An efficient communication strategy for massively parallel computation in CFD

YunBo Wan (wanyb11@163.com)
National University of Defense Technology
Lei He
China Aerodynamics Research and Development Center
Yong Zhang
China Aerodynamics Research and Development Center
Zhong Zhao
China Aerodynamics Research and Development Center
Jie Liu
National University of Defense Technology
HaoYuan Zhang
China Aerodynamics Research and Development Center

Research Article

Keywords: CFD, large scale computation, parallel strategy, parallel efficiency, complex geometry, billions grid

Posted Date: June 9th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1730536/v1

An efficient communication strategy for massively parallel computation in CFD

YunBo Wan[1,3], Lei He[3], Yong Zhang[3], Zhong Zhao[3], Jie Liu[1,2*] and HaoYuan Zhang[3]

[1*]Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073, HuNan, China.

[2*]Laboratory of Software Engineering for Complex Systems, National University of Defense Technology, Changsha, 410073, HuNan, China.

[3*]Computational Aerodynamics Institute, China Aerodynamics Research and Development Center, Mianyang, 621000, SiChuan, China.

*Corresponding author(s). E-mail(s): liujie@nudt.edu.cn;

Abstract

With the growing computing power of high-performance computers, efficient parallel algorithms are becoming increasingly important in the development of Computational Fluid Dynamics(CFD). This research presents a novel parallel strategy based on asynchronous and package communication. This strategy tries to enhance the performance of large-scale computation for realistic complex geometry. The new strategy aggregates all communications and only requires communication once at each iteration step. Convergence of the new strategy is also proved and validated. Three numerical experiments demonstrates the exceptional parallel performance of the novel strategy in simulating complex geometry. When the number of CPU cores approach 26 thousand, strong scale parallel efficiency still remains at 74% based on 10.5 billion mesh elements. With 179,200 CPU cores and 10 billion mesh elements, weak scale parallel efficiency maintains at 90%. This research demonstrates that large-scale parallel computation can be applied efficiently in numerical simulation of a complex aircraft.

Keywords: CFD, large scale computation, parallel strategy, parallel efficiency, complex geometry, billions grid

1 Introduction

Computational fluid dynamics(CFD) is a discipline that adopts numerical methods and computers to analyze flow questions. When compared to traditional wind tunnel experiments, CFD technology has numerous advantages, including low cost and fast prediction speed. Nowadays, CFD has played a key role in aircraft industry aerodynamic development, combustion research, turbomachinery simulation and a variety of other applications.

CFD has achieved great success thanks to the rapid development of computers over the past decades. Based on modern clusters, an entire simulation of a typical case takes only several hours. However, larger-scale calculations are required with increasing demands on fidelity and the development of numerical methods. For instance, the simulation of general geometries typically requires billions of mesh elements when adopting the Direct Numerical simulation method, and the execution time for an entire simulation usually takes several months[1]. According to the forecast, supercomputers can't satisfy the demand for high precision methods until 2050[2].

On the other hand, current CFD software is incapable of making efficient use of modern computing power at all[3]. The development of applications and software is much slower than the hardware of supercomputers. In in the next two years, Exascale-level supercomputers will be available, which marks a new level of hardware. However, robust CFD flow solver scalability even on current multicore platforms is sorely lacking. Few applications can make efficient use of more than O(1000) cores, although the largest machines today are available with O(1,000,000) cores[4]. In large scale computations, the challenges mainly contain I/O, memory, storage, scalability and parallel efficiency et al. Scalability and parallel efficiency are two most crucial aspects of all issues. Scholars carried out a number of studies to address these challenges and improve scalability and parallel efficiency.

Mohammed A. Al Farhan[5] reevaluated the hybrid programming paradigm (MPI+OpenMP) on their fine-tuned version of the flux kernel of PETSc-FUN3D. Strong scalability studies were performed on the KAUST's Cray XC40 system, Shaheen II. 98,304 cores of 3072 compute nodes were launched in the largest case based on 357,900 mesh vertices. The results showed that the flux kernel still scaled well in the distributed-memory systems with the potential of running thousands of hardware cores simultaneously.

Dana A. Jacobsen et al.[6] investigated multi-level parallelism on GPU clusters with MPI-CUDA and hybrid MPI-OpenMP-CUDA parallel implementations. They studied the efficiency and scalability of incompressible flow computations using up to 256 GPUs on a problem with approximately 17.2

billion cells. Their results demonstrated that GPU clusters offered significant benefits for large data sets, and a dual-level MPI-CUDA implementation with maximum overlapping of computation and communication provided substantial benefits in performance.

Igor Menshov [7] proposed a novel CUDA+MPI computational algorithm scalable up to hundreds of GPUs and gave an in-depth analysis of its implementation. This approach could simulate compressible flow problems with complex geometry, and the strong scaling efficiency could be increased from 81 to 92%.

Ahmet Duran et al.[8] tested the scalability of the existing OpenFOAM solver icoFoam. Various sizes of penta-diagonal hepta-diagonal matrices were calculated based on the simulation of blood flow in arteries with a structured mesh domain. They achieved scaled speed-up for large matrices having sizes up to 64 million X 64 million and up to 16384 cores.

Xiazhen Liu et al.[9] introduced a parallel CFD framework software(CCFD) and some parallel technology based on the Sunway TaihuLight heterogeneous architecture. They performed a super-large computational scale test based on the Onera M6 wing model with 1 billion cells. The test achieved a parallel efficiency of 60% with a maximum of 505000 cores across the core group based on 13000 cores.

Thomas D. Economon [10] presented multi-node optimizations of SU2, a widely used open-source CFD application. Based on the results with the well-known ONERA M6 geometry, the hybrid OpenMP+MPI multigrid implementation in multi-node achieved 2X higher parallel efficiency on 256 nodes over conventional Kry-based(GMRES) methods.

Haoqiang Jin studied the MPI+OpenMP hybrid parallel strategy with two full CFD applications used by NASA engineers, OVERFLOW and AKIE. They measured the performance of various hybrid configurations of these codes on several platforms and presented a new approach which can extend the OpenMP model with new data locality extensions to better match the more complex memory subsystems available on modern HPC systems.

