The Implementation of an Implicit LU-SGS Scheme Steady Solver Based on OpenFOAM

Shuai Ye, WenJing Yang*, XinHai Xu, Hao Li, ZhiPeng Lin
College of Computer, National University of Defense Technology
State Key Laboratory of High Performance Computing
Changsha, Hunan, China
e-mail: shuaiye09@163.com, wjyang1088@163.com

***Abstract*—OpenFOAM is a popular open source numerical simulation CFD software. The density-based solvers released by the official are all explicit schemes. LU-SGS is an implicit scheme, LTS is a time marching method that can accelerate the convergence of a steady state problem. In this paper, a density-based implicit solver using LU-SGS scheme and LTS for steady state compressible problems based on OpenFOAM is implemented. The detail of the implementation of the solver is introduced. Oblique shock in OpenFOAM and transonic inviscid compressible flow pass through the NACA0012 airfoil are tested. The comparison of the results has shown a better convergence performance using the LU-SGS-LTS solver.**

Keywords-component; CFD; OpenFOAM; LU-SGS ; LTS; implicit

I. INTRODUCTION

CFD is a branch of fluid mechanics using numerical simulation to solve and analyze problems that involve fluid flows [1], [2]. There are a variety of CFD softwares, including some famous commercial software such as Fluent, CFX, Star-cd, etc [3], and some open source software [4]. OpenFOAM is a widely used framework of the open source C++ CFD toolbox for flexible engineering simulation in CFD. Users can develop their solvers based on OpenFOAM and extend it [5]–[7]. OpenFOAM is very friendly to users and easy to use since it supports DSL (Domain Specific Language). More and more functions are supported by OpenFOAM after decays of development.

Explicit and implicit scheme are two types of schemes that are usually used in time discretization in CFD. In general, the time step of the implicit scheme can be much larger than that of the explicit scheme. LU-SGS is an implicit scheme which has been widely used. Seokkwan yoon and Chen,R.F use the scheme early in reference [8], [9]. The results in their papers shows a faster convergence and smaller residual compared with the explicit scheme.

In the case of numerical simulations of steady state problems, a fixed time step or a dynamic adjustable time step is generally used to advance the time step. LTS (Local Time Step) is a method to accelerate the convergence of the steady state problem [10], [11]. The time step of each cell is calculated separately, it can reduce the number of iterations compared with the fixed time step and the dynamic adjustable time step.

Density based solver and pressure based solver are two types of solvers in OpenFOAM. RhoCentralFoam is a density-based explicit solver for compressible flow in OpenFOAM. It uses the central-upwind schemes of Kruganov and Tadmor in the convective discrete scheme to capture the shocks in the simulations [12]. No implicit density based solvers has yet been released by the official OpenFOAM currently. Kim et al. have developed and validated the density-based LU-SGS implicit solver in the paper [13]. Shen chun et al. have developed a density based LU-SGS implicit solver in 2016 [14]. They have developed lusgsFoam for the steady state and lusgsDualFoam for unsteady state. It is concluded in the paper that lusgsFoam has a better simulation efficiency than rhoCentralFoam. However, LTS is not used in lusgsFoam to accelerate the convergence of steady-state problems.

In this paper, we have developed an LU-SGS implicit solver for steady-state problems that uses LTS based on the OpenFOAM, which is called LU-SGS-LTS. Details of the implementation are introduced in this paper. Two steady state cases are tested to illustrate the effectiveness of LU-SGS-LTS, which can reduce the number of iterations to speed up the problem solving. The main contributions of this paper are:

- An implicit LU-SGS algorithm is designed based on OpenFOAM. Data structures in the LU-SGS algorithm is designed combined with the features of OpenFOAM.
- An implicit LU-SGS-LTS solver for steady state problem is implemented in OpenFOAM. Details of the implementation are presented.
- The correctness and effectiveness of the LU-SGS-LTS solver are verified. Results of LU-SGS-LTS are compared with that of rhoCentralFoam. Additionally, the results of LU-SGS using fixed time step, adjustable time step and LTS are compared to verify the effectiveness of LTS.

