Advances in Engineering Software 91 (2016) 80–88

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Implementation of density-based implicit LU-SGS solver in the framework of OpenFOAM

Chun Shen [a,b,*], Xin-lin Xia [c], Yong-zhen Wang [b], Feng Yu [b], Zhen-wei Jiao [b]

[a] State Key Laboratory of Automotive Simulation and Control, Jilin University, Changchun, China
[b] Department of Thermal Energy Engineering, Jilin University, Changchun, China
[c] School of Energy Science and Engineering, Harbin Institute of Technology, Harbin, China

ARTICLE INFO

Article history:
Received 14 August 2015
Revised 10 October 2015
Accepted 25 October 2015

Keywords:
OpenFOAM
Density-based
Dual-time
Implicit
LU-SGS
Numerical

ABSTRACT

In this paper, in the framework of OpenFOAM, the density-based steady and unsteady dual-time implicit LU-SGS solvers which are termed as *lusgsFoam* and *lusgsDualFoam* are established. The key implementation details of the forward and backward sweep looping process in the LU-SGS approach are introduced. Three typical test cases, i.e. hypersonic airflow across cylinder(steady), subsonic flow over bump (steady) and supersonic forward step flow (unsteady), are used to investigate the performance of these LU-SGS solvers, and the density-based explicit solver, i.e. the *rhoCentralFoam* in the official OpenFOAM, is adopted as the comparison solver. Through the comparison of the numerical results for the first two steady cases, it is found that, the convergence performance of the LU-SGS solver is significantly superior to explicit solver *rhoCentralFoam*. Meanwhile, considering the combined effect of time step and the iteration efficiency, the LU-SGS solver, *lusgsFoam* is significantly more efficient than the explicit solver, *rhoCentralFoam*. For the third test case, i.e. the unsteady supersonic forward step flow, the spatial accuracy of the unsteady dual time *lusgsDualFoam* solver is a little lower than the explicit *rhoCentralFoam*, but overall, this unsteady implicit solver can capture the transient characteristics efficiently.

1. Introduction

OpenFOAM® is a widely spread CFD (Computational Fluid Dynamics) open source library based on object-oriented C++ language [1,2]. Utilizing its high level programming syntax and related released code details, it is convenient for the researchers to reproduce the algorithm in other references recently published or even to develop their own code according to their new idea [3–15]. OpenFOAM originated from pressure-based velocity-pressure correction mode which is devised to solve the incompressible Navier–Stokes equations [2], e.g. PISO [16,17] and SIMPLE [18,19]. After decades of development the relevant pressure-based solving strategies in OpenFOAM have already been abundant. In contrast to pressure-based mode, the density-based algorithm in the framework of OpenFOAM, which is more proper to solve the high speed (especially for the supersonic/hypersonic) compressible flow problem, appears relatively lately. Until now, compared with the commercial CFD software, e.g. Ansys Fluent [20], Fastran [21], the amount of functions of the density-based solvers in the official or the extended version of OpenFOAM is less and the efficiency of them is relatively lower.

So far, the only density-based solver in the official OpenFOAM is *rhoCentralFoam*, of which the convective discrete scheme is the central-upwind schemes of Kurganov and Tadmor [22,23]. Another popular density-based solver is *densityBasedTurbo* series in the extended version of OpenFOAM, of which the convective discrete schemes are the well known Godunov type schemes, e.g. AUSM family and Roe [24]. Based on the *densityBasedTurbo* solvers, for the all-speed flow, the time-derivative preconditioning method with all-speed convective discrete scheme (e.g. preconditioned Roe, AUSM$^+$up, AUSM$^+$P) was added in Refs. [25–27]. However, the density-based solvers above are all explicit, and so their computational efficiency is much lower than the density-based implicit solvers in the commercial software.

Matrix-free LU-SGS (lower-upper symmetric Gauss–Seidel) implicit scheme is widely used because of its low numerical complexity and modest memory requirements [28–30]. Using the LU-SGS algorithm, a density based solver in OpenFOAM was performed by Kim et al. [31], especially focused on steady state. The authors primarily showed the comparison results reproduced by the developed LU-SGS solver, whereas regretfully, they didn't present sufficient details associated with their program architecture within OpenFOAM. Thereby, for others in the OpenFOAM community, it's difficult to follow and improve the solver further. Until now, there is no other density-based implicit solver released in OpenFOAM or related references to declare

* Corresponding author at: State Key Laboratory of Automotive Simulation and Control, Jilin University, Changchun, China. Tel.: +86 431 85095196.
E-mail address: shench@jlu.edu.cn (C. Shen).

http://dx.doi.org/10.1016/j.advengsoft.2015.10.007
0965-9978/

Nomenclature

A_c	convective Jacobian matrix
$A_c^{\pm}$	positive/negative Steger–Warming flux-splitting Jacobian matrix
dS	surface element
D,L,U	diagonal matrix, the lower-diagonal matrix and the the upper-diagonal matrix
F_c	convective flux vectors
F_v	viscous flux vectors
I	identity matrix
$L(i), U(i)$	sets of "owner" and "neighbor" cells which are adjacent to the cell i in OpenFOAM
$\boldsymbol{n}$	unit normal vector at the interface
$N(i)$	set of cells which are adjacent to cell i
R	residual of complete spatial discretization term for the discretized governing equation
t	physical time(s)
V	contravariant velocity
W	vector of conservative variable

Greek symbols

Ω	control volume
Λ_c	spectral radius of the convective flux Jacobian, A_c
ϕ	L2 norm residual
τ	pseudo time step

Subscripts

c	convective; sound speed
i,j	index of cell
ij	interface between cell i and j
l	time level at the current pseudo time
x, y, z	Cartesian coordinates
-1	inverse

Superscripts

n	time level at the current physical time
v	viscous
$\pm$	symbol indicating direction
$*$	temporary variable

the establishment of systematic density-based implicit solver in the framework of OpenFOAM. And so the other new implicit solver needs to be developed to further enrich the density based-solver library within OpenFOAM. The basic motivation of the present work arises from the lack of the density-based implicit solver in the framework of OpenFOAM.

Meanwhile there are many potential advantages due to its characteristics of open source in the framework of OpenFOAM . The most significant advantage of the solver in OpenFOAM is that a lot of other advanced models e.g. turbulence, can be directly added in. In comparison with other analogous solvers in commercial software, it could be also further improved by other researches due to its characteristics of open source. Moreover, it natively supports the unstructured grid for complex geometry and the parallelization to accelerate the computation process.