Feng He et al[11] designed a coarse-grained MPI/OpenMP hybrid parallelism CFD solver framework. This solver framework overlapped nonblocking MPI communication with OpenMP shared memory communication. A performance of "superlinear speedup" was gained with an increasing number of cores and mesh sizes, which showed that the solver framework has both strong and weak scalability.

Due to a lack of space, it is impossible to list all of the related research projects [12–16]. Many well-known CFD softwares such as OpenFOAM, FUN3D, SU2 have made significant efforts to enhance parallel computation performance. Meanwhile, different computer hardware and programming approaches were also investigated in order to enhance computational efficiency. Significant advances in high performance computing in CFD have been made as a result of these initiatives. Nevertheless, there are still a number of challenges with large scale computing in CFD. The number of computational cores is still remaining on the order of thousands of cores which is significantly fewer

than the total numbers of cores in modern supercomputers. The mesh scale utilized is limited up to hundreds of million of elements and simulation with billions of elements is still rarely seen. Furthermore, the geometry adopted is quite simple, which is almost useless in real-world aircraft simulation. Overall, efficient large-scale simulation based on complex geometry is still a critical topic for CFD technology development.

In this paper, we propose and design a novel CFD parallel strategy that is well suitable for modern supercomputers. This parallel strategy is implemented based on an open-source CFD sofware - PHengLEI. The design concept of the parallel strategy is to use asynchronous and package communication mode during distributed computing. The novel parallel strategy has also been proven true in theory. Several simulation cases are performed to verify the effect of the new parallel strategy. The results indicate that the performance of the new parallel strategy significantly outperforms existing traditional methods.

The remainder of the paper is set out as follows. Section 2 describes the key technologies and implementation of the proposed parallel strategy in detail. In section 3, three numerical experiments are conducted and the performance of the results is discussed. The summary is carried out in section 4.

2 A novel parallel strategy

2.1 Introduction to PHengLEI

PHengLEI[17] is a hybrid, open-source CFD platform developed by China Aerodynamics Research and Development Center(CARDC). PHengLEI opened the source code in China in 2020. PHengLEI has been cloned over 1000 times by users from colleges, institutes and companies. PHengLEI is playing a key role in the China's CFD industry.

Based on the C++ programming language, PHengLEI designs a powerful and flexible architecture and data structure.

Full-speed flow problems, including the subsonic, transonic, supersonic, and hypersonic flow problems, can be accurately simulated via PHengLEI. A variety of computational models and numerical methodologies are incorporated in PHengLEI software. The most attractive feature of PHengLEI is that it includes a structured grid solver as well as an unstructured grid solver, allowing it to support both the structured grids and unstructured grids simultaneously. The two solvers can work independently to simulate distinct questions, meanwhile, they can work together to mimic the same problem. This research focuses on an unstructured solver with a laminar model.

2.2 governing equation and numerical solution

The feature of flow problems is dominated by Navier-Stokes equations. The Navier-Stokes equations are solved numerically using numerous approaches in CFD numerical simulation. First, we give a brief overview of the governing equations and discretization schemes used in this paper. The compressible,

laminar flows based on the perfect gas model are focused on. The Navier-Stokes equations can be expressed in integral form as

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{Q}dV+\oint_{\partial\Omega}\mathbf{F}^{i}dS=\oint_{\partial\Omega}\mathbf{F}^{v}dS \tag{1}$$

where $\mathbf{Q}$ represents the conservative variables, $\mathbf{F}^{i}$ and $\mathbf{F}^{v}$ are the convective fluxes and viscous fluxes respectively. Ω is the control volume, $\partial\Omega$ is the outer boundary of the control volume Ω.

This work focuses on transonic laminar flow problems with unstructured grid are focused on. The discretization strategy adopts a finite volume method (FVM) with a typical cell-based data structure. This data structure is built with the control volumes constructed using a cell-centered technique. The numerical solution of Navier-Stokes equations mainly involves spatial discretization and time discretization. The spatial discretization includes three parts: convective flux computing, viscous flux computing ,and source terms computing. Source terms are not taken into account in this study. Time discretization is solving the systems of linear equations obtained from spatial discretization.

The first step of the numerical solution is space discretization of the governing equations. The space term is discrete on the outer boundary of a control volume i and the time term adopts the full implicit method, then one can get the discrete forms of NS equations as:

$$\frac{V_i}{\Delta t}\left(\mathbf{Q}_i^{n+1}-\mathbf{Q}_i^{n}\right)=\sum_{j\in N(i)}\left(-\mathbf{F}_{c,ij}^{n+1}+\mathbf{F}_{v,ij}^{n+1}\right)dS_{ij} \tag{2}$$

where V_i is the volume of control volume i, Δt represents the time step, $\mathbf{Q}_i^{n+1}$ is the conservative variables in element i at time step $n+1$, $\mathbf{Q}_i^{n}$ is the conservative variables in element i at time n, $N(i)$ means the set of neighbor elements of the cell i, j is one element of the neighbor cells set of element i, $\mathbf{F}_{c,ij}^{n+1}$ represents the convective flux on the common face of element i and j at time step $n+1$,$\mathbf{F}_{v,ij}^{n+1}$ is the viscous flux on the common face at time step $n+1$, dS_{ij} is the area of the common face.

It should be noted that the terms labeled n means that they are calculated by the conservative variables at time n, and $n+1$ is similar. Variables at time step n are known at present and variables at time step $n+1$ are required solved. Lower Upper Symmetric Gauss Seidel(LUSGS) method[18] is applied to solve this equation. Formula 2 can be translated into the form using LUSGS as follows:

$$\mathbf{M}_i^{n}\Delta\mathbf{Q}_i^{n}+\sum_{j\in N(i)}\mathbf{M}_{ij}^{n}\Delta\mathbf{Q}_j^{n}=\sum_{j\in N(i)}\left(-\mathbf{F}_{c,ij}^{n}+\mathbf{F}_{v,ij}^{n}\right)dS_{ij} \tag{3}$$

where $\Delta\mathbf{Q}_i^n$ is the variation of the conservative variables in cell i at time step n, $\Delta\mathbf{Q}_j^n$ is the same meaning as $\Delta\mathbf{Q}_i^n$ in cell j, $\mathbf{M}_i^n$ is one coefficient of the formed Matrix at the position of ith row and ith column, $\mathbf{M}_{ij}^n$ is one coefficient of the formed Matrix at the position of ith row and jth column, $\mathbf{F}_{c,ij}^n$ represents the convective flux on the common face of element i and j at time step n, $\mathbf{F}_{v,ij}^n$ is the viscous flux on the common face at time step n, dS_{ij} is the area of the common face.