The rest of the paper is organized as follows: Section II introduces some related work and background, including LU-SGS implicit scheme and LTS time marching method. Section III introduces the implementation of LU-SGS-LTS in OpenFOAM, followed by the experiments in section IV, which shows the efficiency of our method. Finally, Section V concludes the paper.

978-1-5090-6352-9/17/$31.00 ©2017 IEEE

II. BACKGROUND

A. Introduction to OpenFOAM

The solver in OpenFOAM generally includes three procedures, that is, mathematical modeling, equations discretization and linear system solving. The user firstly need to establish the mathematical model of the problem, e.g Euler equations, Navier-Stokes equations etc. Relative data structure such as volScalarField need to be created by user. Then the user need to write the code combined with the model using DSL, which must meet the syntax of OpenFOAM. The procedures of discretization and solving after this procedure is carried out by the kernel of OpenFOAM. OpenFOAM uses finite volume method to discrete the terms in the equations. The matrix coefficient A and the RHS of the linear system b are returned when fvm is used, and field is returned when fvc is used. After all the terms are discretized, the overload operator will return a linear system. The linear system is solved in OpenFOAM with its library.

B. Introduction to the LU-SGS Scheme

Considering the following two-dimensional Euler equations:

$$\partial_t \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix} + div \begin{pmatrix} \rho u & \rho v \\ \rho u^2 + p & \rho uv \\ \rho uv & \rho v^2 + p \\ u(E+p) & v(E+p) \end{pmatrix} = 0 . \tag{1}$$

where ρ is the density, u, v are the velocity components in x and y directions, p is the pressure, and E is the total energy. The first term in the equations is called transient term, and the second is called convective term. After changing Euler equations into a conservation equation of the form, we will have:

$$\partial_t \boldsymbol{u}(\boldsymbol{x},t) + div\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{x},t)) = \boldsymbol{0} . \tag{2}$$

In the process of the numerical discretization, the computational domain Ω is partitioned into multiple computational domains Ω_j . After discretization in each domain, we obtain the following formulation:

$$\left[\frac{\Omega_i}{\Delta t} \boldsymbol{I} + \frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{U}_i} \right] \Delta \boldsymbol{U}_i^n = -\boldsymbol{R}_i^n . \tag{3}$$

where $\boldsymbol{R}_i^n$ is the residual of the variables in cell i in the n-th time step, $\frac{\partial \boldsymbol{R}_i^n}{\partial \boldsymbol{U}_i}$ is the flux Jacobian, $\boldsymbol{I}$ is the identity matrix, $\Delta \boldsymbol{U}_i^n$ is the difference of the variables in cell i, it can be represented by another form:

$$\Delta \boldsymbol{U}_i^n = \boldsymbol{U}_i^{n+1} - \boldsymbol{U}_i^n . \tag{4}$$

After applying the Gauss' law, equation (3) is written as follows:

$$\Omega_i \frac{\boldsymbol{U}_i^{n+1} - \boldsymbol{U}_i^n}{\Delta t_i} + \sum_{j \in N(i)} \boldsymbol{A}_{c,ij} \Delta \boldsymbol{U}_{ij}^n S_{ij} = -R_i^n . \tag{5}$$

where j is the label of the cells shared face with cell i. To support the capturing of shock in nonlinear system, Steger-Warming split flux is used. The split flux is represented as follows:

$$\boldsymbol{A}_{c,ij} \Delta \boldsymbol{U}_{ij}^n S_{ij} = (\boldsymbol{A}_{c,ij}^+ \Delta \boldsymbol{U}_i^n + \boldsymbol{A}_{c,ij}^- \Delta \boldsymbol{U}_j^n) S_{ij} . \tag{6}$$

