

PARALLEL UNSTRUCTURED GRID GMRES+LU-SGS METHOD FOR TURBULENT FLOWS

Hong Luo, Dmitri Sharov, and Joseph D. Baum
Science Applications International Corporation
1710 SAIC Drive, MS 2-6-9
McLean, VA 22102, USA

and

Rainald Löhner
Institute for Computational Sciences and Informatics
George Mason University, Fairfax, VA 22030, USA

ABSTRACT

A parallel, matrix-free implicit GMRES+LU-SGS method is presented on shared-memory, cache-based parallel computers using OpenMP for computing compressible turbulent flow problems. A special grid renumbering technique is used to achieve the parallelization rather than the traditional method of partitioning the computational domain. This renumbering technique helps to avoid inter-processor data dependencies, cache-misses, and cache-line overwrite while allowing pipelining. The resulting code can be used with maximum efficiency and without modifications on traditional scalar computers, vector supercomputers, and shared-memory parallel systems. The developed parallel implicit method has been used to predict drag forces in the transonic regime for both DLR-F4 and DLR-F6 configurations. The numerical results in terms of parallel efficiency indicate that the present implicit method scales reasonably well and is suitable for computing turbulent flows for complex geometries. It is demonstrated that turnaround in a matter of hours for numerical simulation of high Reynolds number turbulent flows past realistic geometries has been achieved.

1. INTRODUCTION

The use of unstructured meshes for computational fluid dynamics problems has become widespread due to their ability to discretize arbitrarily complex geometries and due to ease of adaption in enhancing the solution accuracy and efficiency through the use of adaptive refinement.

Development of computationally efficient algorithms for high Reynolds number viscous flow simulations on highly stretched unstructured grids remains one of the unresolved issues in computational fluid dynamics. Due to the large differences between the convective and diffusive time scales, high Reynolds

number flows produce very stiff systems of equations. This presents a severe challenge to the time advancement procedure. By itself, explicit, multi-stage time advancement is not a computationally viable alternative. A multigrid strategy or an implicit temporal discretization is required in order to speed up convergence. In general, implicit methods can outperform their explicit counterparts by an order-of-magnitude or even more. Unfortunately, this performance is usually achieved at the expense of memory increase, as some efficient implicit methods may need up to an order of magnitude more storage than the explicit methods. This is mainly due to the fact that implicit methods usually require the storage of the left-hand-side Jacobian matrix. Any implicit methods requiring the storage of the Jacobian matrix would be impractical, if not impossible, to use to perform turbulent simulations involving complex, realistic aerodynamic configurations, where millions of mesh points are necessary to represent such engineering-type configurations accurately. The authors have developed a fast, matrix-free implicit method, GMRES+LU-SGS¹, for solving compressible Euler and Navier-Stokes equations on unstructured grids. The developed GMRES+LU-SGS method has proved to be very effective for accelerating the convergence to both steady¹, unsteady², and low Mach number³ flow problems. Despite the striking performance achieved by the present GMRES+LU-SGS method, the overall computational requirements are still very substantial for practical turbulent flow computations for complex geometries. Computing time in-order of days, if not weeks, is typically required to perform high Reynolds number Navier-Stokes calculations past a realistic configuration on a single processor supercomputer. This is certainly not acceptable both for production calculations and during the code developments. As a consequence, practical computations for complex geometries require the use of parallel computing to produce results within an acceptable timescale.

With the advent of massively parallel computers, i.e., computers in excess of 500 nodes, the exploitation of parallelism in solvers has become a ma-

Copyright ©2003 by the authors. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

major focus of attention. Most of the applications ported successfully to parallel computers to date have followed the Single Program Multiple Data (SPMD) paradigm. For grid-based solvers, a spatial subdomain was stored and updated in each processor. For obvious reasons, load balancing⁴⁻⁷ has been a major focus of activity. Despite the striking successes reported to date, only the simplest of all solvers: explicit time-stepping or implicit iterative schemes, perhaps with multigrid added on, have been ported without major changes and/or problems to massively parallel computers with distributed memory. Many code options that are essential for realistic simulations are not easy to parallelize on this type of machine. Among these are local remeshing, repeated h-refinement such as required for transient problems, and chemistry interact, and other applications with rapidly varying load imbalances. Even if 99% of all operations required by these codes can be parallelized, the maximum achievable gain will be restricted to 1:100. If we accept as a fact that for most large-scale codes we may not be able to parallelize more than 99% of all operations, the shared-memory paradigm, discarded for a while as non-scalable, make a comeback. It is far easier to parallelize some of the more complex algorithms, as well as cases with large load imbalance, on shared-memory computers such as the SGI origin 2000/3000.

The objective of the effort discussed in this paper is to implement the GMRES+LU-SGS scheme on shared memory computers using OpenMP. The parallelization technique originally introduced and implemented by Löhner for explicit schemes⁸, and later developed by Sharov et al. for the matrix-free GMRES+LU-SGS method⁹, has been extended for computing high Reynolds number turbulent flows around complex, realistic aerodynamic configurations on unstructured grids. The developed method has been used to compute a number of high Reynolds number turbulent flow problems to evaluate the performance of the developed parallel implicit method. The numerical experience indicates that the turnaround in a matter of hours for numerical simulation of high Reynolds number turbulent flows around complex geometries has been achieved, demonstrating that the developed parallel implicit method can be used in a production environment for complex turbulent flow simulations.

2. GOVERNING EQUATIONS

The Reynolds-averaged Navier-Stokes equations governing unsteady compressible viscous flows can be expressed in the conservative form as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}^j}{\partial \mathbf{x}_j} = \frac{\partial \mathbf{G}^j}{\partial \mathbf{x}_j}, \quad (2.1)$$

where the summation convention has been employed. The flow variable vector \mathbf{U} , inviscid flux vector \mathbf{F} , and viscous flux vector \mathbf{G} , are defined by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix}, \mathbf{F}^j = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p \delta_{ij} \\ u_j (\rho e + p) \end{pmatrix},$$

$$\mathbf{G}^j = \begin{pmatrix} 0 \\ \sigma_{ij} \\ u_i \sigma_{lj} + q_j \end{pmatrix}. \quad (2.2)$$

Here ρ , p , and e denote the density, pressure, and specific total energy of the fluid, respectively, and u_i is the velocity of the flow in the coordinate direction x_i . The pressure can be computed from the equation of state

$$p = (\gamma - 1)\rho(e - \frac{1}{2}u_j u_j), \quad (2.3)$$

which is valid for perfect gas, where γ is the ratio of the specific heats. The components of the viscous stress tensor σ_{ij} and the heat flux vector are given by

$$\sigma_{ij} = (\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}(\mu + \mu_t)\frac{\partial u_k}{\partial x_k}\delta_{ij}, \quad (2.4)$$

$$q_j = \frac{1}{\gamma - 1}\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)\frac{\partial T}{\partial x_j}. \quad (2.5)$$

In the above equations, T is the temperature of the fluid, Pr the laminar Prandtl number, which is taken as 0.7 for air, and Pr_t the turbulent Prandtl number, which is taken as 0.9. μ represents the molecular viscosity, which can be determined through Sutherland's law

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0}\right)^{\frac{3}{2}} \frac{T_0 + S}{T + S}. \quad (2.6)$$

μ_0 denotes the viscosity at the reference temperature T_0 , and S is a constant which for air assumes the value $S=110^\circ\text{K}$. The temperature of the fluid T is determined by

$$T = \gamma \frac{p}{\rho}, \quad (2.7)$$

and μ_t denotes the turbulence eddy viscosity, which is computed using Spalart and Allmaras one equation turbulence model¹⁰

$$\mu_t = \rho \tilde{\nu} f_{v1}. \quad (2.8)$$

The system is closed by including the governing equation for the working variable $\tilde{\nu}$, which can be written as

$$\frac{D\tilde{\nu}}{Dt} = \frac{1}{\sigma} \nabla \cdot ((\nu + (1 + c_{b2})\tilde{\nu})\nabla\tilde{\nu}) - \frac{c_{b2}}{\sigma} \tilde{\nu} \nabla\tilde{\nu}$$

$$+ c_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} - (c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2})\left(\frac{\tilde{\nu}}{d}\right)^2. \quad (2.9)$$

The constants appearing in the above equations take on the standard values recommended in Ref. 10.

3. NUMERICAL METHOD

The numerical algorithms for solving the governing equations (2.1) and (2.9) are based on a loosely coupled approach. In this approach, the flow and turbulence variables are updated separately. At each time step, the flow equations (2.1) are solved first using the previously updated turbulent eddy viscosity. The turbulent equation (2.9) is then solved using the newly updated flow variables to advance the turbulence variable in time. Such a loosely coupled approach allows for the easy interchange of new turbulence models. Another advantage of this approach is that all the working arrays used to solve the flow equations can be used to solve the turbulence equation. As a result, no additional storage is needed for the present implementation.

The mean flow equations are discretized using a hybrid finite volume and finite element method, where a finite volume approximation based on a containment dual control volume¹¹ rather than the more popular median-dual control volume is used to discretize the inviscid fluxes, and a Galerkin finite element approximation with piecewise linear elements is used to evaluate the viscous flux terms. In the present study, the numerical flux functions for inviscid fluxes at the dual mesh cell interface are computed using HLLC scheme^{12,13}. A MUSCL¹⁴ approach is used to achieve high-order accuracy. The Van Albada limiter based on primitive variables is used to suppress the spurious oscillation in the vicinity of the discontinuities. Equation (2.1) can then be rewritten in a semi-discrete form as

$$V_i \frac{\partial \mathbf{U}_i}{\partial t} = -\mathbf{R}_i, \quad (3.1)$$

where V_i is the volume of the dual mesh cell, and \mathbf{R}_i is the right-hand-side residual and equals to zero for a steady state solution.

The discretization for the Spalart-Allmaras turbulence model is very similar to the flow equations, except that the advective terms are discretized using a first-order upwind method for robustness purposes.