We can see that in formula 3, $\Delta\mathbf{Q}^n$ is the only variable to be solved, and other terms are derived by the known variables.

The convective flux and the viscous flux are the most complicated terms in the iterative solution, therefore we'll highlight the evaluation of the two terms first. $\mathbf{F}_{c,ij}^n$ is calculated in convective flux computation. $\mathbf{F}_{v,ij}^n$ is evaluated in viscous flux computation. $\mathbf{M}$ is accessed in time term computation. Three terms are introduced in detail in the following text.

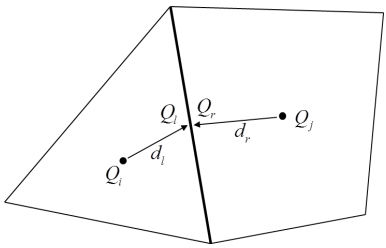


Fig. 1 reconstruction of physical quantity

For estimating the convective flux, the popular Roe approximate Riemann solver[19] is adopted. Evaluation of the convective term involves three steps as indicated in fig 1. The first step is constructing the second-order accuracy face value by the gradient reconstruction method. The gradient calculation utilizes the Green-Gauss method in this study. The second step is to limit the reconstruction value by a limiter coefficient for preserving monotonicity in the solution. This work employs the well known Venkatakrishnan limiter computing approach. Roe flux scheme is adopted to evaluate the convective fluxes in the third step. The details of the Roe scheme can refer to [19]. some variables from the neighbor cells are required to be exchanged if faces for flux calculation are located at the interface between different zones. These flow variables consist of the density(ρ), velocities in three directions (u, v, w), pressure(p) and the limiter coefficient(ϕ).

The central difference method[20] is employed to approximate the viscous fluxes,which is easier than convective flux calculation. The evaluation of viscous flux mainly consists of two steps. The first step is to evaluate primitive values and their gradient values at the face. The weighted average method is adopted in this calculation progress. In the second stage, the viscous flux is then estimated directly using the viscous flux formula. If faces are on the interface of different zones after the domain decomposition, the primitive variables

and their gradients of the neighbor cells are also required to be transferred. These flow variables consist of the density(ρ), velocities (u, v, w), pressure(p) temperature(T) and the viscosity coefficient(μ).

The treatment of time terms plays a role part in the convergence and robustness of the numerical solutions. The widely used LUSGS[18] method is adopted in this paper due to its numerical stability and robustness. LUSGS method a matrix-free which is also easy to code. LUSGS is divided into two parts: forward sweep and backward sweep, with formulas as follows:

$$L\overline{\Delta Q}^n = b, \qquad U\Delta Q^n = \overline{\Delta Q}^n \tag{4}$$

where L and U are respectively lower and upper triangular matrix of the coefficient matrix $\mathbf{M}^n$, $\Delta\mathbf{Q}^n$ is the variable to be solved, $\overline{\Delta\mathbf{Q}}^n$ is the intermediate variable for solving $\Delta\mathbf{Q}^n$.

$\Delta\mathbf{Q}^n$ in the whole flow field can be solved by formula 4. $\Delta\mathbf{Q}^n$ at the interface between different zones is required communicated in LUSGS method.

After solving formula 4, primitive and conservative variables need to be updated via $\Delta\mathbf{Q}^n$. No communication is required in the procedure of updating results. Above all, the iterative solution for the entire laminar flow can be summarized in six steps as follows:

1. Perform Pre-processing such as the mesh I/O and partitioning.
2. Calculate the conservative flux.
3. Calculate the viscous flux.
4. Time marching process.
5. Update results of primitive variables.
6. Output the results.

Where the steps 2 to step 5 are major parts of the iterative computation. There is a lot of communication in these steps which are mainly optimized in this paper.

2.3 CFD basic communication

Efficient communication is a significant and hard challenge in large-scale parallel computing. This section introduces the general communication procedure in CFD parallel simulation. Message Passing Interface(MPI) which is a de-facto standard for parallel programming, is the commonly used communication mode. Despite many other parallel models have been developed, such as OpenMP and OpenAcc, MPI is still the most important parallel model in current high performance computing.

In CFD simulations, tens of millions of mesh elements are typically required for practical engineering geometry. The domain decomposition strategy is a major mean for tackling large-scale challenges. Decomposing the mesh into several sub-zones via partitioning tools such as METIS[21] is the initial stage in Domain decomposition. Second, each sub-zone is allocated to a processor, which builds the connection relationship containing information about neighboring cells. Finally, the CFD application begins an iterative computation and communicates values among processors based on the connection relationship.

In the following text, we will provide an exhaustive explanation via a basic scenario depicted in Fig. 2.

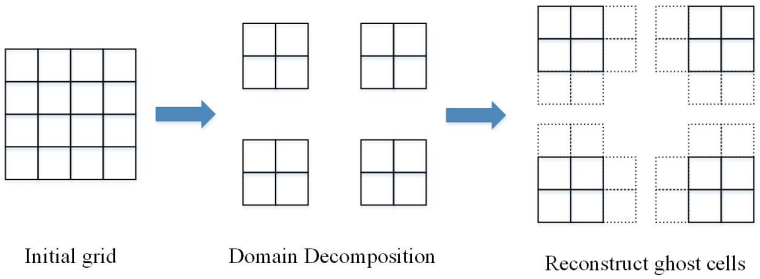
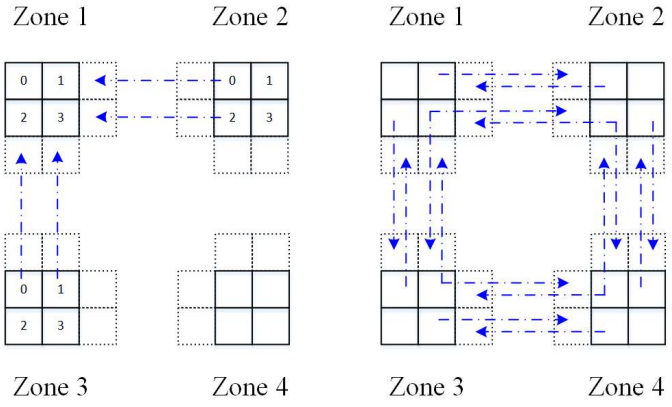


Fig. 2 Domain Decomposition in PHengLEI application

An initial grid with 16 elements is separated into four zones. The four zones are labeled with numerals 1-4, and each zone is arranged to a CPU processor respectively. It is easy to see that each zone owns 4 elements and the load on each processor is balanced. The principle of ensuring load balance is the similar in large-scale problems. Afterward, connection relationship and communication data structure between the four zones are established.

