INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2005; **48**:671–690
Published online 11 March 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.954

A parallel unstructured dynamic mesh adaptation algorithm for 3-D unsteady flows

Young Min Park and Oh Joon Kwon[*,†]

Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea

SUMMARY

An unstructured dynamic mesh adaptation and load balancing algorithm has been developed for the efficient simulation of three-dimensional unsteady inviscid flows on parallel machines. The numerical scheme was based on a cell-centred finite-volume method and the Roe's flux-difference splitting. Second-order accuracy was achieved in time by using an implicit Jacobi/Gauss–Seidel iteration. The resolution of time-dependent solutions was enhanced by adopting an h-refinement/coarsening algorithm. Parallelization and load balancing were concurrently achieved on the adaptive dynamic meshes for computational speed-up and efficient memory redistribution. A new tree data structure for boundary faces was developed for the continuous transfer of the communication data across the parallel subdomain boundary. The parallel efficiency was validated by applying the present method to an unsteady shock-tube problem. The flows around oscillating NACA0012 wing and F-5 wing were also calculated for the numerical verification of the present dynamic mesh adaptation and load balancing algorithm. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: unstructured mesh; dynamic mesh adaptation; parallelization; load balancing; unsteady flow

1. INTRODUCTION

One of the advantages of using unstructured meshes over structured grids is in improving the spatial accuracy of the numerical solution by applying the flexible mesh adaptation capability to the local flow region of interest. This advantage is very attractive also for solving transient flow problems, such as capturing moving shock wave, contact surface, or travelling vortex. During the past decade, several adaptation strategies have been developed [1–12]. Among the various adaptation algorithms, h-refinement/coarsening is regarded as the best method for

*Correspondence to: Oh Joon Kwon, Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea.
†E-mail: ojkwon@kaist.ac.kr

Received 9 January 2004
Revised 24 January 2005
Accepted 7 February 2005

transient problems [2, 4]. In this method, new nodes are added inside the flow region where the solution error is relatively large. This operation is applied only to the local region, while the rest of the computational domain maintains the same global mesh topology. Therefore, this adaptation procedure can be performed more effectively than the mesh movement [11] or the local re-meshing procedure [12].

For unsteady flows, the flow solver and the unstructured dynamic mesh adaptation procedure should be coupled together for the efficient capturing of the continuously varying flow physics. However, this requires very large computational resources, particularly for solving three-dimensional flow problems, which cannot be handled properly even on current large-scale vector machines. Recently, owing to the rapid development of computer hardware technologies, cost-effective parallel machines are easily constructed and can be used to replace the traditional vector machines. At the same time, several studies for implementing dynamic mesh adaptation algorithms under parallel computing environment have been conducted based on various division patterns and data structures [13–16].

A refinement method based on non-predefined subdivision patterns [13] has been previously proposed. In this method, cell division types at the communication boundary are determined first, and then the subdivision of internal cells is made as compatible to the division types of the communication boundary. This method effectively eliminates the cell over-refinement problem, and since iterative communication is not necessary to match the interface boundary, the computational overhead for excessive communication can be significantly reduced. However, this method is susceptible to frequently generating highly skewed cells during the cell refinement and coarsening process. To remedy this problem, cell optimization techniques, such as edge removal, face swapping, and edge swapping, are often adopted after the refinement and coarsening. An optimized edge-collapsing is also used to preserve the mesh quality during cell coarsening. The method was applied to a steady flow around rotor blades and to the unsteady flow within a muzzle brake.

A parallel adaptation and load balancing algorithm based on predefined subdivision patterns was developed using C^{++} and an edge data structure [14, 15]. The developed algorithm was applied to simulating steady flows around helicopter rotor blades in hover over a range of subsonic and transonic tip Mach numbers, and the parallel performance of the dynamic mesh adaptation was measured. A completely parallel algorithm for dynamic auto-adaptive grids [16] was also proposed, which performs the data management, domain decomposition, and data redistribution directly on the network. The parallel performance between the master–slave concept and the slave–slave concept was analysed by applying the method to steady and unsteady flow calculations.

Even though the parallel adaptation algorithms based on predefined subdivision patterns [14–16] were quite successful, practical applications were limited to three-dimensional steady flows and slowly evolving two-dimensional unsteady flows, which do not require frequent data communication. To transfer the information about refined edges on the local subdomain boundary, these algorithms adopted the edge-to-edge communication, which works well for node-based methods. However, using this edge-to-edge communication, the search process for finding the matching edges between adjacent processors requires repeated re-numbering/re-ordering of the boundary edges at every application of the mesh refinement and coarsening. Since the number of new edges generated on the boundary faces increases very quickly, this algorithm is very time-consuming, particularly in the case of rapidly evolving three-dimensional unsteady flows accompanying frequent mesh adaptation.

Since non-hierarchical data structures do not retain the mesh enrichment history [13–16], the mesh quality becomes severely degraded as the level of mesh adaptation and coarsening increases. This difficulty can be relieved by adopting additional mesh optimization procedures, at the expense of the increasing computational overhead. On the contrary, hierarchical data structures [7, 8, 10] require extra memory for retaining the history of mesh enrichment and coarsening. However, refined subcells maintain the mesh quality similar to their parent cells. Also, mesh coarsening can be efficiently performed by using the stored background mesh data.

