AIAA Paper 2007-4432

A GENERAL SUB-DOMAIN BOUNDARY MAPPING PROCEDURE FOR STRUCTURED GRID CFD PARALLEL COMPUTATION

Baoyuan Wang [*], Ge-Cheng Zha[†]
Dept. of Mechanical and Aerospace Engineering
University of Miami
Coral Gables, Florida 33124
E-mail: zha@apollo.eng.miami.edu

Abstract

In the present work, a general sub-domain boundary mapping procedure has been developed for arbitrary topology multi-block structured grids with grid points matched on sub-domain boundaries. The interface of two adjacent blocks is uniquely defined according to each local meshing index system (MIS) which is specified independently. A pack/unpack procedure based on the definition of the interface is developed to exchange the data in 1D array to minimize the communication amount. A safe send/receive procedure is built to remove the possibility of communication block and achieve an optimum parallel computation efficiency.

The procedure is applied to parallelize an in house 3D Navier-Stokes code. The message passing interface (MPI) protocol is used for the data communication. The implementation of this newly developed parallelization procedure is straightforward. The programming work to convert a sequential code to parallel code using this procedure is minimal. The numerical experiments on an MPI based computer cluster show that the general mapping algorithm is robust and has high parallel computation efficiency.

1 Introduction

Parallel computation has been becoming a powerful computational tool for CFD simulations[1, 2]. By interconnecting PCs or workstations, people can easily develop a distributed parallel computing system to increase the computing power. Hence there are many efforts to develop parallel computation codes and to convert legacy sequential codes for multi-processors parallel computation.

For CFD parallel computation, the multi-block structured grid method is widely used since structured grids strategy has its advantages for numerical efficiency and accuracy. The basic idea is to partition a large domain into many smaller sub-domains which decrease the difficulties of grid generation around complex configurations since the structured grids can be generated within each sub-domain independently. The CFD solver written for parallel computation is usually based on

* Ph.D Candidate
† Associate Professor, AIAA Member

single program multiple data (SPMD) strategy. The sub-domains exchange data at domain partitioning boundaries, or called inner boundaries, which are usually treated as boundary conditions[3]. The computation is conducted simultaneously for each sub-domain using the SPMD CFD solver. The inner boundary data is exchanged across the boundaries by a certain mapping procedure after each iteration[4]. The performance of the mapping procedure determines the parallel computation efficiency and the accuracy.

Various mapping procedures for multi-block structured grids have been developed since the parallel computation was introduced to the CFD. However, the procedure and parallel implementation are in general ad-hoc and different code developers may use different methods. The complexity of the procedures depends on the complexity of the problems to be dealt with. For a simple domain partitioning problem, Evans developed a toolkit known as the Computer Aided Parallelization Tools(CAPTools) to transfer a scalar code to a form suitable for parallel implementation with message passing calls[5]. For the complex problems that have different local meshing index systems, it is difficult and time consuming to deal with the inner boundary data exchange. One method is to use database systems to manage CFD parallel computation. For example, the NSMB package, which 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 also used to implement the parallel computation[8]. However, since the database systems are generally dependent on the computer platforms, the portability of the CFD codes hence are quite limited. In addition, such database systems are not necessarily available for the general researchers. Therefore, it is useful to develop a 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 this paper is to develop a general sub-domain boundary mapping procedure for parallel computation using multi-block structured grid CFD code. The procedure is aimed at having simplicity, robustness, reliability, efficiency and portability. The mapping procedure includes the following features: 1) The sub-domain grid can be independently generated and does not need to be within a larger domain; 2) The meshing index system (e.g. i, j, k direction) can be arbitrary and does not need to be the same between the sub-domains and does not need to follow right hand rule. The only condition required is that the grid cells are matched on the domain boundaries to achieve flux conservation and preserve the accuracy of the numerical algorithm. The in house CFD code[9, 10, 11] is implemented for parallel computation using this mapping procedure. Some 2D and 3D flows are calculated on an MPI based computer cluster composed of 54 Dell PowerEdge 1950 Servers 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. This mapping procedure is implemented in the code using MPI and is hence as portable as MPI. The implementation is straightforward and the programming work is minimal.

2 The Mapping Procedure

2.1 Inner Boundary and Relationship Between Adjacent Blocks

The index i, j, k are used to express the mesh index of a general sub-domain block. For any structured grid block with the mesh dimensions of $i = n1, j = n2, k = n3$, we can uniquely define the block, face and edge using the two diagonal points of each entity. For example, the block, face and edge shown in Fig. 1 can be defined as the following

$block: \quad start = 1,1,1, end = n1,n2,n3$
$face\,(i = n1): \quad start = n1,1,1, end = n1,n2,n3$
$edge\,(i = n1, j = 1): \quad start = n1,1,1, end = n1,1,n3$

Where, *start* and *end* are the one dimension arrays with 3 elements, $start\,(3)$ and $end\,(3)$. Any other faces and edges can be defined similarly. With this definition, the mapping relationships between structured grid blocks can be developed. For simplicity, assume that all blocks are numbered with index from 1 to n and there are only two halo layers overlapping grid points for the inner boundaries(Fig. 2). Hence two layers of data need to be communicated for each inner boundary between the adjacent blocks.