The connection relationships include the information of neighbored zones and neighbored cells. Take zone 1 as an example, zone 1 has two neighbored zones, namely zone 2 and 3. Cell 1 in zone 0 is neighbored by cell 0 of zone 2 and cell 2 is neighbored by cell 0 of zone 3. Cell 3 is both neighbored by cell 2 of zone2 and cell 1 of zone 3. Similar connection relationships for other zones are also required to be established. Data structure primarily refer to ghost cells used to store the values received from neighbor zones as labeled by the blue arrow in Fig. 3(a).



(a) Connection relationship. (b) Values passing.

Fig. 3 Communication between different zones

In parallel simulation, the Ghost cell approach makes each zone solve the equations independently like solving a serial problem. The entire communication procedure for the four zones is displayed in Fig. 3(b). CPU 1 sends and receives variables from CPU 2 and 3, respectively. Other processors have a similar communication process with CPU 1. Each zone will set variables in ghost cells by received values following communication.

Communication exists in the entire calculation processes such as preprocessing, iterative computing, and post-processing. Communication in iterative computing is the most important factor for performance. Fig. 4 depicts the detailed communication process in a single iteration step. Limiter coefficients (ϕ) and gradient values of primitive variables ($dqdx, dqdy, dqdz$) are communicated in the convective flux computation. The temperature T and its gradient values ($dTdx, dTdy, dTdz$) are communicated in the computation of viscous flux. The variables of variation of conservative variables (dQ) are communicated in time term discretization. The primitive variables(q) are communicated after updating the results. An iteration step is finished and a new step will follow the same pattern.

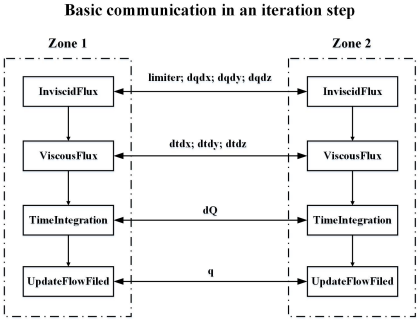


Fig. 4 part of communication process in CFD

2.4 Asynchronous communication

The communication is performed immediately when required in CFD basic parallel mode, therefore a large amount of communication is required in each iteration step. Frequent communication raises the probability of calculation failure and reduces parallel efficiency in large-scale computation. In this section, we propose an asynchronous parallel mode to address the inadequacies of the basic parallel mode. All values are transferred after an iteration step finished, and processers nearly never have to wait for each other. The illustration of asynchronous communication is shown in Fig. 5.

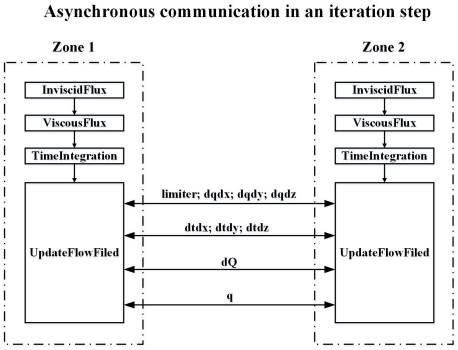


Fig. 5 asynchronous communication

Comparing Fig. 4 and 5 shows that basic method communicates values in respective functions, however asynchronous mode communicates all values after an iteration step finished. It should be noted that the primitive variables q in basic mode and asynchronous mode are all passed after an iteration step. Due to delayed communication, the asynchronous mode has different iterative results from the basic mode at the beginning of the iterative computation. However, the final convergence results of the two approaches are identical. The proof of convergence of asynchronous mode is given in the following text.

The expanded formula for discretization of second order accuracy using cell values is expressed as:

$$\begin{aligned} & M\left(\mathbf{Q}_i^n\right) \Delta \mathbf{Q}_i^n+\sum_{j \in N_i} M\left(\mathbf{Q}_i^n, \mathbf{Q}_j^n\right) \Delta \mathbf{Q}_j^n \\ & =-\sum_{j \in N_i} F_c\left(\mathbf{Q}_i^n+\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_i^n d \mathbf{x}_{ij}, \mathbf{Q}_j^n+\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_j^n d x_{ji}\right) S_{ij} \\ & +\sum_{j \in N_i} F_v\left(\mathbf{Q}_i^n+\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_i^n d \mathbf{x}_{ij}, \mathbf{Q}_j^n+\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_j^n \cdot d \mathbf{x}_{ji}\right) S_{ij} \end{aligned} \tag{5}$$

Where $\mathbf{Q}_i^n$ stands for conservative variables of cell i at n time step, $\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_i^n$ represents the gradient of cell i at n time step, $\Delta \mathbf{Q}_i^n$ is the value to be solve of cell i at n time step, $d\mathbf{x}_{ij}$ is the vector from the center of cell i to the interface ij, and $\mathbf{Q}_j^n$, $\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_j^n$, $d\mathbf{x}_{ji}$ is corresponding term in cell j.