In the equation, $\boldsymbol{A}_{c,ij}^{\pm}$ represents the flux Jacobian on the face shared by cell i and cell j. This formula shows that the flux on the shared face can be split into the sum of flux flowing from cell i and cell j. In the Euler equation, $\boldsymbol{A}_{c,ij}^{\pm}$ is calculated by the following formula:

$$\boldsymbol{A}^{\pm} = \frac{1}{2} \boldsymbol{A}_c \pm \frac{1}{2} (|V| + c) \boldsymbol{I} . \tag{7}$$

where V is the velocity flux flowing through S_{ij} , $|V|+c$ is the spectral radius of the convective flux Jacobian. For the details, we refer the readers to the reference [14]. To obtain $\boldsymbol{A}_c$, which is the convective flux Jacobian, we refer the reader to reference [15]. Put all the cells together, we can get the following equation at n-th time step:

$$\boldsymbol{A} \Delta \boldsymbol{U} = -\boldsymbol{R}^n . \tag{8}$$

It is important to note that A is a large matrix of all the cells, U is a vector of all unknown variables. After dividing the left matrix $\boldsymbol{A}$ into $\boldsymbol{A}$=$\boldsymbol{D}$+$\boldsymbol{L}$+$\boldsymbol{U}$, $\boldsymbol{D}$ is the diagonal of $\boldsymbol{A}$, $\boldsymbol{L}$ is the lower matrix, and $\boldsymbol{U}$ is the upper matrix of $\boldsymbol{A}$, equation (8) can be written:

$$(\boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}) \Delta \boldsymbol{U} = -\boldsymbol{R}^n . \tag{9}$$

Notice that

$$(\boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}) = (\boldsymbol{D} + \boldsymbol{L}) \boldsymbol{D}^{-1} (\boldsymbol{D} + \boldsymbol{U}) - \boldsymbol{L} \boldsymbol{D}^{-1} \boldsymbol{U} . \tag{10}$$

The following equation can be obtained, if $\boldsymbol{L}\boldsymbol{D}^{-1}\boldsymbol{U}\Delta\boldsymbol{U}$ is ignored:

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

Equation (11) can be solved using a two-step sweep LU-SGS, which consists in a forward sweep and a backward sweep.

$$(\boldsymbol{D} + \boldsymbol{L}) \Delta \boldsymbol{U}^* = -\boldsymbol{R}^n . \tag{12}$$

$$(\boldsymbol{D} + \boldsymbol{U}) \Delta \boldsymbol{U}^n = \boldsymbol{D} \Delta \boldsymbol{U}^n . \tag{13}$$

C. Introduction to LTS

In the case of simulation, the time step Δt needs to satisfy certain conditions to make the problem stable. When using an explicit scheme, Δt is strictly constrained by the CFL

condition [16]. The constrain is relatively loose when using an implicit scheme, some scheme is even unconditionally stable. In practical applications, the simulation blows up when the time step is too large. In OpenFOAM, fixed time step and adjustable time step are usually used. When using a fixed time step, regardless of how the flow field changes, Δt^n is set as:

$$\Delta t^n = \Delta t_{user}. \tag{14}$$

This method is simple to implement, the time step does not require additional calculations, but it has a low efficient in time marching, and its setup value depends on the experience of the user. The adjustable time step is calculated based on the current flow field. A minimum time step satisfied the CFL for all the cells is chosen. Δt^n is set as:

$$\Delta t^n = \min_{i \in Cell} \frac{2CFL \bullet V}{\sum_{f \in i} \lambda_f^{\max} A_f}. \tag{15}$$

where f are the faces that surround cell i, λ_f^{max} is the largest eigenvalue of convective flux Jacobian on the face, and $\boldsymbol{A}_f$ is the area of face f. This time marching method requires additional calculations, and the value depends on the minimum time step of all the cells. When the grid size or field speed difference is large, the calculated time step has a large difference between the cells, resulting in the limitation of the execution time of the simulation. In steady state problem, since only the result of the simulation is interested, the time step can march cell by cell. The time step owned by each cell can be calculated separately:

$$\Delta t_i^n = \frac{2CFL \bullet V}{\sum_{f \in i} \lambda_f^{\max} A_f}. \tag{16}$$

LTS also requires additional calcultions, however, since each cell is marching separately, the convergence of the steady state problem can be accelerated.

