JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 5, November 2008

General Subdomain Boundary Mapping Procedure for Structured Grid Implicit CFD Parallel Computation

B. Wang* , Z. Hu† , and G.-C. Zha‡
University of Miami, Coral Gables, Florida 33124

DOI: 10.2514/1.35498

In the present work, a general subdomain boundary mapping procedure has been developed for arbitrary topology multiblock structured grids with grid points matched on subdomain boundaries. The interface of two adjacent blocks is uniquely defined according to each local mesh index system, which is specified independently. A pack/unpack procedure based on the definition of the interface is developed to exchange the data in a one-dimensional array to minimize data communication. A secure send/receive procedure is employed to remove the possibility of blocked communication and achieve optimum parallel computation efficiency. Two terms, "order" and "orientation", are introduced as the logics defining the relationship of adjacent blocks. The procedure is applied to parallelize a three-dimensional Navier–Stokes code, which uses an implicit time marching scheme with line Gauss–Seidel iteration. The partitioning of the implicit matrix is done by discarding the corner matrices, which is easily implemented and is shown to have a small negative effect on the convergence rate. The message passing interface protocol is used for communicating the data. The numerical experiments presented in this paper include two- and three-dimensional flows using Reynolds averaged Navier–Stokes equations and a detached eddy simulation with a fifth-order weighted essentially non-oscillatory scheme. Numerical experiments on a message passing interface based computer cluster show that this general mapping algorithm is robust and has high parallel computing efficiency.

I. Introduction

PARALLEL computing is becoming a powerful tool for computational fluid dynamics (CFD) simulation [1,2]. By interconnecting PCs or workstations one can develop a distributed parallel computing system to increase the computing power. Hence there are many efforts to develop parallel computation codes or to convert legacy sequential codes for multi-processor parallel computation.

Multi-block structured grids are widely used for CFD parallel computation because of their numerical efficiency and accuracy. The basic idea is to partition a large domain into multiple smaller subdomains and conduct the computation in the subdomains simultaneously to save wall clock time. Partitioning to multiple subdomains also decreases the difficulties of grid generation around complex configurations as structured grids can be generated

Received 14 November 2007; revision received 30 April 2008; accepted for publication 13 May 2008.

* Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, Miami Wind™, Gzha@miami.edu, AIAA Member.
† Research Scientist, Department of Mechanical and Aerospace Engineering, Miami Wind™, Center for Computational Science, Ph.D.
‡ Associate Professor, Department of Mechanical and Aerospace Engineering, Miami Wind™, Director of Miami Wind™, AIAA Senior Member.

within each subdomain independently. The CFD solvers written for parallel computation are usually based on a single program multiple data (SPMD) strategy, which uses the same CFD code for each subdomain. The subdomain data exchange at domain partitioning boundaries, or inner boundaries, is usually treated as boundary condition [3]. The data are exchanged across the boundaries by a mapping procedure after each iteration [4]. The mapping procedure determines the parallel computation efficiency, accuracy, and robustness.

Various mapping procedures for multiblock structured grids have been developed since parallel computation was introduced to CFD. However, the procedures and implementation are usually ad-hoc and different code developers may use different methods. The complexity of the procedures depends on the complexity of the geometry to be dealt with. For a simple domain partitioning problem, Evans et al. [5] developed a toolkit known as computer aided parallelization tools (CAPTools) to convert a scalar code to a form suitable for parallel implementation with message passing calls. For complex geometries that have different local mesh index systems, it is difficult and time consuming to treat the inner boundary data exchange. One method to avoid this is to use database systems to manage CFD parallel computation. The NSMB package is a well-known CFD solver developed based on a database system called MEM-COM [6,7]. The portable parallel library APPL and a database manager GPAR are used to implement the parallel computation [8]. However, as database systems are generally dependent on computer platforms, portability of the CFD codes are quite limited. In addition, such database systems are not often available for general CFD code developers. Therefore, it is useful and necessary to develop a general mapping procedure for inner boundary data exchange when a new or legacy structured grid CFD code is to be implemented for parallel computation.

The purpose of the present paper is to develop a general subdomain boundary mapping procedure for parallel computation using multiblock structured grid CFD methods. The procedure is designed for generality, simplicity, robustness, efficiency, and portability. The mapping procedure has the following features:

1) the subdomain grid can be independently generated and does not need to be within a larger domain;
2) the mesh index system (i.e. i, j, k direction) can be arbitrary and does not need to follow the right hand rule.

The only condition required is that the grid cells must be matched on the domain boundaries to achieve flux conservation and to preserve the accuracy of the numerical algorithms.

The in-house CFD code [9–11] is implemented for parallel computation using this mapping procedure. Some two-dimensional (2D) and three-dimensional (3D) flows are calculated on an message passing interface (MPI) based computer cluster composed of 200 Intel Xeon 5150 processors with a floating calculation speed of 2.66 GHz to demonstrate the parallel computation efficiency and the accuracy of the results. The numerical experiments show that the present mapping procedure is robust, accurate, and efficient. The mapping procedure is implemented using MPI and is hence as portable as MPI. The implementation is fairly straightforward.

II. Mapping Procedure

A. Inner Boundary and Relationship Between Adjacent Blocks

The index i, j, k is used to express the mesh index of an arbitrary subdomain block. For any structured grid block with the mesh dimensions of $i_{\max} = n1$, $j_{\max} = n2$, $k_{\max} = n3$ we can uniquely define an arbitrary block, face and edge using the two diagonal points at the opposite corner of each entity. For example, the block, face, and edge shown in Fig. 1 can be defined as the following

$$\begin{aligned} &block: \quad start = 1, 1, 1, end = n1, n2, n3 \\ &face\,(i = n1): \quad start = n1, 1, 1, end = n1, n2, n3 \\ &edge\,(i = n1, j = 1): \quad start = n1, 1, 1, end = n1, 1, n3 \end{aligned}$$

where $start$ and end are one-dimensional (1D) arrays with three elements. For simplicity, assume that the blocks are numbered from 1 to n and there are only two halo layers of overlapping grid points for the inner boundaries (Fig. 2). Thus two layers of data need to be communicated at each inner boundary between the adjacent blocks. The number of halo layers can be arbitrary depending on the accuracy order of the scheme to be used. In our code, we have used up to four halo layers for seventh-order weighted essentially non-oscillatory (WENO) scheme [12].

For an arbitrary block p, uniquely to define its inner boundaries and their relationship with the adjacent blocks, the following information is needed:

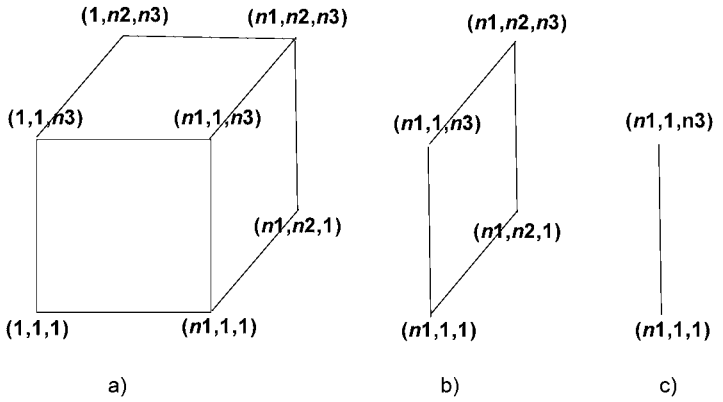

Fig. 1 Definition of a) block, b) face and c) edge.

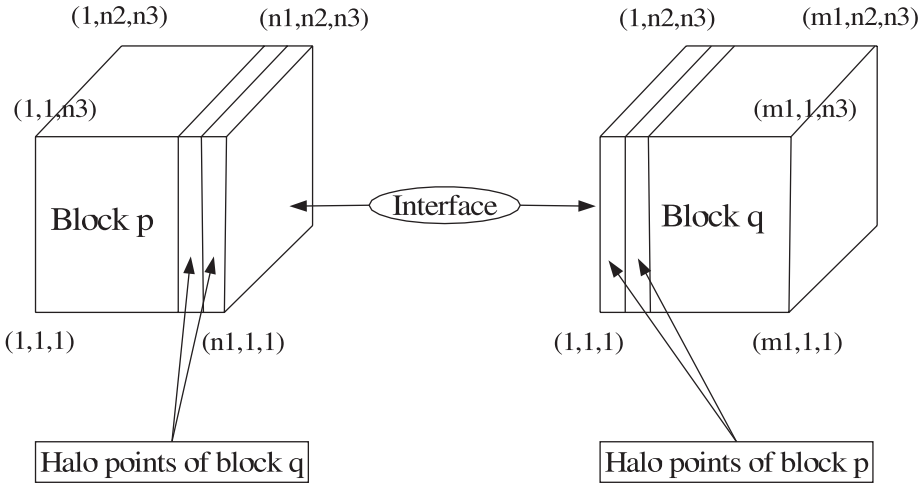

Fig. 2 Inner boundary of two adjacent blocks.

1) the block number of the adjacent block q;
2) the inner boundary definition, that is the two diagonal points of the boundary between block p and q;
3) the relationship of the mesh index system (MIS) between block p and q, which is needed for packing/unpacking the data for exchange.

A term, "order", is introduced to express the MIS of a block. The numbers 1, 2, 3 represent the mesh axis directions of i, j, k respectively for a block. The mesh axis index sequence of the current block p will be always (1, 2, 3). The order of the adjacent block q is always defined based on the mesh axis directions of the current block p. There are six possibilities for the order of block q as shown in Fig. 3. An order is independent of axis directions. For example, in Fig. 3, order 1 of the block q shows two opposite I-axis directions. Hence, for each order there are in total six combinations of index axis directions. In this manner, mesh axis direction does not need to follow the right hand rule. The order is numbered from 1 to 6 as given in Table 1 and Fig. 3. An inner boundary and its relationship with the adjacent block can be uniquely defined by the current block number p, the adjacent block number q, the inner boundary diagonal points of block p and block q, and the order of block q.

The order of block p corresponding to the MIS of block q can be uniquely defined based on the order of q corresponding to block p. The relationship is given in Table 2 based on the relationship indicated in Fig. 3. Table 2 indicates that most of the orders are the same except orders 4 and 5.