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

1) The adjacent block index number q

2) The inner boundary definition, the two diagonal points(*start* and *end*) of the block p and q

3) The relationship of the meshing index system between block p and q

For convenience, a terminology, "Order" is introduced to express the relationship of the MISs between block p and q. The numbers $1, 2, 3$ are used to represent mesh axis directions, i, j, k respectively for a block. For the meshing index sequence $(1, 2, 3)$ of block p, the Order of q corresponding to block p is defined as the corresponding meshing index sequence of block q with which the block p and block q have the same mesh axis directions. There are 6 possibilities for the Order of the block q corresponding to the block p as shown in Fig. 3. The Order is same if the direction of an axis is taken as an opposite direction. The Order is numbered from 1 to 6 as given in table 1. The inner boundary and relationship can be uniquely defined by the current block number p, the adjacent block number q, the inner boundary definition *start* and *end* of the block p, the inner boundary definition *start* and *end* of the block q and the Order of block q corresponding to block p.

The Order of q	(1,2,3)	(1,3,2)	(2,1,3)	(2,3,1)	(3,1,2)	(3,2,1)
Order Number of q	1	2	3	4	5	6

Table 1: The Order of the block q

The Order of p corresponding to the block q can be directly derived from the Order of q corresponding to the block p. The relationship is given by the table 2:

The Order of Block q	(1,2,3)	(1,3,2)	(2,1,3)	(2,3,1)	(3,1,2)	(3,2,1)
The Order of Block p	(1,2,3)	(1,3,2)	(2,1,3)	(3,1,2)	(2,3,1)	(3,2,1)

Table 2: The relative relationship of the Orders for two blocks

Using this relationship, an inner boundary only needs to be defined once. Hence, the probability of making mistakes to define the inner boundary conditions is minimized.

For example, the inner boundary(see Fig. 2) of block p and block q can be written 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 which are used to define the inner boundary. The *start*, *end* and *istart*, *iend* are given according to the local MIS. The *order* represents the Order of block q.

2.2 Pack and Unpack Data Procedures

For CFD parallel computation with multi-block grids, the inner boundary data of the flow field are exchanged after each iteration according to the relationship of the inner boundary defined in the last section. To be efficient, one dimension array is used for data communication. For each block, two operations need to be done for data exchange as the following

1) pack the inner boundary data into a one dimension array and send them to the adjacent block

2) unpack the one dimension array received from the adjacent block and assign it to the inner boundary

For 2D problems, the pack/unpack procedure is simpler than the procedure of 3D problems since the boundaries of 2D problems are edges and the boundaries of 3D problems are faces. Hence, we introduce pack/unpack procedures for 2D and 3D problems separately.

2.2.1 2-D Problems

The pack/unpack procedures are implemented using following rules:

1) The inner boundary data are packed into a one dimension array in inward direction of the interface, i.e. from the outer most face to the inner most face

2) The one dimension array received from the adjacent block are unpacked in reversed (outward) direction

For example, the pack/unpack procedure of the 2D problem shown in Fig. 2($n3 = 1$) for block p is given as the following

```
pack :
    do $i = start(1) - 1, start(1) - l, -1$
      do $j = start(2), end(2)$
        $i1 = i1 + 1$
        $bcb(i1) = x(i, j)$
      end do
    end do

unpack :
    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, $l = 2, start(1) = n1, start(2) = 1, end(2) = n2$, bcb is the one dimension array, x is the data array of the flow field.

2.2.2 3-D problems

For 3D problems, the rules of the pack/unpack procedure are the same as the 2D problems. However, since the inner boundary is a face for 3-D problems, the Order of the adjacent blocks are needed to determine the sequence that the data to be packed and unpacked. As we know, the Order of any block is composed of three numbers: $1, 2$ and 3. These numbers correspond to the MIS of the adjacent blocks, so we call them as axis index numbers. These axis index numbers are used in the pack/unpack procedure

To facilitate the programming work, another terminology, the "Orientation" is introduced to specify the faces of a block. The relationship between the Orientation and faces is given by table 3.

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

Table 3: The relationship between the Orientation and faces

Therefore, any inner boundary of a block has an Orientation numbered from 1 to 6.

Again, the two adjacent blocks p and q are taken as the example as shown in Fig. 2. The inner boundary data of p are packed from the large axis index number to the smaller axis index number of the Order of q according to the Orientation of the inner boundary of block p. For example, if the Orientation of the inner boundary of p is 1, the Order number (1, 3, 5) of q have the same pack procedure because the second axis index number is smaller than the third axis index number, the Order number (2, 4, 6) of q have the same pack procedure because the second axis index number is larger than the third axis index number. For the Orders (1, 3, 5), the data is packed by the following DO loops:

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

For the Orders (2, 4, 6), the data is packed by the following DO loops:

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

The unpack procedure is same for different Orders. It is given by the following DO loops:

```
unpack :
    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
```

Other cases can be handled with the similar method.

2.3 Send/Receive Procedure

As shown in Fig. 4, the send/receive procedure of a block for CFD parallel computation usually is that all blocks send data to all adjacent blocks first, and then all the blocks receive data from all adjacent blocks[12].

This procedure can efficiently avoid the communication deadlock, but the communication block may occur because of the buffer space limitation which used to save the exchange data temporarily. Thereby, a safe send/receive procedure is suggested.

The basic idea of this procedure is to do the send/receive operations in a pair. That is when a block send data, another block will receive the data at the same time(Fig. 5). The procedure is implemented abiding 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 1-1 correspondent relationship between send and receive operations with minimal buffer space required to avoid the communication block. 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 in the same way as in step 2

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 block problem and has high efficiency of data communication in our numerical experiments.

3 Numerical Algorithms

The in house Navier-Stokes code[9, 10, 11] is converted to have parallel computing capability using the suggested general sub-domain boundary mapping procedure. The governing equations are the Reynolds averaged Navier-Stokes equations. The Baldwin-Lomax turbulence model[13] is used to calculate the turbulent eddy viscosity. The finite volume method and upwind schemes[14, 15, 16] are used to discretize the governing equations for steady state solution. To achieve high convergence rate, the implicit time marching scheme is used with the unfactored Gauss-Seidel line relaxation.

Using the finite volume method, the implicit semi-discretized form of the Reynolds averaged Navier-Stokes equations can be written as the following

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