In the present study, an unstructured dynamic mesh adaptation and parallel computing algorithm has been developed for the simulation of three-dimensional unsteady inviscid flows. For this purpose, a new hierarchical face data structure was developed, which enables multi-level mesh adaptation and efficient boundary communication by readily providing the inter-domain information. The information about cells, faces, and nodes can also be extracted easily by using the present face data structure. Predetermined 1:2, 1:4, and 1:8 division types for tetrahedral cells and temporary buffer cells [1, 4] were adopted for mesh refinement/coarsening and mesh quality preservation. Dynamic load balancing was achieved by merging and repartitioning the refined mesh after every application of the adaptation procedure. The parallel mesh adaptation algorithm was demonstrated for a steady flow around an ONERA M6 wing. The unsteady parallel dynamic mesh adaptation and load balancing algorithm was validated for a shock-tube problem, oscillating NACA0012 wing, and F-5 wing.

2. NUMERICAL METHOD

2.1. Flow solver

A three-dimensional unsteady Euler flow solver was developed based on a cell-based finite-volume method. The inviscid flux across each cell face was computed by using the Roe's flux-difference splitting [17]. To obtain high-order spatial accuracy, estimation of the state variables at each cell face was achieved by interpolating the solution with a Taylor series expansion in the neighbourhood of each cell centre. The cell-averaged solution gradient required at the cell centre for the above expansion is computed using the Gauss' theorem by evaluating the surface integral for the closed surface of each tetrahedron. This process can be simplified by using some geometrical invariant features of the tetrahedra [18]. The expansion also requires the nodal value of the solution, which can be computed from the surrounding cell centre data using a second-order accurate pseudo-Laplacian averaging procedure [19].

The discretized governing equations are integrated in time by using the second-order accurate Euler backward-differencing coupled with an implicit Jacobi/Gauss–Seidel relaxation method and dual time stepping [20].

2.2. Parallel implementation of the flow solver

The global computational domain is partitioned into several local subdomains by using the MeTiS library [21], and the subdomain mesh data is allocated to each processor. The flow calculation is made in each computational subdomain by exchanging the solution information across the subdomain boundary. For the present cell-centred scheme, ghost cells are attached to the artificial boundary between adjacent subdomains for the convenience of data communication. Initially, face-centre values of the flow variables are exchanged across the boundary

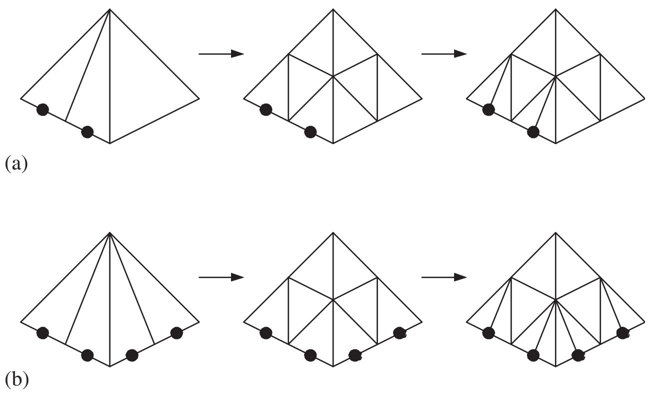

Figure 1. Enrichment procedure of transient cells: (a) Marked type-2 element; and (b) marked type-4 element.

for the evaluation of the flux Jacobian at the subdomain boundary. Next, cell-centre values are exchanged to perform Gauss–Seidel iterations. Boundary node values and the weighting factors for Laplacian averaging are also communicated for the high-order reconstruction. In the present study, cell data are exchanged at every sub-iteration, and face and node data are transferred for every outer iteration.

2.3. *Mesh refinement algorithm*

A solution-adaptive dynamic mesh algorithm was developed for the efficient capturing of time-varying high-gradient flow characteristics. Tagged cells for subdivision under given criteria are divided into eight subcells. To preserve the mesh quality, a transient cell algorithm [1, 4] is applied to cells having either 1:2 or 1:4 division types surrounding the regular 1:8 division cells. These temporary cells are removed and replaced by eight subdivision cells, and then an additional 1:2 or 1:4 division is applied to the subcells containing the edge marked for division. This ensures that the aspect ratio of the divided subcells is not excessively high compared to their parent cell. Figure 1 shows typical cases of the transient cell refinement. Initially, transient cell edges to be divided are identified and marked such that the cell connectivity to surrounding cells is satisfied. If both of the two bottom edges are marked as for the 1:2 division type, the parent cell is divided into eight subcells, and then the subcells with marked sub-edges are divided by an additional level. A similar procedure can also be applied to 1:4 division-type cells.

2.4. *Data structure for communication boundary faces*

Efficient data transfer between computational subdomains through communication and boundary conformation is very complex when the mesh adaptation algorithm is parallelized. To achieve this, edge data structures are usually adopted for node-based schemes [13, 14]. However, for the present cell-based scheme, face-to-face, node-to-node, and cell-to-cell transfer of the flow variables is required. To satisfy this requirement, a new face data structure

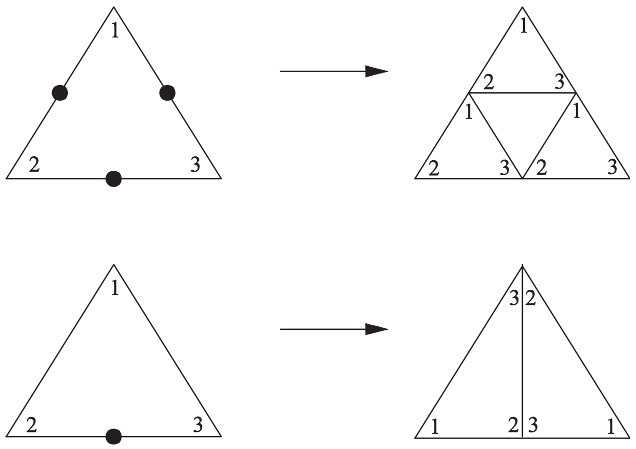

Figure 2. Local node numbering of child faces after division.

is developed, which enables efficient data communication and multi-level mesh adaptation simultaneously without additional post-processing.