In view of lack of the density-based implicit solver in OpenFOAM and the potential advantages of the solver in OpenFOAM, the objective of this paper is to further develop the complete steady and unsteady LU-SGS solver based on the program of Borm et al. [24] and Shen et al. [26], and to show the code architecture of the program properly for futher improvement and perfection by others in the OpenFOAM community. Several test cases are selected to investigate the convergence performance and the computational efficiency of this implicit solver, as the other open source fluid solvers based on OpenFOAM lib [15,23,26,32] or other open source platforms [33–37]. These test cases could cover a wide range of benchmark examples from subsonic to hypersonic and steady to unsteady flow regimes.

2. Physical and mathematical model

2.1. Governing equation

Governing equations in integral form is shown as follows:

$$\int_{\Omega} \frac{\partial \boldsymbol{W}}{\partial t} d\Omega + \oint_{\partial\Omega} (\boldsymbol{F}_c - \boldsymbol{F}_v) dS = 0 \tag{1}$$

where $\boldsymbol{W}$ is the vector of the conservative variables, $\boldsymbol{F}_c$ and $\boldsymbol{F}_v$ are the vectors of the convective fluxes and the viscous fluxes, respectively, and Ω, $\partial\Omega$ and dS denote the control volume, the closed surface related to the particular control volume Ω and a surface element. The specific forms corresponding to the above variables are presented in Refs. [25,27].

2.2. Spatial discretization

By using the finite volume method for each grid cell i, the temporal and spatial discretization of Eq. (1) can be expressed as

$$\left[\frac{\Omega_i}{\Delta t} I + \frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{W}_i}\right] \Delta \boldsymbol{W}_i^n = -\boldsymbol{R}_i^n \tag{2}$$

where $\boldsymbol{R}_i^n$ is the so-called residual-the complete spatial discretization term, the term $\frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{W}_i}$ is referred to as the flux Jacobian, I is identity matrix, and the superscript n denotes time level at the current time t, with

$$\Delta \boldsymbol{W}_i^n = \boldsymbol{W}_i^{n+1} - \boldsymbol{W}_i^n \tag{3}$$

being the solution correction. In this paper, we only consider the contribution of the convective flux Jacobian to the term $\frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{W}_i}$, i.e.

$$\frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{W}_i} \Delta \boldsymbol{W}_i^n = \sum_{j \in N(i)} \left[\left(\frac{\partial \boldsymbol{F}_{c,ij}^{\ n}}{\partial W} \right)_{ij} \Delta S_{ij} \Delta \boldsymbol{W}_{ij}^n \right] = \sum_{j \in N(i)} \left[A_{c,ij} \Delta S_{ij} \Delta \boldsymbol{W}_{ij}^n \right] \tag{4}$$

where A_c is the convective Jacobian matrix, $N(i)$ is the set of cells which are adjacent to cell i. Substituting Eq. (4) for the convective flux Jacobian into Eq. (2), the following form is obtained:

$$\Omega_i \frac{W_i^{n+1} - W_i^n}{\Delta t_i} + \sum_{j \in N(i)} A_{c,ij} \Delta W_{ij}^n S_{ij} = -R_i^n. \tag{5}$$

Then applying the Steger–Warming splitting scheme to discretize implicit operator above

$$A_{c,ij} \Delta W_{ij}^n S_{ij} = \left(A_{c,ij}^+ \Delta W_i^n + A_{c,ij}^- \Delta W_j^n \right) S_{ij} \tag{6}$$

with $A_c^{\pm}$ denoting the positive/negative Steger–Warming flux-splitting Jacobian matrix. Finally, the following discretization form is obtained:

$$\Omega_i \frac{W_i^{n+1} - W_i^n}{\Delta t_i} + \sum_{j \in N(i)} \left(A_{c,ij}^+ \Delta W_i^n + A_{c,ij}^- \Delta W_j^n \right) S_{ij} = -R_i^n. \tag{7}$$

2.3. Temporal discretization for LU-SGS scheme

According to the original theory of LU-SGS scheme, the implicit operator should be divided into the diagonal matrix D, the lower-diagonal matrix L and the the upper-diagonal matrix U. For this purpose, Eq. (7) could be deduced further as follows:

$$\Omega_i \frac{\Delta W_i^n}{\Delta t_i} - \sum_{j\in L(i)} \left(A_{c,i}^- S_{ij}\right)\Delta W_i^n + \sum_{j\in U(i)} \left(A_{c,i}^+ S_{ij}\right)\Delta W_i^n - \sum_{j\in L(i)} \left(A_{c,j}^+ S_{ij}\right)\Delta W_j^n + \sum_{j\in U(i)} \left(A_{c,j}^- S_{ij}\right)\Delta W_j^n = -R_i^n. \quad (8)$$

The split convective flux Jacobian could be evaluated as follows:

$$A^{\pm}\Delta S_{ij} = \frac{1}{2}\left(A_c S_{ij} \pm \Lambda_c I\right)$$
$$\Lambda_c = (|V| + c)S_{ij} \quad (9)$$

where the contravariant velocity $V = \boldsymbol{U} \bullet \boldsymbol{n}_{ij} = n_x u + n_y v + n_z w$, with the unit normal vector $\boldsymbol{n}_{ij} = n_x \boldsymbol{i} + n_y \boldsymbol{j} + n_z \boldsymbol{k}$ corresponding to the face area vector $\boldsymbol{S}_{ij}$, and Λ_c represents the spectral radius of the convective flux Jacobian A_c. Then substituting Eq. (9) into Eq. (8), the diagonal matrix D could be written as

$$D_i = \frac{\Omega_i}{\Delta t_i} I + \sum_{j\in(L(i)+U(i))} \frac{\omega}{2}\Lambda_{c,ij}. \quad (10)$$

Meanwhile, $L_i = -\sum_{j\in L(i)} (A_{c,j}^+ S_{ij})$, with $L(i)$ being the set of "owner" cells which are adjacent to the discrete element i(*cellI*), i.e. $j \in L(i)$; $U_i = \sum_{j\in U(i)} (A_{c,j}^- S_{ij})$, with $U(i)$ being the set of "neighbor" cells which are adjacent to the discrete element i(*cellI*), i.e. $j \in U(i)$.

In OpenFOAM, the superscript "+" and "−" denote the positive and the negative direction corresponding to the face normal vector $\boldsymbol{n}_{ij} = n_x \boldsymbol{i} + n_y \boldsymbol{j} + n_z \boldsymbol{k}$. In other words, for each interface ij, the positive variable is directed from owner cell to neighbor cell and the negative variable has the reverse direction. Specially, it is noted that, if $j \in L(i)$, $\boldsymbol{S}_{ij}$ is the inward-pointing face area vector for cell i, so it means that, according to the Gauss theorem, the sign of the second and forth implicit operator term in the left hand of Eq. (8) should be negative.