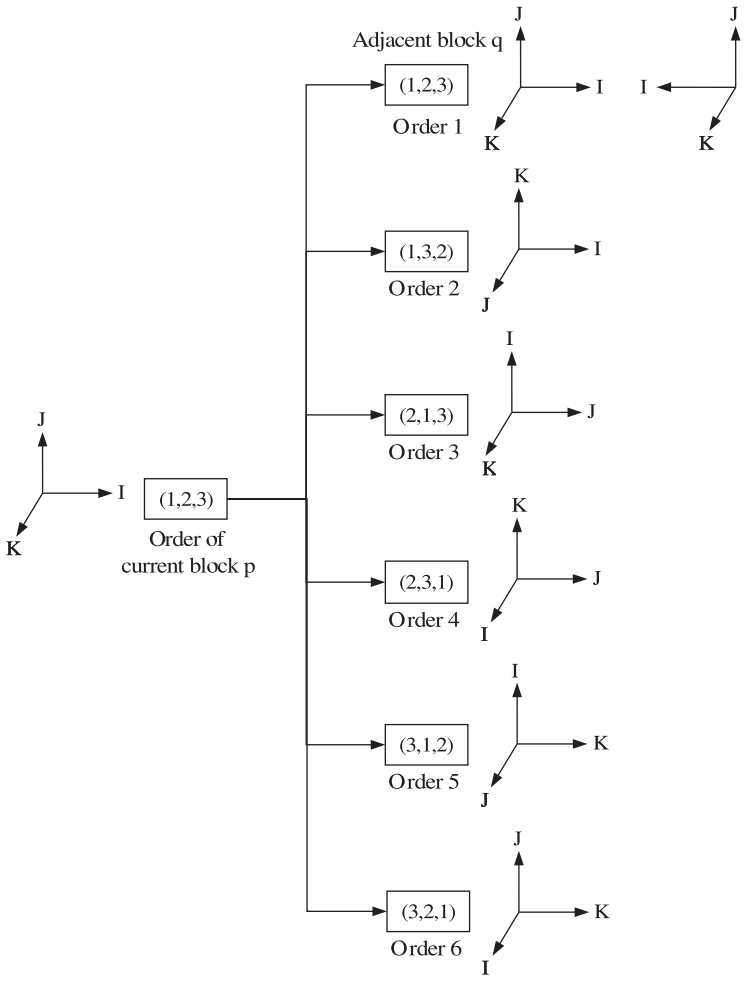

Fig. 3 MIS relationship of adjacent blocks.

Table 1 The order of the block q

The order number of block q	1	2	3	4	5	6
The order of block q	(1,2,3)	(1,3,2)	(2,1,3)	(2,3,1)	(3,1,2)	(3,2,1)

Table 2 The relative relationship of orders for two blocks

Order number of block q	1	2	3	4	5	6
The order of block q based on MIS of block p	(1,2,3)	(1,3,2)	(2,1,3)	(2,3,1)	(3,1,2)	(3,2,1)
The order of block p based on MIS of block q	(1,2,3)	(1,3,2)	(2,1,3)	(3,1,2)	(2,3,1)	(3,2,1)
Order number of block p	1	2	3	5	4	6

With the order information of any two adjacent blocks, an inner boundary only needs to be defined for one block, as the other will take its order according to Table 2. Thus, the probability of making mistakes in defining the inner boundary conditions is minimized.

As an example, the inner boundary (see Fig. 2) between block p and block q can be defined as the following

$$block = p, start = n1, 1, 1, end = n1, n2, n3,$$
$$iblock = q, istart = 1, 1, 1, iend = 1, n2, n3, order = 1, 2, 3$$

where the $block$ and $iblock$ represent the current block number and adjacent block number respectively. The $start$, end, and $istart$, $iend$ are the diagonal points defining the inner boundary. The $start$, end, and $istart$, $iend$ are given according to the local meshindex system (MIS) of the subdomain, which is independent of other subdomains. The order in the above example represents the order of block q corresponding to block p.

B. Pack and Unpack Data Procedures

For CFD parallel computation with multiblock grids the inner boundary data of the flow field are exchanged after each iteration according to the order relationship of the inner boundary defined in the last section. To be efficient, a 1D array is used for data communication. For each block, two operations need to be done for data exchange:

1) pack the inner boundary data into an 1D array and send them to the adjacent block to be used for the halo cells;
2) unpack the 1D array received from the adjacent block and assign it to the halo cells of the inner boundary.

The pack/unpack procedure for a 2D problem is simpler than the 3D one as the 2D boundaries are edges and the 3D boundaries are faces. Hence, we will describe the pack/unpack procedures for 2D and 3D problems separately.

1. 2D Problems

The pack/unpack procedures are implemented using the following rules:

1) inner boundary data are packed into an 1D array in the inward direction of the interface, that is from the outermost edge to the innermost edge;
2) the one-dimensional array received from the adjacent block is unpacked in the reversed (outward) direction.

For example, assuming the number of the halo layer is l and letting $n3 = 1$, the pack/unpack procedure of the 2D problem shown in Fig. 2 for block p is given as the following

Pack:

$i1 = 0$
$do\ i = start(1), start(1) - l + 1, -1$
$do\ j = start(2), end(2)$
$i1 = i1 + 1$
$bcb(i1) = x(i, j)$
$end\ do$
$end\ do$

Unpack:

$i1 = 0$
$do\ i = start(1) + 1, start(1) + l$
$do\ j = start(2), end(2)$
$i1 = i1 + 1$
$x(i, j) = bcb(i1)$
$end\ do$
$end\ do$

where $l = 2$ is used for the example shown in Fig. 2, $start(1) = n1$, $start(2) = 1$, $end(2) = n2$, bcb is the 1D array used for data communication, and x is the data array of the flow field.

2. 3D Problems

For 3D problems the data will be packed from the outermost plane to innermost plane in the current block. The data are unpacked in the reversed direction. However, as the inner boundary is a surface for a 3D problem, the orders of the adjacent blocks are needed to determine the sequence that the data is to be packed and unpacked in the remaining two directions on an inner boundary surface.

Table 3 The relationship between the orientation and faces

Face	$i = 1$	$i = n1$	$j = 1$	$j = n2$	$k = 1$	$k = n3$
Orientation	1	2	3	4	5	6

To define the data packing sequence, it is necessary to introduce another term, "orientation", to specify the orientation and location of the inner boundary interfaces of a block. The orientation is defined based on each inner boundary interface as given in Table 3. Therefore, any inner boundary of a block has an orientation numbered from 1 to 6.

a. Unpacking in 3D First, we need to decide a rule for unpacking for the current block to simplify the pack/unpack procedure. The rule we adopt is that, when the orientation is determined, the data unpacking sequence for the remaining two axis directions is always from the larger axis number to the smaller one. In this way, the unpack procedure will be independent of the order of the adjacent blocks when the orientation is determined. For example, if the orientation of an inner boundary of the current block p is 2, the unpacking is done by the following DO loops:

Unpack:

$i1 = 0$
$do\ i = start(1) + 1, start(1) + l$
 $do\ j = start(2), end(2)$
  $do\ k = start(3), end(3)$
   $i1 = i1 + 1$
   $x(i, j, k) = bcb(i1)$
  $end\ do$
 $end\ do$
$end\ do$

If the orientation of an inner boundary of the current block p is 4, the unpacking is done by the following DO loops:

Unpack:

$i1 = 0$
$do\ j = start(2) + 1, start(2) + l$
 $do\ i = start(1), end(1)$
  $do\ k = start(3), end(3)$
   $i1 = i1 + 1$
   $x(i, j, k) = bcb(i1)$
  $end\ do$
 $end\ do$
$end\ do$

If the orientation of an inner boundary of the current block p is 6, the unpacking is done by the following DO loops:

Unpack:

$i1 = 0$
$do\ k = start(3) + 1, start(3) + l$
 $do\ i = start(1), end(1)$
  $do\ j = start(2), end(2)$
   $i1 = i1 + 1$
   $x(i, j, k) = bcb(i1)$
  $end\ do$
 $end\ do$
$end\ do$

For the odd orientation numbers, the DO loops of the unpack procedure are similar.

b. Packing in 3D To match the unpack procedure, the data packing sequence in the current block must match the MIS of the adjacent block which is defined by the order of the adjacent block.

Assuming that the orientation of the inner boundary of the current block p is 2, the remaining two directions of the data packing for the current block p are then (2, 3). Based on Table 1 and Fig. 3, for order numbers 1, 3, and 5 the corresponding two directions of the adjacent block q are (2, 3), (1, 3) and (1, 2) respectively. To match the data unpacking sequence in the adjacent block q, the packing sequence in the current block then must be from the larger axis number to the smaller one. Specifically, the data packing DO loops are the following:

Pack:

$i1 = 0$
$do\ i = start(1), start(1) - l + 1$
$do\ j = start(2), end(2)$
$do\ k = start(3), end(3)$
$i1 = i1 + 1$
$bcb(i1) = x(i, j, k)$
$end\ do$
$end\ do$
$end\ do$

where i, j and k correspond to 1, 2 and 3 respectively. For order numbers 2, 4, and 6, the corresponding two directions of the adjacent block q are (3, 2), (3, 1), and (2, 1) respectively. To match the data unpacking sequence in the adjacent block q, the packing sequence in the current block must be from the smaller axis number to the larger one, which will have the following DO loops:

Pack:

$i1 = 0$
$do\ i = start(1), start(1) - l + 1$
$do\ k = start(3), end(3)$
$do\ j = start(2), end(2)$
$i1 = i1 + 1$
$bcb(i1) = x(i, j, k)$
$end\ do$
$end\ do$
$end\ do$

The above two types of DO loops cover all the scenarios under orientation 2. For other orientation numbers, the DO loops are similar.