Assume that neighbor cells numbered from 1 to $N(i)-1$ of cell i are in the same zone as cell i, and the Nth cell is in a different zone. Adopting asynchronous communication, the discretization formula5 of left hand term expresses as:

$$\mathbf{LHS}_a=M\left(\mathbf{Q}_i^n\right) \Delta \mathbf{Q}_i^n+\sum_{j \in N_i-1} M\left(\mathbf{Q}_i^n, \mathbf{Q}_j^n\right) \Delta \mathbf{Q}_j^n+M\left(\mathbf{Q}_i^n, \mathbf{Q}_N^n\right) \Delta \mathbf{Q}_N^{n-1} \tag{6}$$

The difference of left hand term between basic and asynchronous communication mode is

$$\mathbf{LHS}_b-\mathbf{LHS}_a=M\left(\mathbf{Q}_i^n, \mathbf{Q}_N^n\right)\left(\Delta \mathbf{Q}_N^n-\Delta \mathbf{Q}_N^{n-1}\right)=\mathbf{M}_{iN}^n \frac{\partial^2 \mathbf{Q}}{\partial t^2}(\Delta t)^2 \tag{7}$$

where $\mathbf{LHS}_b$ and $\mathbf{LHS}_a$ are left hand term of basic and asynchronous communication mode respectively.

The difference between two means of communication is a second-order quantity that can be ignored. Meanwhile, the difference in convective flux between basic and asynchronous communication modes is a second order quantity and the proof progress can be expressed as:

$$\begin{aligned} & \mathbf{F}_{c,b}-\mathbf{F}_{c,a}=F_c\left(\mathbf{Q}_i^n+\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_i^n d x_{iN}, \mathbf{Q}_N^n+\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^n \cdot d x_{Ni}\right) S_{iN} \\ & -F_c\left(\mathbf{Q}_i^n+\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_i^n d x_{iN}, \mathbf{Q}_N^n+\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^{n-1} \cdot d x_{Ni}\right) S_{iN} \\ & =\left(\frac{\partial \mathbf{F}_c}{\partial \mathbf{Q}}\right)_j^n\left[\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^n-\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^{n-1}\right] d x_{Ni} S_{iN} \\ & =\frac{\partial \mathbf{F}_c}{\partial \mathbf{Q}_j^n} \frac{\partial^2 \mathbf{Q}}{\partial x \partial t} S_{iN} d x_{Ni} d t \end{aligned} \tag{8}$$

The difference in viscous flux between two approaches is also second order as follows:

$$\begin{aligned}
&\mathbf{F}_{v,b} - \mathbf{F}_{v,a} = F_v\left(\mathbf{Q}_i^n + \left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_i^n d\mathbf{x}_{iN}, \mathbf{Q}_N^n + \left(\frac{\partial Q}{\partial x}\right)_N^n \cdot d\mathbf{x}_{Ni}\right) S_{iN} \\
&-F_v\left(\mathbf{Q}_i^n + \left(\frac{\partial \mathbf{Q}}{\partial \mathbf{x}}\right)_i^n dx_{iN}, \mathbf{Q}_N^n + \left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^{n-1} \cdot dx_{Ni}\right) S_{iN} \\
&= \left(\frac{\partial \mathbf{F}_v}{\partial \mathbf{Q}}\right)_j^n \left[\left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^n - \left(\frac{\partial \mathbf{Q}}{\partial x}\right)_N^{n-1}\right] dx_{Ni} S_{iN} \\
&= \left(\frac{\partial \mathbf{F}_v}{\partial \mathbf{Q}}\right)_j^n \left(\frac{\partial^2 \mathbf{Q}}{\partial x \partial t}\right)_N^{n-1} S_{iN} d\mathbf{x}_{Ni} dt
\end{aligned} \tag{9}$$

The proof of convergence indicates that the asynchronous communication mode can ensure second-order accuracy. Therefore, the asynchronous communication mode can obtain the same numerical solution as the basic mode. Algorithm 2 provides a concrete introduction to asynchronous communication in PHengLEI. The 'RegisterInterField' function is used to register interface fields used for storing variables before iterative calculation. 'InviscidFlux', 'ViscousFlux', 'LUSGS' and 'Update' are four parts in an iterative step respectively. The function 'UploadInterfaceValue' is used to update variables of interface fields in this part. The function 'DownloadInterfaceValue' is adopted to modify variables in ghost cells. 'CommunicateInterfaceValue' is utilized to transfer all variables via MPI.

Algorithm 1 Asynchronous Communication Strategy

```
1:  RegisterInterField(q, double, size );      ···
2:  for iter = 0 to nMaxIterStep do
3:      InviscidFlux(){
4:          UploadInterfaceValue(dqdx);      ···
5:          DownloadInterfaceValue(dqdx);      ···
6:      }
7:      ViscousFlux(){
8:          UploadInterfaceValue(dtdx);      ···
9:          DownloadInterfaceValue(dtdx);      ···
10:     }
11:     LUSGS(){
12:         UploadInterfaceValue(dQ);
13:         DownloadInterfaceValue(dQ);
14:     }
15:     Update(){
16:         UploadInterfaceValue(q);
17:         DownloadInterfaceValue(q);
18:     }
19:     CommunicateInterfaceValue(q);      ···
20: end for
```

2.5 Package communication

Asynchronous communication transfers all variables together after one iteration step finished. However, total communication times don't decrease, implying that each iteration step still contains 24 MPI 'send-receive' processes. We design a data container named 'DataContainer' to communicate all variables at once in an iteration step. This communication mode is called 'package communication' and the communication concept is displayed in Fig. 6.

Three steps are required in package communication mode. First, all variables that need to be transferred are compressed into DataContainer in each zone. Second, each zone communicates its own DataContainer with others, which means that each zone sends DataContainer to other zones and receives DataContainer from other zones. Third, every zone decompresses the data from the received DataContainer and set the data to ghost cells. Total communication times are reduced to one time in an iteration step using package communication mode.

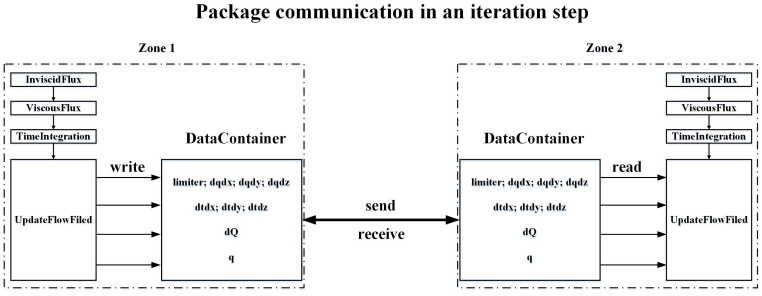


Fig. 6 data exchange using the data container

The purpose of package communication is agglomerating all communications and its implementation depends on the design of DataContainer. DataContainer is claimed to be compatible with all types of data, such as *int*,*bool*, *double*, *float* and so on. The concrete work principle of DataContainer is displayed in Fig. 7.

