower surface of the airfoil, which results in an increasing torque. For this reason, we see an consistent increase of \( C_{p1-\theta} \), a dipole and a small peak on the first half of the \( C_{p1} \) curve. In Fig. 17(e)–(h), a similar process happens as the airfoils travel upward. It is noticed that the first airfoil has somewhat suppressed the vortex formation on the second airfoil. This could also be a factor that has affected the performance of the second airfoil.

Finally, the mean power-extraction coefficients and the efficiencies are reported in Table 3. From \( C_{p,\theta} \) and \( C_{p,\theta} \), it is seen that pitching only contributes fractionally to the performance of the first airfoil, while it causes performance degradation to the second one. These data are consistent with our observation from Fig. 16. The contributions from pressure and viscosity are also given in the same table, and are denoted as \( C_{p,\text{prs}} \) and \( C_{p,\text{vis}} \), respectively. It is seen that viscosity effects are negligible when compared to pressure effects. This indicates that inviscid flow solution is likely to provide a good approximation to this problem. The efficiencies indicate that the first airfoil extracts 28.1% of the available kinetic energy of the free-stream flow within its swept area, and the second airfoil extracts 9.3% of the available free-stream energy. The total efficiency of the system is the sum of these two components, that is, 37.4%. This number is consistent with the vortex distribution in Fig. 15. In Fig. 15, the distance between two successive vortices in the fully developed wake is around 5.5. The time to convet a vortex from its current position to the position of the next vortex in an oscillating period, i.e. 7. The mean flow speed in the wake region can thus be estimated from the vortex distance and the period, which turns out to be around 0.78. This mean flow speed represents a kinetic energy loss of 39%, which is close to the amount of energy the system has extracted.

5. Conclusions

A parallel sliding and deforming spectral difference (SD²) method is successfully created for the 2D compressible Navier–Stokes equations on unstructured quadrilateral grids. By introducing a sliding interface, this method is able to separately deal with translational and rotational motions on a dynamic grid and thus completely remove grid skewness caused by rotating boundaries. The sliding interface is accurately treated using a dynamic curved mortar approach. An algebraic mesh deformation algorithm is employed to deform grid according to prescribed translational motion. The SD² method is high-order accurate for both inviscid and viscous flow. The SD² method is proven to be scalable for even 2D meshes on a distributed memory computer. From the simulation of flow over a plunging and pitching airfoil, it is shown that the SD² method can accurately capture flow structures for large pitching angles while the deforming-only approach can not. The capability of the SD² method has been further demonstrated on two tandem oscillating wings with large plunging and very large pitching amplitudes. The power-extraction analysis show that this two-tandem-wing system can actively harvest energy from the free-stream flow. The SD² method is very suitable for simulation flows that involve boundaries with very large rotating motions.

Acknowledgment

Chunlei Liang is grateful for an Office of Naval Research (ONR) Young Investigator Program Award managed by Dr. Ki-Han Kim with an award number N000141410367.

References

Mohammad
2005;37:239–61
I:
```plaintext
Combustion
```

Persson
2010-0541
```

Linnick
for
```plaintext
Basic
```

Claminar
spectral
```

Wang
2009b
```

DeLaurier
of
```plaintext
M.
```

Ann
```

Liang
1995-671
```

Zhong
1998;30:199–229
```

```