C. Send/Receive Procedure

As shown in Fig. 4, the send/receive procedure for CFD parallel computation is for all blocks to send data to all adjacent blocks first, and then all blocks receive data from all adjacent blocks [13]. This procedure can avoid the communication deadlock, but a communication blockage may occur because of limits to computer buffer space

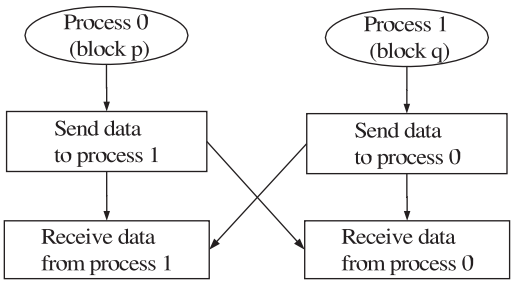

Fig. 4 The send/receive procedure that may create buffer space blockage.

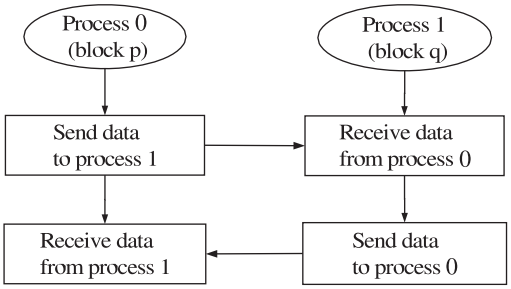

Fig. 5 The secure communication procedure that minimize buffer space usage.

which is used to temporarily save the exchanged data. To avoid communication blockage, a secure send/receive procedure is implemented in the following way.

The basic idea of the secure send/receive procedure is to do the send/receive operations in a pair simultaneously. That is, when a block sends data, the receiving block will receive the data at the same time (Fig. 5). The procedure is implemented based on the following rules for a block:

1) send data to the adjacent blocks with greater block numbers and receive data from the adjacent blocks with smaller block numbers;
2) send data to the adjacent blocks with smaller block numbers and receive data from the adjacent blocks with greater block numbers.

This procedure ensures the one-to-one correspondence between send and receive operations requiring minimal buffer space to avoid communication blockage. Specifically, the procedure is implemented by two DO loops.

Loop 1 (For all interfaces):

1) obtain the block number of the adjacent block.
2) if the block number of the adjacent block is greater than the block number of the current block, send data; otherwise, receive data.

Loop 2 (For all interfaces):

1) obtain the block number of the adjacent block again;
2) If the block number of the adjacent block is smaller than the block number of the current block, send data; otherwise, receive data.

This method is proved to be very effective in removing the communication blockage problem and has a high efficiency of data communication in our numerical experiments.

III. Numerical Algorithms

The in-house Navier–Stokes code [9–11] is converted to have parallel computing capability using the suggested general subdomain boundary mapping procedure. The governing equations are the 3D Navier–Stokes equations. In a generalized coordinate system, the conservative form of the equations is given as the following:

$$\frac{\partial Q}{\partial t}+\frac{\partial \mathbf{E}}{\partial \xi}+\frac{\partial \mathbf{F}}{\partial \eta}+\frac{\partial \mathbf{G}}{\partial \zeta}=\frac{1}{Re}\left(\frac{\partial \mathbf{R}}{\partial \xi}+\frac{\partial \mathbf{S}}{\partial \eta}+\frac{\partial \mathbf{T}}{\partial \zeta}\right) \tag{1}$$

In the above equations

$$Q=\frac{1}{J}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho e\end{bmatrix} \tag{2}$$

$$\mathbf{E}=\begin{bmatrix}\rho U\\ \rho uU+l_x p\\ \rho vU+l_y p\\ \rho wU+l_z p\\ (\rho e+p)U-l_t p\end{bmatrix},\ \mathbf{F}=\begin{bmatrix}\rho V\\ \rho uV+m_x p\\ \rho vV+m_y p\\ \rho wV+m_z p\\ (\rho e+p)V-m_t p\end{bmatrix},\ \mathbf{G}=\begin{bmatrix}\rho W\\ \rho uW+n_x p\\ \rho vW+n_y p\\ \rho wW+n_z p\\ (\rho e+p)W-n_t p\end{bmatrix} \tag{3}$$

$$\mathbf{R}=\begin{bmatrix}0\\ l_k\tau_{xk}\\ l_k\tau_{yk}\\ l_k\tau_{zk}\\ l_k\beta_k\end{bmatrix},\ \mathbf{S}=\begin{bmatrix}0\\ m_k\tau_{xk}\\ m_k\tau_{yk}\\ m_k\tau_{zk}\\ m_k\beta_k\end{bmatrix},\ \mathbf{T}=\begin{bmatrix}0\\ n_k\tau_{xk}\\ n_k\tau_{yk}\\ n_k\tau_{zk}\\ n_k\beta_k\end{bmatrix} \tag{4}$$

where

$$\beta_k=u_i\tau_{ki}-q_k \tag{5}$$

In the equations above, U, V, and W are the contravariant velocities in ξ, η, and ζ directions.

$$\begin{aligned}U&=l_t+\boldsymbol{l}\bullet\boldsymbol{V}=l_t+l_x u+l_y v+l_z w\\ V&=m_t+\boldsymbol{m}\bullet\boldsymbol{V}=m_t+m_x u+m_y v+m_z w\\ W&=n_t+\boldsymbol{n}\bullet\boldsymbol{V}=n_t+n_x u+n_y v+n_z w\end{aligned} \tag{6}$$

where $\boldsymbol{V}=(u, v, w)$ is the velocity vector, $\boldsymbol{l}, \boldsymbol{m}, \boldsymbol{n}$ are the normal vectors on ξ, η, ζ surfaces with their magnitudes equal to the elemental surface area and pointing in the directions of increasing ξ, η, ζ.

$$\boldsymbol{l}=\frac{\nabla\xi}{J}\mathrm{d}\eta\mathrm{d}\zeta,\quad \boldsymbol{m}=\frac{\nabla\eta}{J}\mathrm{d}\xi\mathrm{d}\zeta,\quad \boldsymbol{n}=\frac{\nabla\zeta}{J}\mathrm{d}\xi\mathrm{d}\eta \tag{7}$$

l_t, m_t, n_t stand for the grid moving velocities and are defined as

$$l_t=\frac{\xi_t}{J}\mathrm{d}\eta\mathrm{d}\zeta,\quad m_t=\frac{\eta_t}{J}\mathrm{d}\xi\mathrm{d}\zeta,\quad n_t=\frac{\zeta_t}{J}\mathrm{d}\xi\mathrm{d}\eta \tag{8}$$

When the grid is stationary, $l_t=m_t=z_t=0$.

As $\Delta\xi=\Delta\eta=\Delta\zeta=1$ are used in the current discretization, Eqs. (7) and (8) are then written as

$$\boldsymbol{l}=\frac{\nabla\xi}{J},\quad \boldsymbol{m}=\frac{\nabla\eta}{J},\quad \boldsymbol{n}=\frac{\nabla\zeta}{J} \tag{9}$$

$$l_t=\frac{\xi_t}{J},\quad m_t=\frac{\eta_t}{J},\quad n_t=\frac{\zeta_t}{J} \tag{10}$$

The shear-stress τ_{ik} and total heat flux q_k in Cartesian coordinates can be expressed as

$$\tau_{ik}=(\mu+\mu_t Re)\left[\left(\frac{\partial u_i}{\partial x_k}+\frac{\partial u_k}{\partial x_i}\right)-\frac{2}{3}\delta_{ik}\frac{\partial u_j}{\partial x_j}\right] \tag{11}$$

$$q_k=-\left(\frac{\mu}{Pr}+Re\frac{\mu_t}{Pr_t}\right)\frac{\partial T}{\partial x_k} \tag{12}$$

where Pr is the Prandtl number, Pr_t is the turbulent Prandtl number, μ is the molecular viscosity determined by the Sutherland law, and μ_t is the turbulent viscosity. The Baldwin–Lomax turbulence model [14] or the Spalart–Allmaras one equation turbulence model [15] is used to calculate the turbulent eddy viscosity. The finite volume method is used to discrete the governing equations. To achieve a high convergence rate the implicit time marching scheme is used with the unfactored Gauss–Seidel line relaxation.

Using the finite volume method, the implicit semidiscretized form of the Reynolds averaged Navier–Stokes equations can be written as:

$$\frac{\Delta V}{\Delta t}\left(\mathbf{U}^{n+1}-\mathbf{U}^{n}\right)+\left(\mathbf{E}_{i+\frac{1}{2}}-\mathbf{E}_{i-\frac{1}{2}}\right)^{n+1}+\left(\mathbf{F}_{j+\frac{1}{2}}-\mathbf{F}_{j-\frac{1}{2}}\right)^{n+1}+\left(\mathbf{G}_{k+\frac{1}{2}}-\mathbf{G}_{k-\frac{1}{2}}\right)^{n+1}$$
$$=\left(\mathbf{R}_{i+\frac{1}{2}}-\mathbf{R}_{i-\frac{1}{2}}\right)^{n+1}+\left(\mathbf{S}_{j+\frac{1}{2}}-\mathbf{S}_{j-\frac{1}{2}}\right)^{n+1}+\left(\mathbf{T}_{k+\frac{1}{2}}-\mathbf{T}_{k-\frac{1}{2}}\right)^{n+1} \tag{13}$$

where n and $n+1$ are two sequential time levels, which have a time interval of Δt. ΔV is the volume of the control volume.

