

IMPLEMENTATION OF UNSTRUCTURED GRID GMRES+LU-SGS METHOD ON SHARED-MEMORY, CACHE-BASED PARALLEL COMPUTERS

Dmitri Sharov, Hong Luo, Joseph D. Baum
 Science Applications International Corporation
 1710 Goodridge 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

The implementation of an unstructured grid matrix-free GMRES+LU-SGS scheme on shared-memory, cache-based parallel machines is described. A special grid renumbering technique is used for the parallelization rather than the traditional method of partitioning the computational domain. The renumbering technique helps to avoid inter-processor data dependencies, cache-misses, and cache-line overwrite while allowing pipelining. The resulting source code can be used with maximum efficiency and without modifications on traditional (scalar) computers, vector supercomputers, and shared-memory parallel systems. Special attention has been paid to develop an optimally parallelized preconditioner for the GMRES scheme.

1. INTRODUCTION

Considerable progress has recently been made in the development of implicit schemes for unstructured grids. The implicit methods are widely used to accelerate convergence of steady-state problems as well as to improve the efficiency of unsteady solvers by advancing the solution with substantially larger time steps. The GMRES+LU-SGS implicit scheme proposed by Luo, Baum, and Löhner for steady-state solutions¹ as well as for unsteady problems² can improve the efficiency of traditional explicit methods by one to more than two orders of magnitude. The scheme uses the Lower Upper-Symmetric Gauss-Seidel (LU-SGS) scheme as a preconditioner for the Generalized Minimal Residual (GMRES) method³. The LU-SGS scheme was originally proposed by Jameson and Yoon⁴ on structured grids, and has been successfully generalized and extended to unstructured meshes⁵⁻⁷.

Another way to reduce turn-around time is to use the multiple processors. With the advent of massively parallel machines, i.e. machines in excess of 500 nodes, the exploitation of parallelism in solvers has become a major focus of attention. Most of the applications ported successfully to parallel machines 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 balancing⁸⁻¹¹ has been a major focus of activity.

Despite the striking successes reported to date, only the simplest of all solvers: explicit timestepping 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, we mention local remeshing¹², repeated h-refinement such as required for transient problems¹³, contact detection and force evaluation¹⁴, some preconditioners¹⁵, applications where particles, flow, and chemistry

Copyright © 2000 by the authors. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

interact, and 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 machine (such as the SGI Origin 2000).

The objective of the present research effort is to implement the GMRES+LU-SGS scheme on shared memory parallel computers. Here we will use the shared memory parallelization technique, originally proposed by Löhner and implemented for explicit schemes¹⁶. This method is based on extensive mesh renumbering which provides proper load balancing and avoids cache-misses and cache-line overwrite while allowing pipelining. The advantage of the method over the traditional approach, which is based on domain partitioning, is its ability to be easily used with repeated local mesh refinement and local or global remeshing.

The matrix-free GMRES+LU-SGS implicit scheme uses the LU-SGS approximate factorization as a preconditioner. The parallelization of the LU-SGS algorithm is not an obvious task, because of inherent data dependency. The LU-SGS algorithm can be vectorized for vector processors by using planes $i+i+k=const$ for structured meshes, or by using “hyper plane” edge reordering for unstructured meshes⁷. Unfortunately, for the intended shared-memory parallelization approach, there are very severe penalties to start a loop¹⁶. Hence, a loop can be efficiently parallelized only if its vector length is large enough. For an explicit scheme, if scalability to even 16 processors is to be achieved, the vector loop lengths should be at least 16x1,000. For typical tetrahedral grids there are approximately 22 vector-length groups, indicating that we would need at least 22x16x1,000=352,000 edges to run efficiently. For the LU-SGS scheme this restriction is much more severe. Since we usually have several hundreds of “hyper planes”, 10 times or even 100 times more edges are required to run the code efficiently. Moreover, since the LU-SGS scheme is used as a preconditioner for the GMRES method, even a small inefficiency in parallelization of the LU-SGS scheme may result in severe degradation of overall performance. Thus a comparison of different types of matrix-free parallelized preconditioners has been performed as part of the present effort.

2. GOVERNING EQUATIONS AND THEIR DISCRETIZATION

The Euler equations governing unsteady compressible inviscid flows can be expressed in the conservative form as

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}^j}{\partial x_j} = 0, \quad (2.1)$$

where the summation convention has been employed. The unknown vector \mathbf{Q} , and inviscid flux vector \mathbf{F} are defined by

$$\mathbf{Q} = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix}, \quad \mathbf{F}^j = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p \delta_{ij} \\ u_j (\rho e + p) \end{pmatrix}. \quad (2.2)$$

Here ρ , p , 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 \mathbf{X}_i . This set of equations is completed by the addition of an equation of state.

The governing equations are discretized by the finite volume method based on dual mesh cells associated with the nodes of the mesh, where the control volumes are nonoverlapping dual cells constructed by the median planes of the tetrahedra. In the present study the numerical flux functions for inviscid fluxes at the dual mesh cell interface are computed using the AUSM+ (Advection Upwind Splitting Method) scheme¹⁷. Linear reconstruction of primitive variables is used with Van Albada limiter.

Equation (2.1) can be rewritten in a semi-discrete form as

$$V_i \frac{\partial \mathbf{Q}}{\partial t} = \mathbf{R}_i, \quad (2.3)$$

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.

3. SHARED MEMORY PARALLELIZATION TECHNIQUE

The parallelization technique for explicit schemes¹⁶ will be generalized for implicit computations in this paper. The method requires no explicit domain decomposition, but is based on the combination of several renumbering and data regrouping techniques developed to avoid or considerably minimize cache-misses, cache-line overwrite, and memory contention.

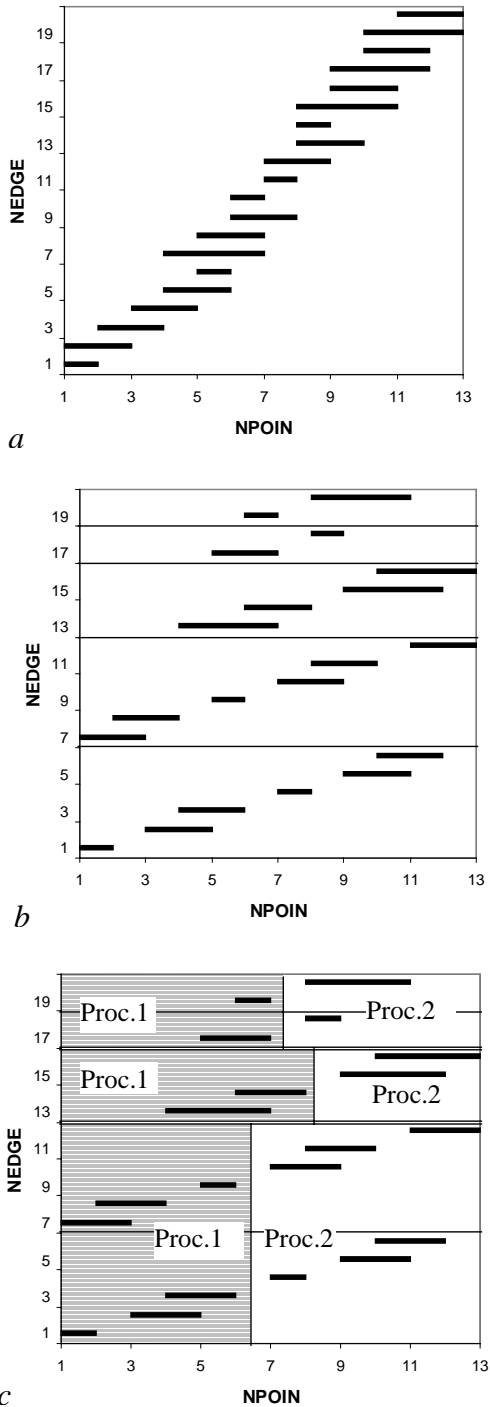


Fig. 1. Edge and node renumbering.
 (a) Renumbering to minimize cache-misses.
 (b) Renumbering to avoid memory contention.
 (c) Renumbering for 2-processor machine.

Renumbering to minimize cache-misses

All unstructured CFD codes contain basic loops over nodes, edges, 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 McKee¹⁸, wavefront¹⁹, or Peano-Hilbert type space-filling curves²⁰, and subsequently renumbering the edges according to the minimum point number on each edge¹⁹. Figure 1a shows an example of the reordered 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.