This face data structure is constructed in a way that the hierarchy of the parent and child faces at the communication boundary is retained. The information about the number of child faces, the attached adjacent processor, and the allocation address is also kept in the main data structure. The generation and deletion of the boundary faces during mesh adaptation are simultaneously updated in the data structure. By using this face data structure, face-to-face information between processors required by the flow solver can be efficiently extracted.

This algorithm can be implemented more effectively by using some geometric patterns of the communication boundary faces. Since the normal vector on the boundary face is directed inward to the subdomain, each subdomain boundary face possesses two opposite directions. The direction normal to the original cell face defined for the global mesh during pre-processing is designated as positive. At the same time, to identify matching edges automatically and to achieve efficient edge-to-edge communication, local nodes attached to the two corresponding boundary faces are re-numbered such that the faces share a common first node.

The node numbering for child faces is made in a way that each face contains symmetry with respect to the first node as shown in Figure 2. By doing this, the edges of the corresponding two child boundary faces between subdomains match automatically when the child faces detect the common first node, as done by their parent face.

Figure 3 shows an example of the data communication between adjacent processors. Initially, bit flags are set for the tagged edge on the face of the left processor. Then, the bit flags are converted into a decimal value, and this decimal value is transferred to the adjacent processor. On receiving, the decimal value is transformed to a new number by considering the node numbering of the face of the receiving processor. In the figure, the left processor sends the information about the division type of ‘1’ to the right processor, and then the value is transformed to ‘4’ and received by the right processor. This decimal value of ‘4’ is converted

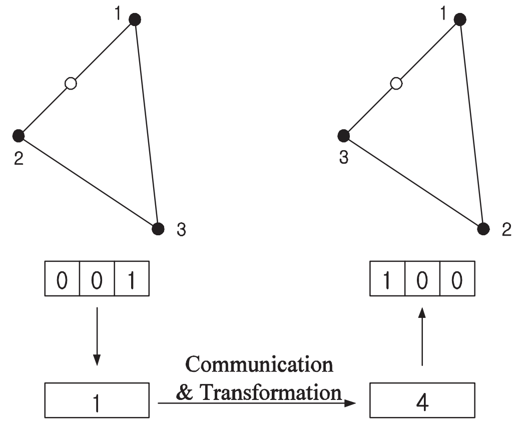

Figure 3. Boundary data communication between adjacent processors.

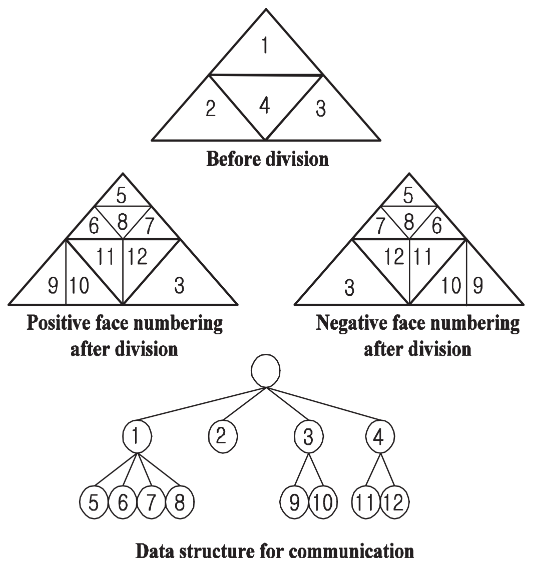
Before division
Positive face numbering after division
Negative face numbering after division
Data structure for communication

Figure 4. Tree structure for data storage at the communication boundary.

back to bit flags, which inform the right processor of the third edge refinement. This process is repeated until no further communication can be found between boundary faces.

Figure 4 shows the tree structure developed for the data storage of the child faces at the communication boundary. In this figure, four boundary faces are presented before and after the typical 1:4 or 1:2 divisions. The two identical boundary faces between adjacent subdomains are shown as the mirror image of each other after the division. The face with a positive rotational direction stores the child faces in the order of 1-2-3-4, while the opposite face stores them as 1-2-4-3. In the case of the face with the 1:2 division type, the face with a positive rotational direction stores the left child face first, while the face with a negative rotational direction starts from the right child face. This method ensures that the matching faces of the two adjacent subdomains maintain the same data structure at the communication boundary, even after the multi-level mesh adaptation. The information about the refined child faces should also be updated in the main data structure so that each child face identifies its global-to-local and local-to-global conversion addresses. By using this data structure, the communication face data can be readily extracted without additional computational overhead for searching and post-processing.

The information about matching nodes can also be obtained easily by using a similar communication process between adjacent processors. However, common nodes shared by multiple processors must be considered separately, since that information is not stored in the communication face data. These nodes can be identified by searching edges shared by more than three processors at pre-processing. When these edges are refined, the information about the added nodes should be distributed to all processors sharing these edges.