In order to obtain a steady-state solution, the spatially discretized Navier-Stokes equations must be integrated in time. Using Euler implicit time-integration, equation (3.1) can be written in discrete form as

$$V_i \frac{\Delta \mathbf{U}_i^n}{\Delta t} = -\mathbf{R}_i^{n+1}, \quad (3.2)$$

where Δt is the time increment, and $\Delta \mathbf{U}^n$ the difference of unknown vector between time levels n and $n+1$, i.e.,

$$\Delta \mathbf{U}^n = \mathbf{U}^{n+1} - \mathbf{U}^n. \quad (3.3)$$

Equation (3.2) can be linearized in time as

$$V_i \frac{\Delta \mathbf{U}_i^n}{\Delta t} = -(\mathbf{R}_i^n + \frac{\partial \mathbf{R}_i^n}{\partial \mathbf{U}} \Delta \mathbf{U}_i), \quad (3.4)$$

where \mathbf{R}_i is the right-hand-side residual and equals to zero for a steady state solution. Writing the equation for all nodes leads to the delta form of the backward Euler scheme

$$\left(\frac{V}{\Delta t} \mathbf{I} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \Delta \mathbf{U} \right) = -\mathbf{R}. \quad (3.5)$$

The following simplified flux function is used to obtain the left-hand-side Jacobian matrix

$$\begin{aligned} \mathbf{R}_i = \sum_j \frac{1}{2} [\mathbf{F}(\mathbf{U}_i, \mathbf{n}_{ij}) + \mathbf{F}(\mathbf{U}_j, \mathbf{n}_{ij}) \\ - |\lambda_{ij}| (\mathbf{U}_j - \mathbf{U}_i)] | \mathbf{s}_{ij} |, \end{aligned} \quad (3.6)$$

where

$$|\lambda_{ij}| = |\mathbf{V}_{ij} \cdot \mathbf{n}_{ij}| + C_{ij} + \frac{\mu_{ij} + \mu_{t_{ij}}}{\rho_{ij} |\mathbf{x}_j - \mathbf{x}_i|}, \quad (3.7)$$

where \mathbf{s}_{ij} is the area vector normal to the control-volume interface associated with the edge ij , $\mathbf{n}_{ij} = \mathbf{s}_{ij} / |\mathbf{s}_{ij}|$ its unit vector in the direction \mathbf{s}_{ij} , C_{ij} the speed of sound, and the summation is over all neighboring vertices j of vertex i . Using an edge-based data structure, the left-hand-side Jacobian matrix is stored in upper, lower, and diagonal forms, which can be expressed as

$$U_{ij} = \frac{1}{2} (J(\mathbf{U}_j, \mathbf{n}_{ij}) - |\lambda_{ij}| |\mathbf{I}| | \mathbf{s}_{ij} |), \quad (3.8)$$

$$L_{ij} = \frac{1}{2} (-J(\mathbf{U}_i, \mathbf{n}_{ij}) - |\lambda_{ij}| |\mathbf{I}| | \mathbf{s}_{ij} |), \quad (3.9)$$

$$D_{ii} = \frac{V}{\Delta t} \mathbf{I} + \sum_j \frac{1}{2} (J(\mathbf{U}_i, \mathbf{n}_{ij}) + |\lambda_{ij}| |\mathbf{I}| | \mathbf{s}_{ij} |). \quad (3.10)$$

Equation (3.5) represents a system of linear simultaneous algebraic equations and needs to be solved at each time step. The most widely used methods to solve this linear system are iterative solution methods and approximate factorization methods. The LU-SGS approximate factorization method is attractive because of its good stability properties and competitive computational cost. In this method, the matrix A is split in three matrices, a strict lower matrix L , a diagonal matrix D , and a strict upper matrix U . This system is approximately factored by neglecting the last term on the right hand side of equation (3.11). The resulting equation can be solved in the two steps shown in equations (3.12) and (3.13), each of them involving only simple block matrix inversions.

$$(D + L)D^{-1}(D + U)\Delta \mathbf{U} = -\mathbf{R} + (LD^{-1}U)\Delta \mathbf{U} \quad (3.11)$$

Lower (forward) sweep:

$$(D + L)\Delta\mathbf{U}^* = -\mathbf{R} \quad (3.12)$$

Upper (backward) sweep:

$$(D + U)\Delta\mathbf{U} = D\Delta\mathbf{U}^*. \quad (3.13)$$

It is clear that the above algorithm involves primarily the Jacobian matrix-solution incremental vector product. Such operation can be approximately replaced by computing increments of the flux vector $\Delta\mathbf{F}$:

$$J\Delta\mathbf{U} \approx \Delta\mathbf{F} = \mathbf{F}(\mathbf{U} + \Delta\mathbf{U}) - \mathbf{F}(\mathbf{U}). \quad (3.14)$$

The forward sweep and backward sweep steps can then be expressed as

$$\begin{aligned} \Delta\mathbf{U}_i^* &= D^{-1}[-\mathbf{R}_i \\ &- \sum_{j:j<i} \frac{1}{2}(\Delta\mathbf{F}(\mathbf{u}_j^*, \mathbf{n}_{ij}) - |\lambda_{ij}| \Delta\mathbf{U}_j^*) | s_{ij} |], \quad (3.15) \\ \Delta\mathbf{U}_i &= \Delta\mathbf{U}_i^* - D^{-1} \sum_{j:j>i} \frac{1}{2}(\Delta\mathbf{F}(\mathbf{U}_j, \mathbf{n}_{ij}) \\ &- |\lambda_{ij}| \Delta\mathbf{U}_j) | s_{ij} |. \quad (3.16) \end{aligned}$$

The most remarkable achievement of this approximation is that there is no need to store the upper- and lower- matrices U and L , which substantially reduces the memory requirements.

Although the LU-SGS method is more efficient than its explicit counterpart, a significant number of time steps are still required to achieve the steady state solution, due to the nature of the approximation factorization schemes. One way to speed up the convergence is to use iterative methods. The present authors have developed a GMRES+LU-SGS method, where GMRES algorithm combined with LU-SGS preconditioner is used to solve the linear system (3.5). The resulting method is found to offer substantial CPU time savings over the best current implicit methods for a variety of flow problems. A clear advantage of the LU-SGS preconditioner is that it uses the Jacobian matrix of the linearized scheme as a preconditioner matrix, as compared with ILU preconditioner. Consequently it does not require any additional memory storage and computational effort to store and compute the preconditioner matrix. Furthermore, as GMRES only requires matrix-vector products, the same technique used in the LU-SGS method can be applied to eliminate the storage of the upper and lower matrices, which leads to a fast, low-storage implicit algorithm.

The turbulence model equation is integrated in time using an implicit method similar to that of the mean flow equations. The same GMRES+LU-SGS

procedure is then used to solve the resulting linear system. No matrix-free approach is attempted to solve the turbulence equation, as the storage of the left-hand-side Jacobian for the single turbulence equation is not very demanding. Since the turbulence equation is solved separately from the mean flow equations, all the working arrays used to solve the flow equations can be used to solve the turbulence equation without any additional storage.

4. PARALLEL IMPLEMENTATION

For maximum portability, OpenMP is chosen as an Application Program Interface (API) for implementation of the parallel method. Although the current platform for parallel computations is SGI Origin 2/3000 computer, the OpenMP is supported by all major vendors for platforms ranging from the desktop to the supercomputer. OpenMP is the portable standard for multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms.

The implicit matrix-free GMRES+LU-SGS algorithm can be divided into two basic operations: construction of right-hand-side residual R , left-hand-side diagonal block matrix D , and numerical fluxes (Eqn. (3.14)), and LU-SGS preconditioning. The parallel implementation of these two operations is described on shared-memory, cache-based computers below. Note that the parallel implementation of the first operation can be obtained without compromising the single processor algorithm. The parallelization is based on the combination of several renumbering and data regrouping techniques⁸ developed to avoid or considerably minimize cache-misses, cache-line overwrite, and memory contention.

4.1 Renumbering to minimize cache-misses

The essential elements of the first operation contain basic loops over nodes, edges, faces, and elements. If a loop over the edges is considered, and cache-misses are a concern, then the storage locations for the required point information should be as close as possible in memory when required by an edge. At the same time, as the loop progresses through the edges, the point information should be accessed as uniformly as possible. This may be achieved by first renumbering the points using a bandwidth-minimization technique such as the reverse Cuthill McKee¹⁵, wavefront¹⁶, or Peano-Hilbert type space-filling curves¹⁷, and subsequently renumbering the edges according to the minimum point number on each edge¹⁶. Figure 1a shows an example of the re-ordered edges and points. The same type of renumbering is done for all entities, which serve as basic loops in the code (e.g. elements, boundary faces, etc.). All of these algorithms are of complexity $O(N)$ or at most $O(N \log N)$, and well worth the effort.