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=LNOED(1,IEDGE)
  IPOI2=LNOED(2,IEDGE)
  REDGE=F(IPOI1,IPOI2)
  RHSP0(IPOI1)=RHSP0(IPOI1)+REDGE
  RHSP0(IPOI2)=RHSP0(IPOI2)-REDGE
1600 CONTINUE
```

Since one and the same point can be accessed more than once from different edges, the information in RHSP0 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)
C$DIR IVDEP !PIPELINING DIRECTIVE
DO 1600 IEDGE=NEDG0,NEDG1
  IPOI1=LNOED(1,IEDGE)
```

```

IPOI2=LNOED(2,IEDGE)
REDGE=F(IPOI1,IPOI2)
RHSPO(IPOI1)=RHSPO(IPOI1)+REDGE
RHSPO(IPOI2)=RHSPO(IPOI2)-REDGE
1600 CONTINUE
1400 CONTINUE

```

```

1400 CONTINUE
RETURN

```

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. 16, 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. 16. 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 PARALLELIZATION DIRECTIVE
C$DOACROSS LOCAL(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)
C$DIR IVDEP !PIPELINING DIRECTIVE
DO 1600 IEDGE=NEDG0,NEDG1
IPOI1=LNOED(1,IEDGE)
IPOI2=LNOED(2,IEDGE)
REDGE=F(IPOI1,IPOI2)
RHSPO(IPOI1)=RHSPO(IPOI1)+REDGE
RHSPO(IPOI2)=RHSPO(IPOI2)-REDGE
1600 CONTINUE

```

There is no doubt that this algorithm can be applied to any explicit CFD solver. In the sequel, we consider the extension of this method to an implicit scheme.

4. IMPLICIT TIME INTEGRATION

In order to obtain a steady-state solution, the spatially discretized equations must be integrated in time. Using Euler implicit time-integration, Eq.(2.1) can be written in discrete form as

$$V_i \frac{\Delta \mathbf{Q}_i^n}{\Delta t} = \mathbf{R}_i^{n+1}, \quad (4.1)$$

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

$$\Delta \mathbf{Q}^n = \mathbf{Q}^{n+1} - \mathbf{Q}^n. \quad (4.2)$$

Equation (4.1) can be linearized in time as

$$V_i \frac{\Delta \mathbf{Q}_i^n}{\Delta t} = \mathbf{R}_i^n + \frac{\partial \mathbf{R}_i^n}{\partial \mathbf{Q}} \Delta \mathbf{Q}_i, \quad (4.3)$$

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

$$A \Delta \mathbf{Q} = \mathbf{R}, \quad (4.4)$$

where

$$A = \frac{V}{\Delta t} \mathbf{I} - \frac{\partial \mathbf{R}^n}{\partial \mathbf{Q}}. \quad (4.5)$$

We use a simplified flux function to obtain the left-hand side Jacobian matrix,

$$\mathbf{R}_i = -\frac{1}{2} \sum_j (\mathbf{F}(\mathbf{Q}_i, \mathbf{n}_{ij}) + \mathbf{F}(\mathbf{Q}_j, \mathbf{n}_{ij}) - \lambda_{ij} (\mathbf{Q}_j - \mathbf{Q}_i)) s_{ij} \quad (4.6)$$

where λ_{ij} is the spectral radius,

$$\lambda_{ij} = |\mathbf{v}_{ij} \cdot \mathbf{n}_{ij}| + c_{ij}, \quad (4.7)$$

where \mathbf{n}_{ij} is the unit vector normal to the cell interface, \mathbf{v}_{ij} is the velocity vector, and c_{ij} is the speed of sound.

Using an edge-based data structure, the left-hand side Jacobian matrix A is stored in lower, upper, and diagonal forms, which can be expressed as

$$A = L + U + D, \quad (4.8)$$

where

$$L_{ij} = \frac{1}{2} \left(-\frac{\partial \mathbf{F}(\mathbf{Q}_i, \mathbf{n}_{ij})}{\partial \mathbf{Q}} - \lambda_{ij} \mathbf{I} \right) s_{ij}, \quad (4.9)$$

$$U_{ij} = \frac{1}{2} \left(\frac{\partial \mathbf{F}(\mathbf{Q}_j, \mathbf{n}_{ij})}{\partial \mathbf{Q}} - \lambda_{ij} \mathbf{I} \right) s_{ij}, \quad (4.10)$$

$$D_{ii} = \frac{V_i}{\Delta t} \mathbf{I} + \frac{1}{2} \sum_j \left(\frac{\partial \mathbf{F}(\mathbf{Q}_i, \mathbf{n}_{ij})}{\partial \mathbf{Q}} + \lambda_{ij} \mathbf{I} \right) s_{ij}. \quad (4.11)$$

Equation (4.4) 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. In Ref. 1 it has been shown that the matrix-free GMRES+LU-SGS method results in very good convergence for unstructured meshes.

Since our goal is not to solve our system entirely by the LU-SGS approximate factorization but rather use the GMRES with appropriate preconditioner, the preconditioner must be very fast, and at the same time it should resemble the original Jacobian matrix A as close as possible. Preconditioning will be cost-effective only if the additional computational work incurred for each subiteration is compensated for by a reduction in the total number of iterations to convergence. Thus, even a moderate inefficiency in parallelization of the preconditioner can be critical.

Next, the following matrix-free methods are considered as candidates for the GMRES preconditioner:

1. The LU-SGS;