Finally, Eq. (8) could be transformed into

$$(D + L)D^{-1}(D + U)\Delta W^n = -R^n. \quad (11)$$

The solving process is divided into two steps: forward sweep and backward sweep, where forward and backward sweep processes are directly solving Eqs. (12) and(13), respectively,

$$(D + L)\Delta W^* = -R^n, \quad (12)$$
$$(D + U)\Delta W^n = D\Delta W^*. \quad (13)$$

2.4. Dual time-stepping approach

Dual time-stepping approach is very often employed for the unsteady flow. In this approach, pseudo-time term, i.e. $\Omega_i \frac{\Delta W_i^*}{\Delta \tau}$, should be added in as follows:

$$\Omega_i \frac{\Delta W_i^l}{\Delta \tau} + \Omega_i \frac{3\Delta W_i^l + 3W_i^l - 4W_i^n + W_i^{n-1}}{2\Delta t} + \sum_{j\in N(i)} \left(F_{c,ij}^{l+1} - F_{v,ij}^l\right)S_{ij} = 0 \quad (14)$$

with

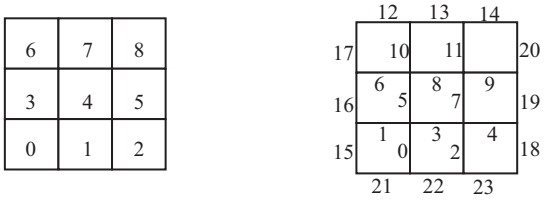
$$\Delta W_i^l = W_i^{l+1} - W_i^l \quad (15)$$

where superscript l denotes the current pseudo-time level. Similar as the discretization Eq. (2) for the physical time implicit scheme, Eq. (16) is obtained:

$$\left[\Omega_i\left(\frac{1}{\Delta\tau} + \frac{3}{2\Delta t}\right) + \frac{\partial R_i^l}{\partial W_i^l}\right]\Delta W_i^l = -(R_i^*)^l \quad (16)$$

where the residual term for dual time-stepping approach is

$$\left(R_i^*\right)^l = R_i^l + \left(\Omega_i \frac{3W_i^l - 4W_i^n + W_i^{n-1}}{2\Delta t}\right). \quad (17)$$



(a) cell index i(*cellI*) (b) face index (*faceI*)

Fig. 1. Cell and face index number of 3 − 3 mesh (2-D structured) in OpenFOAM.

Table 1
Mapping relationship between internal face *faceI* and its adjacent owner cell *cellI*.

	Effective index for LU-SGS scheme loop											Ineffective index			
faceI	0	1	2	3	4	5	6	7	8	9	10	11	**12**	**13**	**14**	...
cellI	0	0	1	1	2	3	3	4	4	5	6	7	**6**	**7**	**8**	...

Table 2
Mapping relationship between internal face *faceI* and its adjacent neighbor cell *cellI*.

	Effective index for LU-SGS scheme loop											
faceI	0	1	2	3	4	5	6	7	8	9	10	11
cellI	1	3	2	4	5	4	6	5	7	8	7	8

The discretization procedure to solve Eq. (16) is similar as that shown in Section 2.3.

3. Implementation of LU-SGS scheme in OpenFOAM

For the LU-SGS scheme, the sweeping process could be conveniently implemented. Eqs. (12) and (13) are transformed into the following form:

$$\Delta W^* = D^{-1}(-R^n - L\Delta W^*), \quad (18)$$
$$\Delta W = \Delta W^* - D^{-1}(U\Delta W). \quad (19)$$

As mentioned above, Eq. (18) is related to the forward sweep that $L\Delta W^*$ is known in advance, and in OpenFOAM, it is equivalent to that, corresponding to the cell i(*cellI*) to be solved, the solutions of the variable at its adjacent owner cells are already known. Eq. (19) is related to the backward sweep that $U\Delta W$ is known beforehand, and in OpenFOAM, the solutions of the variable at its adjacent neighbor cells are known in advance.

Subsequently, the detail of the implementation of LU-SGS scheme in OpenFOAM is presented, taken the simple 2-D structured 3 × 3 mesh system, shown in Fig. 1. Fig. 1(a) shows the cell index i or *cellI*, and Fig. 1(b) shows the face serial index number *faceI*. Tables 1 and 2 present mapping relationship between the internal face *faceI* (not

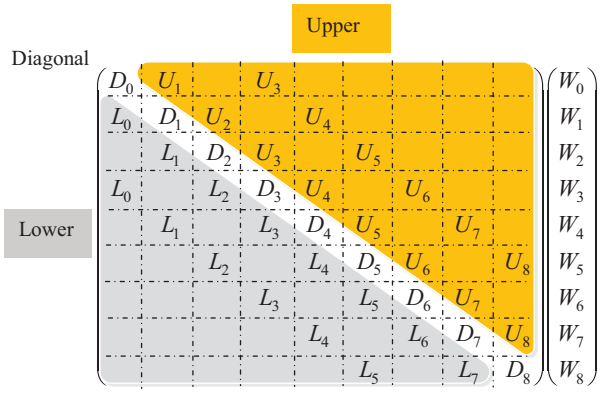


Fig. 2. Implicit operator matrix for 3 − 3 mesh (2-D structured) in OpenFOAM.

Algorithm 1
Extract list of owner and neighbor cell index.

```
.....
// Extract lists of owner and neighbor cell index related to cell cellI in Tables 1 and 2.
    const unallocLabelList & owner = mesh.owner();
    const unallocLabelList & neighbour = mesh.neighbour();
      const surfaceVectorField & Sf = mesh.Sf();          //face area vector
      const surfaceScalarField & magSf = mesh.magSf();//face area
.....
```