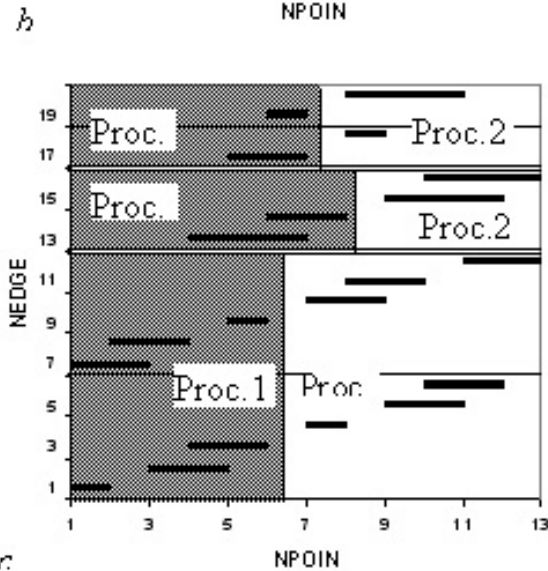
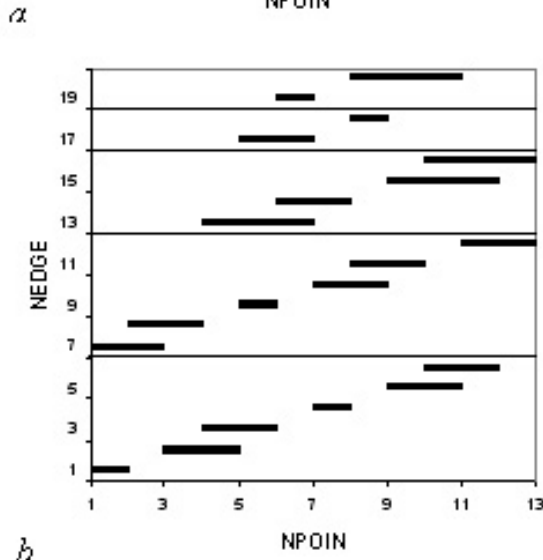
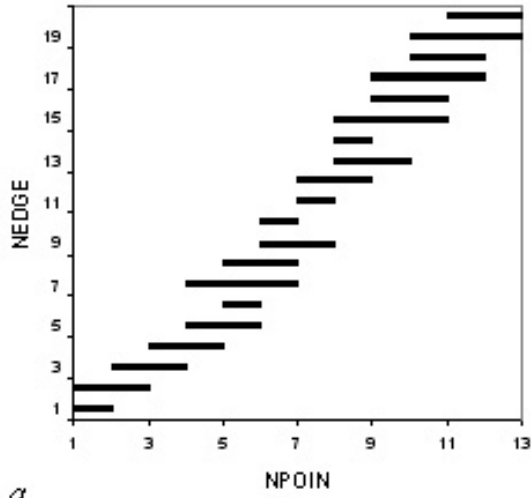


Figure 1. Edge and point renumbering illustration. (a) Renumbering to minimize cache-misses. (b) Renumbering to avoid memory contention. (c) Renumbering for 2-processor computer.

4.2 Data and loop rearrangements to avoid

memory contention

In order to achieve pipelining or vectorization, memory contention issues must be avoided. The memory contention can arise for instance in a loop over the edges while writing to corresponding points. The following example is a typical simplified loop:

```

Loop 1
do 1600 iedge = 1, nedge
  ipoi1 = inpoed(1,iedge)
  ipoi2 = inpoed(2,iedge)
  redge = f(ipoi1,ipoi2)
  rhspo(ipoi1) = rhspo(ipoi1) + redge
  rhspo(ipoi2) = rhspo(ipoi2) - redge

```

1600 continue

Since one and the same point can be accessed more than once from different edges, the information in rhspo may be corrupted in the pipeline. To make sure that no point is accessed more than once, the loop can be split into several contention-free loops over renumbered edges, see Fig. 1b:

Loop 2

```

do 1400 ipass = 1, npass
  nedg0 = edpas(ipass)+1
  nedg1 = edpas(ipass+1)
!$dir ivdep                                !$pipelining directive
do 1600 iedge = nedg0, nedg1
  ipoi1 = inpoed(1,iedge)
  ipoi2 = inpoed(2,iedge)
  redge = f(ipoi1,ipoi2)
  rhspo(ipoi1) = rhspo(ipoi1) + redge
  rhspo(ipoi2) = rhspo(ipoi2) - redge

```

1600 continue

1400 continue

4.3 Data and loop rearrangements to avoid cache-line overwrite

An auto-parallelizing compiler can parallelize the inner loop in loop 2. However, as has been mentioned in Ref. 8, such parallelization is not efficient, because of both start-up penalties and cache-line overwrite. The start-up penalties are associated with launching of a parallel loop. To minimize the penalties, the number of passes npass should be as small as possible and therefore, the vector-length should be large. However, when large vector-lengths are used, the probability that different processors access the same cache-line is increased. If the cache-line overwrite takes place, all processors must update this line, leading to a large increase of interprocessor communication, severe performance degradation, and non-scalability. To keep vector-lengths short and enjoy small start-up cost, a special edge renumbering has been proposed in Ref. 8. Figure 1c illustrates the idea of this renumbering for the case of two processors. The actual loop may look like:

Loop 3

```
do 1000 imacg = 1, npasg, nproc
  imac0 = imacg
  imac1 = min(npasg,imac0+nproc-1)
c OpenMP directive
!$omp parallel do private(ipasg)
  do 1200 ipasg = imac0, imac1
    call loop3p(ipasg)
```

```
1200 continue
```

```
1000 continue
```

loop3p becomes subroutine of the form:

```
subroutine loop3p(ipasg)
  npas0 = edpag(ipasg)+1
  npas1 = edpag(ipasg+1)
  do 1400 ipass = npas0, npas1
    nedg0 = edpas(ipass)+1
    nedg1 = edpas(ipass+1)
!$dir ivdep                !$pipelining direc-
tive
    do 1600 iedge = nedg0, nedg1
      ipoi1 = inpoed(1,iedge)
      ipoi2 = inpoed(2,iedge)
      redge = f(ipoi1,ipoi2)
      rhspo(ipoi1) = rhspo(ipoi1) + redge
      rhspo(ipoi2) = rhspo(ipoi2) - redge
1600 continue
1400 continue
  return
```

Clearly, this type of parallel implementation can be applied to any elements involved in the first operation such as computation of the right-hand-side residual, left-hand-side diagonal block matrix, and numerical fluxes in a straightforward way. In the sequel, the parallel implementation of the LU-SGS preconditioning will be discussed.

4.4 Parallelization of the LU-SGS algorithm

The parallelization of LU-SGS algorithm is not straightforward due to inherent data dependencies. There are two approaches to the solution of this problem: a) Use a special scheduling algorithm which enables data parallelism by regrouping edges¹⁸. This method has the advantage of producing exactly the same result as the single processor case, but it suffers from severe overhead penalties for parallel loop initiation, heavy interprocessor communications, and poor load balance. Our experience has shown that this method does not provide good scalability on shared memory computers. b) Split the computational domain into several non-overlapping regions according to the number of processors, and apply the LU-SGS method inside of each region with (or without) some special interprocessor boundary treatment^{19–23}. This approach may suffer from convergence degradation but takes advantage of minimal parallelization overhead and good load balance. Keep in mind that this

convergence degradation only affects the GMRES iterations, and not the convergence of the parallel implicit method, as shown below.

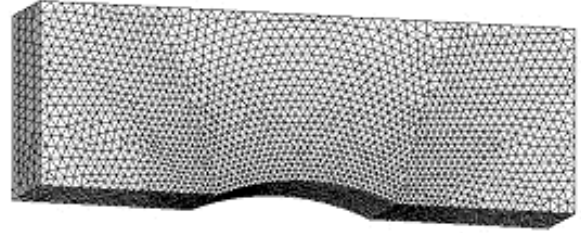


Figure 2. unstructured surface mesh for the channel with a circular bump.

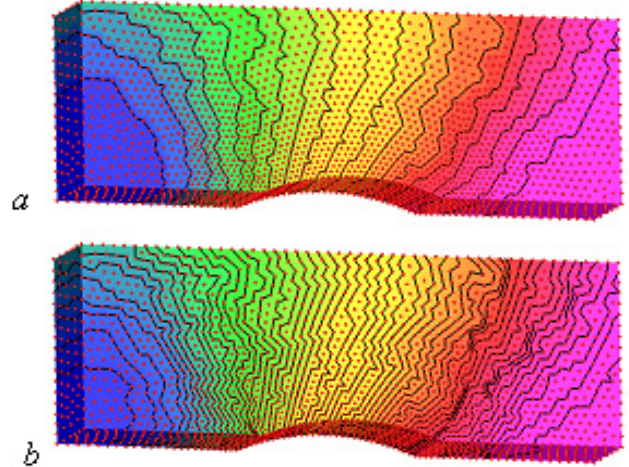


Figure 3. Partitioning with the wavefront renumbering. (a) 20 blocks. (b) 50 blocks.

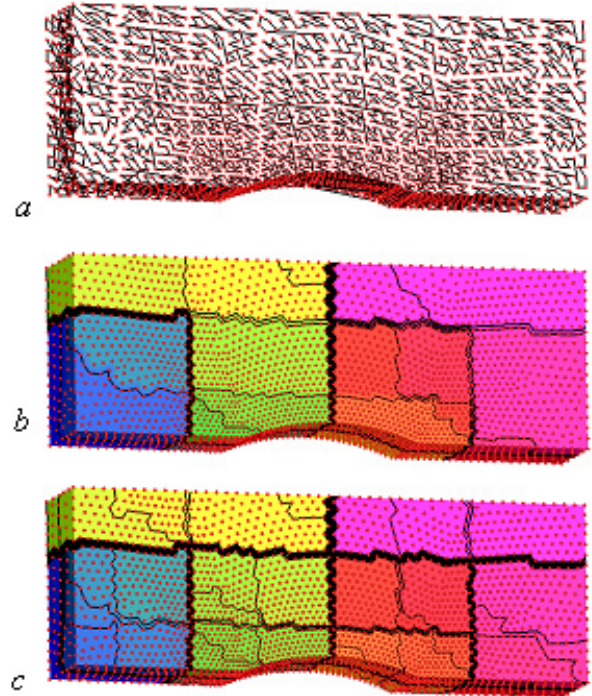


Figure 4. (a) Peano-Hilbert-Morton space-filling

curve (b) 20 blocks obtained with the Peano-Hilbert renumbering. (c) 50 blocks obtained with the Peano-Hilbert renumbering.

There are several methods to obtain a good partitioning of computational domain into blocks^{24–26}. In our case we use the fact that the grid nodes are already renumbered to minimize bandwidth, so we cut the entire array of nodes into equally sized pieces corresponding to the number of processors. This technique is very simple and provides perfect load balancing. Though the method does not provide good control over minimization of interprocessor boundary, this issue can be addressed by using alternative node renumbering techniques. In addition, since the shared-memory platforms are considered, the interprocessor communication overhead is not tightly connected to the area of the interprocessor boundaries. As an illustrative example, the partitioning into 20 blocks for a channel configuration with a circular bump on the lower wall (Fig. 2) using the wavefront renumbering is shown in Fig. 3a. Figure 3b shows similar partitioning into 50 blocks. The wavefront renumbering results in very narrow slices, thus a Peano-Hilbert type space-filling curve was also considered to renumber the points. An example of such curve is shown in Fig. 4a. This curve was obtained using Morton’s algorithm¹⁷. The 20 blocks partitioning corresponding to the Peano-Hilbert renumbering is shown in Fig. 4b, while 50 blocks partitioning is shown in Fig. 4c. Our experience⁹ has shown that the Peano-Hilbert renumbering gave the best results for the efficiency of parallelization for the GMRES+LU-SGS method, and therefore is used in the present work.