III. The Implementation of LU-SGS-LTS in OpenFOAM

A. The Procedure of LU-SGS-LTS in OpenFOAM

The procedure of LU-SGS-LTS is different from that of the standard solver in OpenFOAM. In the LU-SGS-LTS solver, the discretization of the DSL language and the solving of the linear system are replaced by the LU-SGS-LTS method. The procedure of LU-SGS is used in the procedures of discretization and solving instead of using the OpenFOAM's library.

B. The Data Structure in LU-SGS-LTS Method

The variables in OpenFOAM are stored in the form of fields, the fields contain some other information including the grid information. The entity of the fields is a one-dimensional array. The data structures in OpenFOAM do not distinguish simulation cases between one-dimensional, two-dimensional and three-dimensional, all the implementations are designed based on three-dimensional. In LU-SGS-LTS, the Euler equation has five formula items. For each cell in the grid, the second term in equation (5) needs to calculate a matrix vector multiplication. Where Jacobian matrix $\boldsymbol{A}$ is a 5×5 matrix, $\Delta \boldsymbol{U}_i^n$ is a vector of length five. In the implementation, a two-dimensional array of 5 in length and cellNumber in width to store the $\Delta \boldsymbol{U}^n$ generated during the procedure of LU-SGS-LTS. To make the implementation simple, the data structure is designed as shown in Fig.3. In addition, a matrix window of 5×5 is designed to store the Jacobian matrix $\boldsymbol{A}$ for each cell.

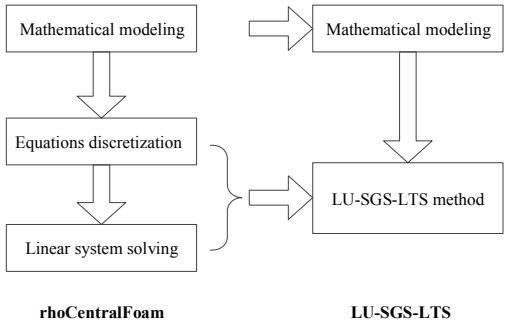


Figure 1. The change of procedures from rhoCentralFoam to LU-SGS-LTS.

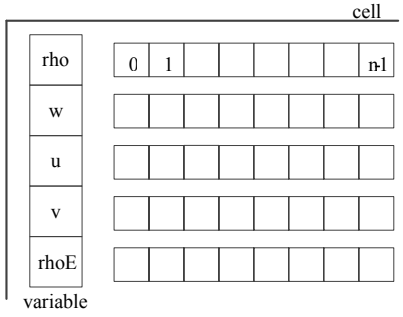


Figure 2. The data structure of $\Delta \boldsymbol{U}$.

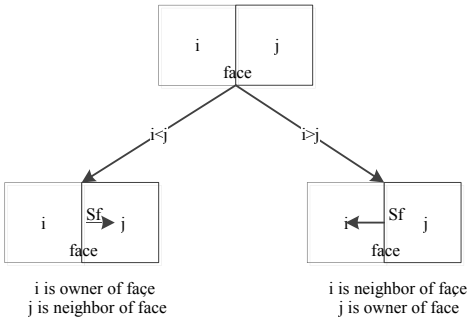


Figure 3. The owner and neighbor of face in OpenFOAM.

C. The Algorithm of LU-SGS-LTS Method

In OpenFOAM, the grid cells are numbered from small to large. Variables can be stored in the center of the grid cell, the data is organized in the form of fields. Each face of the

grid cell is shared by two cells, which are labeled by owner and neighbor of the face in OpenFOAM. The owner of the face is always the cell having a smaller label between these two share-face cells, and the neighbor of the face always has a larger label. The rule of face owner and face neighbor is shown in Fig. 3.