Algorithm 2
Definition of Lower matrix, and performance of forward sweep.

```
// define Lower matrix, and perform forward sweep.
forAll(mesh.cells(), cellI)
{
    ....................
    const labelList& cellFaces = mesh.cells()[cellI];      //obtain list of face index enclosed cell cellI.
    forAll (cellFaces, i)
    {
        label faceI = cellFaces[i];      //obtain face index faceI related to the i-th member in the list cellFaces.
        if(mesh.isInternalFace(faceI))      //judge whether the face faceI is internal face or not.
        {
            label own = owner[faceI];      //obtain the indexes of the owner and neighbor cell related to face faceI.
            label nei = neighbour[faceI];
            if(own! = cellI)                  //Judge the relationship between owner cell and cell cellI. If owner cell is not cell cellI, proceed to following calucation.
            {
                // Calculate the diagonal matrix D.
                D11[cellI]+ = 0.5*ac_nei[faceI]*(mag(Sf[faceI]));
                D22[cellI] + = 0.5*ac_nei[faceI]*(mag(Sf[faceI]));
                .........
                // Calculate the lower matrix L.
                L11.value() = ...; L12.value() = ...;      .....
                L21.value() = ...; L22.value() = ...;      .....
                .....
                // Calculate the operator LΔW*.
                R1_tem[cellI) = ...;
                R2_tem[cellI) = ...;
                ...
                R5_tem[cellI) = ...;
            }
            if(nei!=cellI)                  // Judge the relationship between neighbor cell and cell cellI. If neighbor cell is not cell cellI, proceed to following calucation.
            {
                // Calculate the diagonal matrix D.
                D11[cellI] + =0.5*ac_own[faceI]*(mag(Sf[faceI]));
                D22[cellI] + = 0.5*ac_own[faceI]*(mag(Sf[faceI]));
                ...
            }
        }
    }
    ...
    // Calcualte D^-1, i.e. the inversion of diagonal matrix D.
    c.inv ();
    Dinv11.value() = ...
    .....
    Dinv55.value() = ...;
    .....
    //Calculate the operator (−R^n − LΔW*) (note: R*Volume)
    R1_tem[cellI] = − R1_tem[cellI] + R1[cellI]*mesh.V()[cellI];
    ...
    R5_tem[cellI] = −R5_tem[cellI] + R5[cellI]*mesh.V()[cellI];
    // Calculate the operator ΔW* = D^-1(−R^n − LΔW*)
    delta_W_star1[cellI] = ...;
    ...
    delta_W_star5[cellI] = ...;
...
}
...
```

including the boundary faces) and its adjacent owner and neighbor cells.

Fig. 2 shows the implicit operator matrix (diagonal matrix D, the lower-diagonal matrix L and the upper-diagonal matrix U) for the 3×3 mesh in Fig. 1.

Algorithms 1–3 show some details corresponding to the LU-SGS scheme in OpenFOAM in brief. Firstly, the lists of owner and neighbor cell index, i.e. *cellI* lists in Tables 1 and 2, should be obtained, as shown in Algorithm 1. These two lists are the basis to get cells adjacent to each cell *cellI*.

Algorithm 2 shows the key ingredient of the LU-SGS scheme looping in OpenFOAM, i.e. definition of the Lower matrix, and performance of the forward sweep.

The process corresponding to the backward sweep is similar to the forward sweep as shown in Algorithm 2. It is noted that the reverse cell looping command *forAllReverse*

Algorithm 3
Definition of upper matrix, and performance of backward sweep.

```
.....
    //define Upper matrix, and perform backward sweep
    forAllReverse(mesh.cells(),cellI)
    {
        // The process is similar as the forward sweep.
        .....
    }
.....
```

substitutes the positive cell looping command *forAll,* as shown in Algorithm 3.

Fig. 3 shows the block diagram of forward sweeping process in the framework of OpenFOAM.

3. Results and discussion

3.1. Case 1: hypersonic airflow across cylinder

Hypersonic airflow across blunt body, e.g. blunt cone and cylinder, is a typical test case to investigate the performance of the numerical methods to capture the shock discontinuity in the hypersonic flow regime [23,38,39]. In this paper, Hypersonic airflow across cylinder is employed, and the inlet velocity of the hypersonic airflow stream is 6.47Ma, and other computational conditions, e.g. the geometric sizes are the same as that presented in Ref. [23]. It is noted that the independence of grid size was also discussed in Ref. [23]. The numerical results produced by the LU-SGS solver, which called *lusgsFoam* solver in the following context, are compared with the corresponding results generated by the *rhoCentralFoam* solver in the

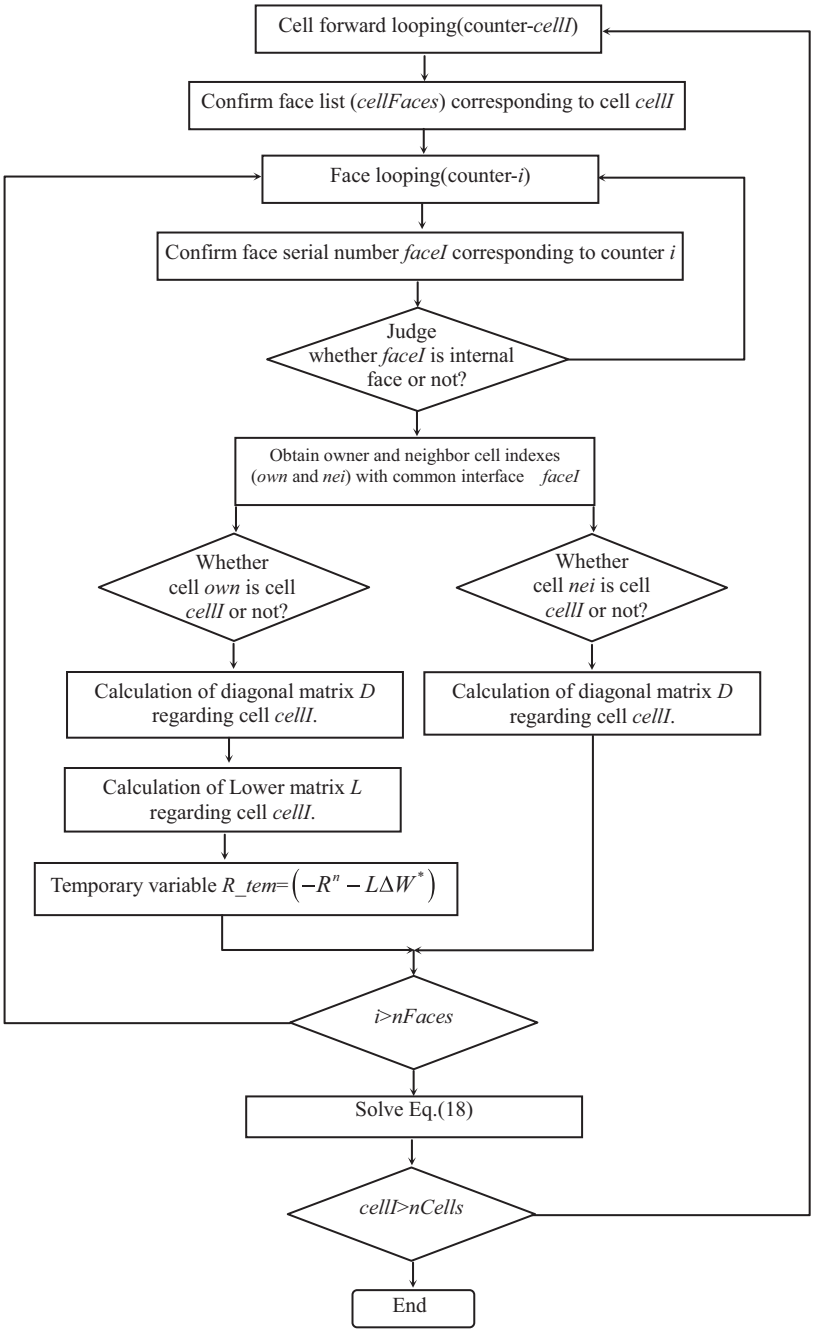


Fig. 3. Block diagram of forward sweeping process in OpenFOAM.