The implementation of the LU-SGS scheme on parallel non-overlapped blocks without interprocessor communications is extremely easy and simple. Figure 5a shows an example of a grid point i surrounded by nodes belonging to the same block. All surrounded nodes are divided into two groups L and U for lower and upper matrix computations correspondingly. The LU-SGS is used locally on each processor without any contribution from interprocessor boundaries. Consider point i , which has neighbors belonging to different blocks (Fig. 5b). If there is no any exchange between the blocks, the L and U sets will look as shown in Fig. 5b, and contribution from the three gray-colored nodes of processor A are not computed. The interprocessor data exchange can be achieved via so-called hybrid approach where LU-SGS scheme is used for edges inside of each block, and Jacobian scheme for interprocessor edges. This method is schematically illustrated in Figs. 5c-5d. Figure 5c corresponds to the forward sweep, and Fig. 5d corresponds to the backward sweep of the LU-SGS procedure. In the LU-SGS scheme, when the forward sweep is performed, upper matrix computation has no data

dependency. Conversely, when the backward sweep is performed, the lower matrix computation has no data dependency. Note that in the worst scenario when the

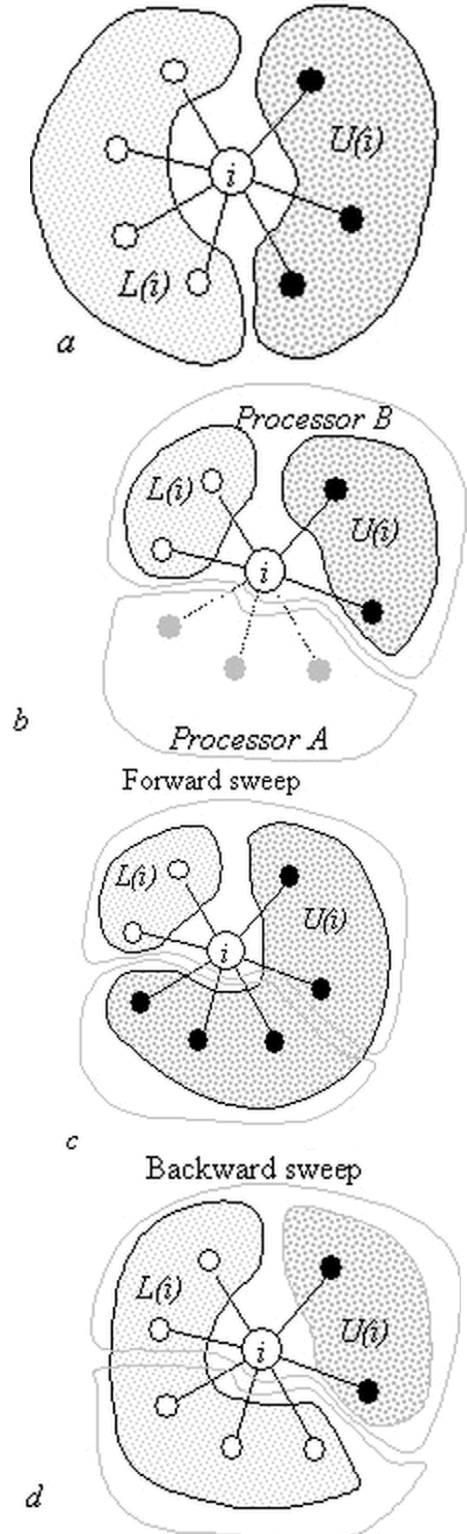


Figure 5. Stencil for LU-SGS scheme. (a) Internal point. (b) Interface point without interprocessor communications. (c) Hybrid LU-SGS forward sweep. (d) Hybrid LU-SGS backward sweep.

number of blocks is equal to the number of the grid points, the present implementation of the LU-SGS method represents the Jacobian method, which corresponds to the block-diagonal preconditioner. Our experience⁹ has shown that although the hybrid approach benefits for a large of blocks, LU-SGS scheme without interprocessor data exchange works almost as well as its hybrid counterpart for moderate number of processors, and therefore is used in the present work.

5. NUMERICAL RESULTS

All computations were initiated as a uniform flow at free-stream conditions, and advanced with a CFL number of 200. The relative L_2 norm of the density residual is taken as a criterion to test convergence history. The solution tolerance for GMRES is set to 0.1 with 10 search directions and 20 iterations. We observed that during the first few time steps, more iterations are required to solve the system of the linear equations: even 20 iterations can not guarantee that the stopping criterion will be satisfied for some cases. However, it takes less than 20 iterations to solve the linear equations at a later time, and global convergence is not affected by a lack of linear system convergence during the first few time steps. All computations were performed on a SGI origin 2000 with R12000 processors (400MHz) computer at ARL MSRC.

A. M6 wing configuration

The first test case is the well-documented case of transonic flow over a ONERA M6 wing configuration. Flow solution is presented at a Mach number of 0.84, a Reynolds number of 18.2 millions, and an angle of attack of 3.06° . The computation was performed on two different grids.

The first mesh obtained by generating prismatic elements in the boundary layer and tetrahedral elements elsewhere and then dividing the prismatic elements into the tetrahedral elements, consists of 2,953,333 elements, 504,790 grid points, and 24,166 boundary points. The minimum normal spacing at the wall is about $2.0E-06$ and the biggest stretching ratio of the tetrahedra is 11,961. The surface mesh used in the computation and the computed pressure contours on the upper surfaces are shown in Figs.6a-6b, respectively. The convergence history for 1, 2, 4, 8, an 16 processors is shown in Fig. 6c, where no convergence degradation is observed, demonstrating that the matrix-free implicit GMRES+LU-SGS method is insensitive to the number of partitioning blocks. The resulting speed-ups for this computation is shown in Fig. 6d, where the speedup is measured by timing total CPU on different number of processors.

The second mesh, generated by a new mesh generator¹⁵ with mesh clustering toward the corners, consists of 4,746,431 elements, 811,125 grid points,

and 32,018 boundary points. The upper surface meshes are shown in Fig. 7e. The computed pressure contours on the upper surfaces, displayed in Fig. 7f, clearly show the sharply captured lambda-type shock structure formed by the two inboard shock waves, which merge together near 90% semispan to form the single strong shock wave in the outboard region of the wing. Figs. 6g-6l show the comparison of experimental data¹⁶ and computed pressure coefficient distributions obtained using these two meshes at six spanwise stations, respectively. Clearly, the second mesh yields a much better solution in the vicinity of the trailing edge while the first mesh yields a nonphysical solution with a huge C_p jump at the trailing edge. The convergence history for 1, 2, 4, 8, an 16 processors is shown in Fig. 6m, where no convergence degradation is observed, demonstrating that the matrix-free implicit GMRES+LU-SGS method is insensitive to the number of partitioning blocks. The resulting speed-ups for this computation is shown in Fig. 6n. Note that it only takes about 74 minutes on 16 processors to get the converged solution.

Considering that more iterations, and therefore more CPU time are needed to achieve GMRES convergence for more partitioning blocks, the scalability is astonishingly good for this test case.

B. DLR-F4 configuration

As an example of application, the developed parallel method has been used to compute the first test case for the AIAA Drag Prediction Workshop¹⁷ held in Anaheim, CA June 2001. The workshop challenge was to compute the lift, drag, and pitching moment for the DLR-F4 wing-body configuration for different sets of conditions. Flow solution was computed at $M_\infty=0.75$, $C_L=0.5$, and at a Reynolds number of 3 millions. The standard unstructured grid used in the workshop contains 9,650,674 elements, 1,641,451 points, and 48,339 boundary points, as shown in Figure 7a. The computed pressure contours displayed in Fig. 7b, clearly show good resolution of the upper surface shock. Figs. 7c-h show the comparison of experimental data¹⁸ and computed pressure coefficient distributions at six spanwise stations, respectively. One can see that the results obtained compare closely with experimental data, although the upper surface pressure peak is lower than experimental data, and the post-shock Mach number is too high, as most of the participants in the DPW obtained. Table 1 summarizes the results obtained for case 1 by experiments, drag prediction workshop, and present computation. Although there is a considerable amount of scatter in the data obtained by the drag prediction workshop, our computational results are close to the average data, especially for drag and pitching moment. The convergence history of density residual and lift coefficient for 1, 2, 4, 8, an 16 processors is

shown in Fig. 7i, where no convergence degradation is observed, demonstrating that the matrix-free implicit GMRES+LU-SGS method is not very sensitive to the number of partitioning blocks. The resulting speed-ups for this computation is shown in Fig. 7j. Again, a fairly good scalability is obtained for this run. Note that it only takes about 5 hours on 16 processors to get the converged solution in 600 time-steps.

Table 1. Comparison of computational results with experimental data and Drag Prediction Workshop data for case 1

Data	α	C_L	C_D	C_M
ONERA	0.192	0.50	0.02896	-0.1260
NLR	0.153	0.50	0.02889	-0.1301
DRA	0.179	0.50	0.02793	-0.1371
Workshop(Ave)	-0.237	0.5002	0.03037	-0.1559
Workshop(Min)	-1.000	0.4980	0.02257	-0.2276
Workshop(Max)	1.223	0.5060	0.04998	0.0481
Present	-0.002	0.5009	0.03058	-0.1513

C. DLR-F6 configuration

Finally, the present parallel implicit method has been used to compute the turbulent flow past the DLR-F6 aircraft configuration with a conventional nacelle. Flow solution is presented at a Mach number of 0.75, a Reynolds number of 3 millions, and an angle of attack of 1° . The computational mesh, shown in Figure 3a, consists of 14,304,595 elements, 2,419,388 grid points, and 70,580 boundary points. The computed pressure contours is displayed in Fig. 3b. Figs. 3c-f show the comparison of experimental data²⁶ and computed pressure coefficient distributions at four spanwise stations, respectively, where two sections are located inside of the pylon and two outside of the pylon. One can see that the results obtained compare closely with experimental data, although for the pylon inboard section ($\eta=0.331$) a slightly over-prediction of C_p can be found at the leading edge on the upper side. The computed lift coefficient of 0.485 and drag coefficient of 0.0342 are compared reasonably well with the experimental values of 0.5 and 0.0338. The computation requires about 10 hours on 16 processors to get the converged solution in 1000 time-steps.