2. Data-Parallel Lower-Upper Relaxation (DP-LUR) method²¹, which by its nature is a Jacobi iterative method;
3. Symmetric Gauss-Seidel (SGS) relaxation method. The LU-SGS approximate factorization scheme is just a subset of the SGS method and corresponds to the SGS scheme with $k=1$.

1. The LU-SGS approximate factorization is described as following.

$$(D + L)D^{-1}(D + U)\Delta\mathbf{Q} = \mathbf{R} + (LD^{-1}U)\Delta\mathbf{Q} \quad (4.12)$$

Neglecting the last term on the right-hand side of Eq. (4.12), and assuming that

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Q}} \Delta\mathbf{Q} \approx \Delta\mathbf{F} = \mathbf{F}(\mathbf{Q} + \Delta\mathbf{Q}) - \mathbf{F}(\mathbf{Q}), \quad (4.13)$$

the system can be solved in the two steps. First, a lower (forward) sweep:

$$(D + L)\Delta\mathbf{Q}^* = \mathbf{R} \quad (4.14)$$

or, in matrix-free form:

$$\Delta\mathbf{Q}_i^* = D^{-1} \left[\mathbf{R}_i - \frac{1}{2} \sum_{j: j \in L(i)} (\Delta\mathbf{F}_j^* - \lambda_{ij} \Delta\mathbf{Q}_j^*) s_{ij} \right]; \quad (4.15)$$

and second, an upper (backward) sweep:

$$(D + U)\Delta\mathbf{Q} = D\Delta\mathbf{Q}^* \quad (4.16)$$

or:

$$\Delta\mathbf{Q}_i = \Delta\mathbf{Q}_i^* - D^{-1} \frac{1}{2} \sum_{j: j \in U(i)} (\Delta\mathbf{F}_j - \lambda_{ij} \Delta\mathbf{Q}_j) s_{ij} \quad (4.17)$$

The most remarkable feature 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. It is found that this approximation does not compromise any numerical accuracy, and the extra computational cost is negligible.

These sweeps can be vectorized with long vector lengths by using special ordering technique⁷, but parallelization of the LU-SGS algorithm is not straightforward due to inherent data dependencies.

2. The DP-LUR method has been successfully used in Ref. 21 as a substitute for the LU-SGS method for massively parallel computer implementation. The method has no inherent data dependencies, so it can be easily parallelized in the same way as an explicit scheme. The method can be described in the following way:

The first subiteration:

$$\Delta\mathbf{Q}^0 = D^{-1}\mathbf{R} \quad (4.18)$$

Then the k_{\max} subiterations are made using

$$\Delta\mathbf{Q}^{k+1} = D^{-1}(\mathbf{R} - (U + L)\Delta\mathbf{Q}^k), \quad (4.19)$$

which can be written in matrix-free form as

$$\Delta \mathbf{Q}_i^{k+1} = D^{-1} \left[\mathbf{R}_i - \frac{1}{2} \sum_j (\Delta \mathbf{F}_j^k - \lambda_{ij} \Delta \mathbf{Q}_j^k) s_{ij} \right], \quad (4.20)$$

where k is the subiteration number.

With this approach, the data that is required for each subiteration has already been computed during the previous subiteration. Therefore, the entire subiteration may be performed simultaneously, and there are no data dependencies.

3. Symmetric Gauss-Seidel relaxation.

First, zero the $\Delta \mathbf{Q}$ array:

$$\Delta \mathbf{Q}^0 = 0. \quad (4.21)$$

Then the k_{\max} subiterations are made using forward sweep:

$$(D + L)\Delta \mathbf{Q}^{k+1/2} = \mathbf{R} - U\Delta \mathbf{Q}^k \quad (4.22)$$

and then a backward sweep:

$$(D + U)\Delta \mathbf{Q}^{k+1} = \mathbf{R} - L\Delta \mathbf{Q}^{k+1/2} \quad (4.23)$$

which can be written in matrix-free form as forward sweep:

$$\Delta \mathbf{Q}_i^{k+1/2} = D^{-1} \left(\mathbf{R}_i - \frac{1}{2} \sum_{j \in L(i)} (\Delta \mathbf{F}_j^{k+1/2} - \lambda_{ij} \Delta \mathbf{Q}_j^{k+1/2}) s_{ij} - \frac{1}{2} \sum_{j \in U(i)} (\Delta \mathbf{F}_j^k - \lambda_{ij} \Delta \mathbf{Q}_j^k) s_{ij} \right) \quad (4.24)$$

and backward sweep:

$$\Delta \mathbf{Q}_i^{k+1} = D^{-1} \left(\mathbf{R}_i - \frac{1}{2} \sum_{j \in U(i)} (\Delta \mathbf{F}_j^{k+1} - \lambda_{ij} \Delta \mathbf{Q}_j^{k+1}) s_{ij} - \frac{1}{2} \sum_{j \in L(i)} (\Delta \mathbf{F}_j^{k+1/2} - \lambda_{ij} \Delta \mathbf{Q}_j^{k+1/2}) s_{ij} \right) \quad (4.25)$$

For one subiteration ($k_{\max}=1$), the SGS method is equivalent to the LU-SGS approximate factorization method.

5. PARALLELIZATION OF THE PRECONDITIONER

Since parallelization of the DP-LUR method is straightforward, we will discuss only the parallelization of the Symmetric Gauss-Seidel methods. There are two approaches to the solution of the problem:

1. Use a special scheduling algorithm which enables data parallelism by regrouping edges²². 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.

2. Split the computational domain into several nonoverlapping regions according to the number of processors, and apply the SGS method inside of each region with (or without) some special interprocessor boundary treatment²³⁻²⁷. This approach may suffer from convergence degradation but takes advantage of minimal parallelization overhead and good load balance.

Our experience with the shared memory SGI Origin 2000 computer has shown that the first method doesn't provide good scalability, so we will consider the second approach here.

For testing purposes we computed a transonic flow in a channel with a 10% circular bump on the lower wall. The length of the channel is 3, its height is 1, and its width is 0.5. The inlet Mach number is 0.675. This is a three-dimensional simulation of a two-dimensional flow. The tetrahedral mesh was automatically generated by the advancing front technique and contains 13,256 grid points, 64,595 elements, and 8,756 boundary triangles. The mesh and computed pressure contours are shown in Figs. 2a and 2b. All computations were run with essentially infinite time step (CFL=10⁴).

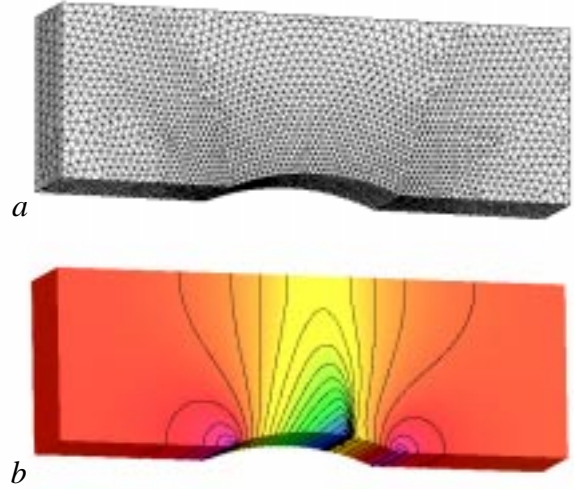


Fig. 2. Flow in channel with circular bump. (a) Surface mesh. (b) Computed pressure contours on the channel surface at $M=0.765$.

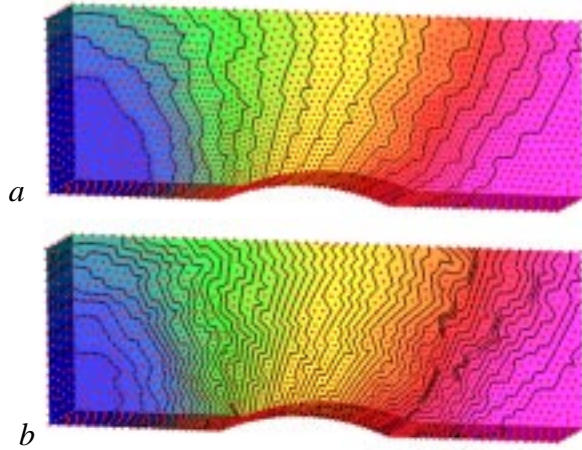


Fig. 3. Blocks with the wavefront renumbering. (a) 20 blocks. (b) 50 blocks.

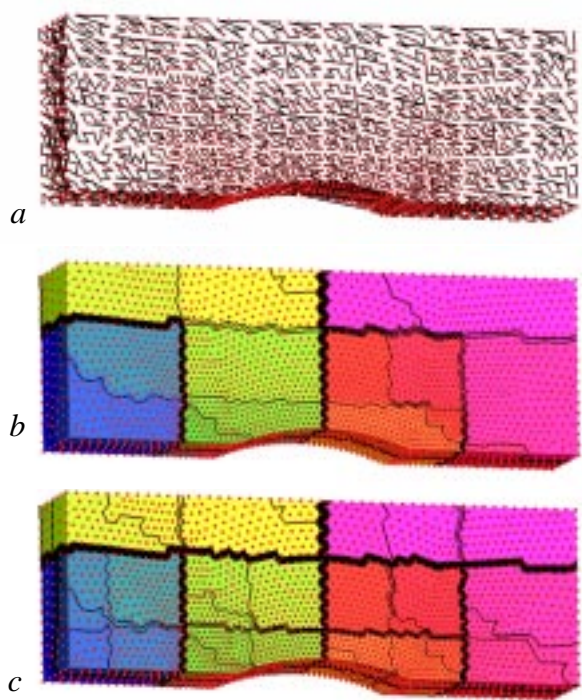


Fig. 4. (a) Peano-Hilbert-Morton space-filling curve (b) 20 blocks obtained with the Peano-Hilbert renumbering. (c) 50 blocks obtained with the Peano-Hilbert renumbering.

There are several methods to obtain a good partitioning of computational domain into blocks²⁸⁻⁵⁰. 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 doesn't provide good control over minimization of interprocessor boundary, it will be shown that 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. The partitioning into 20 blocks using the wavefront¹⁹ 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 algorithm²⁰. The 20 blocks partitioning corresponding to the Peano-Hilbert renumbering is shown in Fig. 4b, while 50 blocks partitioning is shown in Fig. 4c.

Next, the implementation of the LU-SGS scheme on parallel nonoverlapped blocks is considered. 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 (see Eqs.(4.24-25)).

At first, the SGS 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.

This approach has been tested using the LU-SGS scheme (without the GMRES) on 20 and 50 blocks, with the wavefront node renumbering. The test computation was performed on a single processor Pentium III PC. Figure. 6 demonstrates that convergence severely degrades for 20-block case and stalls for 50-block case.

The second approach for parallelization is the so-called hybrid LU-SGS or HLU-SGS. A similar scheme was used in Ref. 27 for structured grids. This scheme uses the DP-LUR for interprocessor edges, and regular SGS scheme for edges internal to each block. It is easier to consider Eqs. (4.24-25) to understand the

method. Schematically, the method is shown in Figs. 5c and 5d. Figure 5c corresponds to the forward sweep, and Fig. 5d corresponds to the backward sweep of the SGS procedure. In the 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.

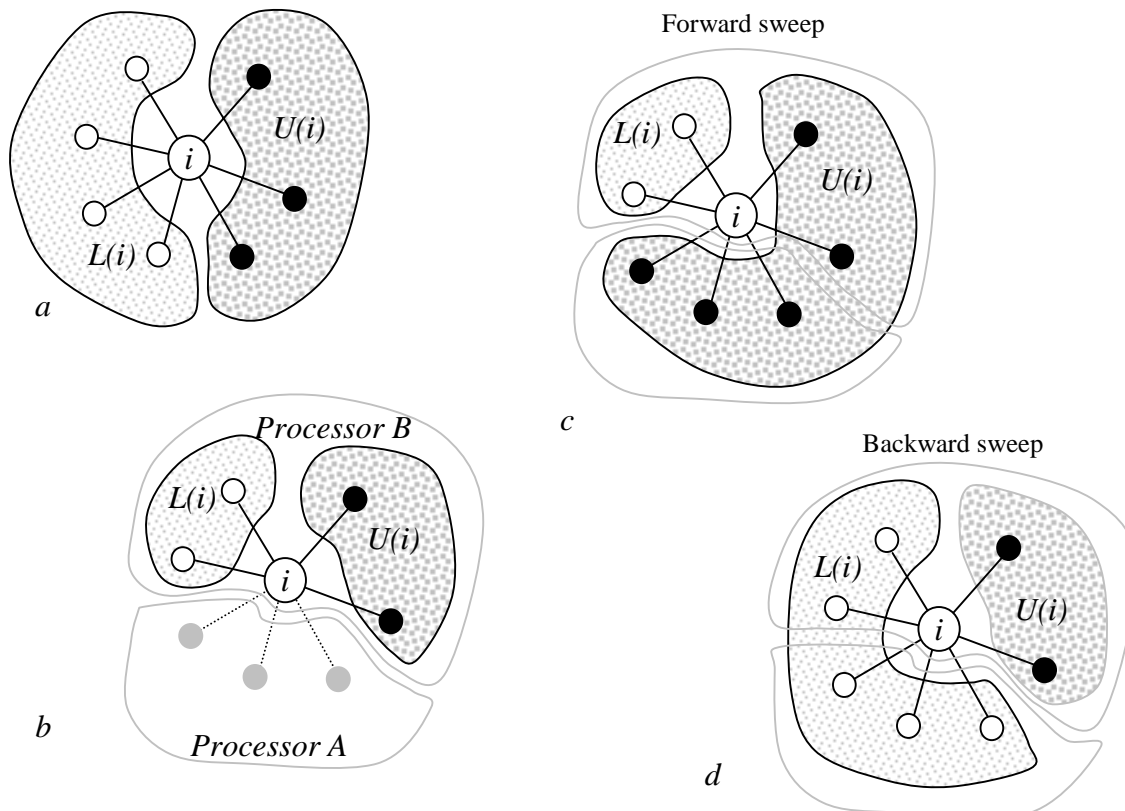


Fig. 5. Stencil for Gauss-Seidel scheme. (a) Internal point. (b) Interface point without interprocessor communications. (c) Hybrid SGS forward sweep. (d) Hybrid SGS backward sweep.

The results of HLU-SGS computation (with $k=1$) on 20 and 50 blocks are shown in Fig. 6. It is demonstrated that the hybrid scheme has some advantages over the LU-SGS scheme.

Next, consider how the LU-SGS, HLU-SGS, and DP-LUR schemes work as a preconditioner for the GMRES method. In our computations we used the same version of GMRES as in Ref. 1 with 10 search directions, 20 iterations and solution tolerance set to 0.1.

Results of the DP-LUR scheme as preconditioner are shown in Fig. 7a. The advantages of this method are its easy parallelization and lack of dependency on the number of processors. The influence of the number of