Such that the second term in equation (5) can be written as:

$$\sum_{j\in N(i)} \boldsymbol{A}_{c,ij}\Delta\boldsymbol{U}_{ij}^{n}S_{ij} = \sum_{i=owner} \boldsymbol{A}_{c,ij}\Delta\boldsymbol{U}_{ij}^{n}S_{ij} + \sum_{i=neighbor} \boldsymbol{A}_{c,ij}\Delta\boldsymbol{U}_{ij}^{n}S_{ij} \quad . \tag{17}$$

According to equation (6), the coefficient of $\Delta\boldsymbol{U}_i$ will be filled in matrix $\boldsymbol{D}$, and the coefficient of $\Delta\boldsymbol{U}_j$ will be filled in the matrix $\boldsymbol{L}$ or $\boldsymbol{U}$. Specifically, when i is the owner of the face, which means that j is larger than i, the coefficient will be filled in $\boldsymbol{U}$, and when i is the neighbor of the face, the coefficient will be filled in $\boldsymbol{L}$. It should be noted that the direction of the face vector is always point from the smaller cell to the larger cell, that is, always from the owner cell pointing to neighbor. It need to pay attention to the signs of the terms in the equations. We use a two-steps iteration to solve equation (12) and equation (13):

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

$$\Delta\boldsymbol{U}^{n} = \Delta\boldsymbol{U}^{*} - \boldsymbol{D}^{-1}(\boldsymbol{U}\Delta\boldsymbol{U}^{n}) \,. \tag{19}$$

In forward sweep, the $\Delta\boldsymbol{U}^{*}$ in matrix $\boldsymbol{L}$ is needed for calculating $\Delta\boldsymbol{U}^{*}$ of cell i. The cell appears in $\boldsymbol{L}$ always has a smaller label than cell i, that is, the $\Delta\boldsymbol{U}^{*}$ in matrix $\boldsymbol{L}$ will be calculated in advance when calculating $\Delta\boldsymbol{U}^{*}$ of cell i. similarly, in backward sweep, the coefficient of cell in matrix $\boldsymbol{U}$ is needed when calculating $\Delta\boldsymbol{U}^{n}$ of cell i, and this coefficient is obtained in advance when we traverse from the back. The specific algorithm is shown in algorithm 1.

Algorithm 1 Procedure of LU-SGS-LTS in time step n

procedure of calculating LTS
forward sweep:
for i=0 to cellAll-1 **do**
  if use LTS **then**
    D_i = *cellVolume*[*i*]/*LTS*Δ*t*[*i*], tmp=0
  else
    D_i = *cellVolume*[*i*]/Δ*t*, tmp=0
  end if
  for all faceI surround cell i **do**
    modify D_i
    label j= mesh.faceOwner()[i]
    if cell i is neighbor of faceI **then**
      calculate $\boldsymbol{L}$
      $tmp+=\boldsymbol{L}*\Delta\boldsymbol{U}_j^{n}$
    end if
  end for
  $\Delta\boldsymbol{U}_i^{*} = \boldsymbol{D}_i^{-1}(-\boldsymbol{R}_i^{n} - tmp)$
end for
backward sweep:
for i=cellAll-1 to 0 **do**
  if use LTS **then**
    D_i = *cellVolume*[*i*]/*LTS*Δ*t*[*i*], tmp=0
  else
    D_i = *cellVolume*[*i*]/Δ*t*, tmp=0
  end if
  for all faceI surround cell i **do**
    modify D_i
    label j= mesh.faceNeighbor()[i]
    if cell i is owner of faceI **then**
      calculate $\boldsymbol{U}$
      $tmp+=\boldsymbol{U}*\Delta\boldsymbol{U}_j^{n}$
    end if
  end for
  $\Delta\boldsymbol{U}_i^{n} = \boldsymbol{U}_i^{*} - tmp*\boldsymbol{D}_i^{-1}$
end for

IV. Experiment and Result

A. Experiment Set up

The operating system of the computer we use in this experiment is Red Hat Enterprise Linux Server 6.5, with OpenFOAM version 4.0 installed on it. In this experiment, we have tested two steady state cases.