6. CONCLUSIONS

A parallel, matrix-free implicit GMRES+LU-SGS method has been presented on shared-memory, cache-based parallel computers. The parallelization is based on a special grid renumbering technique rather than the traditional domain partitioning approach. The method can be easily be combined with mesh refinement and remeshing procedures. OpenMP is used for implementing the parallel method in order to achieve maximum portability. The developed method has been used to compute turbulent flows for complex

configurations on unstructured grids. The numerical results in terms of parallel efficiency indicate that the scalability of the current parallel method on the SGI origin 2000 computer, while not excellent is certainly acceptable. It is demonstrated that the turnaround in a matter of hours for numerical simulation of high Reynolds number turbulent flows around realistic geometries has been achieved using the present parallel, matrix-free implicit method.

REFERENCES

- ¹H. Luo, J.D. Baum, and R. Löhner, "A Fast, Matrix-free Implicit Method for Compressible Flows on Unstructured Grids," *Journal of Computational Physics*, Vol. 146, 1998.
- ²H. Luo, J.D. Baum, and R. Löhner, "An Accurate, Fast, Matrix-free Implicit Method for Computing Unsteady Flows on Unstructured Grids," *Computers & Fluids* Vol. 30, No. 2, pp. 137-159, 2001.
- ³H. Luo, J.D. Baum, and R. Löhner, "A Fast, Matrix-free Implicit Method for Computing Low Mach Number Flows on Unstructured Grids," *International Journal of Computational Fluid Dynamics* Vol. 14, pp. 133-157, 2001.
- ⁴D. Williams, "Performance of Dynamic Load Balancing Algorithms for Unstructured Grid Calculations," Cal Tech Rep. C3P913, 1990.
- ⁵H. Simon, "Partitioning of Unstructured Problems for Parallel Processing," NASA Ames Tech. Rep., RNR-91-008, 1991.
- ⁶P. Mehrota, J. Saltz, and R. Voigt (eds.), "Unstructured Scientific Computation on Scalable Multiprocessors," MIT press, 1992.
- ⁷A. Vidwans, Y. Kallinderis, and V. Venkatakrisnan, "A Parallel Load Balancing Algorithm for 3-D Adaptive Unstructured Grids," AIAA Paper 93-3313, 1993.
- ⁸R. Löhner, "Renumbering Strategies for Unstructured-Grids Solvers Operating on Shared-memory, Cache-Based Parallel Machines," AIAA Paper 97-2045, 1997.
- ⁹D. Sharov, H. Luo, J.D. Baum, and R. Löhner, "Implementation of Unstructured Grid GMRES+LU-SGS Method on Shared-memory, Cache-Based Parallel Computers," AIAA Paper 2000-0927, 2000.
- ¹⁰P.R. Spalart, and S.R. Allmaras, "A One-equations Turbulence Model for Aerodynamic Flows", AIAA Paper 92-0439, 1992.
- ¹¹H. Luo, D. Sharov, J.D. Baum, and R. Löhner, "On the Computation of Compressible Turbulent Flows on Unstructured Grids," *International Journal for Computational Fluid Dynamics*, Vol. 14, pp. 253-270, 2001.
- ¹²Toro, E.F., Spruce, M., and Speares, W., "Restoration of the Contact Surface in the HLL Riemann Solver," *Shock Waves*, 4, 25, 1994.

- ¹³P. Batten, M. A. Leschziner, and U. C. Goldberg, "Average-State Jacobians and Implicit Methods for Compressible Viscous and Turbulent Flows", *Journal of Computational Physics*, 137, pp. 38-78, 1997.
- ¹⁴B. van Leer, "Towards the Ultimate Conservative Difference Scheme, II. Monotonicity and Conservation Combined in a Second Order Scheme," *Journal of Computational Physics*, Vol. 14, 1974.
- ¹⁵E. Cuthill, and J. McKee, "Reducing the Bandwidth of Sparse Symmetrical Matrices", *Proc. ACM Nat. Conf.*, pp.157-172, New York 1969.
- ¹⁶R. Löhner, "Some Useful Renumbering Strategies for Unstructured Grids", *Int. J. Num. Meth. Eng.*, Vol. 36, pp. 3259-3270, 1993.
- ¹⁷H. Sagan, "Space-Filling Curves", Springer Verlag, New York, 1994.
- ¹⁸A. Povitsky, P. J. Morris, "Parallel Compact Multi-Dimensional Numerical Algorithm with Application to Aeroacoustics", AIAA 99-3271, 1999.
- ¹⁹C. B. Jansen, "Implicit Multiblock Euler and Navier-Stokes Calculations", *AIAA Journal*, Vol. 32, No. 9, pp. 1808-1814, 1994.
- ²⁰C. Sheng, D. Hyams, et al., "Three-Dimensional Incompressible Navier-Stokes Flow Computations About Complete Configurations Using a Multiblock Unstructured Grid Approach", AIAA-99-0778, 1999.
- ²¹P. Stoll, P. Gerlinger, D., Bruggemann, "Domain Decomposition for an Implicit LU-SGS Scheme using Overlapping Grids", AIAA-97-1896, pp. 479-487, 1997.
- ²²A. W. Wissink, A. S. Lyrintzis, and R. C. Strawn, "Parallelization of a Three-Dimensional Flow Solver for Euler Rotorcraft Aerodynamics Predictions", *AIAA Journal*, Vol. 34, No. 11, pp. 2276-2283, 1996.
- ²³A. W. Wissink, A.S. Lyrintzis, and A.T. Chronopoulos, "A Parallel Newton-Krylov Method for Rotorcraft Flowfield Calculations", AIAA-97-22049, pp. 1060-1070, 1997.
- ²⁴J. Flower, S. Otto, and M. Salama, Optimal Mapping of irregular Finite Element Domains to Parallel Processors, pp.239-250, 1990.
- ²⁵V. Venkatakrishnan, H. D. Simon, T. J. Barth, "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids", *NASA Ames Tech. Rep.* RNR-91-024, 1991.
- ²⁶R. Löhner, and R. Ramamurti, "A Load Balancing Algorithm for Unstructured Grids", *Comp. Fluid Dyn.*, 5, pp.39-58, 1995.
- ¹⁵D. Sharov, H. Luo, J.D. Baum, and R. Löhner, "Unstructured Navier-Stokes Grid Generation at Corners and Ridges," AIAA Paper 2001-2600, 2001.
- ¹⁶V. Schmitt, and F. Charpin, "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers," *Experiment Data Base for Computer Program Assessment*, AGARD AR-138,1979.
- ¹⁷AIAA Drag Prediction Workshop. Anaheim, CA.
- <http://www.aiaa.org/tc/apa/dragpredworkshop/dpw.html>, June 2001.
- ¹⁸G. Redeker, "DLR-F4 Wing Body Configuration," Technical Report, AGARD AR-303, Vol. II, 1994.

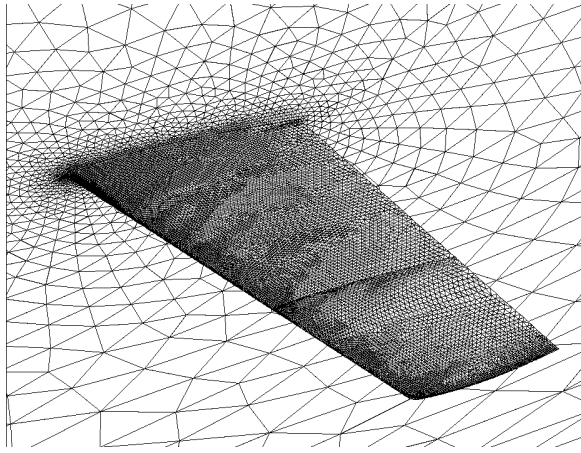


Fig. 6a: Mesh used for computing turbulent flow past M6 wing (nelem=2,953,333, npoin=504,790, nboun=24,166).

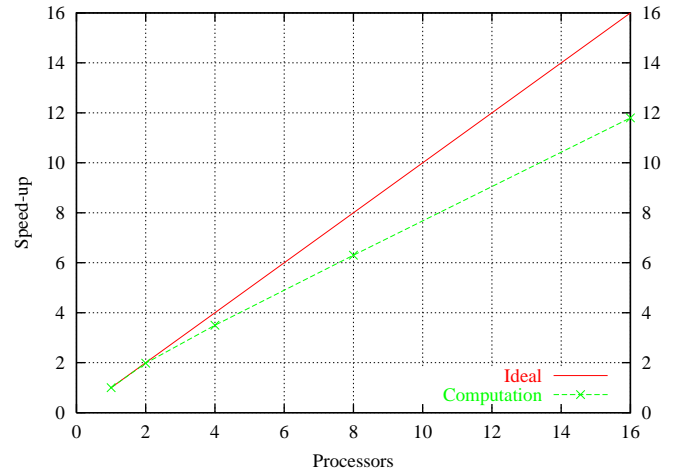


Fig. 6d: Speedups for computation of turbulent flow over M6 wing using different processors.

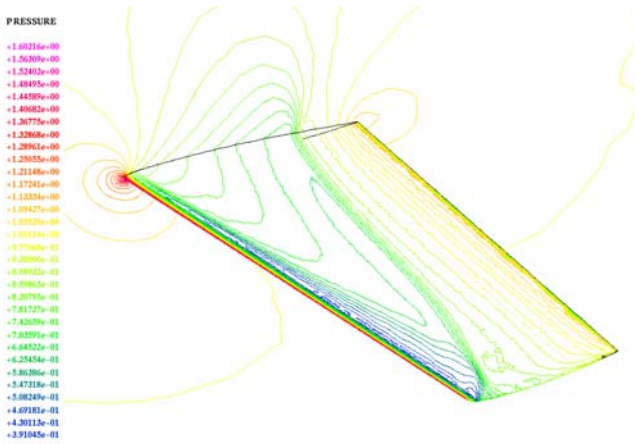


Fig. 6b: Computed pressure contours on the surface of the M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$, and $Re=18,200,000$.