During the parallel adaptation process, the information about the error-marking procedure, the propagation of subdivision patterns, and the extraction of the data for matching faces and nodes should be communicated between adjacent processors as necessary.

When hybrid meshes containing pyramidal or prismatic cells are used, an additional numbering sequence of nodes and refined child faces for quadrilateral faces, consistent to that for triangular faces, needs to be considered.

2.5. Main data structure

For the present cell-based solver, the main data structure contains the information that links each tetrahedral cell to its faces, edges, and nodes [6, 10]. The connectivity information that links each triangular face to its edges, nodes, and the cell is also stored in the data structure, along with the edge-to-node linkage.

A 2^n tree data structure is used to store the information about each cell. Each object divides into 2^n sub-objects under their parent object, where 'n' denotes either 1, 2, or 3 depending on the cell division type of 1:2, 1:4, or 1:8, respectively. New edges generated additionally during face or cell refinement should also be included in the tree data structure. Since the faces divided at the communication boundary always remain in the plane of their own parent face, the boundary face data can also be handled by using the 2^n tree data structure. This tree data structure is updated simultaneously with the main data structure during refinement or coarsening, and thus additional post-processing or re-ordering is not needed after the mesh adaptation process.

This hierarchical data structure for handling the mesh adaptation and the boundary communication is stored in local processors. The dynamic mesh adaptation procedure and the main data structure require approximately 500 Mbytes (125 Mwords) of memory/1 million cells. Additional memory of approximately 3% is also used for the boundary face data structure. For the present implicit time integration scheme, the flow variables are temporarily stored in each processor, because the required memory for storing the flow variables is approximately 3 times more than that of the mesh. As a result, each processor having 256 Mbytes of memory can handle up to 100 000 cells approximately.

After every application of the dynamic mesh adaptation, the host processor collects the local mesh and solution data. Then the updated global mesh and solution data are repartitioned for load balancing by using the MeTiS library [21], and the host processor redistributes the updated information to each processor. The elapsed time for data transfer and repartition of this load balancing procedure is negligible compared to the total computational time.

3. RESULTS AND DISCUSSION

3.1. Parallel performance of the flow solver

A steady flow around an ONERA M6 wing at a transonic fight speed was calculated to evaluate the parallel performance of the flow solver. The initial mesh consisted of 53 460 tetrahedral cells, 10 911 nodes, and 5541 surface triangles. The free stream Mach number was 0.84, and the angle of attack was set to 3.06°. The calculation was made on a PC-cluster composed of AMD 900 MHz processors connected with 100 Mbps fast Ethernet network cards. To reach a steady-state solution, approximately 300 iterations were required using local time stepping.

Iterative schemes, such as the Gauss–Seidel method adopted for the present study, typically perform sub-iterations to achieve satisfactory convergence of the solution. Under the parallel environment, it is also important to guarantee the unified convergence property regardless of the number of processors used, especially for time-accurate unsteady calculations. In Figure 5, it is shown that this dependency of the solution convergence on the number of processors can be eliminated by allowing proper data communication between subdomains. However, performing the full data communication at every sub-iteration requires large computational overhead for three-dimensional unsteady flow simulations. Figure 6 shows the effect of the number of sub-iteration communications on the solution convergence. It is observed that at least three communications are required to achieve a proper convergence of the solution, which was used for all calculations presented in the present paper.

Next, the shock-capturing capability of the present mesh adaptation procedure was tested. The total elapsed time for the calculation using 16 processors was approximately 1400 CPUs,

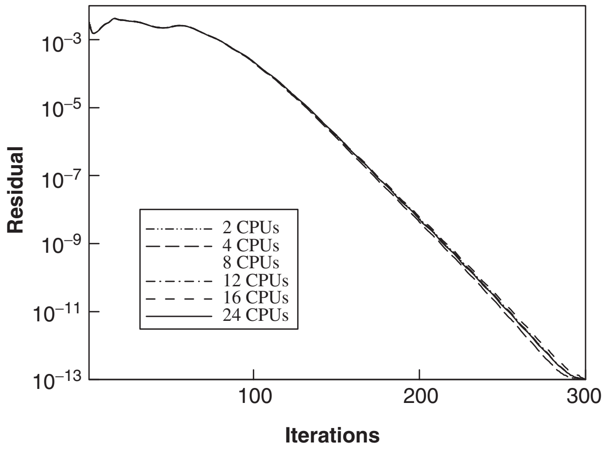

Figure 5. Effect of number of processors on the solution convergence.

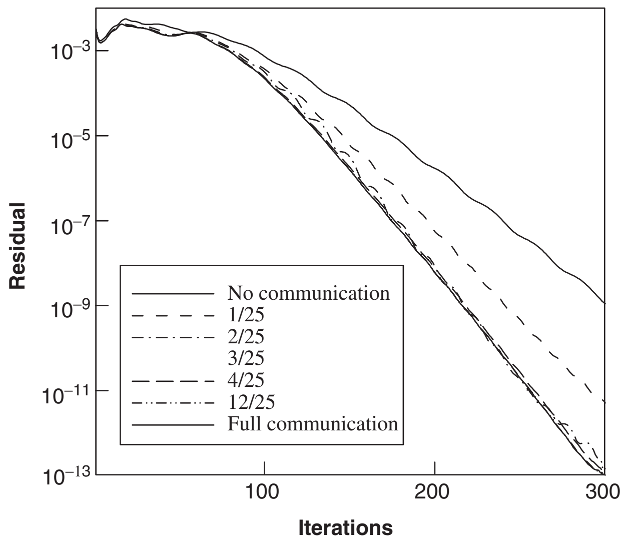

Figure 6. Effect of number of sub-iteration communications on the solution convergence.