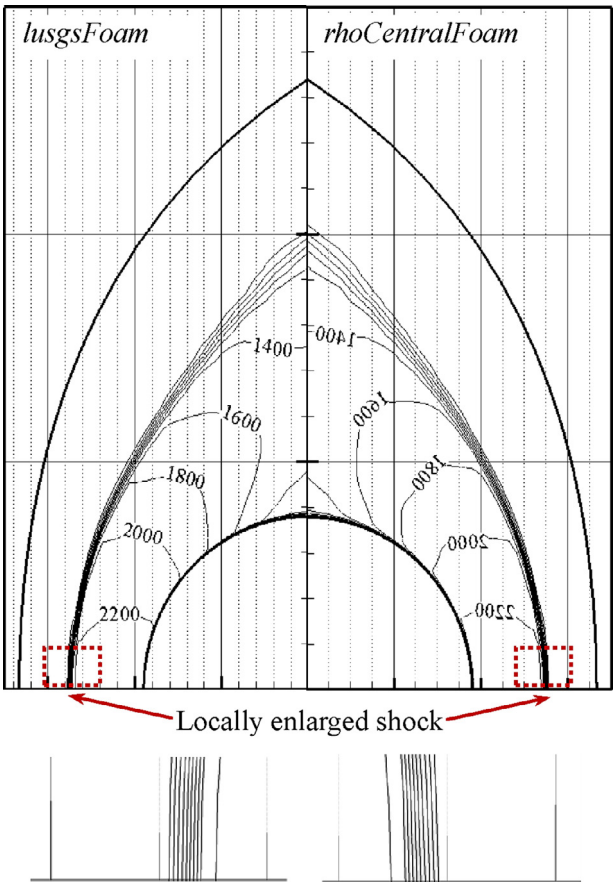


Fig. 4. Comparison of temperature contour lines for Case 1.

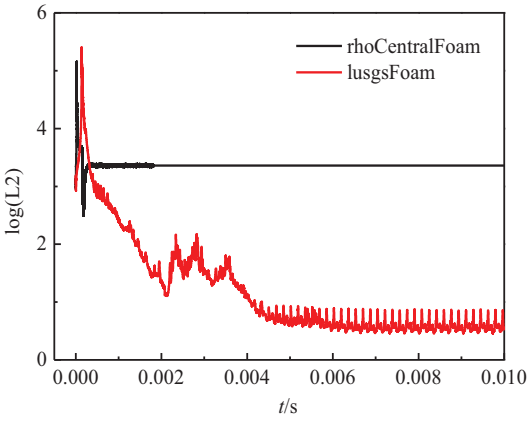


Fig. 5. Comparison of convergence history for Case 1.

official OpenFOAM platform. For these two solvers, the one-dimension reconstruction scheme with minmod TVD limiter is utilized to enhance the spatial accuracy to second order [19,26,28].

Fig. 4 shows the comparison of temperature contour lines predicted by the two solvers mentioned above. The locations of corresponding contour lines, especially for the locations at shocks, are

Table 3
Consumption of computational execution time for Case 1.

Physical time ($t/10^{-7}$ s)	Execution time (unit:s)					
	1	2	3	4	5	Average
lusgsFoam	0.73	1.23	1.72	2.22	2.71	0.542
rhoCentralFoam	3.06	5.91	8.8	11.72	14.55	2.91

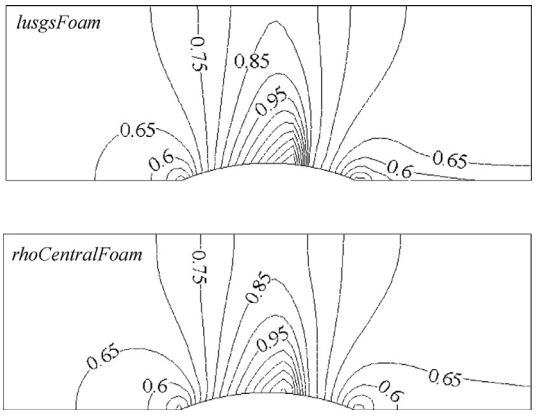


Fig. 6. Comparison of Mach contour lines for Case 2.

Table 4
Consumption of computational execution time for Case 2.

Physical time ($t/10^{-3}$ s)	Execution time (unit: s)					
	1	2	3	4	5	Average
lusgsFoam	0.28	0.5	0.66	0.83	1.01	0.206
rhoCentralFoam	1.09	2.11	3.13	4.16	5.22	1.104

nearly coincident with each other. The spacings of the shocks predicted by these two solvers are almost uniform.

Except the spatial accuracy, the convergence history and the computational efficiency regarding the execution time, i.e. CPU time, are two additional key points to be discussed for the numerical solver. In this paper, L2 norm residual ϕ is chosen as the convergence criterion, which is shown as follows:

$$\phi = \sqrt{\sum_{cellI} |(\nabla \cdot (\rho \boldsymbol{U}))_{cellI}|^2} = \sqrt{\sum_{cellI} \left| \sum_{facel}^{cellI} \left(\boldsymbol{S} \cdot (\rho \boldsymbol{U})_{facel} \right) \right|^2}. \quad (20)$$

The convergence histories in terms of physical time t are shown in Fig. 5. For the *lusgsFoam*, the magnitude of the residual is nearly converged at the value 0.5, and it is much smaller than the converged value (larger than 3) produced by the *rhoCentralFoam* solver.