where n and $n+1$ are two sequential time levels, which have a time interval of Δt. The 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 $(+\frac{1}{2})$ interface 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[14, 15, 16]. The viscous fluxes are evaluated by using central difference scheme. Finally, the implicit discretized form are written as,

$$\begin{aligned}&\bar{B}\Delta\mathbf{U}^{n+1}_{i,j,k}+A^{+}\Delta\mathbf{U}^{n+1}_{i+1,j,k}+A^{-}\Delta\mathbf{U}^{n+1}_{i-1,j,k}+B^{+}\Delta\mathbf{U}^{n+1}_{i,j+1,k}\\&+B^{-}\Delta\mathbf{U}^{n+1}_{i,j-1,k}+C^{+}\Delta\mathbf{U}^{n+1}_{i,j,k+1}+C^{-}\Delta\mathbf{U}^{n+1}_{i,j,k-1}=\mathbf{RHS}^{n}\end{aligned} \tag{2}$$

The Gauss-Seidel line iteration is applied in each direction (i, j, k) respectively and is swept forward and backward on each direction. Take j direction for example, the equation form for Gauss-Seidel iteration following lines along j direction (i and k are constant) is written as,

$$B^{-}\Delta\mathbf{U}^{n+1}_{i,j-1,k}+\bar{B}\Delta\mathbf{U}^{n+1}_{i,j,k}+B^{+}\Delta\mathbf{U}^{n+1}_{i,j+1,k}=\mathbf{RHS}' \tag{3}$$

where

$$\mathbf{RHS}'=\mathbf{RHS}^{n}-A^{+}\Delta\mathbf{U}^{n+1}_{i+1,j,k}-A^{-}\Delta\mathbf{U}^{n+1}_{i-1,j,k}-C^{+}\Delta\mathbf{U}^{n+1}_{i,j,k+1}-C^{-}\Delta\mathbf{U}^{n+1}_{i,j,k-1} \tag{4}$$

The Eq. (3) can be written in a matrix form:

$$\left[\begin{array}{cccc|ccc}\bar{B}_1 & B_1^{+} & & & & & \\ B_2^{-} & \bar{B}_2 & B_2^{+} & & & 0 & \\ & & \ddots & & & & \\ & & B_m^{-} & \bar{B}_m & B_m^{+} & & \\ \hline & & & B_{m+1}^{-} & \bar{B}_{m+1} & B_{m+1}^{+} & \\ & 0 & & & & \ddots & \\ & & & & & B_n^{-} & \bar{B}_n\end{array}\right]\left[\begin{array}{c}\Delta\mathbf{U}_1\\ \Delta\mathbf{U}_2\\ \vdots\\ \Delta\mathbf{U}_m\\ \hline \Delta\mathbf{U}_{m+1}\\ \vdots\\ \Delta\mathbf{U}_n\end{array}\right]=\left[\begin{array}{c}\mathbf{RHS}'_1\\ \mathbf{RHS}'_2\\ \vdots\\ \mathbf{RHS}'_m\\ \hline \mathbf{RHS}'_{m+1}\\ \vdots\\ \mathbf{RHS}'_n\end{array}\right] \tag{5}$$

When using multi-block parallel computation, for example two blocks, the Eq. (5) will be split into two sub-matrices as the following