including 40 s for the mesh adaptation. The final mesh after two-level mesh enrichment resulted in 88 290 nodes and 419 184 cells. Refinement and coarsening of the cells were made after 200 and 400 iterations by examining the magnitude of the density gradient at cell edges. Figure 7 shows the surface triangulation and the density contours before and after the mesh adaptation. The partitioned subdomain boundary is represented as bold lines. After the cell refinement, the local subdomains were redistributed at the wing leading edge and along the shock wave as a result of the load balancing between processors. It is also shown that the λ-shaped shock-wave pattern is defined better on the refined mesh.

The parallel efficiency of the flow solver obtained from full and partial communications is presented in Figure 8 for both initial and refined meshes. It shows that the performance of the coarse initial mesh became severely lower than that of the fine mesh as the number of processors increased. This is because the CPU time for communication was relatively larger than the pure solution time for small mesh size. A significantly improved parallel efficiency was obtained by using the fine mesh. The figure also shows that much higher efficiency can be obtained at less computational time by performing the partial communication than the full communication during sub-iteration. Owing to the cache RAM efficiency of the PC-cluster, the performance higher than the ideal speed-up was also observed when an optimum number of cells was allocated to processors.

3.2. Unsteady shock-tube simulation

The accuracy of the dynamic mesh adaptation and load balancing procedure was evaluated by simulating an unsteady shock-tube problem that contains several time-dependent flow features, such as the moving shock wave, expansion fan, and contact surface. The propagation of the communication boundary resulting from the mesh adaptation and the subsequent dynamic load

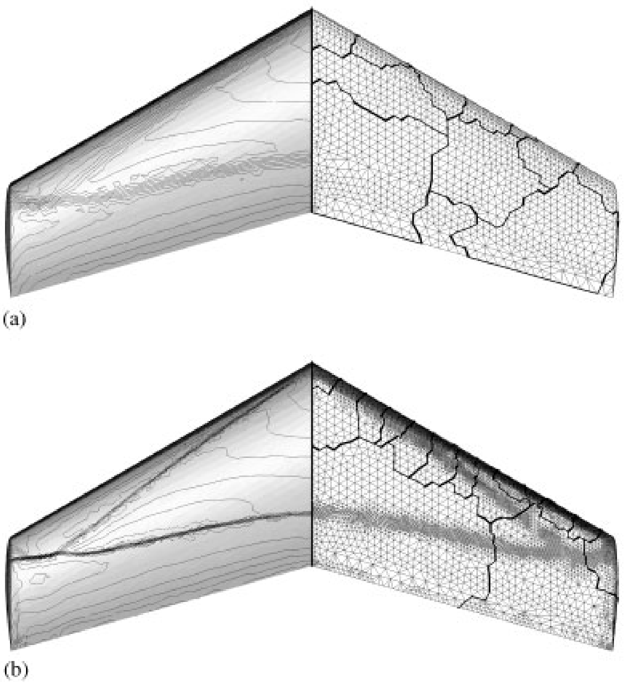

Figure 7. Surface triangulation and upper surface density contours for the ONERA M6 wing at $M_{\infty} = 0.84$ and $\alpha = 3.06^{\circ}$: (a) Initial mesh; and (b) after two-level adaptation.

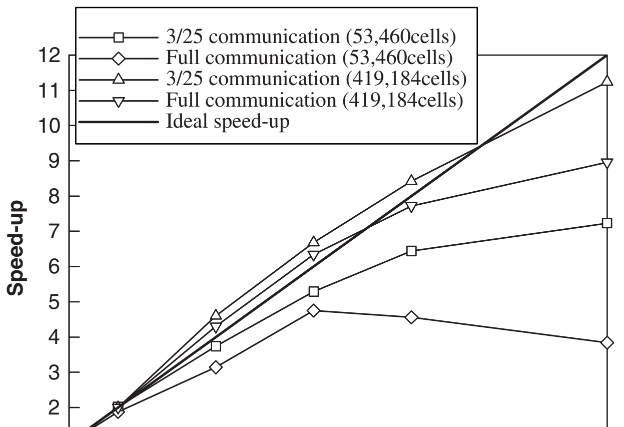

Figure 8. Parallel efficiency of the solver for the steady ONERA M6 wing calculation.

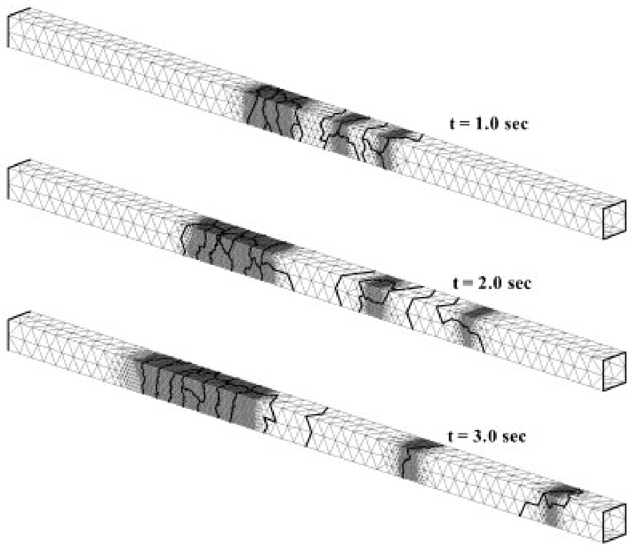

Figure 9. Surface mesh and communication boundary for the shock-tube problem at three time levels (16 domains).

balancing could also be checked from this test problem. Initially, the velocity was set to zero on both sides of the tube, and the density and pressure difference across the discontinuity was set to five. The mesh had 1732 cells and 554 nodes in the beginning, and its size increased to 224 841 cells and 42 989 nodes at the end of the calculation with three levels of mesh adaptation. The time-accurate calculation was made at a non-dimensional time step of 0.001 using 16 processors. The mesh was refined at every 40 time steps based on the density gradient.