The first case called oblique shock is a tutorial case of rhoCentralFoam solver provided in OpenFOAM. This case simulates two flows flowing from the top and left of a field in an angle. Shocks will appear and stay steady in the field in this case. We use a structured grid in this test, as is shown in Fig. 4. Notice that the grid scale of the bottom is smaller than the grid scale of the top.

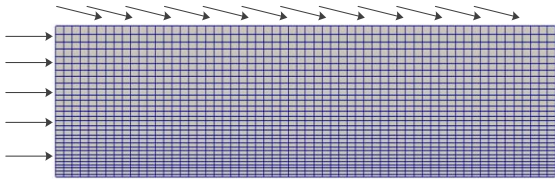


Figure 4. The grid and flow direction in oblique shock test case.

The second case is a transonic inviscid laminar flow over the NACA0012 airfoil with M_∞ = 0.8, α = 1.25. This problem has a steady state solution under these conditions. Shocks on the upper and lower surfaces appear when the solution is steady. This case is usually used to test the ability of a numerical method to capture a shock wave. It is the test case of the international workshop on high order CFD methods. We use an unstructured grid in this test. Fig.5 and Fig.6 have shown the grid of far field and near field. Also notice the grid scale of near field is smaller than the grid scale of far field. We use Euler equation introduced in section II-B as the mathematic model to solve these test cases.

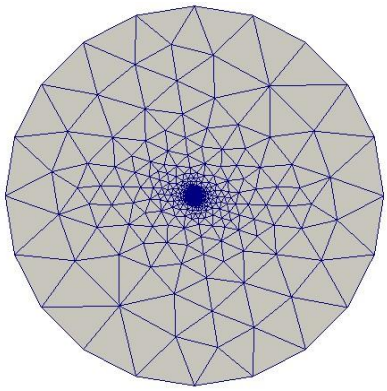


Figure 5. The far field of NACA0012 with *Ncell* =6821.

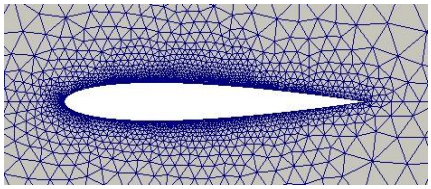


Figure 6. The near field of NACA0012 with *Ncell* =6821.

B. Methodology

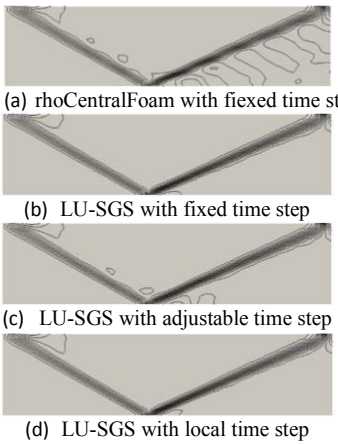
To illustrate the correctness of our implicit LU-SGS solver, we firstly simulate the oblique shock test case and NACA0012 with rhoCentralFoam, which is the official solver in OpenFOAM. The results of these cases using rhoCentralFoam are compared with the results using LU-SGS solver. The shock positions in these results should be consistent. In our experiments, we focus on the convergence rate in these two cases. Convergence critical are different between cases. In the oblique shock case, we define the relative L2 normal of residual *Res* in time step n as follows:

$$Res^n = \frac{\sqrt{\sum_{i=0}^{N} (\frac{\rho_i^n - \rho_i^{n-1}}{\Delta t})^2 / N}}{Res^0}$$

Notice that the Δt is different in different time marching methods. In the NACA0012 case, which is much more complicated, we focus on the drag coefficient of the airfoil apart from the residual. To illustrate the efficient of the LTS, we will test these two cases with rhoCentralFoam and LU-SGS with three time marching methods, as introduced in II-C. All the courant number is set 0.5 in these cases.