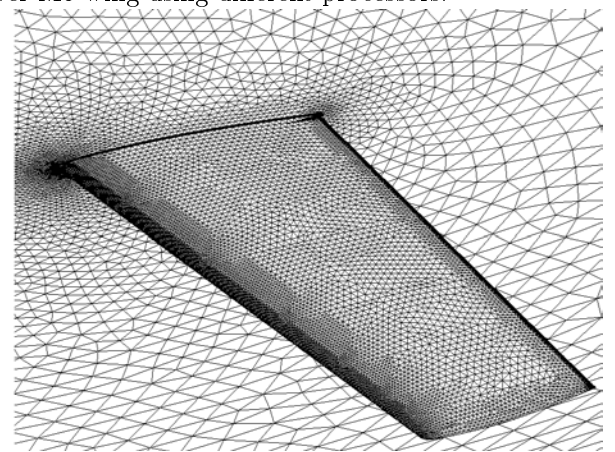


Fig. 6e: Mesh used for computing turbulent flow past M6 wing (nelem=4,746,431, npoin=811,125, nboun=32,018).

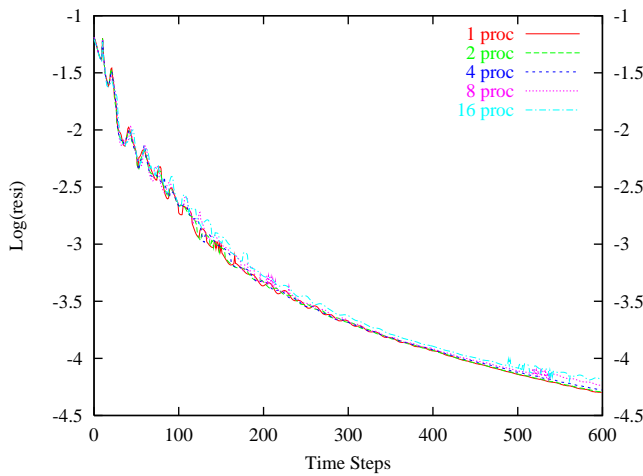


Fig. 6c: Residual convergence history versus time steps for turbulent flow over M6 wing using different processors.

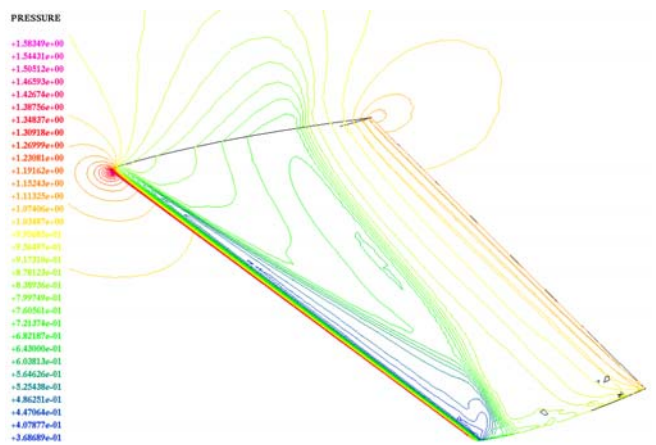


Fig. 6f: Computed pressure contours on the surface of the M6 wing at $M_\infty = 0.84$, $\alpha = 3.06^\circ$, and $Re=18,200,000$.

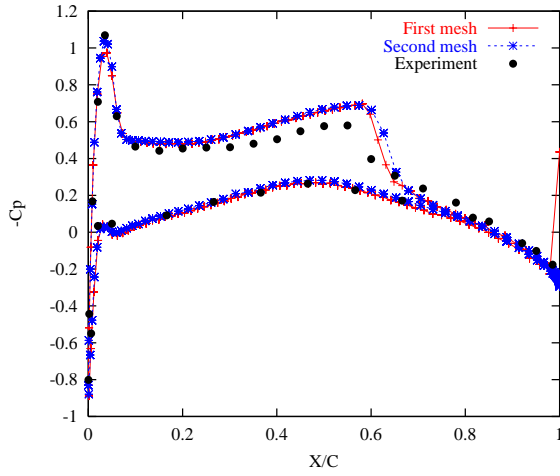


Fig. 6g: Comparison between experimental and computed pressure coefficient distributions for wing section at 20% semispan.

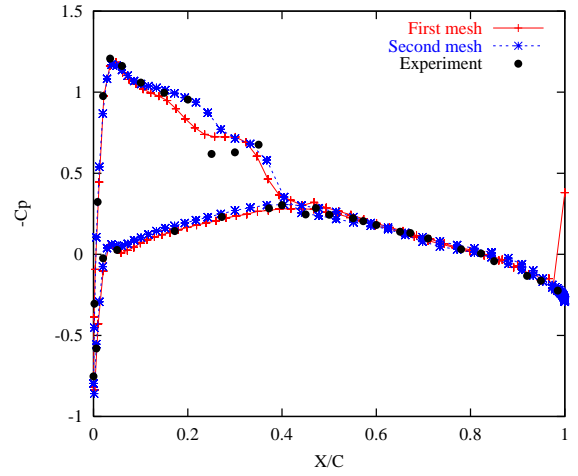


Fig. 6j: Comparison between experimental and computed pressure coefficient distributions for wing section at 80% semispan.

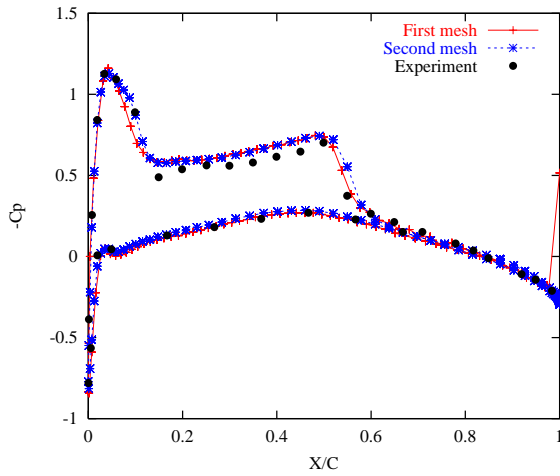


Fig. 6h: Comparison between experimental and computed pressure coefficient distributions for wing section at 44% semispan.

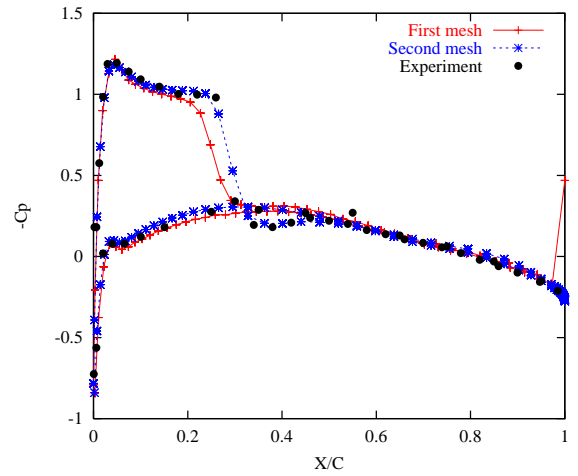


Fig. 6k: Comparison between experimental and computed pressure coefficient distributions for wing section at 90% semispan.

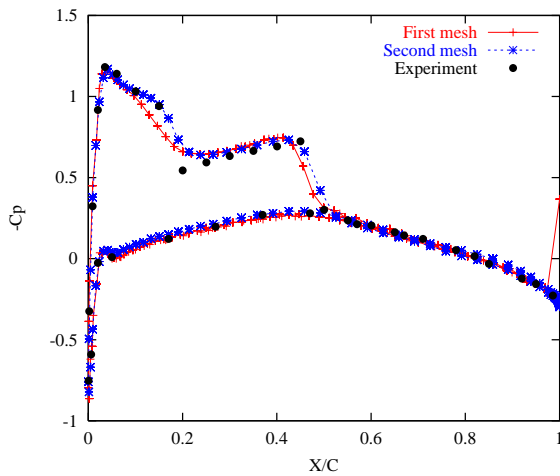


Fig. 6i: Comparison between experimental and computed pressure coefficient distributions for wing section at 65% semispan.

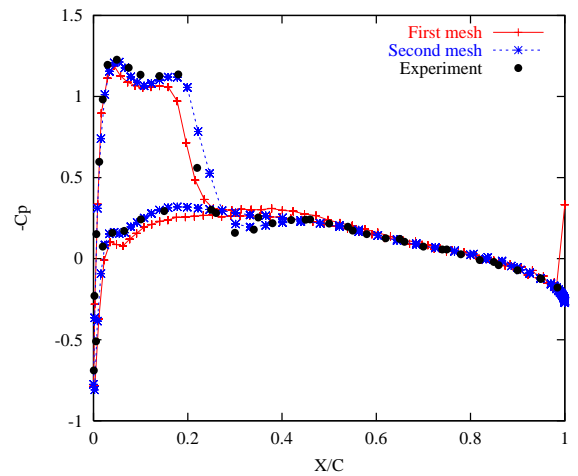


Fig. 6l: Comparison between experimental and computed pressure coefficient distributions for wing section at 95% semispan.