The inviscid fluxes $\mathbf{E}, \mathbf{F}, \mathbf{G}$ and viscous fluxes $\mathbf{R}, \mathbf{S}, \mathbf{T}$ at the control volume left $(-\frac{1}{2})$ and right $(+\frac{1}{2})$ interfaces are computed based on the flow variables at the current cell and its neighboring cells. The inviscid fluxes are evaluated following the characteristic directions by using upwind schemes [16–18] with a third-order MUSCL interpolation [19] or a fifth-order WENO scheme [20]. The viscous fluxes are evaluated by using a second-order or fourth-order central differencing scheme. A first-order scheme is used for the implicit inviscid terms to enhance diagonal dominance. Finally, the implicit discretized form of Eq. (13) is written as

$$\bar{B}\Delta\mathbf{U}^{n+1}_{i,j,k}+A^{+}\Delta\mathbf{U}^{n+1}_{i+1,j,k}+A^{-}\Delta\mathbf{U}^{n+1}_{i-1,j,k}+B^{+}\Delta\mathbf{U}^{n+1}_{i,j+1,k}$$
$$+B^{-}\Delta\mathbf{U}^{n+1}_{i,j-1,k}+C^{+}\Delta\mathbf{U}^{n+1}_{i,j,k+1}+C^{-}\Delta\mathbf{U}^{n+1}_{i,j,k-1}=\mathbf{RHS}^{n} \tag{14}$$

Gauss–Seidel line iteration is applied in each direction (i, j, k) and is swept one time step forward and backward in each direction. For example, the equation for Gauss–Seidel iteration following lines along direction i with the index from small to large is written as

$$B^{-}\Delta\mathbf{U}^{n+1}_{i,j-1,k}+\bar{B}\Delta\mathbf{U}^{n+1}_{i,j,k}+B^{+}\Delta\mathbf{U}^{n+1}_{i,j+1,k}=\mathbf{RHS}' \tag{15}$$

where

$$\mathbf{RHS}'=\mathbf{RHS}^{n}-A^{+}\Delta\mathbf{U}^{n}_{i+1,j,k}-A^{-}\Delta\mathbf{U}^{n+1}_{i-1,j,k}-C^{+}\Delta\mathbf{U}^{n}_{i,j,k+1}-C^{-}\Delta\mathbf{U}^{n+1}_{i,j,k-1} \tag{16}$$

Eq. (15) can be written in a matrix form (ignore the dash lines here)

$$\left[\begin{array}{ccccc|ccc}
\bar{B}_1 & B_1^{+} & & & & & & \\
B_2^{-} & \bar{B}_2 & B_2^{+} & & & & 0 & \\
 & & \ddots & & & & & \\
 & & B_m^{-} & \bar{B}_m & & B_m^{+} & & \\
-- & -- & -- & -- & + & -- & -- & -- \\
 & & & B_{m+1}^{-} & & \bar{B}_{m+1} & B_{m+1}^{+} & \\
 & 0 & & & & & \ddots & \\
 & & & & & & B_n^{-} & \bar{B}_n
\end{array}\right]\left[\begin{array}{c}\Delta\mathbf{U}_1\\ \Delta\mathbf{U}_2\\ \vdots\\ \Delta\mathbf{U}_m\\ --\\ \Delta\mathbf{U}_{m+1}\\ \vdots\\ \Delta\mathbf{U}_n\end{array}\right]=\left[\begin{array}{c}\mathbf{RHS}_1'\\ \mathbf{RHS}_2'\\ \vdots\\ \mathbf{RHS}_m'\\ --\\ \mathbf{RHS}_{m+1}'\\ \vdots\\ \mathbf{RHS}_n'\end{array}\right] \tag{17}$$

When using multiblock parallel computation, for example two blocks, Eq. (17) will be partitioned into two submatrices as indicated by the dash lines in Eq. (17). The variables $\Delta\mathbf{U}_1 \rightarrow \Delta\mathbf{U}_m$ and $\Delta\mathbf{U}_{m+1} \rightarrow \Delta\mathbf{U}_n$ are solved on two separate processors by conducting the matrix inversion on each submatrix. A simple treatment is to discard the corner matrices B_m^{+} on the first submatrix and B_{m+1}^{-} on the second submatrix. They are the coefficients computed based on the variables from the adjacent subdomain. In the present work, these two coefficients are treated as zero

in the implicit solver for the subdomain computations. The two subdomain matrix systems obtained are

$$\begin{bmatrix} \bar{B}_1 & B_1^+ & & \\ B_2^- & \bar{B}_2 & B_2^+ & \\ & & \ddots & \\ & & B_m^- & \bar{B}_m \end{bmatrix} \begin{bmatrix} \Delta\mathbf{U}_1 \\ \Delta\mathbf{U}_2 \\ \vdots \\ \Delta\mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_1' \\ \mathbf{RHS}_2' \\ \vdots \\ \mathbf{RHS}_m' \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} \bar{B}_{m+1} & B_{m+1}^+ & & \\ B_{m+2}^- & \bar{B}_{m+2} & B_{m+2}^+ & \\ & & \ddots & \\ & & B_n^- & \bar{B}_n \end{bmatrix} \begin{bmatrix} \Delta\mathbf{U}_{m+1} \\ \Delta\mathbf{U}_{m+2} \\ \vdots \\ \Delta\mathbf{U}_n \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_{m+1}' \\ \mathbf{RHS}_{m+2}' \\ \vdots \\ \mathbf{RHS}_n' \end{bmatrix} \tag{19}$$

The advantages of discarding the matrices B_m^+ and B_{m+1}^- is avoiding exchanging the matrices across the subdomain boundaries which reduces communication time. The disadvantage is that it does not preserve the exact matrices as does in the single processor computation which may affect the convergence efficiency. However, the computation experiments indicate that the slowing of convergence owing to this non-exact treatment is small, particularly when the mesh size is large. It should be pointed out that discarding the matrices B_{m+1}^- and B_m^+ will not affect the accuracy of the solution when it is converged. This is because the variables at the subdomain boundaries are exchanged exactly matching the result of the single domain calculation to determine the right-hand side (RHS) of Eqs. (14), which determines the accuracy of the solutions.

IV. Results and Discussion

A. RAE2822 Transonic Airfoil

The RAE2822 transonic airfoil is calculated to examine the accuracy of the CFD solver and parallel computation efficiency for 2D problems. The multiblock grids (Fig. 6) are obtained by partitioning a single block O-grid with dimensions of 259×56. The Reynolds number is 6.5×10^6 based on the chord length. The Mach number is 0.729. The angle of attack is 2.31°. Figure 7 shows the convergence history with different number of processors (blocks) from 1 through 8. The convergence rate with multiple processors is slightly affected by the approximate implicit treatment at the subdomain boundaries. Figure 8 presents the comparison of pressure coefficients between the experiment and computation with 1 and 8 processors. The computed results are identical and agree very well with the experiment. Excellent speed up and efficiency for parallel computation are obtained as shown in Table 4 and Fig. 9, which indicate that a super-linear scalability is achieved up to eight processors.

B. Co-Flow Jet Airfoil

A co-flow jet (CFJ) flow control airfoil [21–23] shown in Fig. 10 cannot be calculated using a single block grid. A CFJ airfoil has a jet injection from the slot near the leading edge and the same amount of jet mass flow is sucked in near the trailing edge. The CFJ airfoil is partitioned to five block grids as shown in Fig. 10. The dimensions of the blocks are 33×49, 97×97, 17×97, 23×97 and 59×193, respectively. The free-stream Mach number is 0.1024. The Reynolds number is 380 000 based on the chord length. Figure 11 shows the contours of the Mach number at angle of attack (AoA) = 10°. Figure 12 shows the C_L vs AoA compared with experiment. The jet effect is included in the C_L calculation [23]. The predicted lift coefficient agrees reasonably well with experiment at AoA less than 20°. At high AoA, the lift is significantly under the predicted value. The reason may be that the Reynolds averaged Navier-Stokes (RANS) model cannot predict the turbulence mixing well at high AoA, which may have a large vortex structure.

C. Transonic ONERA M6 Wing

This case is to examine the CFD solver accuracy and parallel computation efficiency for 3D problems. The multiblock grids are obtained by partitioning a single block O–H-grid with the dimensions of $145 \times 61 \times 41$ (Fig. 13). The Mach number is 0.8395. The Reynolds number is 1.97×10^7 based on the averaged chord. The angle of attack is 0°.

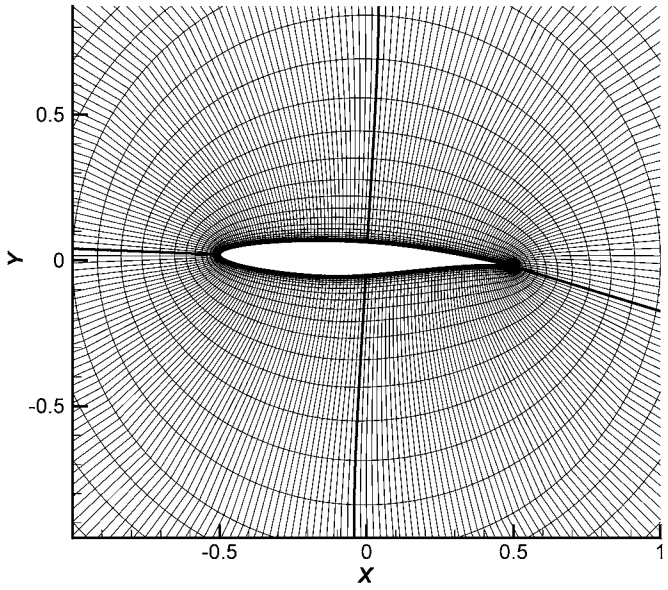

Fig. 6 Zoomed 4-Block grids for RAE2822 airfoil.

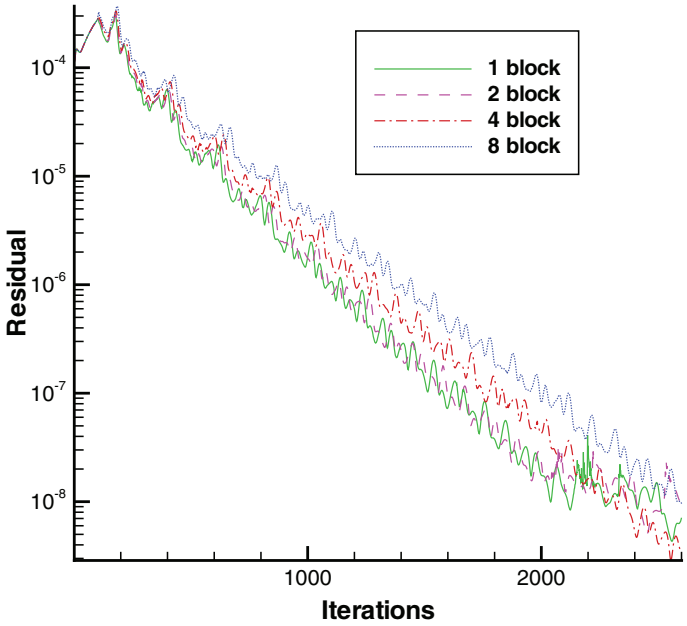

Fig. 7 The L2 residual convergence history of RAE2822 airfoil.