In Figure 9, the surface mesh distributions are presented at three subsequent time levels after the breakdown of the discontinuity. It is shown that the position of the subdomain boundary represented as bold lines changed in time to cope with the varying local cell density and to achieve the dynamic load balancing between processors. The present dynamic mesh adaptation procedure is well demonstrated in the figure by showing cells added and deleted as necessary along the moving discontinuities. The precise marking procedure, the propagation of division types, and the proper data transfer between matching boundary faces under the time-varying flow environment are also well demonstrated. The calculated temporal and spatial behaviours of the density profiles are in good agreement with the exact solution within the accuracy of the present numerical method as shown in Figure 10.

Figure 11 shows the variation of the number of cells allocated to each processor as a function of the computational time. It is observed that each subdomain had highly unbalanced local meshes when the dynamic load balancing was not applied, which severely degraded the efficiency of the parallel computation. On the other hand, when the dynamic load balancing was applied, the number of cells was distributed equally to each processor and the high parallel efficiency was obtained. The computational time without applying the dynamic load balancing was approximately 3 times more than that with the load balancing.

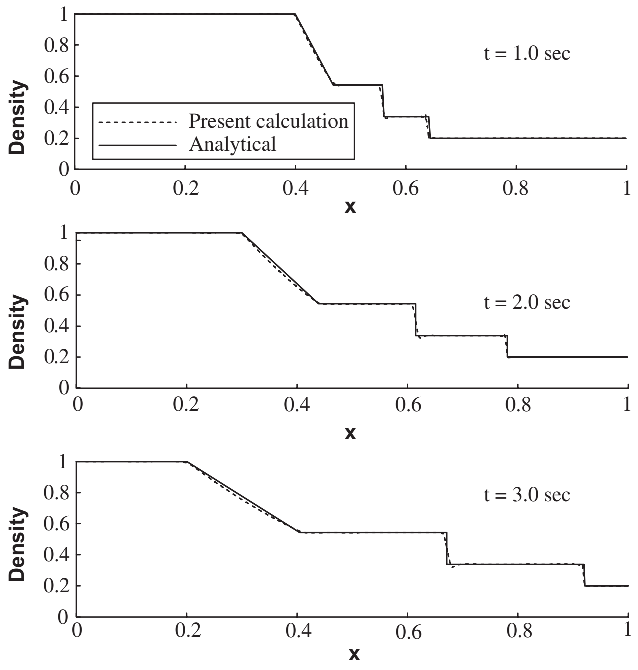

Figure 10. Comparison of calculated and exact density profiles along the shock-tube problem.

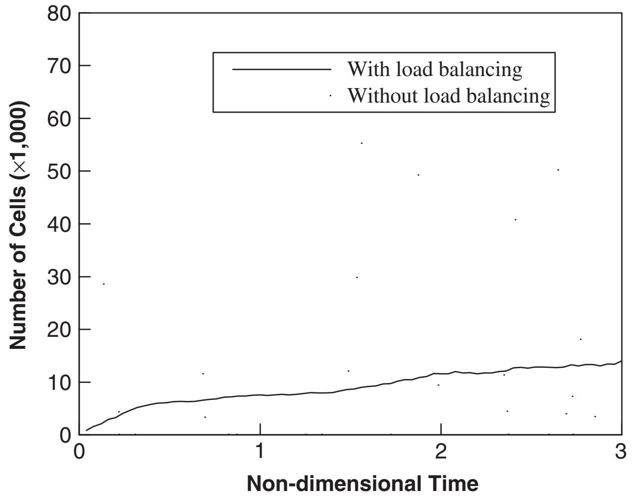

Figure 11. Cell number history of each processor with and without load balancing for the shock-tube problem.

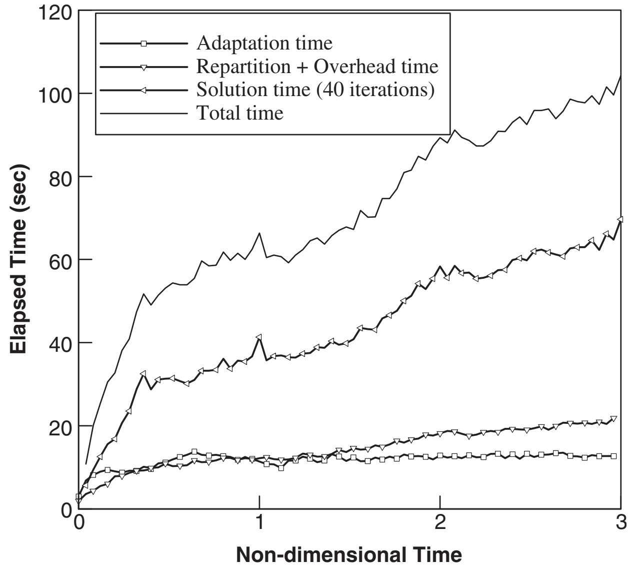

Figure 12. Elapsed execution time spent for 40 iterations during the shock-tube simulation.