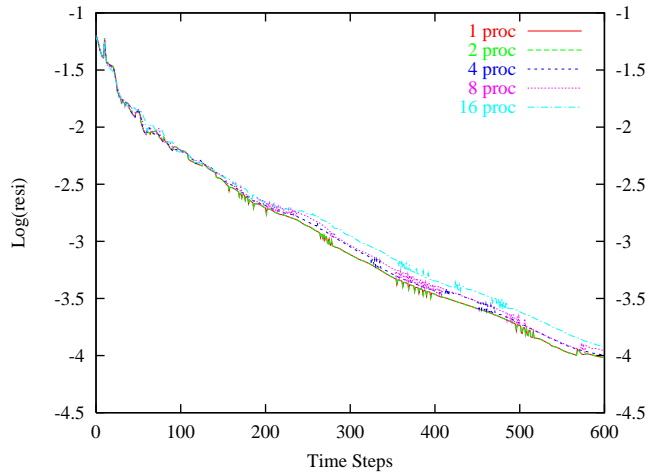


Fig. 6m: Residual convergence history versus time steps for turbulent flow over M6 wing using different processors.

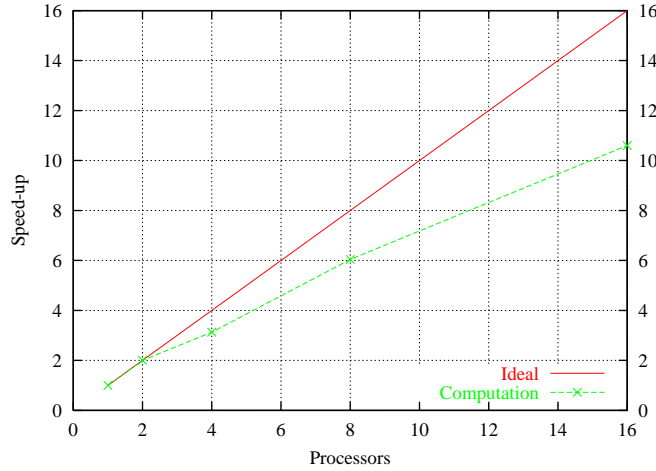


Fig. 6n: Speedups for computation of turbulent flow over M6 wing using different processors.

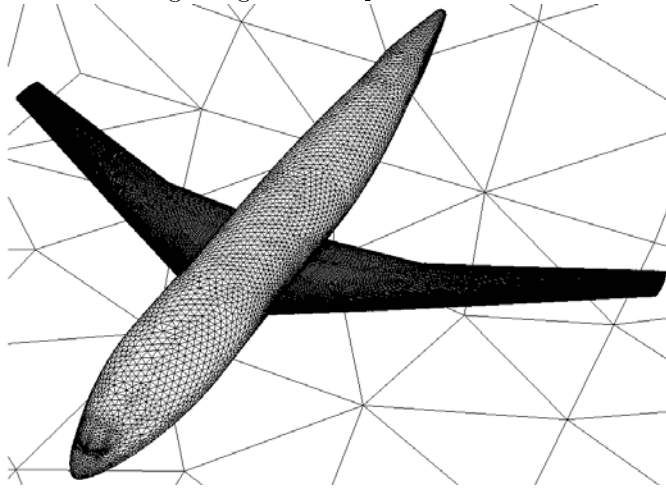


Fig. 7a: Unstructured surface mesh used for computing turbulent flow past DLR-F4 configuration (nelem=9,650,674, npoin=1,641,451, nboun=48,339).

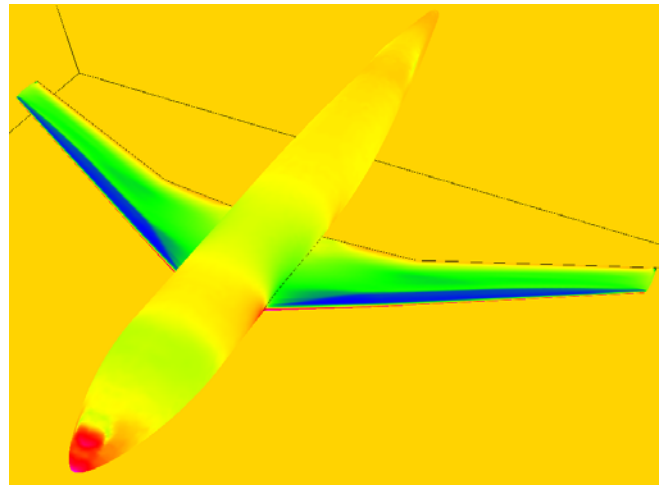


Fig. 7b: Computed pressure contours on the surface of DLR-F4 configuration at $M_\infty = 0.75$, $\alpha = 0.93^\circ$, and $Re=3,000,000$.

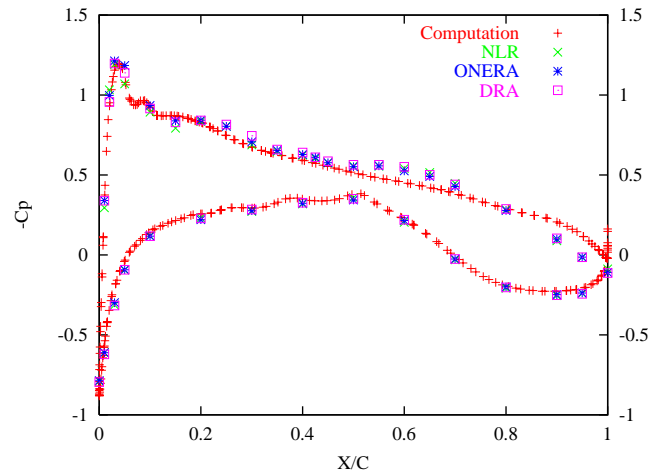


Fig. 7c: Comparison between experimental and computed pressure coefficient distributions for wing section at 18.5% semispan.

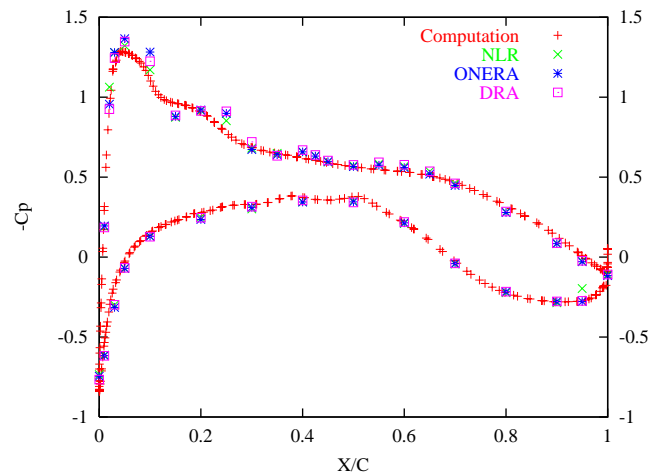


Fig. 7d: Comparison between experimental and computed pressure coefficient distributions for wing section at 23.8% semispan.

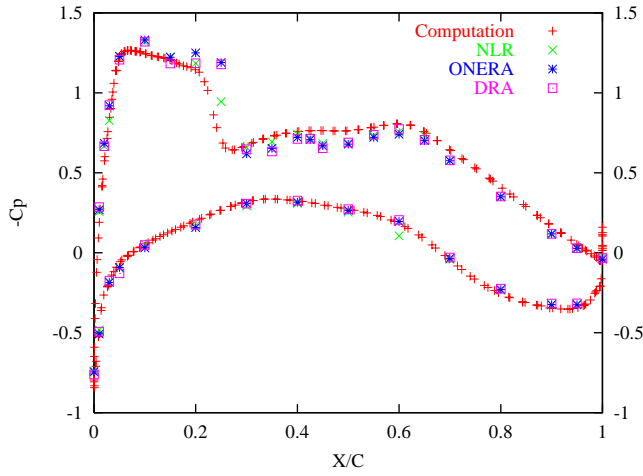


Fig. 7e: Comparison between experimental and computed pressure coefficient distributions for wing section at 41.1% semispan.

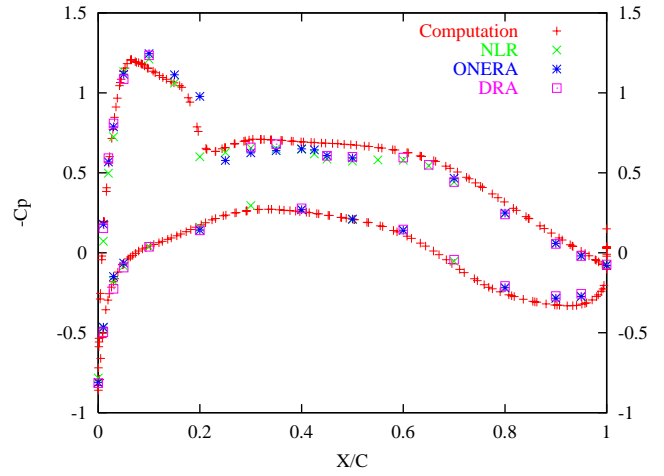


Fig. 7h: Comparison between experimental and computed pressure coefficient distributions for wing section at 84.7% semispan.

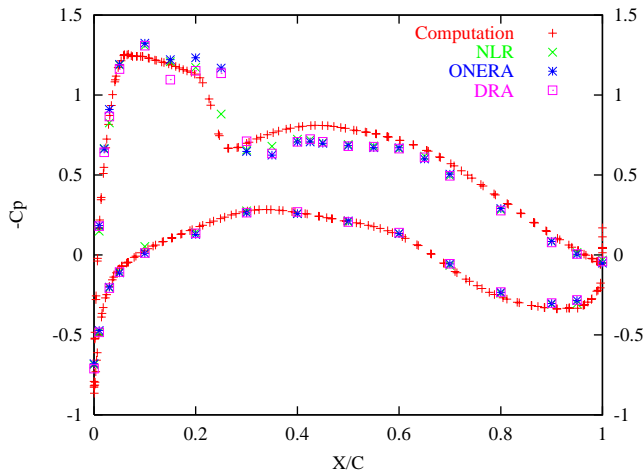


Fig. 7f: Comparison between experimental and computed pressure coefficient distributions for wing section at 51.4% semispan.