In this case, the computational time steps $\Delta t = 10^{-7}$ s and 0.5×10^{-8}s are utilized for the *lusgsFoam* and *rhoCentralFoam* solvers, respectively, and these two time spans are the maximum allowable time steps for these two solvers, respectively. The consumptions of the execution time at the same physical moment t are shown at Table 3. Considering the combined effect of physical time step and the iteration efficiency, the consumption of the CPU execution time

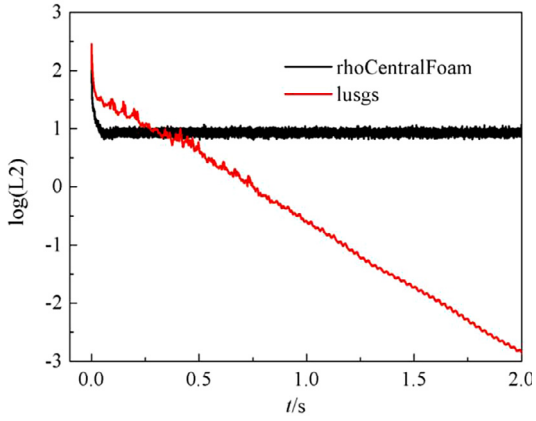
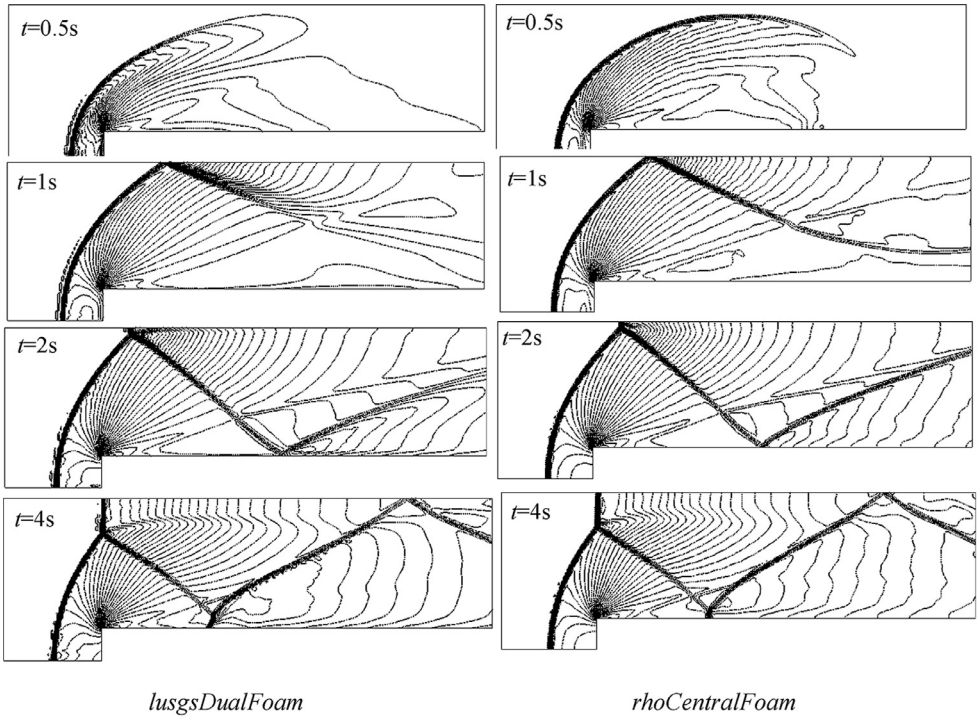
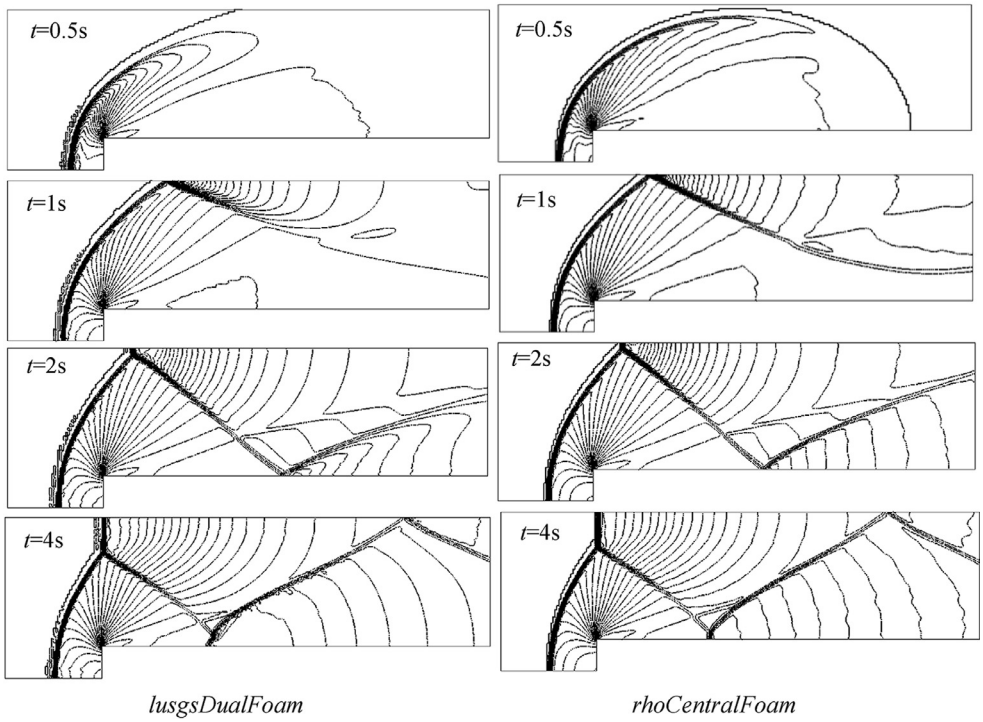


Fig. 7. Comparison of convergence history for Case 2.



(a) density



(b) pressure

Fig. 8. Comparison of density and pressure contour lines for Case 3.

corresponding to the *lusgsFoam* is less than one fifth of that cost by the *rhoCentralFoam*.

3.2. Case 2: subsonic flow over bump

The subsonic flow over bump is another frequently used as compressible steady test case [26]. In this paper, the inlet Mach number is 0.675. In order to enhance the spatial accuracy to second order, the multi-dimension reconstruction scheme with Venkatakrishnan limiter [24,26,28] is adopted for the *lusgsFoam* solver, and one-dimension reconstruction scheme with minmod limiter function is adopted for the *rhoCentralFoam* solver. As shown in Fig. 6, the difference of the contour lines predicted by these two solvers is significant. Some contour lines produced by the *rhoCentralFoam* can't expand enough as the corresponding lines generated by *lusgsFoam*, especially for the lines regarding the magnitude Ma = 0.6 and 0.85. However, the area

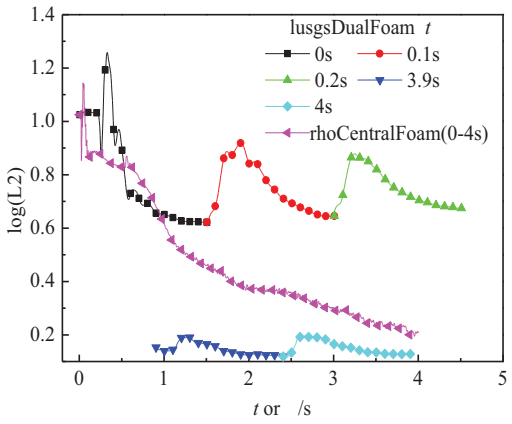


Fig. 9. Comparison of convergence history for Case 3 (t for rhoCentralFoam; τ for lusgsDualFoam).

enclosed by other contour lines, e.g. Ma = 0.65 ahead of and behind the bump, are a little larger that corresponding lines predicted by the *lusgsFoam* solver.