Figure 12 shows the comparison of the elapsed execution time for performing 40 iterations between each mesh adaptation. The rapid increase of the total CPU time at the initial stage of the calculation was mostly for refining the mesh along the expansion fan, contact surface, and shock wave. At the subsequent iterations, the CPU time increased gradually because the number of cells increased consistently to refine the expanding expansion fan region. The CPU time spent for mesh adaptation was similar to that for repartitioning, and both took approximately 30–40% of the total calculation time.

The time-averaged parallel efficiency of the present shock-tube simulation is presented in Figure 13. The resultant performance based on the total elapsed time was relatively low because of the overhead time for mesh adaptation, data communication, and mesh repartitioning. However, the solution time alone showed a good performance. It is expected that the parallel performance would improve further as the calculation proceeds, since the net solution time increases faster than that for the mesh adaptation and repartitioning as shown in Figure 12. The performance can also be enhanced by reducing the number of mesh adaptation at the expense of the solution accuracy.

3.3. Oscillating NACA0012 wing

The next validation was made for an unsteady flow around a rectangular NACA0012 wing, harmonically oscillating at 0.016° mean angle of attack and the amplitude of oscillation of 2.52°. The free stream Mach number was 0.755, and the reduced frequency of oscillation was 0.0814. The wing aspect ratio was set to 0.5, and a symmetric boundary condition was imposed on both sides of the wing tip so that the results can be compared with the two-dimensional experimental data [22]. The calculation was made using 16 processors.

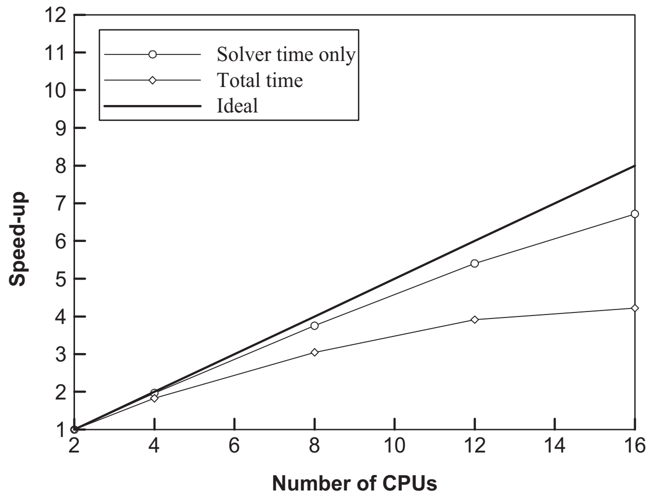

Figure 13. Parallel efficiency of the unsteady shock-tube problem.

Initially, a steady-state solution was obtained at the mean angle of attack on a coarse mesh having 26 632 cells and 5861 nodes. Then, the unsteady calculation was made using a non-dimensional time step of 0.015, which was equivalent to proceeding approximately 3400 time steps to complete one cycle of the motion. Two levels of the dynamic mesh adaptation were adopted to capture high-gradient flow regions near the leading edge and along the moving shock wave. The mesh refinement and coarsening were applied at every 50 time steps based on the magnitude of the density gradient.

Figure 14 shows the instantaneous surface meshes at four typical angles of attack during the cycle. The total number of cells periodically varied approximately from 340 000 to 380 000, depending on the shock wave formation and its strength. Due to the varying local mesh density and as a result of the application of the dynamic load balancing, the position of the subdomain boundary changed continuously in time as observed with the bold lines in the figure.

The history of the number of cells in each processor is presented for two periods of oscillation in Figure 15. The maximum number of cells allocated to the processors without applying the dynamic load balancing was 80 324 after two levels of mesh adaptation. This number was approximately 50 times more than the minimum number. The subdomain located near the leading edge maintained the maximum cell number without large deviation in time. On the other hand, the number of cells in the subdomain located at the mid code showed large abrupt changes due to the migration of the shock wave. Once the dynamic load balancing was applied, every processor shared an approximately equal number of cells. The total elapsed time to complete one period of oscillation was 15 822 s, including 5344 s for mesh adaptation and repartitioning. The total elapsed time without applying the dynamic load balancing increased to 37 159 s, which was approximately 2.4 times more than that with the dynamic load balancing.

Figure 16 shows the comparison of the elapsed execution time for 50 iterations performed between mesh adaptations. The execution time for the solution iteration and the mesh repartitioning showed a periodic behaviour due to the varying total number of cells. The CPU time