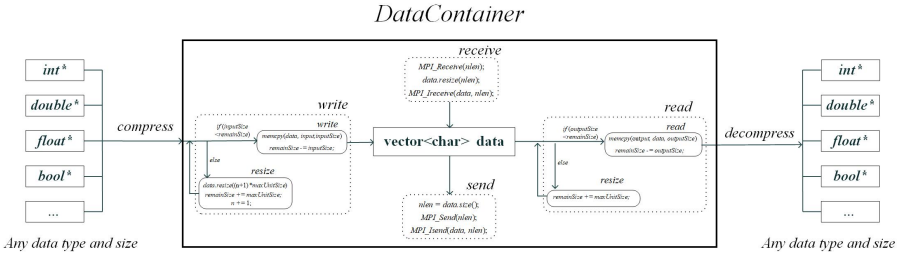


Fig. 7 work principle of DataContainer

The class DataContainer owns a *vector* type 'data' that is responsible for storing the communication variables. The four basic attributes of DataContainer are 'write', 'read', 'send', and 'receive', respectively. The four functions are used to compress, decompress, MPI sends and MPI receives in communication.

The 'write' function is applied to compress all variables into DataContainer. Before writing the data, DataContainer first checks whether the remaining memory space is sufficient or not. DataContainer resizes a defined length of memory and copies the data into DataContainer if the remaining space is insufficient. If the data is sufficient, DataContainer copies it directly into DataContainer. The copy operation is performed at the memory level, thus any type of data can be compressed into DataContainer.

The 'read' is a correspondingly function employed to decompress values from 'data'. It copies data from DataContainer to an array prepared in advance. After the copying operation, the size of the remaining data in DataContainer will decrease until all data has been read. It should be noted that all data must be decompressed in the same order as compressed into DataContainer.

The 'send' function is used to send 'data' to other processors through MPI.

The length of all data is initially sent to other processors that are utilized to open memory space in the opponent's processors. The DataContainer then transfers all of the data to the other processors, and the opponent's processors will read data from the received DataContainer.

The 'receive' function is correspondingly used to receive 'data' from other processors. Each processor will receive a data length and open memory space used for storing the received DataContainer.

Concrete illustration for package communication in PHengLEI is listed in algorithm 2. Zone 1 compresses and sends DataContainer to Zone 2, which receives the DataContainer and decompresses the data from DataContainer. Communication in complex simulation with more zones is similar to that of this case as algorithm 2.

Algorithm 2 Package Communication Strategy

progress in zone 1:

DataContainer cdata:
for $iVar = 0$ to $nTotalVars$ **do**
    cdata → write($iVar$)
end for
cdata → send(zone 2);

progress in zone 2:

DataContainer cdata;
cdata → receive(zone 1);
for $iVar = 0$ to $nTotalVars$ **do**
    cdata → read($iVar$)
end for

3 Results and Discussion

In this section, we perform three numerical experiments to verify the effect of the novel communication strategy. ShanHe, a supercomputer constructed by JiNan's national supercomputer facility, serves as the computing platform. ShanHe cluster holds 5400 computing nodes connected by InfiniBand SDR network. Each computing node is equipped with two Intel Xeon Gold 6285R processors and each processor holds 28 cores. There is a total of 192GB of memory on each computing node. The MPI library version is mpich-3.3.2, and the compiler is gcc 7.5.0.

The first experiment simulates a typical airfoil flow problem in order to compare the performance of the new parallel strategy to the basic method. The second experiment performs a strong scale parallel efficiency test via realistic geometry with 10.5 billion elements. The third experiment conducts a weak scale efficiency test based on the same problem as the second experiment.

3.1 3D transonic flow past an ONERA M6 wing.

In this experiment, ONERA M6 wing[22] widely used validation case for numerical solution, is adopted. The experiment includes the verification of correctness and efficiency. Correctness verification is utilized for revealing the precision of asynchronous communication. Efficiency verification is used to indicate that the new strategy can increase parallel performance. The flow condition is that Mach number equals 0.8395, the reference temperature is 255K, reference pressure is equal to 315,979Pa, and the angel of attack is 3.06 degrees. The mesh utilized holds 2.4 million elements as detailed in Fig. 8. Ten thousand iteration steps are performed to ensure the algorithm converges.

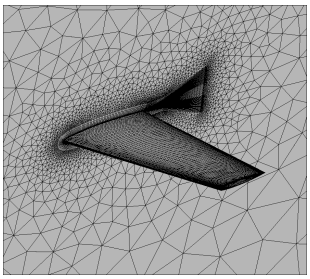


Fig. 8 Mesh of ONERA M6 wing

The first step is to verify the correctness of the code using four CPU cores. Results of several different methods can be observed in Fig. 9. The curve represents the distribution of wall pressure along a line cutting at a constant z position. The pressure distributions at the position of $z/b = 0.2$ and $z/b = 0.65$ are shown below. The new strategy could gain exactly the same results as the basic method, and results of both methods agree well with experiments which illustrates that the novel strategy enables accuracy simulation.

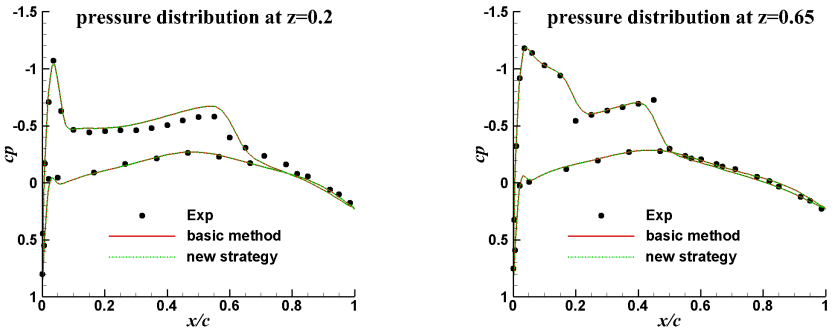


Fig. 9 comparison of pressure distribution at position of 20%

Another point to consider is the time for the two approaches to simulate. The execution time of the novel and basic methods is 9.5 hours and 9.7 hours respectively. The novel strategy is slightly faster than the basic method based on four CPU cores. This case is used as the baseline of the next experiment for parallel performance. The performance compare experiment focuses on strong scale parallel efficiency. The parallel efficiency of strong scale is calculated as:

$$e_p = \frac{t_b n_b}{t_p n_p}$$

where n_b is the number of cores in the base case, t_b is the baseline computational time, n_p is the number of cores in parallel cases, t_p is the computational time of the base case, and e_p is the strong scale parallel efficiency.