Figure 14 presents the comparison of surface pressure distributions between the experiment and computation at different span-wise sections. The location of $z/b = 0.2$ is near the root and $z/b = 0.99$ is near the tip of the wing. The computation results agree well with the experimental data.

Again, excellent speed up and efficiency for the parallel computation are obtained as shown in Table 5 and Fig. 15.

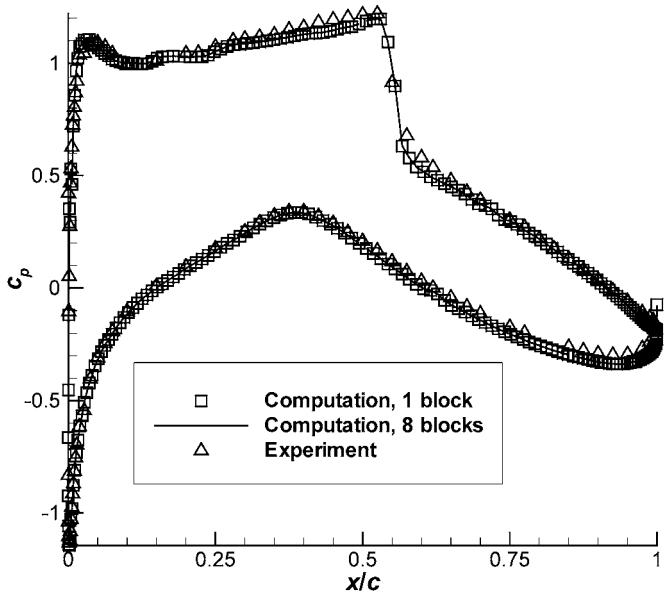

Fig. 8 The surface distribution of pressure coefficient of the RAE2822 airfoil.

Table 4 The parallel computing performance for 2D RAE2822 airfoil

Number of nodes	1	2	4	8
Time (s/step)	0.355	0.161	0.077	0.039
Speed up		2.205	4.610	9.103
Efficiency (%)		110.3	115.3	113.8

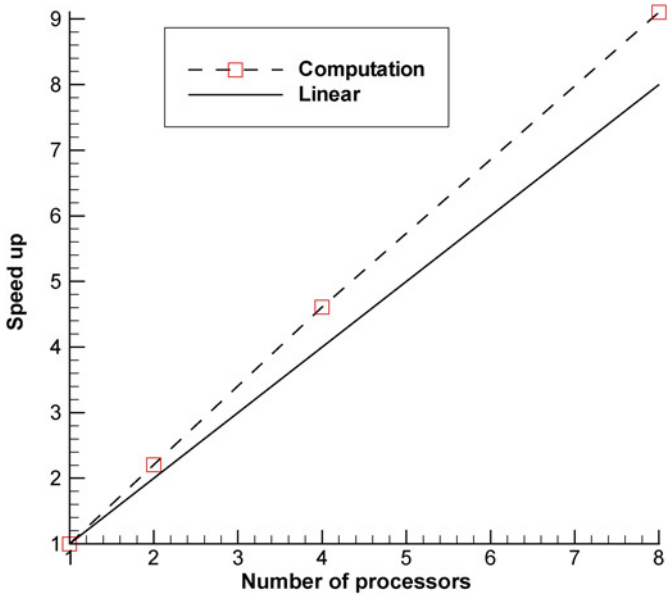

Fig. 9 Speedup of parallel computation for RAE2822 airfoil.

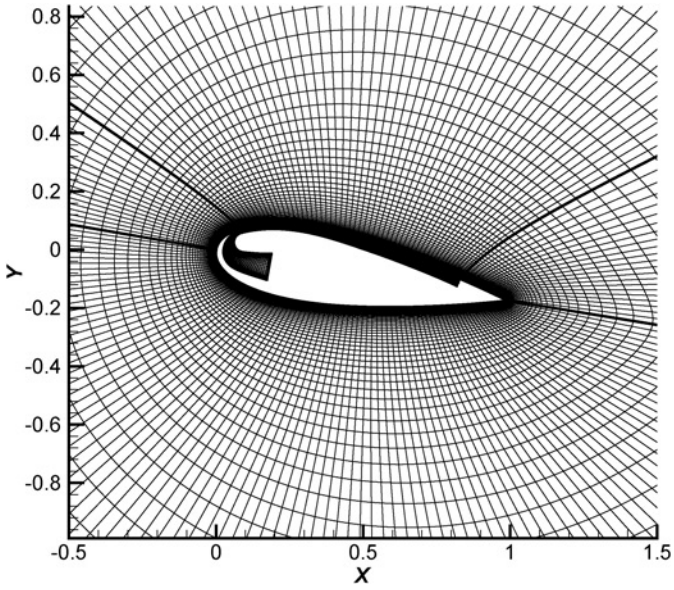

Fig. 10 Zoomed five-block grids for the co-flow jet airfoil, CFJ0025-065-196 [21].

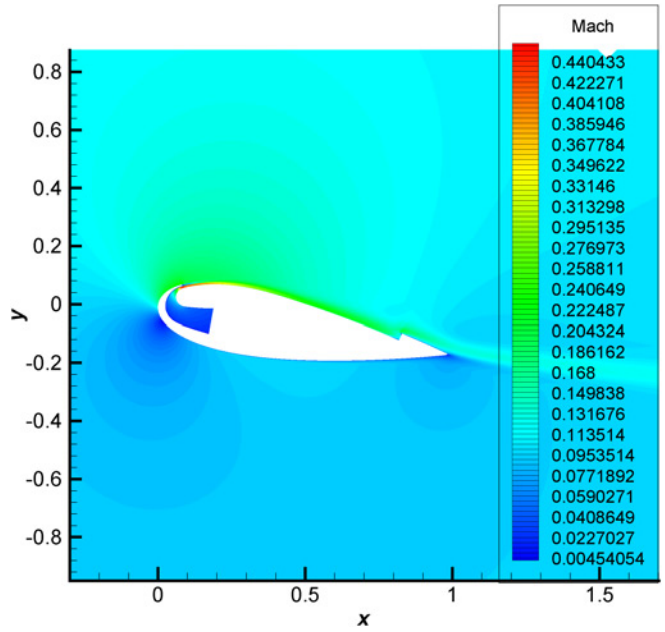

Fig. 11 The contour of the Mach number of the CFJ0025-065-196 airfoil.

D. "Engineless" 3D Co-Flow Jet Aircraft

The "Engineless" CFJ aircraft is a flying wing using a CFJ airfoil [21–23]. Again, a configuration with jet slots such as this can not be simulated using a single block grid. In total, nine block grids are used in the parallel computation(see Fig. 16). The Reynolds number is 2×10^6 and the Mach number is 0.1. The range of angle of attack varies from $-5°$ to $45°$.

Figure 17 presents the coefficient of lift vs AoA and shows that flow separates at a very high angle of attack, about $35°$. Figure 18 shows the drag polar plot. It indicates that the drag coefficient remains negative within a range

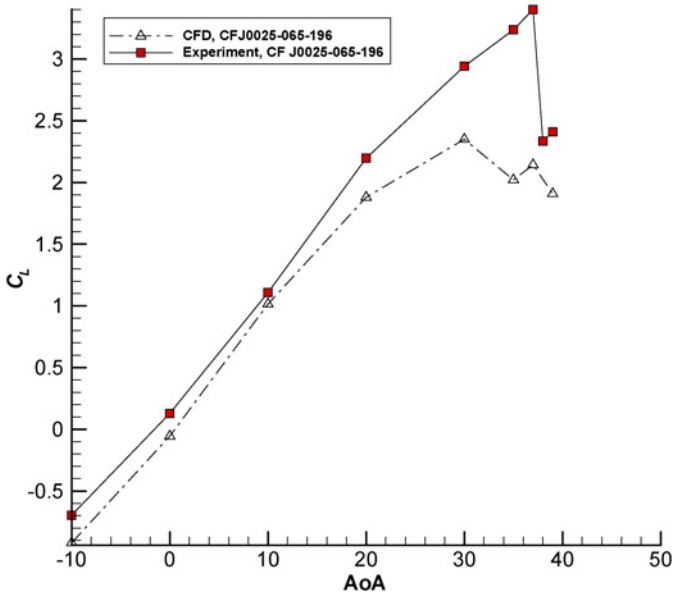

Fig. 12 Lift coefficients of CFJ airfoil CFJ0025-065-196 at different angle of attack.

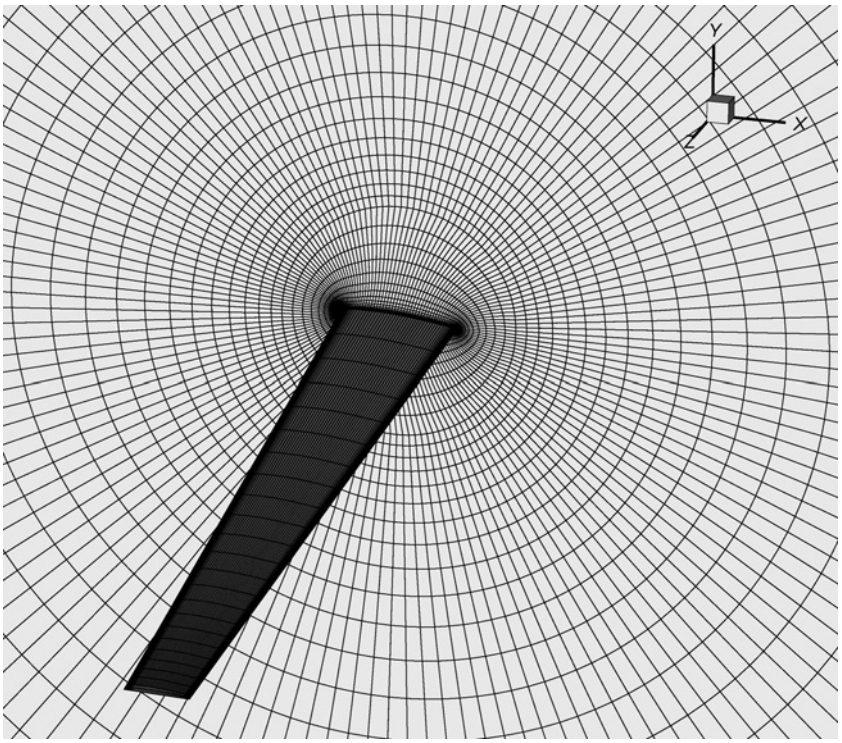

Fig. 13 The 3D M6 wing mesh.

of angles of attack from $-5°$ to $10°$ for the "Engineless" CFJ aircraft. After this point, the form drag is large enough to offset the thrust produced by the CFJ device.