$$\begin{bmatrix} \bar{B}_1 & B_1^+ & & & \\ B_2^- & \bar{B}_2 & B_2^+ & & \\ & & \ddots & & \\ & & & B_m^- & \bar{B}_m \end{bmatrix} \begin{bmatrix} \Delta\mathbf{U}_1 \\ \Delta\mathbf{U}_2 \\ \vdots \\ \Delta\mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_1' \\ \mathbf{RHS}_2' \\ \vdots \\ \mathbf{RHS}_m' \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \bar{B}_{m+1} & B_{m+1}^+ & & & \\ B_{m+2}^- & \bar{B}_{m+2} & B_{m+2}^+ & & \\ & & \ddots & & \\ & & & B_n^- & \bar{B}_n \end{bmatrix} \begin{bmatrix} \Delta\mathbf{U}_{m+1} \\ \Delta\mathbf{U}_{m+2} \\ \vdots \\ \Delta\mathbf{U}_n \end{bmatrix} = \begin{bmatrix} \mathbf{RHS}_{m+1}' \\ \mathbf{RHS}_{m+2}' \\ \vdots \\ \mathbf{RHS}_n' \end{bmatrix} \tag{7}$$

The variables $\Delta\mathbf{U}_1 \rightarrow \Delta\mathbf{U}_m$ and $\Delta\mathbf{U}_{m+1} \rightarrow \Delta\mathbf{U}_n$ are correspondingly solved on two processors by conducting the matrix inversion on the above two sub-matrices. Comparing Eqs. (6) and (7) with the original Eq. (5) for the single processor computation, B_m^+ on the first sub-matrix and B_{m+1}^- on the second sub-matrix are discarded. They are the coefficients computed based on the variables from the neighboring sections. In the present work, these two coefficients are treated as zero in the implicit solver for the sub-domain computations. The variables are updated independently on the two sub-domains. This treatment does not preserve the exact matrices as in the single processor computation, which will negatively affect the convergence efficiency in steady state calculation. However, the computation experience indicates that the slow down of the convergence due to this non-exact treatment is small, in particular when the mesh size is large.

4 Results and Discussion

4.1 RAE2822 Transonic Airfoil

This case is to examine the accuracy of the CFD solver and parallel computation efficiency for 2D problems. The multi-block 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 2.31^0. In parallel computation, one block is assigned to each CPU processor. Fig. 7 shows the solution residuals with different number of processors(blocks). The convergence rate with multiple processors is slightly affected due to the approximate implicit treatment at the sub-domain boundaries. Fig. 8 presents the comparison of pressure coefficients between the experimental data and computation results with 1 and 8 processors. The speed up and efficiency for parallel computation are given in table 4.