There are six cases in this experiment, each case with 4, 8, 16, 32, 64, and 128 cores respectively. The grid and flow conditions in this test are identical to those in the previous test. The performance comparison is illustrated in Fig. 10. When a basic communication method is adopted, the efficiency drops sharply with the increasing number of cores. When the core number goes up to 128 cores, parallel efficiency is only 50%. However, the parallel efficiency curve with the new strategy can remains almost flat as the number of cores increases. The results show that the asynchronous parallel strategy significantly enhances parallel performance as compared to the basic methods.

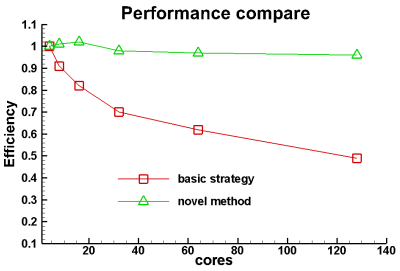


Fig. 10 performance comparison of two communication methods

3.2 strong scale parallel efficiency

CHN-T(CHiNa-Transport) model[23] is a standard single-channel transporter designed by China Aerodynamics Research and Development Center(CARDC). The CHN-T model is primarily used to validate and verify CFD software and methodologies. The CHN-T model consists of airfoils, body, flat tail, vertical tail, pylon, nacelle, and other components. The simplified CHN-T model without nacelle and pylon is adopted in this test shown in 11.

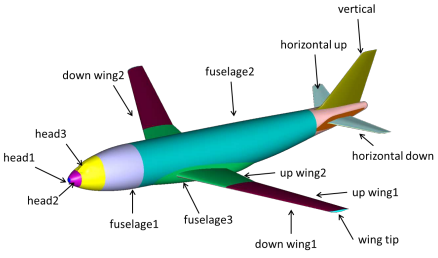


Fig. 11 CHN-T transporter model

The flow condition in this experiment is that the Mach number equals 0.2, the Reynold number per meter is 6.5e6, and the reference temperature is 288.15K. The computational grid contains 10.5 billion elements which requires 11 Terabytes of hard disk space. The grid is created by refining a coarse mesh with 20 million elements three times, each time gaining an 8X increase in the number of elements.

The number of cores used for the strong scale performance test ranges from 11200 cores, to 257600 cores. Each case runs three times for the average execution time. Each case performs 100 iteration steps and the execution time is measured by average time between 10 and 100 steps. The execution time using 11200 cores is serves as the benchmark for all other cases.

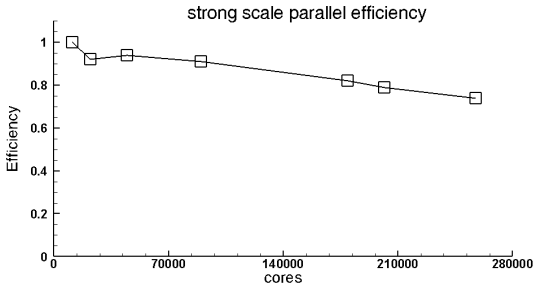


Fig. 12 strong parallel efficiency

Fig. 12 illustrates the performance of parallel efficiency along with the number of cores based on the novel strategy. Parallel efficiency decreases slowly along with increasing numbers of cores. The parallel efficiency remains at 74% when using 257600 cores. Cluster stability is another factor that affects parallel performance. Parallel performance significantly suffers from the cluster stability when the number of cores exceeds 100 thousand in ShanHe cluster. Performance of parallel efficiency should have notable improvements on a more stable cluster. When the number of elements reaches 10.5 billion, the simulation with basic communication mode always fails. As a result, results based on the basic communication mode are not shown here.

3.3 weak scale parallel efficiency

Weak scale parallel efficiency is another common notion of scalability. Weak scale is defined as how the computation time varies with the number of computational cores based on a fixed problem size per computational core. The same computational model and flow conditions as the strong scale test are adopted in this weak scale test. The number of computational cores employed in this test is 350 cores, 2800 cores, 22400 cores, and 179200 cores, respectively. The mesh size ranges from 20 million to 10.5 billion elements, with a load of

58,746 elements on each core. The case with 350 cores and 20 million elements is employed as the baseline of the weak scale efficiency test.

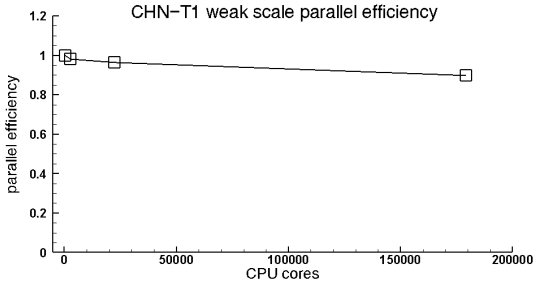


Fig. 13 weak parallel efficiency

As we can see from Fig. 13, the performance of weak scale parallel efficiency decreases slightly along with the increasing number of cores. The last case with 17920 cores and 10.5 billion elements can still maintain an 89.8% efficiency. The performance loss might be overlooked when considering the instability of the cluster. The advantage of the novel strategy are further validated by the perfect performance of weak scale parallel efficiency.

4 Conclusion

Large-scale parallel computation is a crucial issue in CFD simulation for complex geometry. This paper proposes a new parallel strategy that includes asynchronous and package communication. Asynchronous communication transfers all variables after an iteration step is completed and decreases the delay of waiting. Furthermore, the convergence of asynchronous communication is proven to guarantee computational accuracy. In Package communication, all variables are compressed into a DataContainer, and then the DataContainer is transferred between processors. Package communication achieves communicating once in a single iteration step, considerably reducing the transfer times.