C. Experimental Results

1) ***Oblique shock:*** The contour lines of the ρ field is shown in Fig.7. These three results are compared with the result of original solver rhoCentralFoam in OpenFOAM. To get a result with the residual less than 0.1, we set $\Delta t = 0.001$ in the fixed time step method. Notice the positions of the shocks are consistent. Also notice that the result of LU-SGS scheme is more stable than rhoCentralFoam. The consistent in these results show the correctness of our implicit LU-SGS solver. The convergence histories of residual in terms of execution time is shown in Fig. 8. Execution time here is CPU time.



(a) rhoCentralFoam with fiexed time step

(b) LU-SGS with fixed time step

(c) LU-SGS with adjustable time step

(d) LU-SGS with local time step

Figure 7. The comparison of contour lines of ρ in oblique shock test.

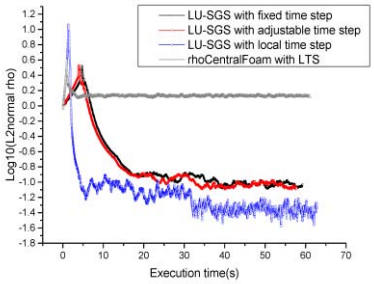
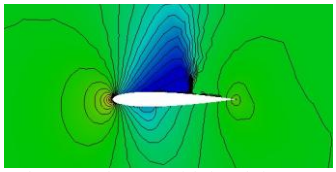


Figure 8. The residuals of oblique shock with different scheme and time marching methods.

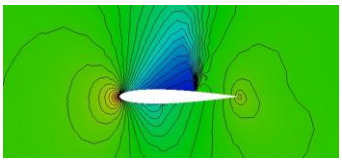
Results of the LU-SGS solver have smaller residuals than rhoCentralFoam. Since the grid in this test case has uniform scale, the adjustable time step method has a weak advantage over fixed time step. Local time step method has a faster convergence rate than the other two. Notice that the residual of local time step is also smaller.

2) ***NACA0012****:* The grid used in this case is unstructured mesh. The grid scale is smaller when it is nearer to the airfoil. The contour lines of ρ are shown in Fig. 9. All the scheme and time marching methods can obtain a steady result. Notice that LU-SGS-LTS solver has more smooth contour lines along the up tail of the airfoil.

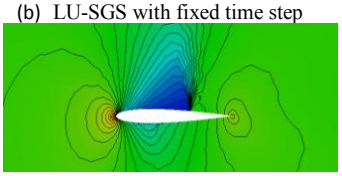
These results also verify the ability of our LU-SGS solver in capturing shock. The convergence histories of residual in terms of execution time is shown in Fig. 10. The LU-SGS-LTS solver has a smaller residual compared with rhoCentralFoam.



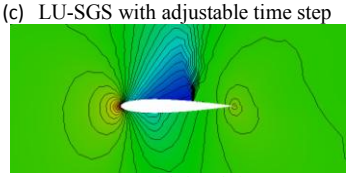
(a) rhoCentralFoam with local time step



(b) LU-SGS with fixed time step



(c) LU-SGS with adjustable time step



(d) LU-SGS with local time step

Figure 9. The comparison of contour lines of ρ in NACA0012 test.

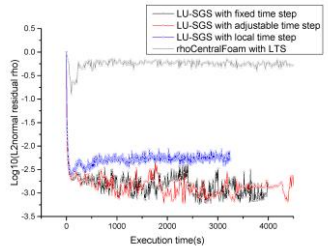


Figure 10. The residuals of NACA0012 with different scheme and time marching methods.

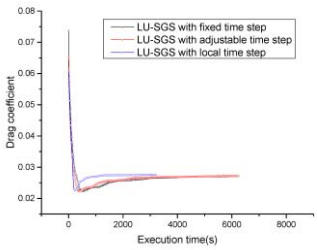


Figure 11. The drag coefficient of the airfoil with three different time marching methods.