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

Table 4: The parallel performance of the computation for RAE2822

Fig. 9 shows a good linear scalability for parallel computation.

4.2 The Co-Flow Airfoil

This case is used to test the robustness of the parallel code using the present mapping procedure. The co-flow jet (CFJ) airfoil[17, 18] and the five block grids are shown in Fig. 10. The dimensions of the blocks are (33×49), (97×97), (17×97), (23×97) and (59×193). The free-stream Mach number is 0.1024; the Reynolds number is 380000.0 based on the cord length. Fig. 11 shows the contours of the Mach number at AoA=10^0. Fig 12 shows the C_L vs. AoA.

4.3 Transonic ONERA M6 Wing

This study is to examine the accuracy of the CFD solver and parallel computation efficiency for 3D problems. The multi-block grids are also 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. The angle of attack is 0^0.

Fig. 15 presents the comparison of the pressure distribution between the experiment and computation at the different 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 except at the section of z/b=0.8, where the double-shock pattern is not simulated well.

The speed up and efficiency for the parallel computation are given in table 5. Fig. 14 shows a good linear scalability for this case.

Number of nodes	1	2	4	8	16
time (sec/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

Table 5: The parallel performance of the computation for M6 wing

5 Conclusions

This paper has developed a general sub-domain boundary mapping procedure for arbitrary topology multi-block structured grid parallel CFD computation. The grid points are required to match on the inner boundaries. The meshing index system of a sub-domain block can be arbitrary. The concepts of block Order and Orientation are introduced to uniquely define the relationship of the mesh indexing systems between the adjacent blocks and the current block and make the programming work easier. A pack/unpack procedure is developed to exchange the data in 1D array to minimize the communication amount. A safe send/receive procedure is developed to minimize the buffer space use which basically removes the possibility of communication block.