subiterations k_{\max} has also been represented in Fig. 7a. Note that $k_{\max}=0$ is equivalent to a diagonal preconditioner. It is inefficient to use more than one subiteration, since increase of k_{\max} yields no improvement in the convergence rate. When the DP-LUR scheme is used not as a preconditioner, the result is reversed: increase in the number of subiterations improves performance.

Figure 7b illustrates the influence of number of subiterations in the SGS preconditioner on convergence. These computations were performed using one single block. The test with $k=1$, which is equivalent to the LU-SGS preconditioner, gives the best performance overall, in contrast with results obtained with the SGS scheme alone, which converges better

when more subiterations are used (usually up to 20). This can be explained by the fact that the GMRES iterative procedure is more efficient than the SGS iterations.

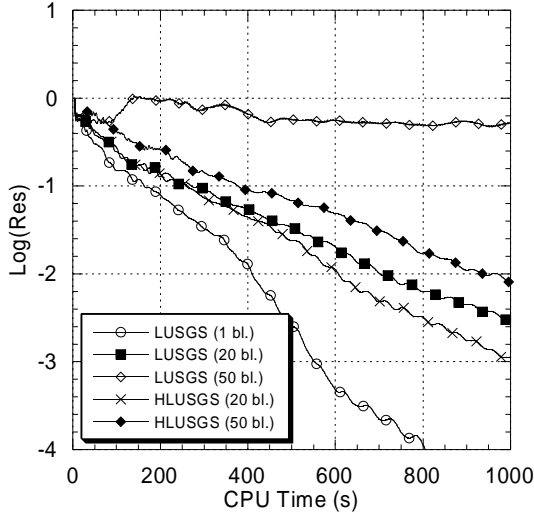


Fig. 6. Convergence history for LU-SGS and hybrid LU-SGS schemes without GMRES on 1, 20, and 50 blocks with wavefront renumbering.

Next, increasing the number of blocks is considered. Previously, some authors²⁷ used several preconditioner subiterations to reduce its degradation. Our results using 50 blocks (Fig. 7c), show that the increasing the number of subiterations actually leads to slower convergence.

It was demonstrated that with increasing of number of blocks, the LU-SGS scheme suffers from performance degradation. Let's check how this fact influences the behavior of the GMRES scheme. Figure 8a shows convergence rates comparisons for 1, 10, 20, and 50 blocks using a simple LU-SGS preconditioner without hybrid treatment of interprocessor boundaries and with the wavefront node renumbering. The diagonal preconditioner result is also shown because it represents the worst scenario of the LU-SGS scheme, when the number of blocks is equal to the number of grid points. Figure 8b shows the corresponding results for the hybrid LU-SGS scheme. The hybrid scheme is a better choice for large number of blocks. The worst case for the hybrid scheme corresponds to the DP-LUR preconditioner with $k_{\max}=1$.

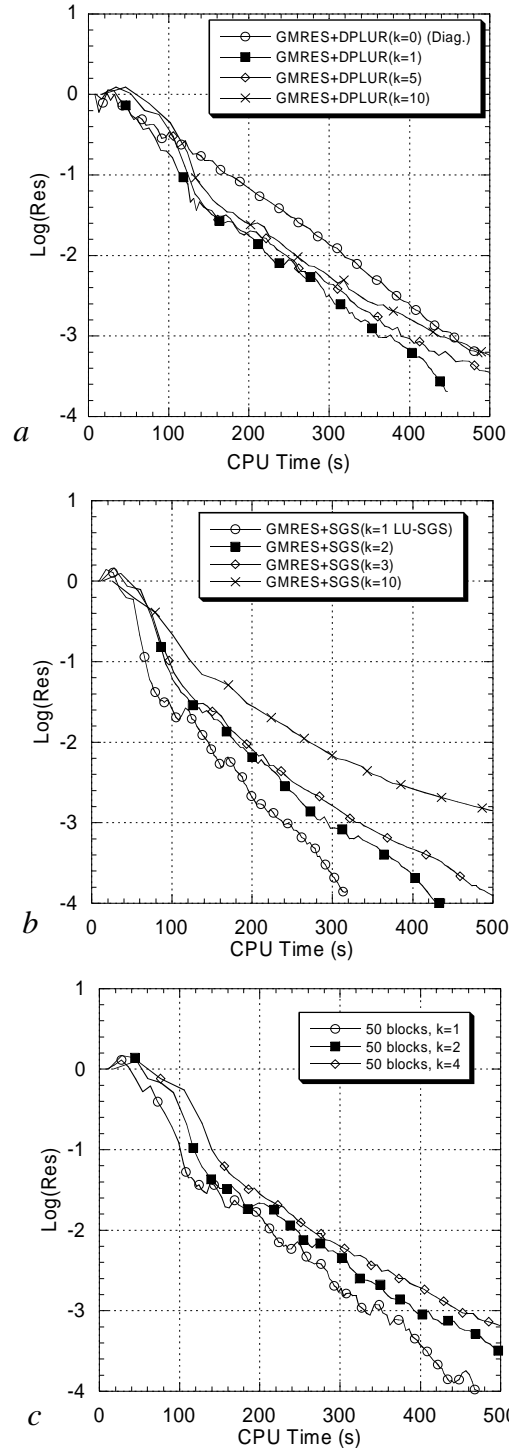


Fig. 7. Convergence history. (a) GMRES+DP-LUR scheme, single block. (b) GMRES+SGS scheme, single block. (c) GMRES+Hybrid SGS scheme, 50 blocks.

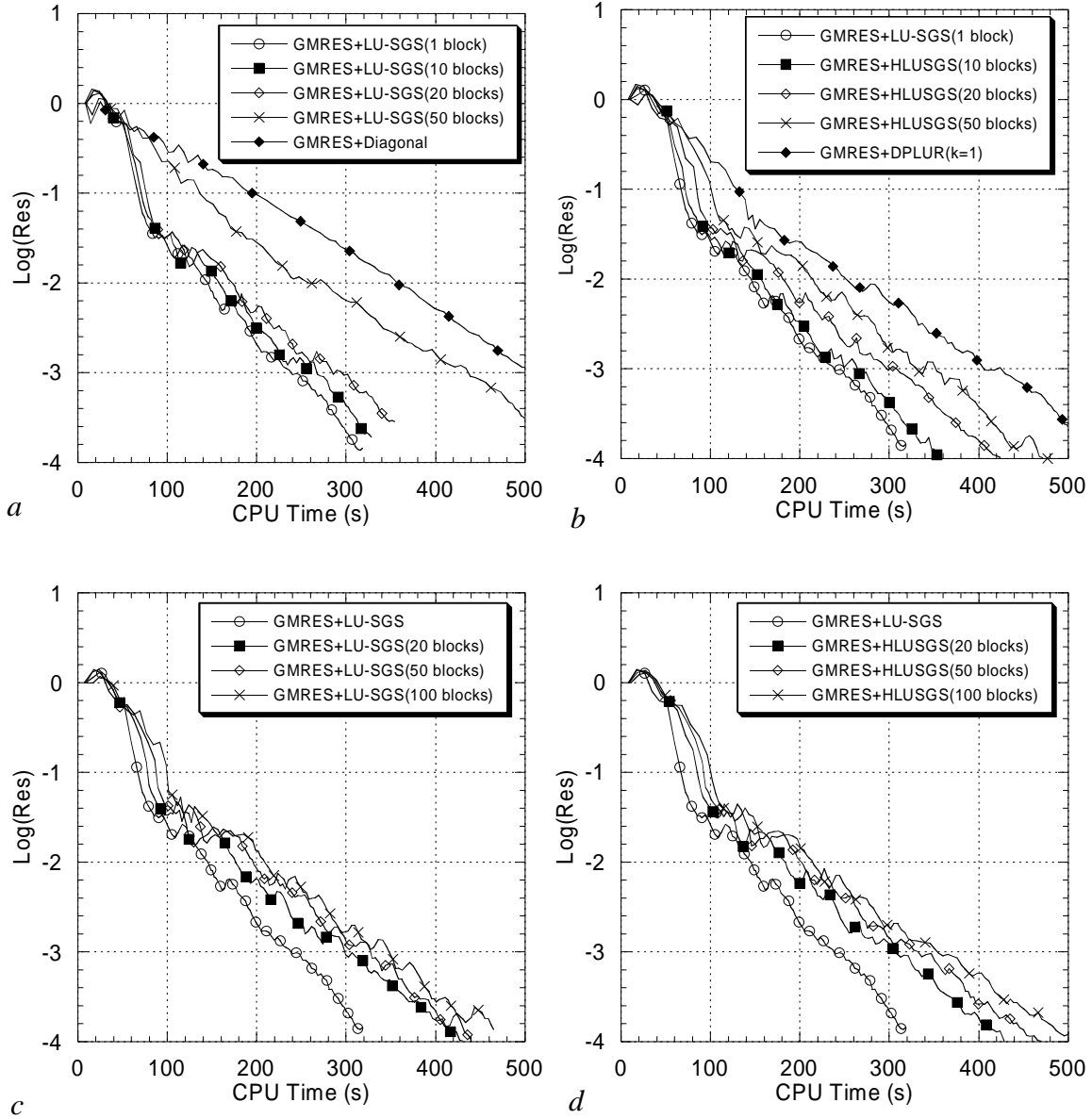


Fig. 8. Convergence history. (a) GMRES+LU-SGS with wavefront renumbering. (b) GMRES+hybrid LU-SGS with wavefront renumbering. (c) GMRES+LU-SGS with Peano-Hilbert renumbering. (d) GMRES+hybrid LU-SGS with Peano-Hilbert renumbering.

Figures 8c and 8d show the results of computations with the Peano-Hilbert renumbering. Comparison with the results shown in Figs.8a and 8b shows that the gain from a good renumbering technique is much more important than gain from the hybrid SGS scheme.

We conclude that the LU-SGS scheme being used as a preconditioner for the GMRES is not very sensitive to partitioning into blocks. Both the LU-SGS scheme

and the HLU-SGS scheme can be used as a preconditioner. When large number of processors is required it is better to pay attention to domain-splitting technique, in our case Peano-Hilbert reordering gives good results. If the number of processors doesn't exceed 20, it is not important how the domain is divided into blocks.

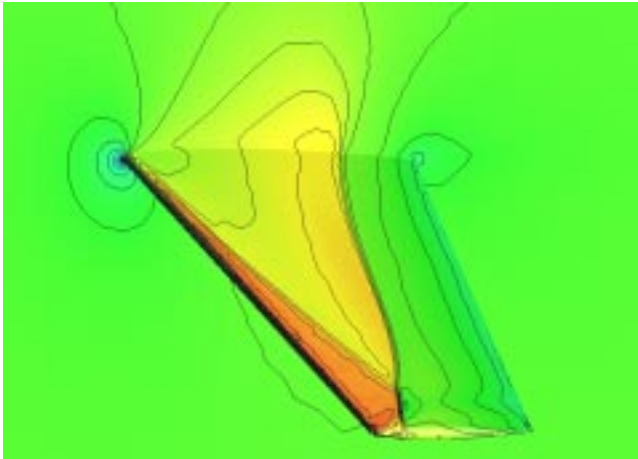


Fig. 9. ONERA M6 wing. Absolute velocity contours. $M=0.84$, angle of attack 3.06°

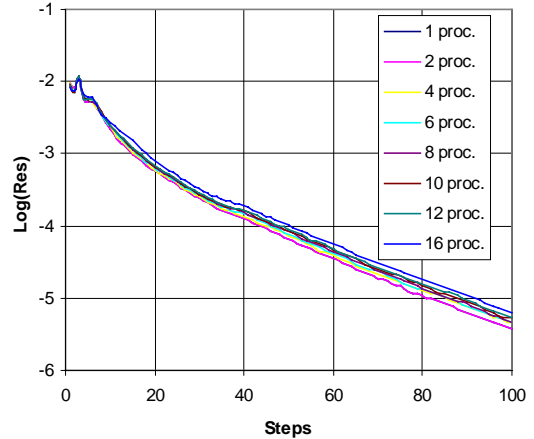


Fig. 11. Residual convergence history versus time steps for ONERA M6

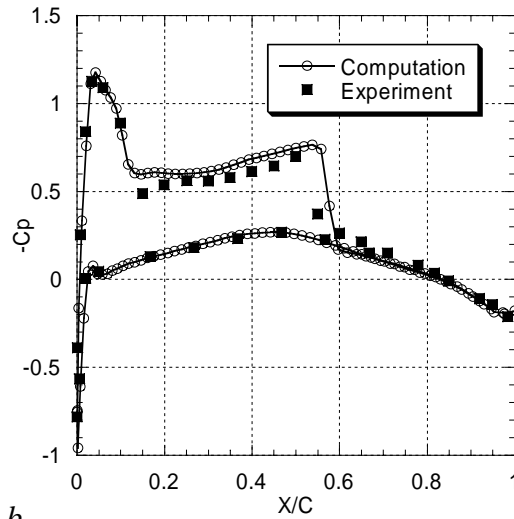
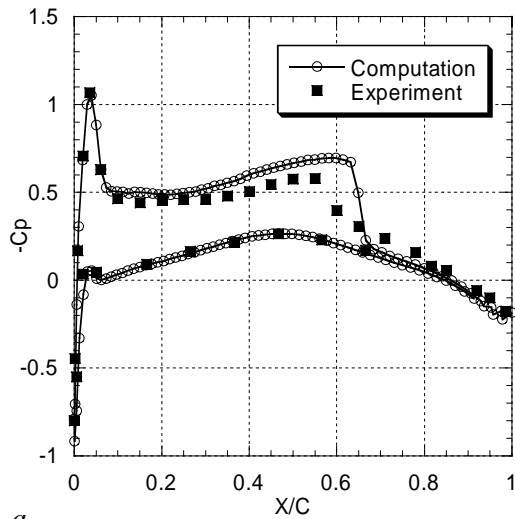


Fig.10. ONERA M6. C_p profile at 20% semispan (a) and 44% semispan (b).

6. APPLICATION OF THE GMRES+LU-SGS SCHEME TO LARGE-SCALE COMPUTATIONS

The computations were performed on a SGI Origin 2000 computer with R10000 processors.

ONERA M6 Wing Configuration

The first application is an inviscid transonic flow over a ONERA M6 wing. The M6 wing has a leading-edge sweep angle of 30° , an aspect of 3.8, and a taper ratio of 0.562. The airfoil section of the wing is the ONERA "D" airfoil, which is a 10% maximum