The convergence history in terms of physical time t is shown in Fig. 7. As we can observe, the convergence residual line which is produced by the *lusgsFoam* with multi-dimensional interface reconstruction scheme continuously decline rapidly, while the corresponding line generated by *rhoCentralFoam* with vanleer limiter oscillated up and down at the value of 1.

In this case, the computational step for the *lusgsFoam* is 2×10^{-4} s, while the computational step for the *rhoCentralFoam* is 2×10^{-5} s. Due to neglecting the derivative of the Jacobian matrix during the flux splitting process, the CFL number should be restricted, so the time step 2×10^{-4} s is not the largest time step for this *lusgsFoam* with multi-dimensional reconstruction scheme [28], while 2×10^{-5} s is the largest time step for the *rhoCentralFoam* with minmod limiter. The consumption of the execution time at the same physical time t is shown at Table 4. Considering the combined effect of time step and the iteration efficiency, the consumption of the execution time corresponding to the *lusgsFoam* is still less than one fifth of that cost by the *rhoCentralFoam*, similar as Case 1.

3.3. Case 3: supersonic forward step flow for unsteady problem

The performance of iteration efficiency of lusgsDualFoam solver is nearly the same as lusgsFoam. Employing Case 3, it is intended to indicate that the lusgsDualFoam solver can capture the unsteady characteristics for the transient flow problem. As for its entire efficiency (summing up both inner iteration-pseudo time stepping and out iteration-physical time stepping), it depends the physical time step Δt and the inner iteration number. If Δt and the inner iteration number are larger, the entire computational efficiency will be higher, and the accuracy will be lower. Herein smaller physical time step $\Delta t = 0.1$ s is adopted to ensure the accuracy.

The subsonic flow over bump is selected to evaluate the performance of dual-time *lusgsDualFoam* solver for the unsteady flow. Fig. 8(a) presents comparison of density and pressure contour lines produced by the *lusgsFoam* and *rhoCentralFoam* solvers both with vanleer limiter. As one can see in Fig. 8(a), at $t = 0.5$ s, the contour lines generated by *lusgsDualFoam* solver are less sharp than those predicted by *rhoCentralFoam*. There are existing pre-shock ahead of the shock in front of the forward step, and the pre-shock is gradually vanished with the physical time proceeding. Meanwhile, at $t = 4$ s, downstream of the shock reflected from the step, the oscillation is produced by the *lusgsDualFoam* solver. Compared with the data from Ref. [26], the pre-shock and the oscillation phenomena are a little weaker than the results produced by the original *densityBasedTurb* solvers. Fig. 8(b) presents comparison of pressure contour lines. The numerical phenomena shown in Fig. 8(b) are similar as that presented in Fig. 8(a), and the pre-shock is much clearer.

It should be noted that, in Fig. 9, the meanings of x-axis are different. For lusgsDualFoam solver, x-axis represents the pseudo-time, and for rhoCentralFoam solver, the x-axis represents physical time. For the sake of clarity, only the convergence residual curves at five physical time step ($t = 0$, 0.1, 0.2, 3.9 and 4 s) corresponding to the lusgsDualFoam are shown in Fig. 9. For the convenience of comparison, wherein the curves corresponding to $t = 3.9$ and 4 s are translated forward and the tail of curve corresponding to $t = 4$ s is aligned with the tail of curve corresponding to rhoCentralFoam solver.

As shown in Fig. 9, different from the previous two test cases, at the terminal time $t = 4$ s, the residuals produced by the *lusgsDualFoam* and *rhoCentralFoam* both are nearly the same, i.e. nearly both converging to the value of 0.2. For the previous two cases, it is required that the computational process must proceed to the steady state that the convergence residuals should be small enough. For this forward step case, due to the unsteady feature, there is no need to guarantee that the residual is small sufficiently. That's why the residuals generated by these two solvers are nearly converged to the same value.

4. Conclusion

In this paper, in the framework of OpenFOAM, the density based steady and unsteady dual-time implicit LU-SGS solvers which are termed as *lusgsFoam* and *lusgsDualFoam* are established. The key implementation details of the forward and backward sweep looping process in the LU-SGS approach are introduced. Three typical test cases, i.e. hypersonic airflow across cylinder (steady), subsonic flow over bump (steady) and supersonic forward step flow (unsteady), are used to investigate the performance of this LU-SGS solver, and the density based explicit solver, i.e. the *rhoCentralFoam* in the official OpenFOAM, is adopted as the comparison solver.

Through the comparison of the numerical results for the first two steady cases, it is found that, the convergence performance of the LU-SGS solver, *lusgsFoam* is significantly superior to explicit solver *rhoCentralFoam*. Meanwhile, considering the combined effect of time step and the iteration efficiency, the consumption of the execution time corresponding to the *lusgsFoam* solver is still less than one fifth of that cost by the *rhoCentralFoam* solver, and in other words, the LU-SGS solver, *lusgsFoam* has much higher computational efficiency than the explicit solver, *rhoCentralFoam*.

For the third test case, i.e. the unsteady supersonic forward step flow, the spatial accuracy of the unsteady dual-time *lusgsDualFoam* solver is a little lower than the explicit *rhoCentralFoam*, but overall, this unsteady implicit solver can capture the transient characteristics efficiently.

The most significant advantage of the density-based implicit solver in OpenFOAM is that a lot of other advanced models e.g. turbulence, can be directly added in. In comparison with other analogous solvers in commercial software, it could be also further improved by other researches due to its characteristics of open source. Moreover, it natively supports the unstructured grid for complex geometry and the parallelization to accelerate the computation process. However, until now, it needs much more test cases to verify and does not support the multigrid method, and this is the work that the authors should do in the future.

Acknowledgment

This study was supported by the National Natural Science Foundation of China (Grant no. 51176038 and Grant no. 51309115), and the

Special Fund for Agroscientific Research in the Public Interest (Grant no. 201503135).