Figure 19 presents the pressure contours and shows that the low pressure of the leading edge suction effect contributes significantly to generate the thrust.

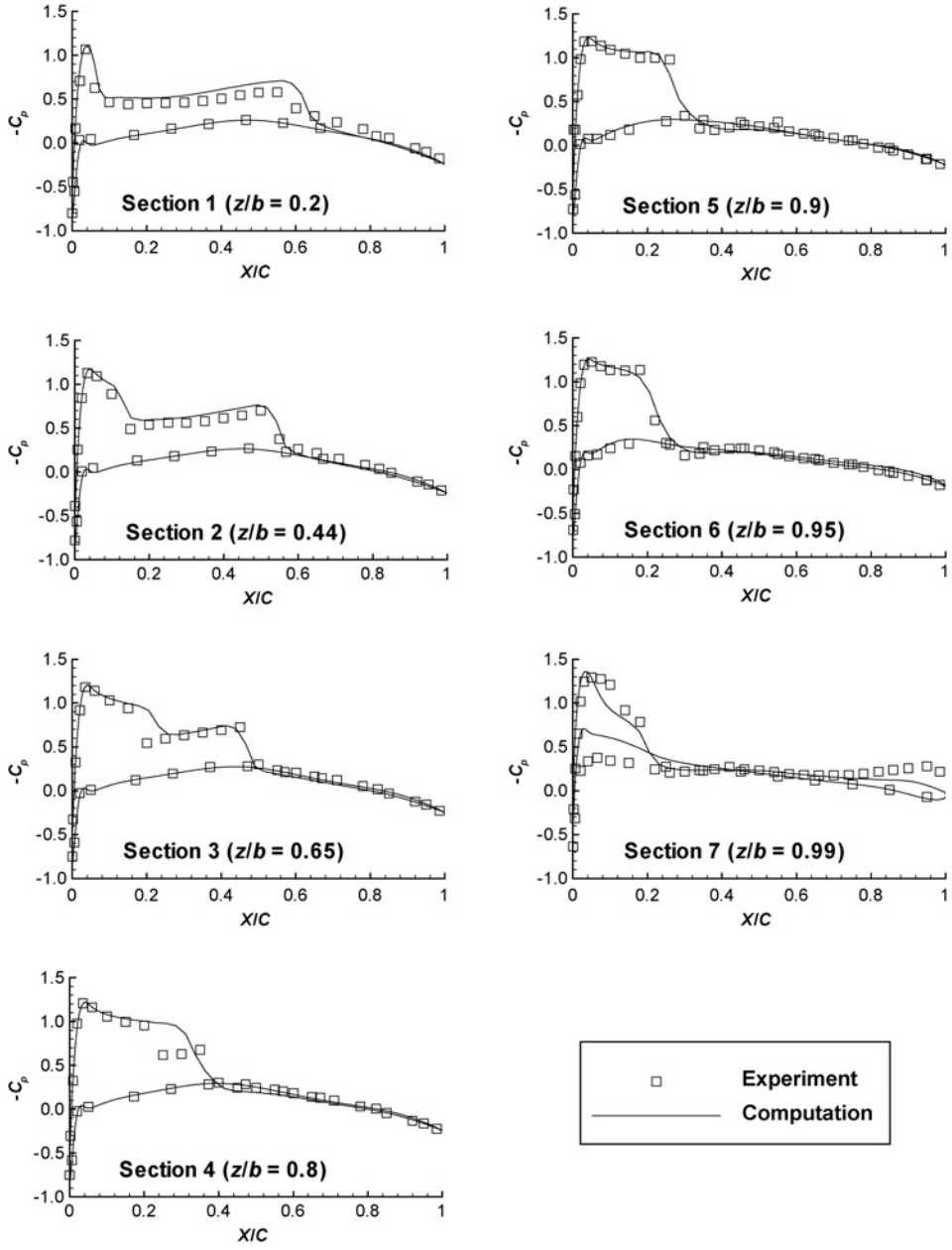

Fig. 14 Surface pressure distribution of M6 wing at different span-wise locations.

Table 5 The parallel computing performance for 3D M6 wing

Number of nodes	1	2	4	8	16
Time (s/step)	10.747	4.912	2.484	1.298	0.697
Speed up		2.188	4.326	8.280	15.419
Efficiency (%)		109.4	108.2	103.5	96.37

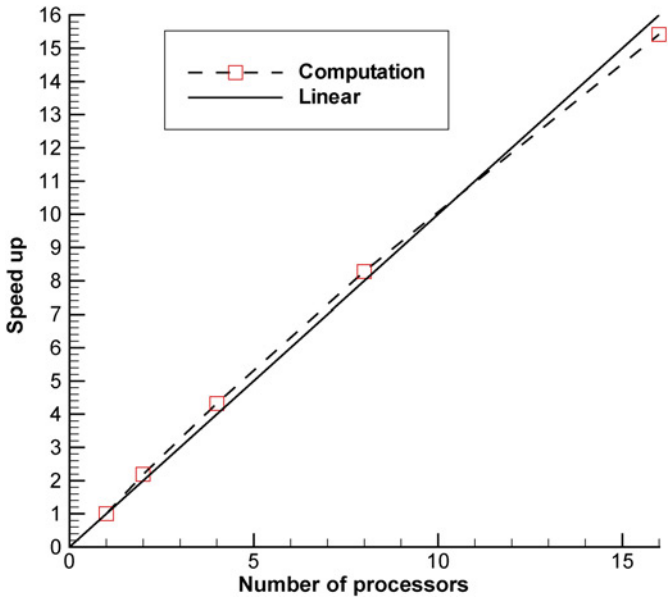

Fig. 15 Speedup of the parallel computation for M6 wing.

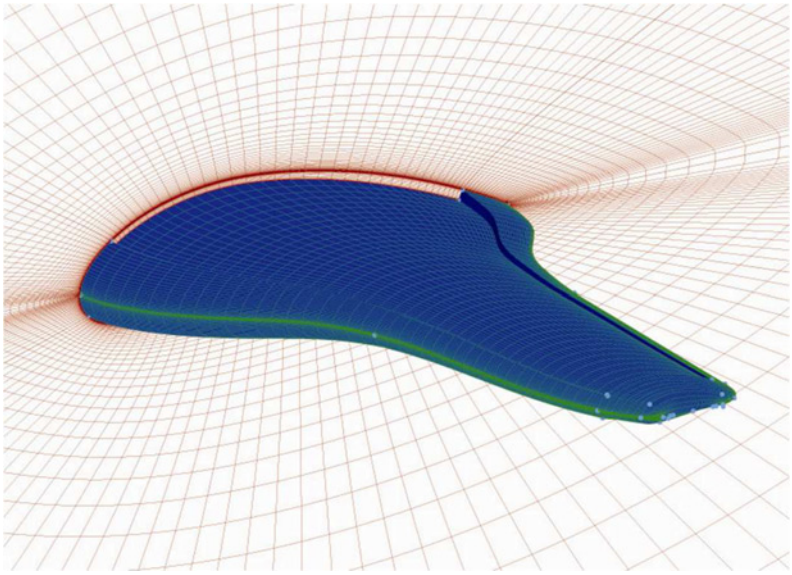

Fig. 16 Zoomed nine-block grids for the CFJ aircraft.

E. Detached Eddy Simulations of a Circular Cylinder

The flow around a cylinder at a Reynolds number 3900 is calculated by detached eddy simulation (DES) using the proposed mapping procedure. DES is an unsteady 3D flow calculation so it is very CPU intensive and parallel computation is necessary. The Mach number is 0.2. The span wise length is $2\pi D$, where D is the cylinder diameter. The computation grid is composed of three blocks. The dimensions of each block are $(41 \times 81 \times 65)$ (see Fig. 20). A fifth order WENO scheme is used in the computation and hence three halo layers are needed. The dual time stepping method for unsteady problems is used [24] and a nondimensional time step of 0.01 is adopted for this calculation.

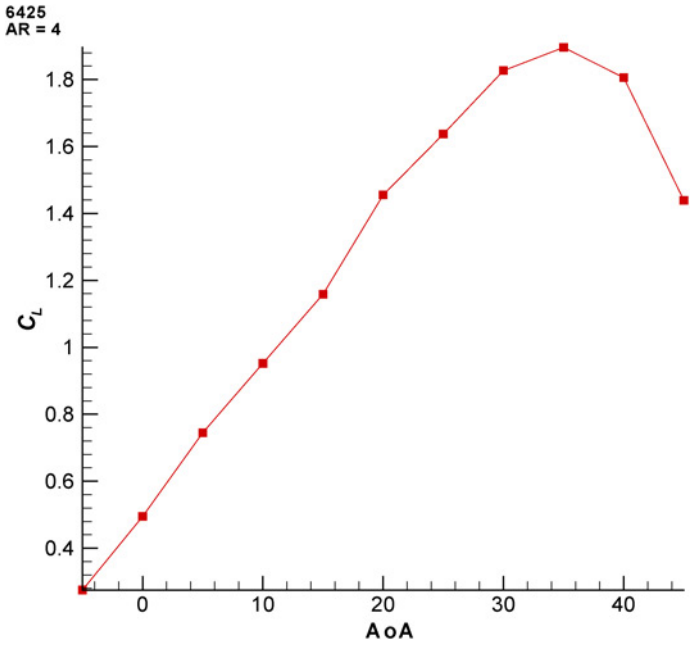

Fig. 17 Coefficient of lift vs angle of attack for CFJ aircraft.