thickness-to-chord ratio conventional section. The flow solutions are presented at a Mach number of 0.84 and an angle of attack of 3.06° . The mesh used in the computation consists of 741,095 elements, 136,051 grid points, and 20,762 boundary points. The computed absolute velocity contours on the wing surface are displayed in Fig. 9. The upper surface contours clearly show the sharply captured Lambda-type shock structure formed by the two inboard shock waves, which merge near the 80% semispan to form the single strong shock wave in the outboard region of the wing. The computed pressure coefficient distributions are compared with experimental data³¹ in Fig. 10. We can observe that

there is only one grid point within the shock structure; this demonstrates the sharp shock-capturing ability of the AUSM+ scheme. The results obtained compare closely with the experimental data. The convergence history for 1, 2, 4, 6, 8, 10, 12, and 16 processors is shown in Fig. 11. No serious convergence degradation is observed.

Wing/Pylon/Finned-Store (Eglin) Configuration

Another test case was conducted for the wing/pylon/finned-store configuration reported in Ref. 32, which consists of a clipped delta wing, 45° sweep, composed of a constant NACA 64010 symmetric airfoil section. The wing has a root chord of 16in, a semispan of 13in, and a taper ratio of 0.134. The pylon is located at the midspan station. The width of the pylon is 0.294in. A constant NACA0008 airfoil section with a leading-edge sweep of 45° and a truncated tip defines the four fins of the store. The mesh used in the computation is shown in Fig.12a. It contains 1,329,694 elements, 239,547 grid points, and 27,359 boundary points. The flow solutions are presented at a Mach number of 0.95 and an angle of attack of 0°. Figures 12b and 12c show the pressure contours on the upper and lower wing surfaces, respectively. The convergence history for the computation with 6 processors is shown in Fig. 13.

The resulting speedups for the bump case, ONERA M6 case, and the Eglin case are shown in Fig. 14. The speedup was measured by timing CPU time of one time step on different number of processors. The performance degrades with the number of processors. This is to be expected, as the increasing number of passes results in higher relative loop costs. The resulting speedup is very similar to one obtained for the explicit scheme, see Ref. 16.

Time Accurate Simulation of Aircraft Canopy Trajectory

For the time accurate implicit computation we use the same method applied in Ref. 2. The method is based on pseudo-timesteps. In this method, Eq. (2.3) is transformed to the following form:

$$\frac{V^{n+1}\mathbf{Q}^{n+1} - V^n\mathbf{Q}^n}{\Delta t} + \frac{\partial V\mathbf{Q}}{\partial \tau} = (1-\alpha)\mathbf{R}^{n+1} + \alpha\mathbf{R}^n \quad (6.1)$$

where τ is the pseudo time variable, n denotes the time level. If $\alpha=1$, the scheme is the backward Euler method. If $\alpha=0.5$, the resulting scheme known as Crank-Nicholson method is second-order in time.

This yields the following system of linear equations

$$\left(\frac{V^{n+1}}{1-\alpha} \left(\frac{1}{\Delta t} + \frac{1}{\Delta \tau} \right) \mathbf{I} - \frac{\partial \mathbf{R}^m}{\partial \mathbf{Q}} \right) \Delta \mathbf{Q}^m = \mathbf{R}^m - \frac{V^{n+1}\mathbf{Q}^n}{(1-\alpha)\Delta t} + \frac{1}{1-\alpha} \left(\frac{V^n\mathbf{Q}^n}{\Delta t} + \alpha\mathbf{R}^n \right) \quad (6.2)$$

It has been shown in Ref. 2 that the fully implicit scheme is more accurate than its linearized counterpart, since it requires several subiterations to achieve convergence on each time step.

The new method was used to compute an F/A18-C/D fighter canopy ejection. During the initial opening of the canopy, a number of topological changes occur in the geometry. The problem was computed with Mach number 0.76. The computation was started at $t=26$ ms, when the canopy just started to move, see Fig. 15, and ended at $t=105$ ms when the canopy has moved to the tail part of the plane. At initial stage the canopy was hinged to the plane. After rotating 45 degrees (at $t=40$ ms), the canopy was released and allowed to move in response to the forces exerted on it. The mesh at several time instances as well as velocity field is shown in Fig. 16. The