The procedure is applied to parallelize a in house 3D Navier-Stokes code. The message passing interface (MPI) protocol is used for the data communication. The code is portable to any platform as long as the MPI is available. The implementation is straightforward and need minimal programming work to convert a legacy sequential code. The numerical experiments on an MPI based computer cluster show that the developed general mapping procedure is robust and has high parallel computation efficiency.

References

[1] S. Tu and S. Aliabadi, "A Robust Parallel Implicit Finite Volume Solver for High-speed Compressible Flows." AIAA Paper 2005-1396, 2005.

[2] Cai J, Tsai HM, Liu F, "A parallel viscous flow solver on multi-block overset grids," *Journal of Computers & Fluids*, vol. 35, pp. 1290–1301, 2006.

[3] T. Ohta, "An object-oriented programming paradigm for parallel computational fluid dynamics on memory distributed parallel computers ." D.R. Emerson, A. Ecer, J. Periaux, N. Satofuka and P. Fox, eds., Parallel Computational Fluid Dynamics, 1998.

[4] S. Allwright, "Multiblock techniques for transonic flow computation about complex aircraft configuration ." K.W. Morton and M.J. Baines, eds., Numerical Methods for Fluid Dynamics, III, 1988.

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

[6] S. Merazzi, "MEM-COM An Integrated Memory and Data Management, System mem-com User Manual Version 6.0 ." SMR TR-5060, Mar. 1991.

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

[8] A. Ecer, H.U. Akay, W.B. Kemle, H. Wang, D. Ercoskun, "Parallel computation of fluid dynamics problems ," *Computer Methods in Applied Mechanics and Engineering*, vol. 112, pp. 91–108, 1994.

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

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

[11] Chen, X.-Y. and Zha, G.-C. and Yang, M.-T., "Numerical Simulation of 3D Wing Flutter with Fully Coupled Fluid-Structural Interration," *Journal of Computers & Fluids*, vol. 36, pp. 856–867, 2007.

[12] Daniela di Serafino, "A parallel implementation of a multigrid multiblock Euler solver on distributed memory machines," *Parallel Computing*, vol. 23, pp. 2095–2113, 1997.