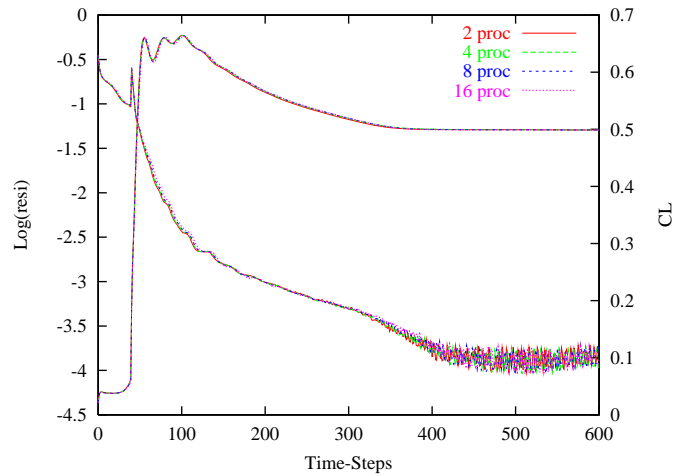


Fig. 7i: Residual convergence history versus time steps for turbulent flow over DLR-F4 configuration wing using different processors.

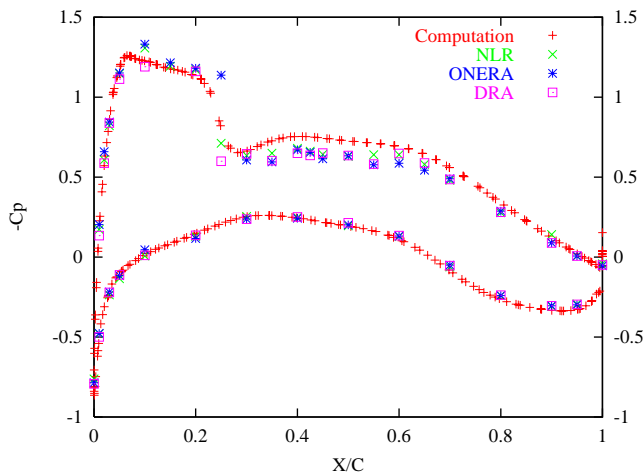


Fig. 7g: Comparison between experimental and computed pressure coefficient distributions for wing section at 63.8% semispan.

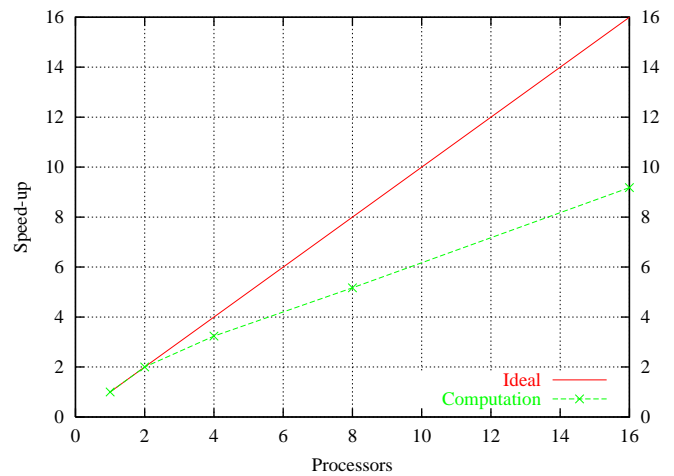


Fig. 7j: Speedups for computation of turbulent flow over DLR-F4 configuration wing using different processors.

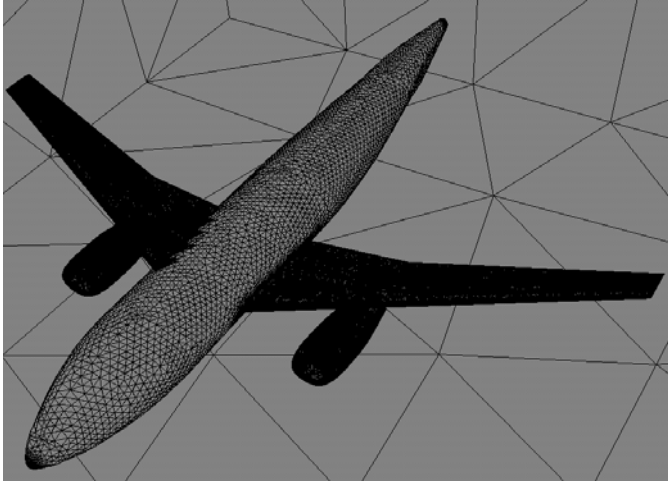


Fig. 3a: Unstructured surface mesh used for computing turbulent flow past DLR-F6 configuration (nelem = 14,304,595, npoin = 2,419,388, nboun = 70,580).

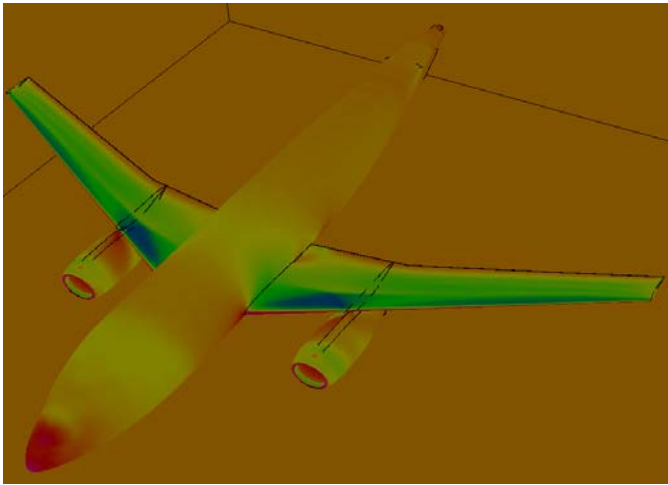


Fig. 3b: Computed pressure contours on the surface of DLR-F6 configuration at $M_\infty = 0.75$, $\alpha = 1^\circ$, and $Re=3,000,000$.

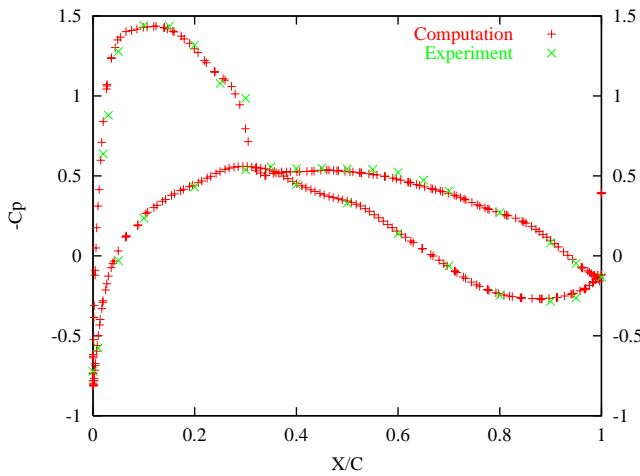


Fig. 3c: Comparison between experimental and computed pressure coefficient distributions for wing section at 23.9% semispan.

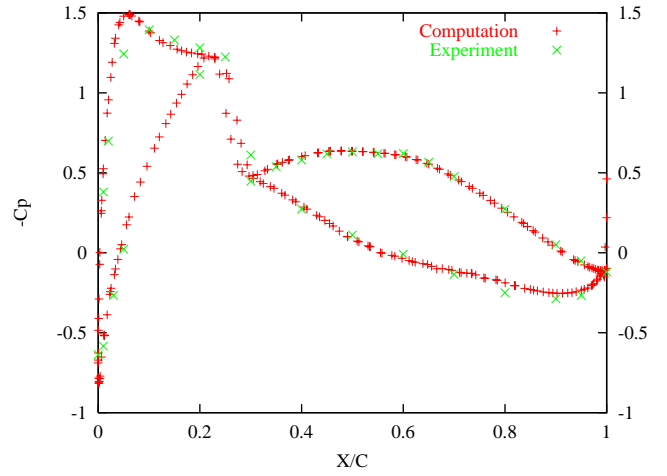


Fig. 3d: Comparison between experimental and computed pressure coefficient distributions for wing section at 33.2% semispan.

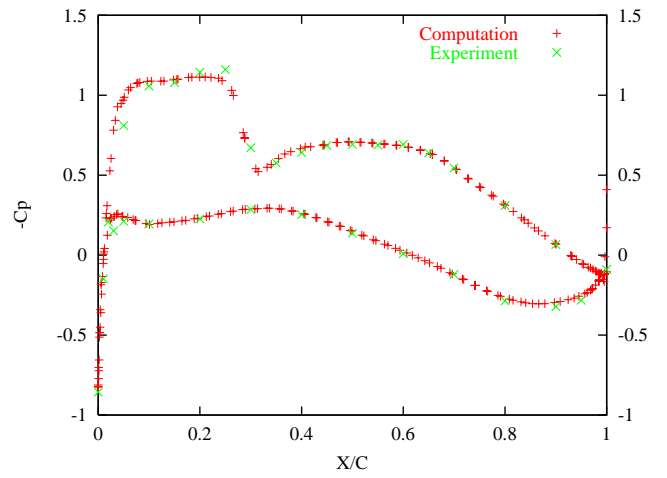


Fig. 3e: Comparison between experimental and computed pressure coefficient distributions for wing section at 37.7% semispan.

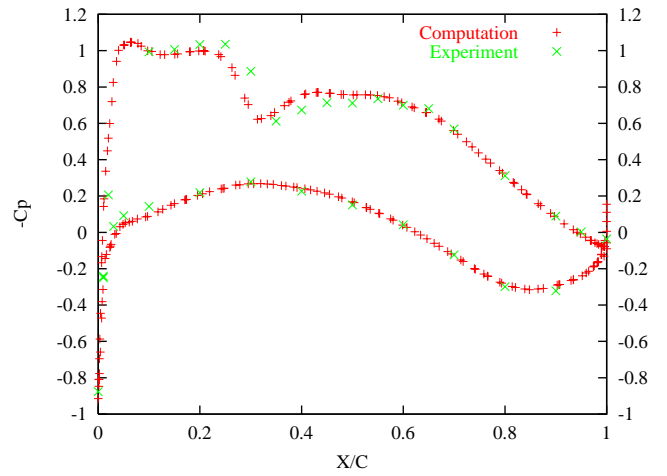


Fig. 3f: Comparison between experimental and computed pressure coefficient distributions for wing section at 44.1% semispan.