average size of the mesh was 250,000 points and 1,300,000 elements. Mesh size varied a little during the computation. More detailed data on this problem can be found in Ref. 33. A time step corresponding to the CFL number of 50 was used in the computation. The simulation required approximately 4 CPU hours on 8 processors of SGI Origin 2000. The plot of CPU time vs. problem time is shown in Fig. 17. The curve is not straight but rather has 8 steps attributed to global remeshing required by the algorithm. The remeshing was done by a new parallel algorithm described in Ref. 34. This computation is more than 10 times faster as compared to our explicit computation, see Ref. 33. Only 184 time steps were required (compare with approximately 32,000 time steps with explicit scheme). This significant reduction in CPU requirements is attributed to the implicit GMRES scheme and parallel remeshing. Figure 18 shows the speedup of the time accurate computation. This speedup was computed by measuring the CPU time for a single timestep. The time accurate speedup result is very similar to those obtained for the steady-state cases.

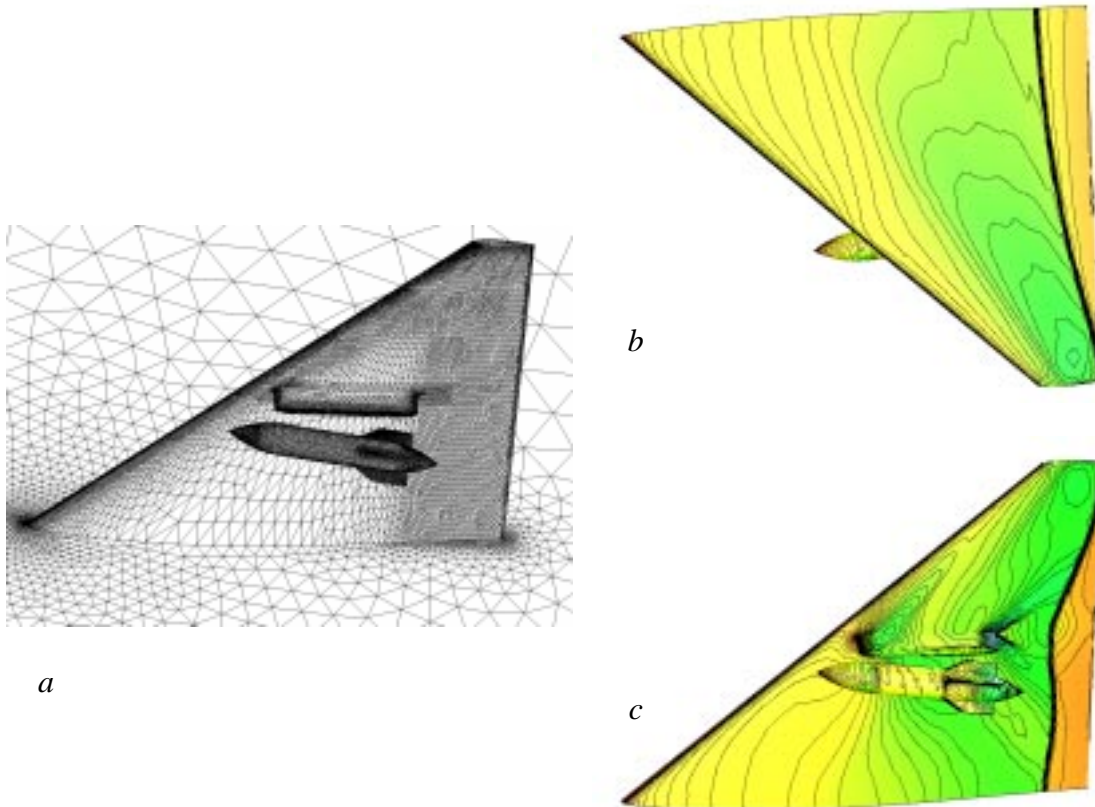


Fig. 12. (a) Surface mesh used for Wing/Pylon/Finned-Store configuration. (b) Computed pressure contours on the upper surface. (c) Computed pressure contours on the lower surface.

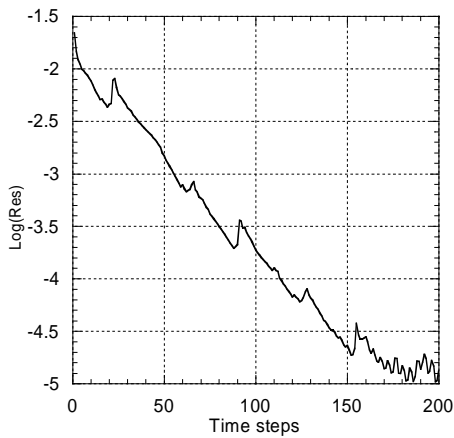


Fig. 13. Residual convergence history versus time steps for Wing/Pylon/Finned-Store configuration on 6 processors.

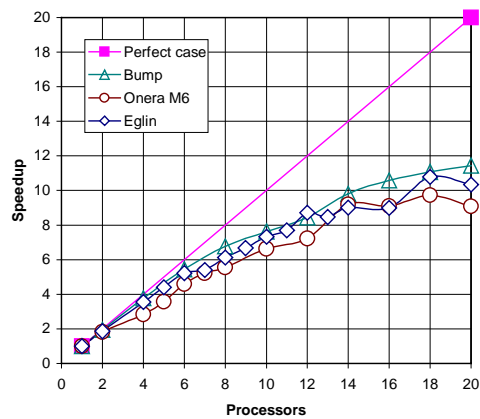


Fig. 14. Speedups in computations of the channel with circular bump, ONEA M6, and Wing/Pylon/Finned-Store (Eglin) configurations.

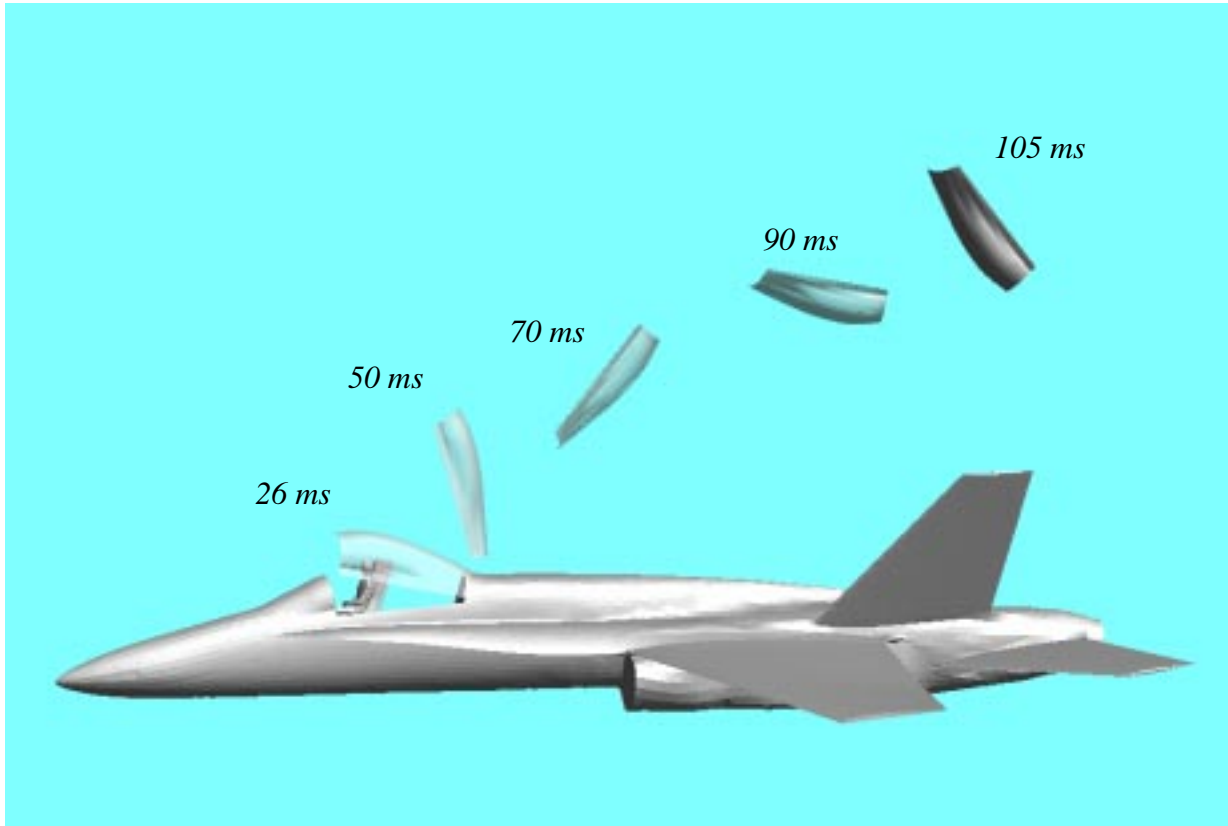


Fig. 15. F/A18-C/D fighter canopy ejection.

CONCLUSIONS

A parallelization technique for matrix-free GMRES+LU-SGS unstructured grid method on shared-memory machine is proposed. The method requires no direct domain partitioning and can easily be combined with mesh refinement and remeshing procedures. Special attention is given to parallel implementation of GMRES preconditioner. It is shown that for moderate number of processors, the LU-SGS method without interprocessor data exchange is a good choice. The hybrid LU-SGS scheme works slightly better for higher number of processors. The proper node renumbering is critical to efficiency of the method. For parallelization

of the implicit scheme the Peano-Hilbert type renumbering demonstrated the best results.

Even though the method's efficiency degrades with increasing the number of processors, the degradation is proven to be small and the method always maintains its stability since the worst case corresponds to the GMRES scheme with the diagonal preconditioning, which is proven to be stable for the Euler computations.

The method has been successfully applied to several steady-state and time-accurate 3-D simulations. Significant savings in CPU time are achieved as compared to the previous version of the code, which utilized the explicit Runge-Kutta time integration.

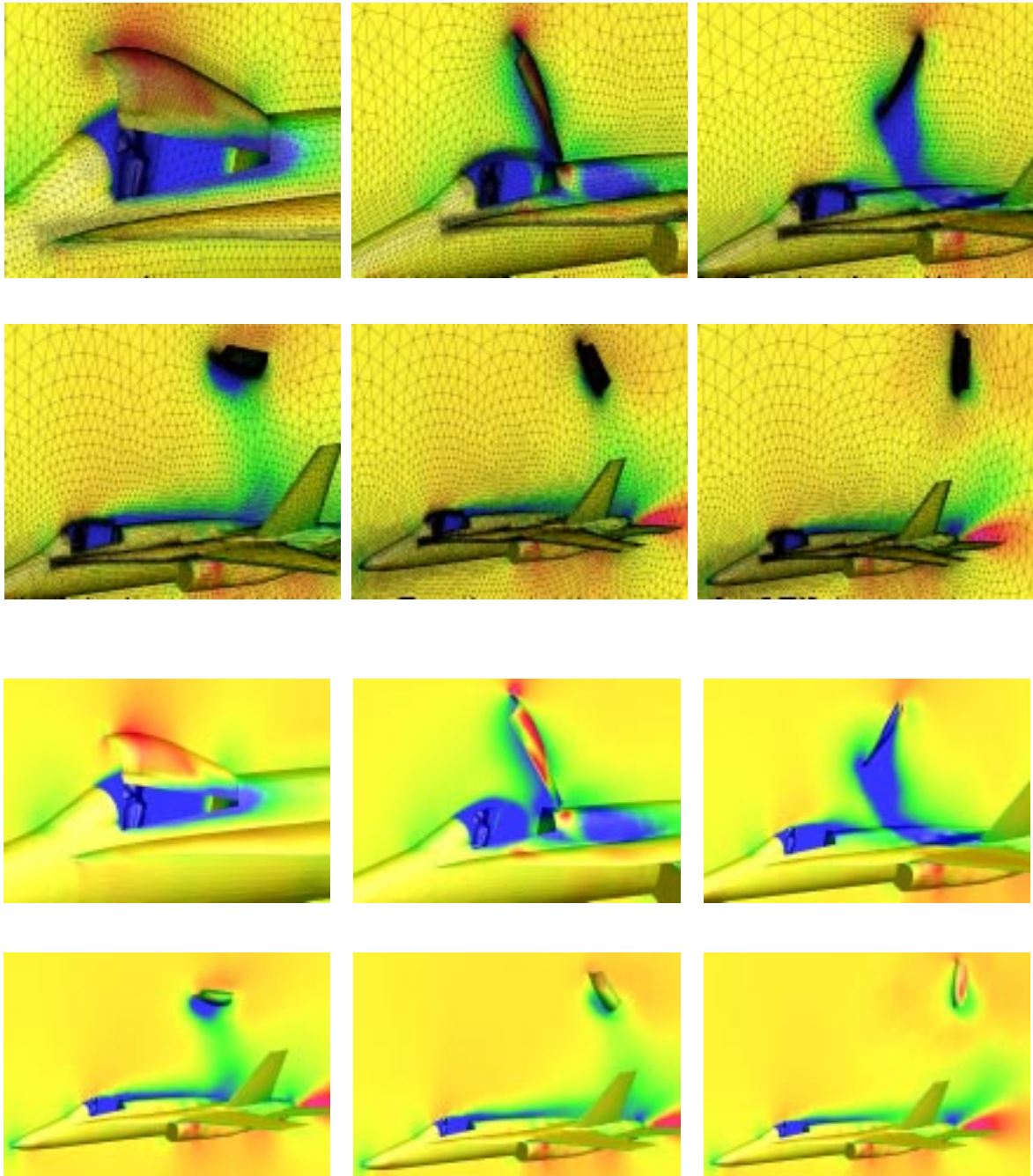


Fig. 16. F/A18-C/D fighter canopy ejection. Surface mesh and absolute velocity contours.

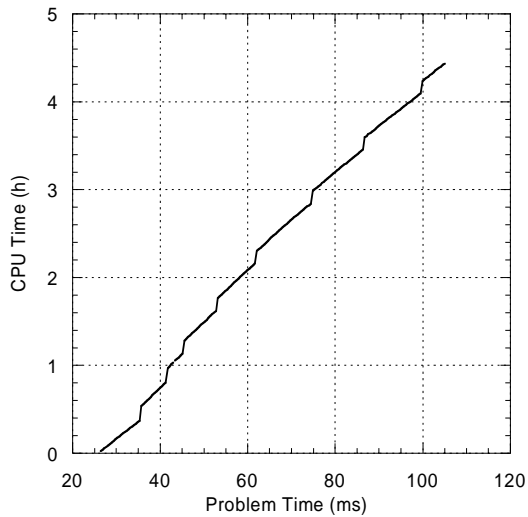


Fig. 17. Canopy ejection. CPU time on 8 processors versus problem time

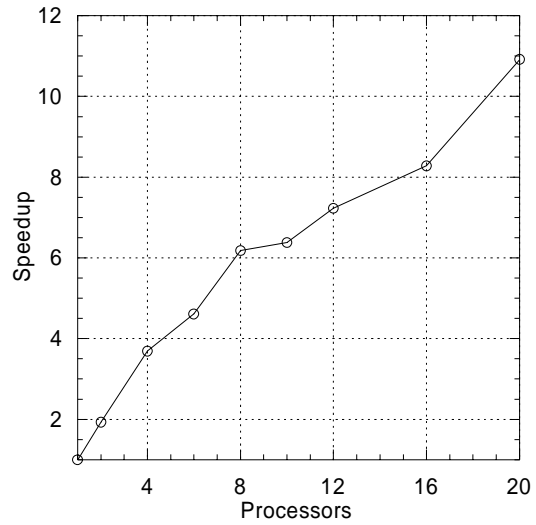


Fig. 18. Speedup for the canopy ejection case.

REFERENCES

1. Luo, H., Baum, J.D., and Löhner, R., A Fast, matrix-free Implicit Method for Compressible Flows on Unstructured Grids, *Journal of Computational Physics*, Vol. 146, pp.664-690,1998.
2. Luo, H., Baum, J.D., and Löhner, R., An Accurate, Fast Matrix-Free Implicit Method for Computing Unsteady Flows on Unstructured Grids, *AIAA 99-0937*, 1999.
3. Saad, Y., and Schultz, M.H., GMRES: a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear systems, *SIAM J. Sci. Stat. Comp.*, Vol. 7, No 3 (1988), pp. 89-105.
4. Jameson, A., and Yoon, S., Lower-Upper Implicit Schemes with Multiple Grids for the Euler equations, *AIAA J.*, Vol. 25, No7, pp.929-935, 1987.
5. Soetrisno, M., Imlay, S.T., and Roberts, D.W., A Zonal Implicit Procedure for Hybrid Structured-Unstructured Grids, *AIAA 94-0645*, 1994