The convergence histories of drag coefficient in terms of execution time is shown in Fig. 11. The drag coefficient of the airfoil shows that the LU-SGS method with local time step method has a faster convergence rate than the other two.

V. CONCLUSION

This paper has implemented an implicit LU-SGS solver for steady state problems. Result of an oblique shock case is compared with rhoCentralFoam to verify the correctness of our LUSGS solver. LTS is applied to the LU-SGS solver. Oblique shock and NACA0012 problems are simulated to illustrate the efficiency of our LU-SGS with local time step method. Both results of these two cases have shown a better convergence rate compared with fixed time step and adjustable time step method. This paper has some limitations since LTS is only suitable for steady state problem. Further more, LU-SGS solver is executed in sequence in this paper, the parallel algorithm is to be developed in the future.

ACKNOWLEDGMENT

The authors would like to thank the Science Challenge Project (No.TZ2016002), the National Key Research and Development Program of China (No.2016YFB0201301).

REFERENCES

[1] J. D. Anderson and J. Wendt, *Computational fluid dynamics*. Springer, 1995, vol. 206.

[2] C. Hirsch, Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics. ButterworthHeinemann, 2007.

[3] Z. Jian-hua, "Review of commercial cfd software [j]," *Journal of Hebei University of Science and Technology*, vol. 2, pp. 160–165, 2005.

[4] H. K. Versteeg and W. Malalasekera, An introduction to computational fluid dynamics: the finite volume method. Pearson Education, 2007.

[5] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, "Openfoam: A c++ library for complex physics simulations," in *International workshop on coupled methods in numerical dynamics*, vol. 1000. IUC Dubrovnik, Croatia, 2007, pp. 1–20.

[6] Q. Wang, X. Xu, C. Li, H. Ji, X. Ren, and X. Yang, "Coupling strategies investigation of hybrid atomistic-continuum method based on state variable coupling," 2016.

[7] T. T. Zhang, W. J. Yang, Y. F. Lin, Y. Cao, M. Wang, Q. Wang, and Y. X. Wei, "Numerical study on flow rate limitation of open capillary channel flow through a wedge," Advances in Mechanical Engineering, vol. 8, no. 4, 2016.

[8] S. Yoon and A. Jameson, "Lower-upper symmetric-gauss-seidel method for the euler and navier-stokes equations," *Aiaa Journal*, vol. 26, no. 9, p. 1025, 1988.

[9] R. F. Chen and Z. J. Wang, "Fast , Block Lower-Upper Symmetric Gauss-Seidel Scheme Introduction," 2000.

[10] AIAA, "Multigrid time-accurate integration of Navier-Stokes equations," *Canadian journal of veterinary research*, vol. 75, no. 2, pp. 122–127, 1993.

[11] S. Osher and R. Sanders, "Numerical approximations to nonlinear conservation laws with locally varying time and space grids," *Mathematics of Computation*, vol. 41, no. 164, pp. 321–336, 1983.

[12] C. Shen, F. Sun, and X. Xia, "Analysis on capabilities of density-based solvers within openfoam to distinguish aerothermal variables in difusion boundary layer," *Chinese Journal of Aeronautics*, vol. 26, no. 6, pp. 1370–1379, 2013.

[13] J. Kim and K. Kim, "A development and verification of density based solver using lu-sgs algorithm in openfoam," 2014.

[14] C. Shen, X. L. Xia, Y. Z. Wang, F. Yu, and Z. W. Jiao, "Implementation of density-based implicit lu-sgs solver in the framework of openfoam," *Advances in Engineering Software*, vol. 91, pp. 80–88, 2016.

[15] A. Rohde, "Eigenvalues and eigenvectors of the euler equations in general geometries," *Aiaa Journal*, 2001.

[16] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *Ibm Journal of Research & Development*, vol. 11, no. 2, pp. 215–234, 2010.