Three tests are performed to verify the performance of the new parallel strategy. Based on the simulation case of the M6 wing model, the novel parallel strategy enables getting the same accurate results as the basic communication mode. Meanwhile, it enhances the performance of parallel efficiency compared to the basic communication mode. Both the strong and weak scale tests indicate that the new strategy exceeds the scale limits of mesh sizes and processing cores. Moreover, it enables a perfect performance of parallel efficiency on large-scale problems.

This research provides a strategy for high fidelity simulation of complex geometry using high accuracy methods. The achievement is significant in terms of the rapid and accurate design of aircraft. However, this research only considers the performance of parallel efficiency, and an entire simulation of large-scale

complex geometry remains a significant task. The entire simulation involves a slew of other issues, such as visualization of computational results, long time stable running and so on, which will be further researched.

Acknowledgments. This paper was supported by the National Key Research and Development Program of China(2017YFB0202104), the National Key Research and Development Program of China(2018YFB0204301), National Numerical Windtunnel(NNW) Project, and the national supercomputer center in JiNan.

Declarations

- All authors agreed with the content and all gave explicit consent to submit and they obtained consent from the responsible authorities at the institute where the work has been carried out, before the work is submitted.
- All authors agreed with the content for publication.
- The datasets generated during and analysed during the current study are available from the corresponding author on reasonable request.
- None Competing interests
- This study was funded by National Key Research and Development Program of China(2017YFB0202104, 2018YFB0204301) and National Numerical Windtunnel(NNW) Project.
- YunBo Wan wrote the manuscript; Lei He performed the data analyses ; Yong Zhang performed the experiment; Zhong Zhao contributed significantly to analysis and manuscript preparation; Jie Liu contributed to the conception of the study; HaoYuan Zhang helped perform the analysis with constructive discussions.

References

[1] Witherden, F.D., Jameson, A.: Future directions in computational fluid dynamics. In: 23rd AIAA Computational Fluid Dynamics Conference (2017)

[2] Spalart, P.R.: Strategies for turbulence modelling and simulations. International Journal of Heat and Fluid Flow **21**(3), 252–263 (2000)

[3] Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., Mavriplis, D.: Cfd vision 2030 study: A path to revolutionary computational aerosciences. mchenry county natural hazards mitigation plan (2014)

[4] Top 500 supercomputer sites; http://www.top500.org/

[5] Al Farhan, M.A., Kaushik, D.K., Keyes, D.E.: Unstructured computational aerodynamics on many integrated core architecture. Parallel Computing **59**, 97–118 (2016)

[6] Jacobsen, D.A., Senocak, I.: Multi-level parallelism for incompressible flow computations on gpu clusters. Parallel Computing **39**(1), 1–20 (2013)

[7] Menshov, I., Pavlukhin, P.: Highly scalable implementation of an implicit matrix-free solver for gas dynamics on gpu-accelerated clusters. Journal of supercomputing (2017)

[8] Duran, A., Celebi, M.S., Piskin, S., Tuncel, M.: Scalability of openfoam for bio-medical flow simulations. Journal of supercomputing **71**(3), 938–951 (2015)

[9] Liu, X., Lu, Z., Yuan, W., Ma, W., Zhang, J.: Massively parallel cfd simulation software: Ccfd development and optimization based on sunway taihulight. Scientific Programming **2020**(5), 1–17 (2020)

[10] Economon, T.D., Mudigere, D., Bansal, G., Heinecke, A., Palacios, F., Park, J., Smelyanskiy, M., Alonso, J.J., Dubey, P.: Performance optimizations for scalable implicit rans calculations with su2. Computers and Fluids, 146–158 (2016)

[11] He, F., Dong, X., Zou, N., Wu, W., Zhang, X.: Structured mesh-oriented framework design and optimization for a coarse-grained parallel cfd solver based on hybrid mpi/openmp programming. The Journal of Supercomputing **76**(4), 2815–2841 (2020)

[12] Lee, S., Gounley, J., Randles, A., Vetter, J.S.: Performance portability study for massively parallel computational fluid dynamics application on scalable heterogeneous architectures. Journal of Parallel and Distributed Computing **129**, 1–13 (2019)

[13] Xue, W., Jackson, C.W., Roy, C.J.: An improved framework of gpu computing for cfd applications on structured grids using openacc. Journal of Parallel and Distributed Computing (2021)

[14] Wang, Y., Yan, X., Zhang, J.: Research on gpu parallel algorithm for direct numerical solution of two-dimensional compressible flows. The Journal of Supercomputing, 1–21 (2021)

[15] Kissami, I., Cerin, C., Benkhaldoun, F., Scarella, G.: Towards parallel cfd computation for the adapt framework (2021)

[16] Shang, Z.: Large-scale cfd parallel computing dealing with massive mesh. Journal of Engineering,2013,(2013-4-28) **2013**, 1–6 (2013)

[17] Zhong Zhao, L.H.X.H.Y.G.Q.X. Laiping Zhang: Phenglei: A large scale parallel cfd framework for arbitrary grids. Chinese journal of computers **42**(11), 2368–2383 (2019)

[18] Jameson, A., Yoon, S.: Lower-upper implicit schemes with multiple grids for the euler equations. AIAA Journal (1987)

[19] Roe, P.L.: Approximate riemann solvers, parameter vectors, and difference schemes. Journal of Computational Physics **43**(2), 357–372 (1981)

[20] Jalali, A., Sharbatdar, M., Ollivier-Gooch, C.: Accuracy analysis of unstructured finite volume discretization schemes for diffusive fluxes. Computers Fluids **101**, 220–232 (2014)

[21] Karypis, George, Kumar, Vipin: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing (1998)

[22] Mani, M., Ladd, J., Cain, A., Bush, R.: An assessment of one- and two-equation turbulence models for internal and external flows. In: 28th Fluid Dynamics Conference (1997)

[23] Wang Y T, C.Z.B. Liu G: Summary of the first aeronautical computational fluid dynamics redibility workshop. Acta Aerodynamica Sinica **37**(2), 247–261 (2019)