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

[14] P. Roe, “Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes,” *Journal of Computational Physics*, vol. 43, pp. 357–372, 1981.

[15] B. Van Leer, “Flux-Vvector Splitting for the Euler Equations,” *Lecture Note in Physics*, vol. 170, pp. 507–512, 1982.

[16] G.-C. Zha, “A Low Diffusion E-CUSP Upwind Scheme for Transonic Flows.” AIAA Paper 2004-2707, to appear in AIAA Journal, 34th AIAA Fluid Dynamics Conference, June 28 - July 1 2004.

[17] G.-C. Zha, B. Carroll, C. Paxton, A. Conley, and A. Wells, “High Performance Airfoil with Co-Flow Jet Flow Control.” AIAA-2005-1260, 2005.

[18] Zha, G.-C. and Gao, W., and Paxton, C. and Conley, A., “Effect of Injection Slot Size on the Performance of Coflow Jet Airfoil,” *Journal of Aircraft*, vol. 43, pp. 987–995, 2006.

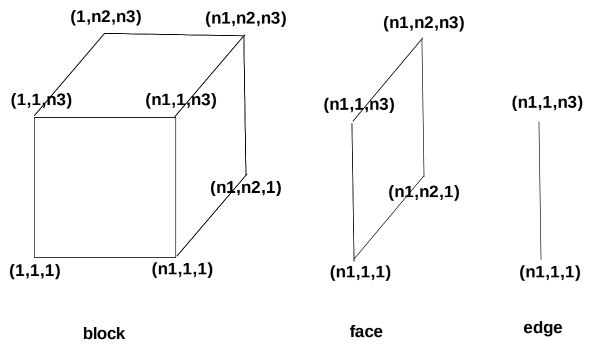

Figure 1: Definition of a block, faces and edges

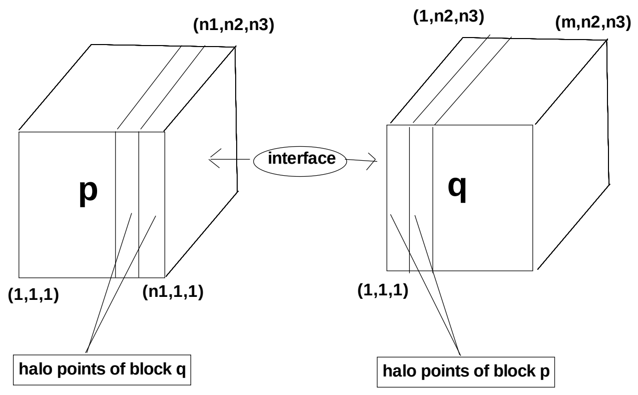

Figure 2: Inner boundary of two adjacent blocks

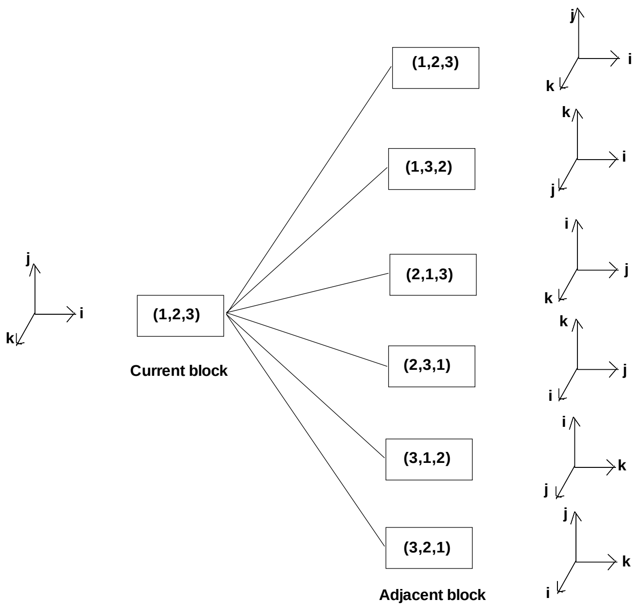

Figure 3: MIS relationship of adjacent blocks

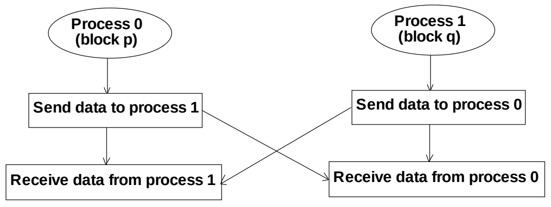

Figure 4: Unsafe communication procedure

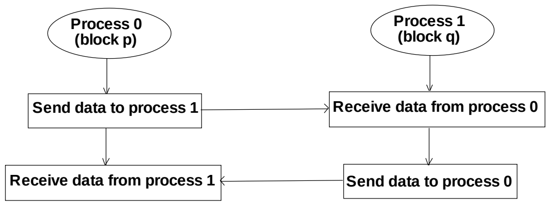

Figure 5: Safe communication procedure

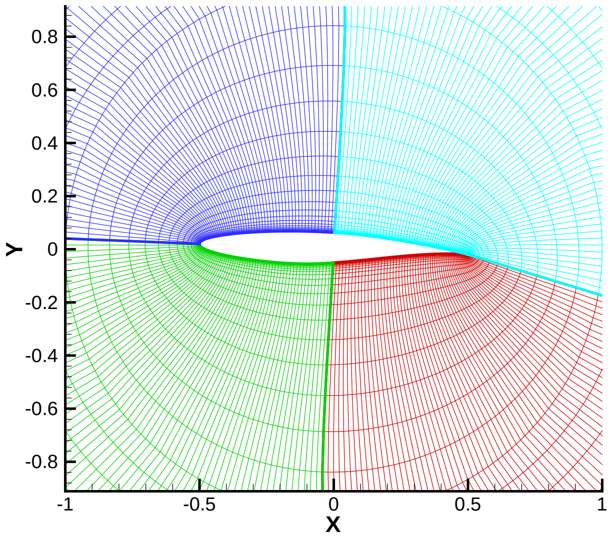

Figure 6: 4-Block grids for RAE2822

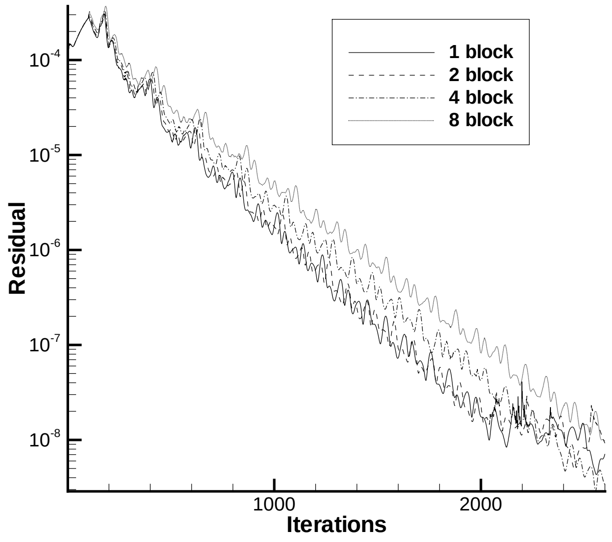

Figure 7: The L2 solution residual history of RAE2822

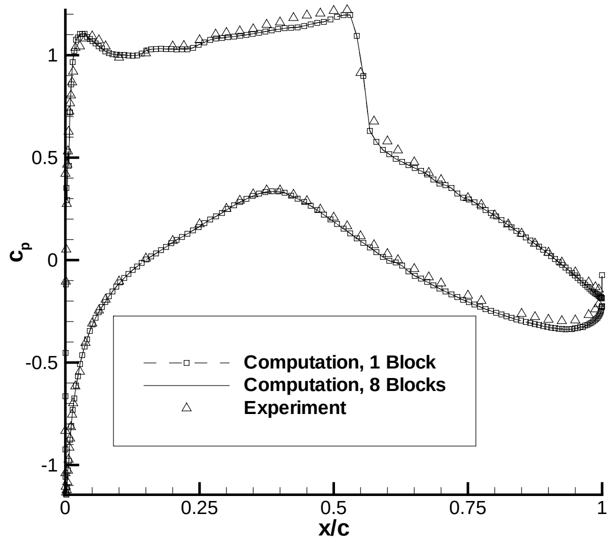

Figure 8: The distribution of cp

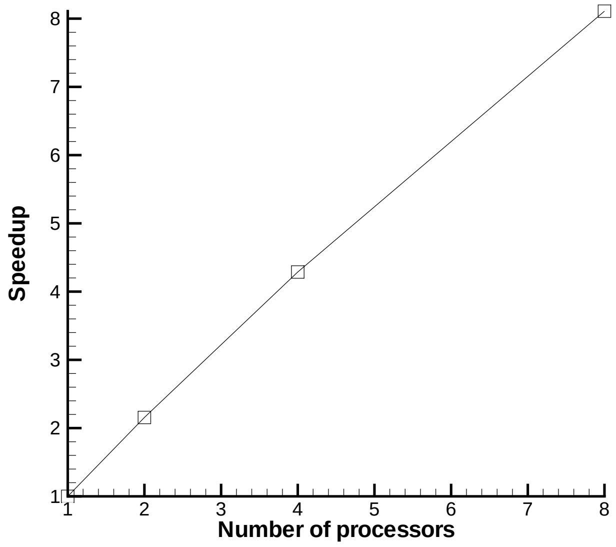

Figure 9: Speedup of parallel computation for RAE2822

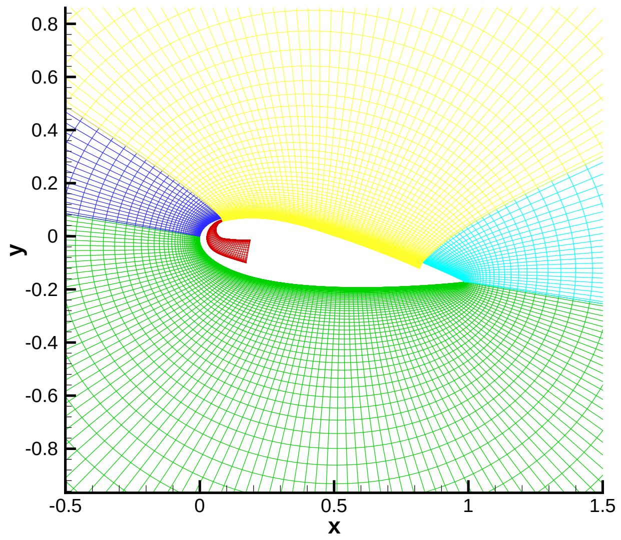

Figure 10: 5-Block grids for co-flow jet airfoil

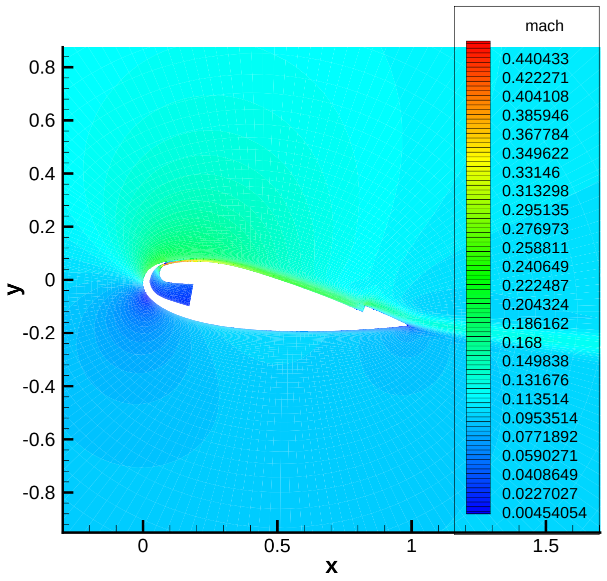

Figure 11: The contour of the Mach number

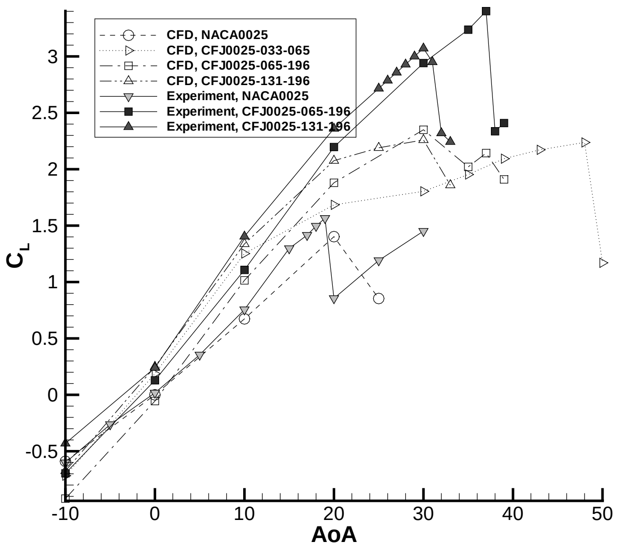

Figure 12: Lift coefficient of CFJ airfoils

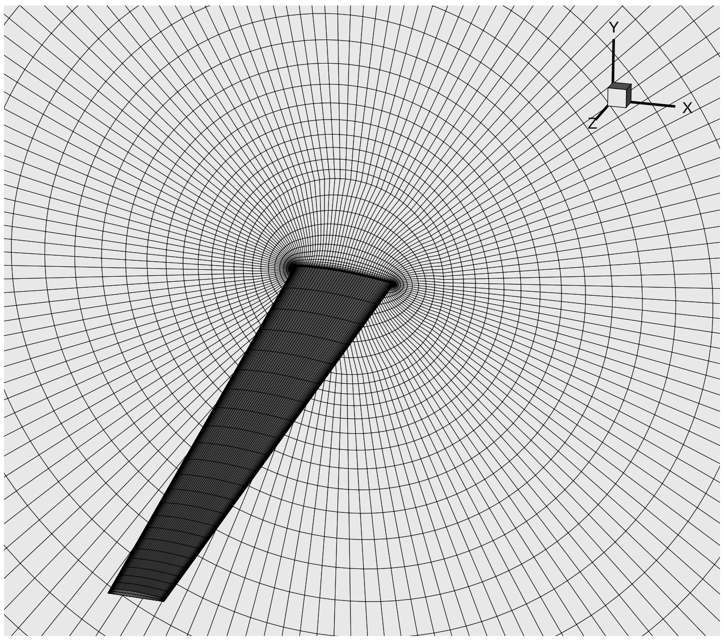

Figure 13: M6 wing mesh

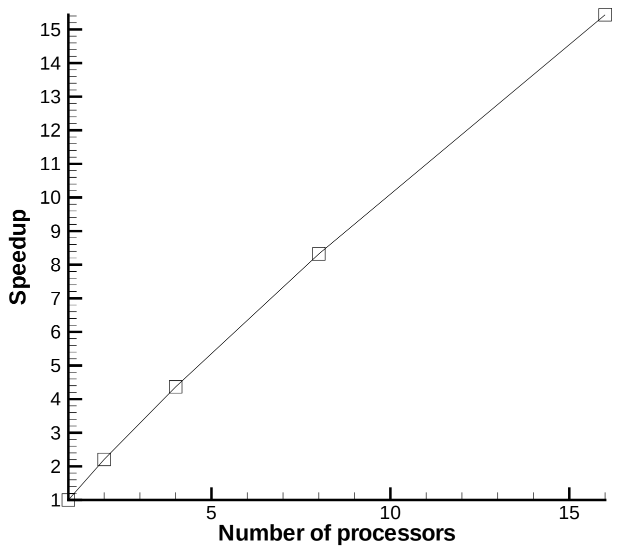

Figure 14: Speedup of parallel computation for M6 wing

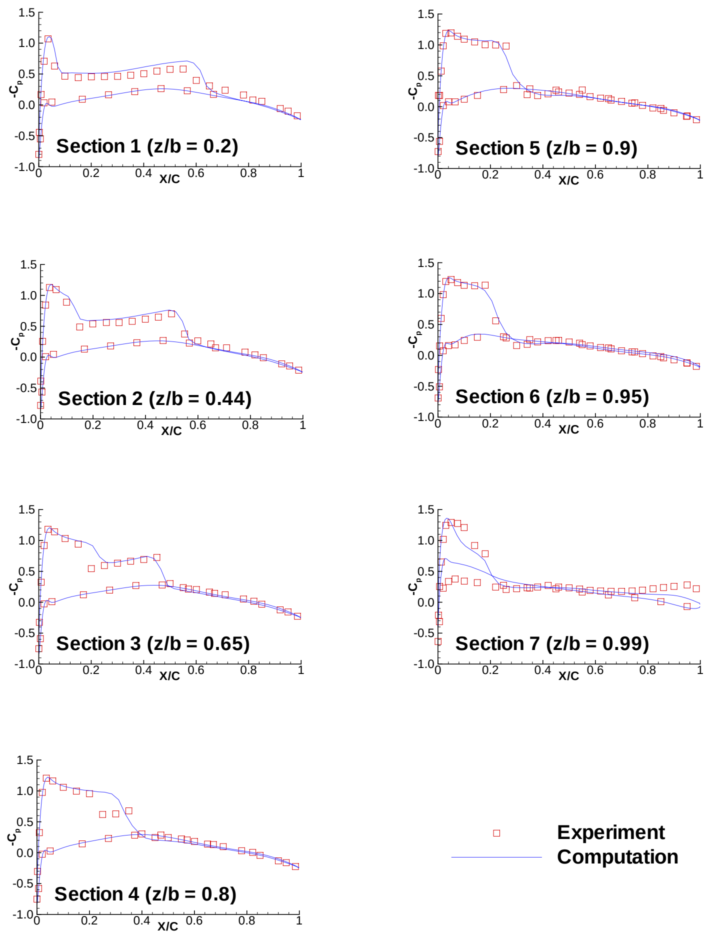

Figure 15: cp distribution of M6 wing