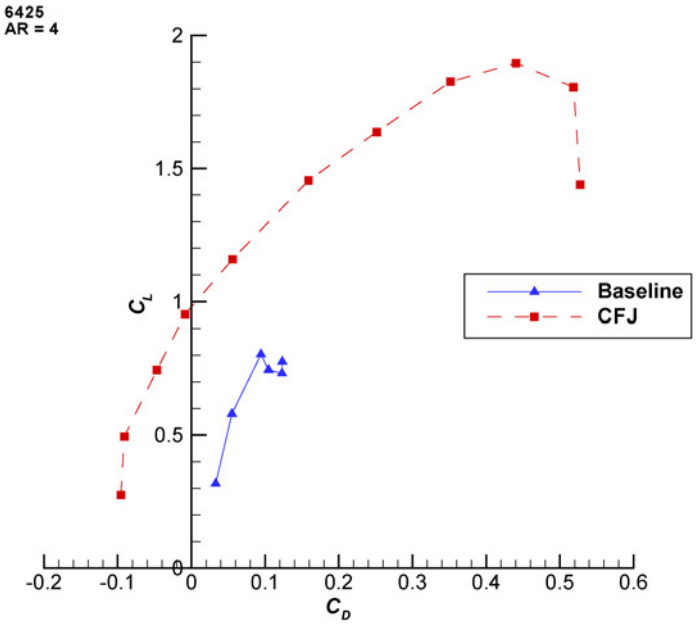

Fig. 18 Drag polar compared with baseline for CFJ aircraft.

The computation begins from a uniform flow field. All the results are time-averaged from $100T$ to $300T$, where T is the flow characteristic time.

Figure 21 shows the mean pressure coefficients on the cylinder surface. The computed mean pressure coefficient agrees very well with experiment at $0 \leqslant \theta \leqslant 60°$. The present result is better than the large eddy simulation (LES) result of Kasliwal et al. [25] in this region. In the region from 60° to 180°, the computed pressure lies among the experimental results.

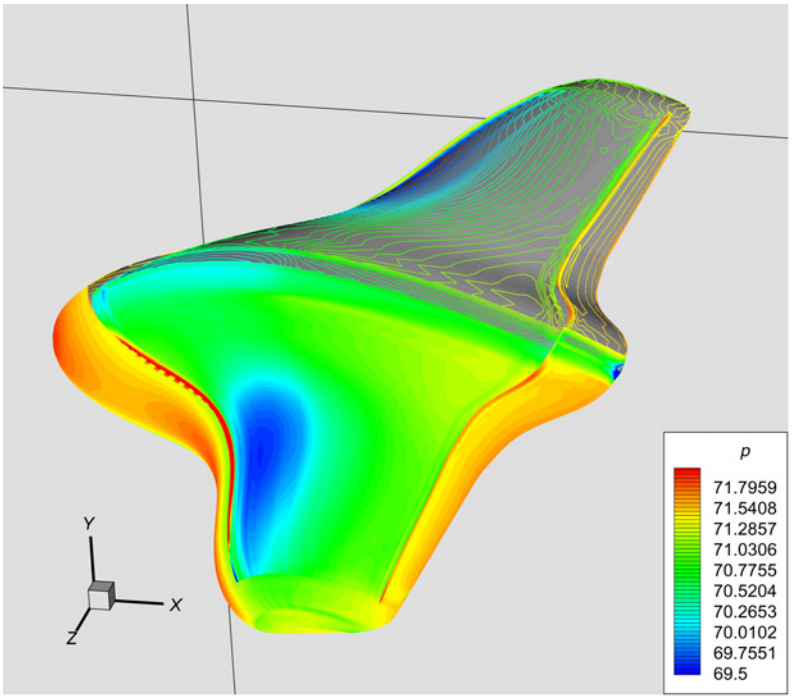

Fig. 19 Surface pressure contours at AoA = 0° for CFJ aircraft.

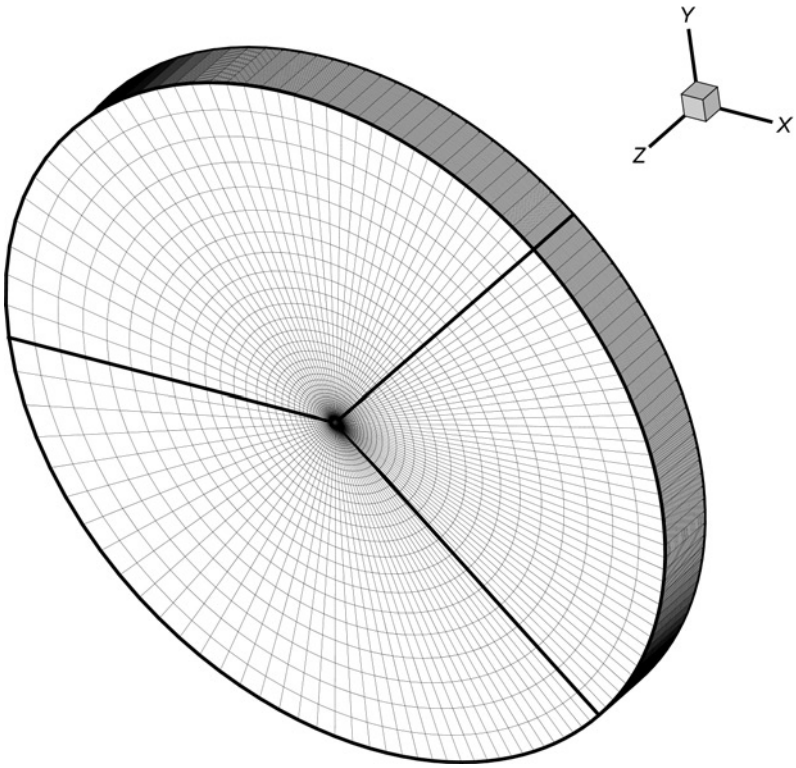

Fig. 20 Three-block grids for cylinder.

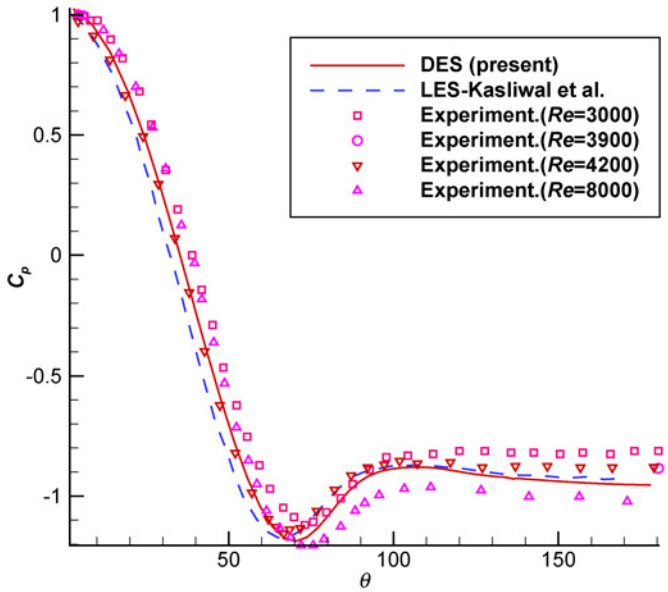

Fig. 21 Mean pressure coefficient variation on the surface of the cylinder.

Figure 22 is the averaged mean stream-wise velocity on the centerline in the wake of the cylinder. The present result agrees better with experiment [26] than those of LES conducted by Kasliwal et al. [25] and Kravchenko and Moin [27].

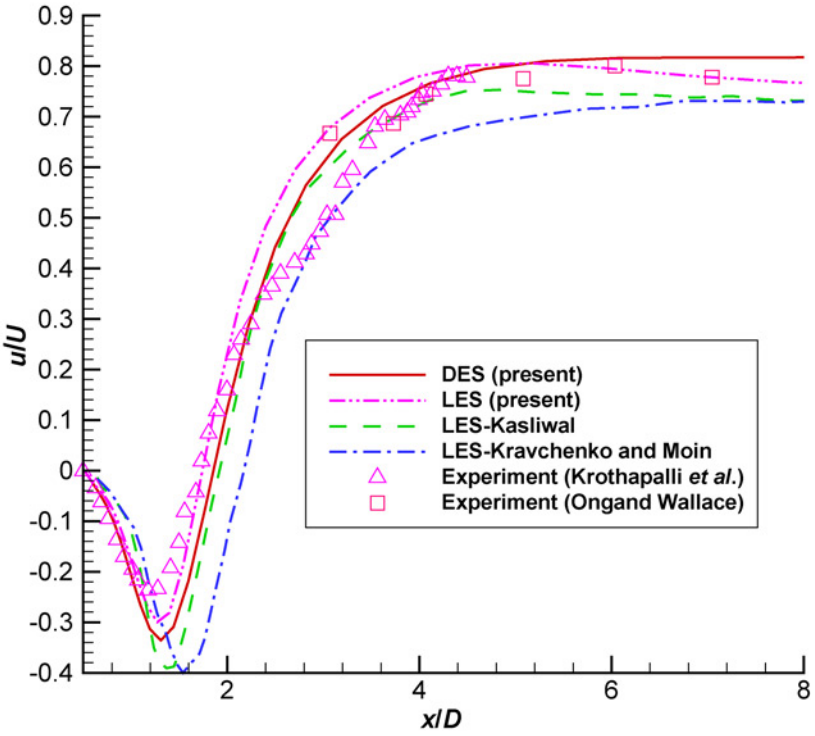

Fig. 22 Stream-wise velocity in the wake at $y/D = 0$.