6. Men'shov, I., Nakamura, Y., An Implicit Advection Upwind Splitting Scheme for Hypersonic Air Flows in Thermochemical Nonequilibrium, *6th Int. Symp. on CFD*, pp.815-820, 1995.
7. Sharov, D., Nakahashi, K., Reordering of Hybrid Unstructured Grids for Lower-Upper Symmetric Gauss-Seidel Computations, *AIAA J.*, vol.36, No 3, pp.484-486, 1998.
8. Williams, D., Performance of Dynamic Load Balancing Algorithms for Unstructured Grid Calculations; *CalTech Rep. C3P913* (1990).
9. Simon, H., Partitioning of Unstructured Problems for Parallel Processing; *NASA Ames Tech. Rep. RNR-91-008* (1991).
10. Mehrota, P., Saltz, J., Voigt, R. (eds.), *Unstructured Scientific Computation on Scalable Multiprocessors*; MIT Press (1992).
11. Vidwans, A., Kallinderis, Y, Venkatakrishnan, V., A Parallel Load Balancing Algorithm for 3-D Adaptive Unstructured Grids; *AIAA-93-3313-CP* (1993).

12. Löhner, R, Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing; *Computer Systems in Engineering* 1, 2-4, 257-272 (1990).
13. Löhner, R, and Baum, J.D., Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems; *Int. J. Num. Meth. Fluids*, 14, pp.1407-1419 (1992).
14. Haug, E., Charlier, H., et.al., Recent Trends and Developments of Crashworthiness Simulation Methodologies and their Integration into the Industrial Vehicle Design Cycle; *Proc. Third European Cars/Trucks Simulation Symposium (ASIMUTH)*, Oct.28-30 (1991).
15. Ramamurti, R., and Löhner, R, Simulation of Flow Past Complex Geometries Using a Parallel Implicit Incompressible Flow Solver; pp.1049-1050, *Proc. 11th AIAA CFD Conf.*, Orlando, FL, July (1993).
16. Löhner, R, Renumbering Strategies for Unstructured-Grid Solvers Operating on Shared-Memory, Cache-Based Parallel Machines, AIAA 97-2045, 1997, pp.1015-1025.
17. Liou, M.S, Progress towards an Improved CFD Method: AUSM+, AIAA 95-1701, (1995).
18. Cuthill, E., and McKee, J., Reducing the Bandwidth of Sparse Symmetric Matrices; *Proc. ACM Nat. Conf.*, New York 1969, pp.157-172, (1969).
19. Löhner, R, Some Useful Renumbering Strategies for Unstructured Grids; *Int. J. Num. Meth. Eng.*, 36, pp.3259-3270, (1993).
20. Sagan, H., *Space-Filling Curves*, Springer Verlag, New York, 1994.
21. Candler, G.V., and Wright, M.J., Data-Parallel Lower-Upper Relaxation Method for Reacting Flows, *AIAA Journal*, 32, No12, pp2380-2386, 1994.
22. Povitsky, A., Morris, P.J., Parallel Compact Multi-Dimensional Numerical Algorithm with Application to Aeroacoustics, AIAA 99-3271, (1999).
23. Jenssen, C.B., Implicit Multiblock Euler and Navier-Stokes Calculations, *AIAA Journal*, 32, No 9, pp.1808-1814, 1994.
24. Sheng, C., Hyams, D. et al., Three-Dimensional Incompressible Navier-Stokes Flow Computations About Complete Configurations Using a Multiblock Unstructured Grid Approach, AIAA 99-0778, (1999).
25. Stoll, P., Gerlinger, P., Bruggemann, D., Domain Decomposition for an Implicit LU-SGS Scheme using Overlapping Grids, AIAA-97-1896, pp.479-487, (1997).
26. Wissink, A.W., Lyrantzis, A.S., and Strawn, R.C., Parallelization of a Three-Dimensional Flow Solver for Euler Rotorcraft Aerodynamics Predictions, *AIAA Journal*, 34, No.11, pp.2276-2283, 1996.
27. Wissink, A.W., Lyrantzis, A.S., Chronopoulos, A.T., A Parallel Newton-Krylov Method for Rotorcraft Flowfield Calculations, AIAA-97-22049, pp.1060-1070, 1997.
28. Flower, J., Otto, S., Salama, M., Optimal Mapping of irregular Finite Element Domains to Parallel Processors; pp.239-250 (1990).
29. Venkatakrishnan, V., Simon, H.D., Barth, T.J., A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids; NASA Ames Tech. Rep. RNR-91-024 (1991).
30. Löhner, R, Ramamurti, R., A Load Balancing Algorithm for Unstructured Grids; *Comp. Fluid Dyn.*, 5, pp.39-58 (1995).
31. Schmitt, V., Charpin, F., Pressure Distributions on the ONERA M6 Wing at Transonic Mach Numbers, *Experiment Data Base for Computer Program Assessment*, AGARD AR-138,1979.
32. Ilem, E.R., CFD Wing/Pylon/Finned Store Mutual Interference Wind Tunnel Experiment, AEDC-TSR-91-P4, Arnold Engineering Development Center, Arnold AFB, TN, Jan., 1991.
33. Baum, J.D., Löhner, R, Marquette, T.J., Luo, H., Numerical Simulation of Aircraft Canopy Trajectory, AIAA-97-1885, (1997).
34. Löhner, R., A Parallel Advancing Front Grid Generation Scheme, AIAA-2000-1005, (2000).