References

[1] OpenCFD Ltd.. User and programmer's guide. OpenCFD Ltd.; 2012. http://www.openfoam.com .
[2] Weller HG, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 1998;12:620–31.
[3] Vuorinen V, Chaudhari A, Keskinen JP. Large-eddy simulation in a complex hill terrain enabled by a compact fractional step OpenFOAM® solver. Advances in Engineering Software 2015;79:70–80.
[4] Vuorinen V, Keskinen JP, Duwig C, Boersma B. On the implementation of low dissipative Runge–Kutta projection methods for time dependent flows using OpenFOAM®. Comp Fluids 2014;93:153–63.
[5] Horguea P, Soulaine C, Franc J, Guibert R, Debenest G. An open-source toolbox for multiphase flow in porous media. Comput Phys Commun 2015;187:217–26.
[6] Novaresio V, García-Camprubí M, Izquierdo S, Asinari P, Fueyo N. An open-source library for the numerical modeling of mass-transfer in solid oxide fuel cells. Comput Phys Commun 2012;183:125–46.
[7] Cosden IA, Lukes JR. A hybrid atomistic–continuum model for fluid flow using LAMMPS and OpenFOAM. Comput Phys Commun 2013;184:1958–65.
[8] Le NTP, White C, Reese JM, Reese AM, Myong RS. Langmuir–Maxwell and Langmuir–Smoluchowski boundary conditions for thermal gas flow simulations in hypersonic aerodynamics. Int J Heat Mass Transf 2012;55:5032–43.
[9] Su JW, Gu ZL, Xu XY. Discrete element simulation of particle flow in arbitrarily complex geometries. Chem Eng Sci 2011;66:6069–88.
[10] Tukovic Z, Jasak H. A moving mesh finite volume interface tracking method for surface tension dominated interfacial fluid flow. Comput Fluids 2012;55:70–84.
[11] Flores F, Garreaud R, Munoz RC. CFD simulations of turbulent buoyant atmospheric flows over complex geometry: solver development in OpenFOAM. Comput Fluids 2012;80:408–22.
[12] Lysenko D, Ertesvg I, Rian K. Modeling of turbulent separated flows using OpenFOAM. Comput Fluids 2012;80:1–13.
[13] Kong B, Olse MG, Fox RO, Hill JC. Predictive capability of large eddy simulation for point-wise and spatial turbulence statistics in a confined rectangular jet. Chem Eng Sci 2012;69:240–56.
[14] Franke J, Sturm M, Kalmbach C. Validation of OpenFOAM 1.6.x with the German VDI guideline for obstacle resolving micro-scale models. J Wind Eng Ind Aerodyn 2012;104-106:350–9.
[15] Komen E, Shams A, Camilo L, Koren B. Quasi-DNS capabilities of OpenFOAM for different mesh types. Comput Fluids 2014;96:87–104.
[16] Issa RI. Solution of the implicitly discretized fluid flow equations by operator splitting. J Comput Phys 1985;62:40–65.
[17] Jasak H. Error analysis and estimation for the finite volume method with applications to fluid flows [Ph.D. dissertation]. Imperial College of Science, Technology and Medicine; 1996.
[18] Patankar SV. Numerical heat transfer and fluid flow. New York: McGraw-Hill; 1980.
[19] Ferziger JH, Peric M. Computational methods for fluid dynamics. Berlin\New York: Springer Verlag; 1995.
[20] Fastran, http://www.esi-cfd.com, 2015.
[21] Ansys, http://www.ansys.com/, 2015
[22] Greenshields CJ, Weller HG, Gasparini L, Reese JM. Implementation of semi-discrete, non-staggered central schemes in a collocated, polyhedral, finite volume framework, for high-speed viscous flows. Int J Numer Methods Fluid 2010;63:1–21.
[23] Shen C, Sun FX, Xia XL. Analysis on capabilities of density-based solvers within OpenFOAM to distinguish aerothermal variables in diffusion boundary layer. Chin J Aeronaut 2013;26(6):1370–9.
[24] Borm O, Jemcov A, Kau HP. Density based Navier–Stokes solver for transonic flows. In: Proceedings of the 6th OpenFOAM Workshop, USA. PennState University; 2011.
[25] Saegeler S.F., Mundt C., Advanced numerical simulation of mixing hot core and cold bypass flow in modern propulsion systems with internal lobed forced mixer, AIAA Paper AIAA-2013-2424.
[26] Shen C, Sun FX, Xia XL. Implementation of density-based solver for all speeds in the framework of OpenFOAM. Comput Phys Commun 2014;185:2730–41.
[27] Heinrich M, Schwarze R. All-Mach number density based flow solver for OpenFOAM. In: Proceedings of the ASME turbo expo 2014: turbine technical conference and exposition, vol. 2B: turbomachinery, Düsseldorf, Germany; June 16–20; 2014 [ASME Paper No. GT2014-26220].
[28] Blazek J. Computational fluid dynamics: principles and applications. second ed. Elsevier; 2005.
[29] Wang G, Jiang YW, Ye ZY. An improved LU-SGS implicit scheme for high Reynolds number flow computations on hybrid unstructured mesh. Chin J Aeronaut 2012;25(1):33–41.
[30] Kitamura K, Shima E, Fujimoto K, Wang ZJ. Performance of low-dissipation Euler fluxes and preconditioned LU-SGS at low speeds. Commun Comput Phys 2011;10(1):90–119.
[31] Kim J, Kim K. A development and Verification of density based solver using LU-SGS algorithm in OpenFOAM. In: Proceedings of the 29th congress of the International Council of the Aeronautical Sciences (ICAS 2014), St. Petersburg, Russia; September 7-12, 2014 [Paper No. ICAS2014_0128].
[32] Scanlon TJ, Roohi E, White C, Darbandi M, Reese JM. An open source, parallel DSMC code for rarefied gas flows in arbitrary geometries. Comput Fluids 2010;39:2078–89.
[33] Li XL, Fu DX, Ma YW, Liang X. Direct numerical simulation of compressible turbulent flows. Acta Mech Sin 2010;26:795–806.
[34] Li XL. Direct numerical simulation techniques for hypersonic turbulent flows. Acta Aeronaut Astronaut Sin 2015;36(1):147–58 [In Chinese].
[35] Leng Y, Li XL, Fu DX, Ma YW. The effect of numerical schemes in the frame of finite volume method. Phys Gases: Theory Appl 2012;7(1):33–41 [In Chinese[.
[36] Li XL, Dexun F, Ma YW. Direct numerical simulation of hypersonic boundary layer transition over a blunt cone with a small angle of attack. Phys Fluids 2010;22:025105.
[37] Li XL, Dexun F, Ma YW. Direct numerical simulation of hypersonic boundary-layer transition over a blunt cone. AIAA J 2008;46(11):2899–913.
[38] Joon H.L., Oh H.R. Accuracy of AUSM+ scheme in hypersonic blunt body flow calculations, AIAA Paper AIAA-98-1538; 1998.
[39] Rashidi MM, Bastani MM, Beg OA. Numerical simulation of axisymmetric supersonic viscous flow over a blunt cone with a diagonal fourth-order finite difference method. Proc Inst Mech Eng Part G: J Aerosp Eng 2012;226(3):310–26.