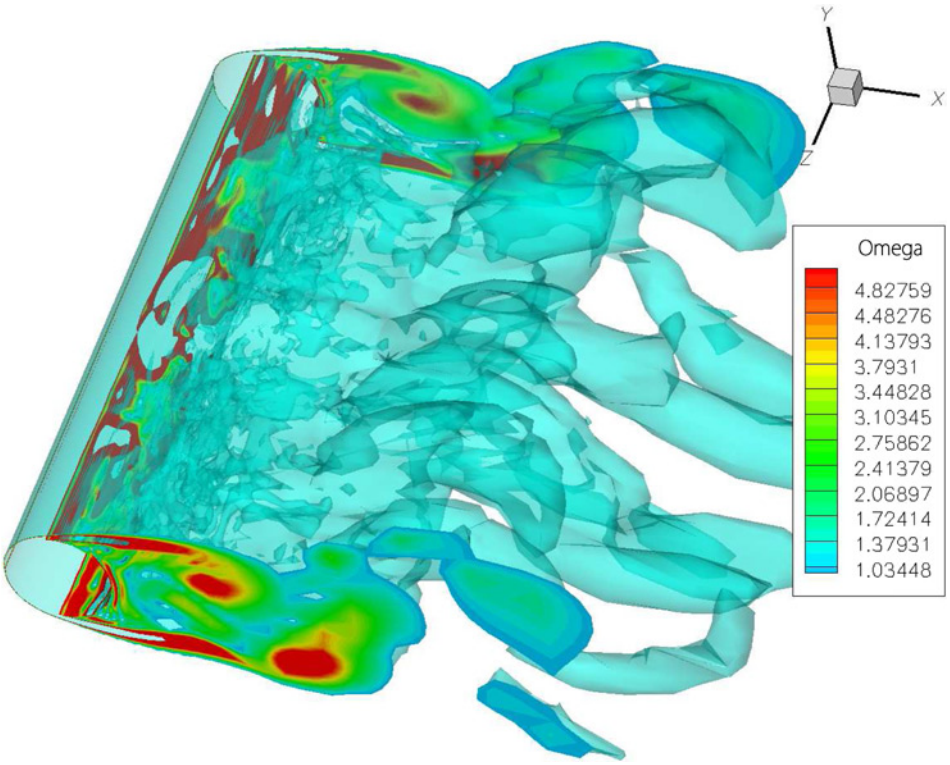

Fig. 23 Contours of instantaneous vorticity at $t = 300T$.

An instantaneous vorticity magnitude at $300T$ is shown in Fig. 23. Figure 23 shows that DES resolves some small-scale vortex structures.

V. Conclusions

This paper has developed a general subdomain boundary mapping procedure for arbitrary topology multiblock structured grid parallel CFD computation. The grid points are required to match on the subdomain boundaries to preserve conservation and accuracy. The mesh index system of a subdomain block can be arbitrary. The terms block order and domain boundary orientation are introduced. These terms uniquely define the relationship of the mesh index systems between the adjacent blocks for data exchange. A pack/unpack procedure is developed to exchange the data in 1D arrays to minimize the amount of data communication. A secure send/receive procedure is adopted to minimize buffer space use and essentially to remove the possibility of communication blockage.

The procedure is applied to parallelize an inhouse implicit 3D Navier–Stokes code. The partitioning of the implicit block tridiagonal matrix inversion is done by discarding the corner matrices. The numerical experiments indicate that the effect of this implicit treatment on the convergence rate is small. The MPI protocol is used for the data communication. The code is portable to any platform as long as the MPI is available. The numerical experiments including 2D and 3D RANS and DES simulations show that the general mapping procedure developed in this paper is robust and has high parallel computational efficiency.

Acknowledgment

This work is supported by AFOSR Grant FA9550-06-1-0198 monitored by Fariba Fahroo and by Miami Wind™ Research Center at University of Miami.

References

[1] Tu, S., and Aliabadi, S., "A Robust Parallel Implicit Finite Volume Solver for High-Speed Compressible Flows," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 10–13 Jan. 2005, AIAA, Reston, VA, 2005, AIAA Paper 2005-1396.

[2] Cai J., Tsai, H.M., and Liu, F., "A Parallel Viscous Flow Solver on Multi-Block Overset Grids," *Journal of Computers and Fluids*, Vol. 35, No. 10, 2006, pp. 1290–1301.
doi: 10.1016/j.compfluid.2005.02.006

[3] Ohta, T., "An Object-Oriented Programming Paradigm for Parallel Computational Fluid Dynamics on Memory Distributed Parallel Computers," *Parallel Computational Fluid Dynamics*, edited by D. R. Emerson, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, Elsevier Publishing Co., 1998.

[4] Allwright, S., "Multiblock Techniques for Transonic Flow Computation about Complex Aircraft Configuration," *Numerical Methods for Fluid Dynamics, III*, edited by K. W. Morton and M. J. Baines, Calarendon Press, 1988.

[5] Evans, E.W., Johnson, S.P., Leggett, P.F., and Cross M., "Automatic Generation of Multi-Dimensionally Partitioned Parallel CFD Code in A Parallelisation Tool," *Parallel Computational Fluid Dynamics*, edited by D. R. Emerson, A. Ecer, J. Periaux, N. Satofuka and P. Fox, Elsevier Publishing Co., 1998.

[6] Merazzi, S., "MEM-COM An Integrated Memory and Data Management, System Mem-Com User Manual Version 6.0," SMR TR-5060, SMR Corporation, March 1991.

[7] Legland, P., Vos, J.B., Van Kemenade, V., and Ytterstrom, A., "NSMB: A Modular Navier–Stokes Multiblock Code for CFD," AIAA Reston, VA, 1995, AIAA Paper 1995-0568.

[8] Ecer, A., Akay, H.-U., Kemle, W.-B., Wang, H., and Ercoskun, D., "Parallel Computation of Fluid Dynamics Problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 112, No. 1–4, 1994, pp. 91–108.
doi: 10.1016/0045-7825(94)90020-5

[9] Zha, G.-C., and Hu, Z.-J., "Calculation of Transonic Internal Flows Using an Efficient High Resolution Upwind Scheme," *AIAA Journal*, Vol. 42, No. 2, 2004, pp. 205–214.
doi: 10.2514/1.3113

[10] Hu, Z.-J., and Zha, G.-C., "Calculations of 3D Compressible Using an Efficient Low Diffusion Upwind Scheme," *International Journal for Numerical Methods in Fluids*, Vol. 47, Nov. 2004, pp. 253–269.
doi: 10.1002/fld.810

[11] Chen, X.-Y., Zha, G.-C., and Yang, M.-T., "Numerical Simulation of 3D Wing Flutter with Fully Coupled Fluid-Structural Interration," *Journal of Computers and Fluids*, Vol. 36, No. 5, 2007, pp. 856–867.

[12] Shen, Y.-Q., and Zha, G.-C., "A Robust Seventh Order WENO Scheme and Its Applications," *46th AIAA Aerospace Sciences Meeting*, AIAA, Reston, VA, 2008, AIAA Paper 2008-0757.
doi: 10.1016/j.compfluid.2006.08.005

[13] di Serafino D., "A Parallel Implementation of A Multigrid Multiblock Euler Solver on Distributed Memory Machines," *Parallel Computing*, Vol. 23, No. 13, 1997, pp. 2095–2113.

[14] Baldwin, B., and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA, Reston, VA, 1978, AIAA Paper 78-257.

[15] Spalart, P., and Allmaras, S., "A One-equation Turbulence Model for Aerodynamic Flows," AIAA, Reston, VA, 1992, AIAA-92-0439.

[16] Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.
doi: 10.1016/0021-9991(81)90128-5

[17] Van Leer, B., "Flux-Vector Splitting for the Euler Equations," *Lecture Note in Physics*, Vol. 170, 1952, pp. 507–512.

[18] Zha, G.-C., "A Low Diffusion Efficient Upwind Scheme," *AIAA Journal*, Vol. 43, No. 5, 2005, pp. 1137–1140.
doi: 10.2514/1.7726

[19] Van Leer, B., "Towards the Ultimate Conservative Difference Scheme, III," *Journal of Computational Physics*, Vol. 23, No. 3, 1977, pp. 263–275.

[20] Shen, Y.-Q., Wang, B.-Y., and Zha, G.-C., "Implicit WENO Scheme and High Order Viscous Formulas for Compressible Flows," *25th AIAA Applied Aerodynamics Conference*, 25–28 June 2007, AIAA, Reston, VA, 2007, AIAA Paper 2007-4431.

[21] Zha, G.-C., Paxton, C., Conley, A., and Wells, A., "High Performance Airfoil Using Co-Flow Jet Flow Control," *AIAA Journal*, Vol. 45, No. 8, 2007, pp. 2087–2090.
doi: 10.2514/1.20926

[22] Zha, G.-C., Paxton, C., Conley, A., Wells, A., and Carroll, B., "Effect of Injection Slot Size on High Performance Co-Flow Jet Airfoil," *AIAA Journal of Aircraft*, Vol. 43, No. 4, 2006, pp. 987–995.

[23] Zha, G.-C., Gao, W., and Paxton, C., "Jet Effects on Co-Flow Jet Airfoil Performance," *AIAA Journal*, Vol. 45, No. 6, 2007, pp. 1222–1231.
doi: 10.2514/1.23995

[24] Wang, B.-Y., Zha, G.-C., and Shen, Y.-Q., "Detached Eddy Simulations of a Circular Cylinder Using a Low Diffusion E-CUSP and High-Order WENO Scheme," *38th AIAA Fluid Dynamics Conference*, AIAA, Reston, VA, 2008, AIAA Paper 2008-3855.

[25] Kasliwal, A., Ghia, K., and Ghia, U., "Higher-order Accurate Solution for Flow Past a Circular Cylinder at $Re = 13{,}4000$." AIAA-2005-1123, *43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA, Reston, VA, 2005, AIAA-2005–1123.

[26] Krothapalli, A., Shih, C., and Lourenco, L., "The Near Wake of a Circular Cylinder at $0.3 < M_\infty < 0.6$: A PIV Study," *32nd Aero Sciences Meeting and Exhibit*, AIAA, Reston, VA, 1994, AIAA Paper 94-0063.

[27] Kravchenko, G., and Moin P., " Numerical Studies of Flow over a Circular Cylinder at $Re_D = 3900$," *Physics of Fluids*, Vol. 12, No. 2, 2000, pp. 403–417.
doi: 10.1063/1.870318

Christopher Rouff
Associate Editor