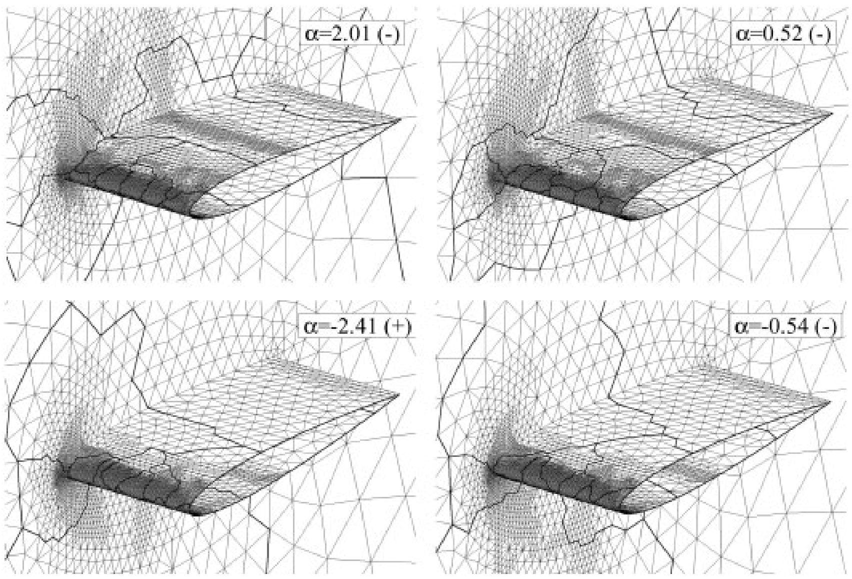

Figure 14. Surface meshes at four instantaneous angles of attack for the oscillating NACA0012 wing (+: pitch-up).

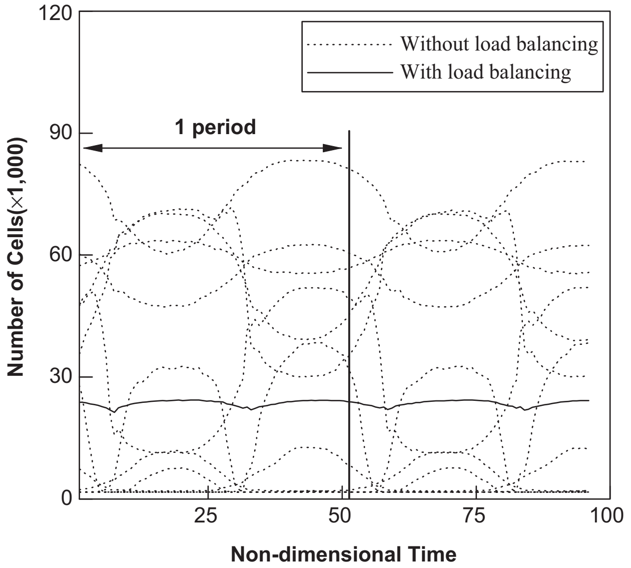

Figure 15. History of the number of cells of each processor for two cycles of oscillation.

required for the dynamic mesh adaptation was almost unchanged throughout the calculation. Approximately one-third of the total execution time was used to perform the mesh adaptation and the repartitioning for the present two-level mesh adaptation.

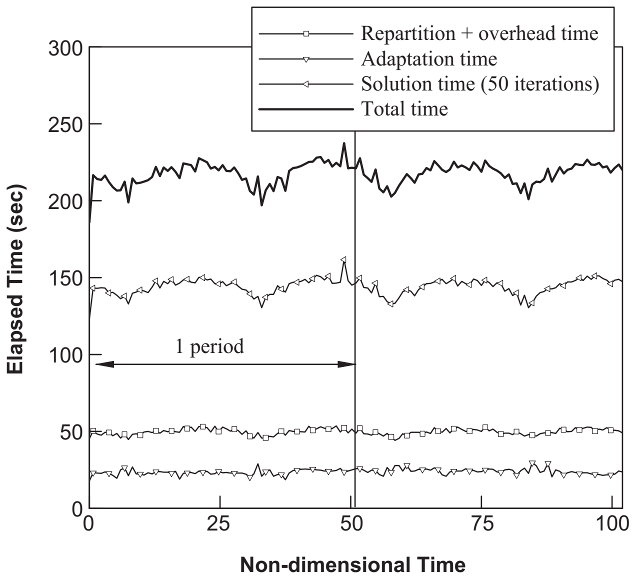

Figure 16. Elapsed execution time for 50 iterations for the oscillating NACA0012 wing simulation.

The pressure distributions on the wing surface at four instantaneous angles of attack are shown in Figure 17. Good comparison was observed between the present results and the experimental data [22], demonstrating the validity of the present unsteady calculation. The effect of the mesh adaptation for shock capturing is also well confirmed in the figure.

3.4. *Oscillating F-5 wing*

The unsteady flow around an F-5 wing at a free stream Mach number of 0.95 was calculated using 24 processors. The wing was oscillating at a reduced frequency of 0.132 with an amplitude of 0.532°. The initial calculation was made on a coarse mesh having 22 434 nodes and 116 595 tetrahedral cells. The final mesh after two levels of mesh adaptation contained 80 673 nodes and 399 935 cells. The total elapsed time to obtain a steady-state solution at the mean angle of attack was approximately 1030 s, including 26 s of overhead time for mesh adaptation. The surface mesh distribution and the density contours at the steady state are shown in Figure 18.

The unsteady calculation triggered from the steady-state solution took approximately 2.1 h of CPU time/cycle with the two-level dynamic mesh adaptation. The normalized time step size used for the present unsteady calculation was 0.015, and approximately 1670 time steps were needed to complete one cycle of the motion. The mesh refinement and coarsening were applied at every 50 time steps based on the magnitude of the density gradient. The maximum number of cells reached up to 600 000 approximately.

The real and imaginary components of the unsteady surface pressure coefficient were obtained by applying the Fourier series analysis, and the results at four spanwise sections of the wing are compared with the experiment [23] in Figure 19. It is shown that the imaginary

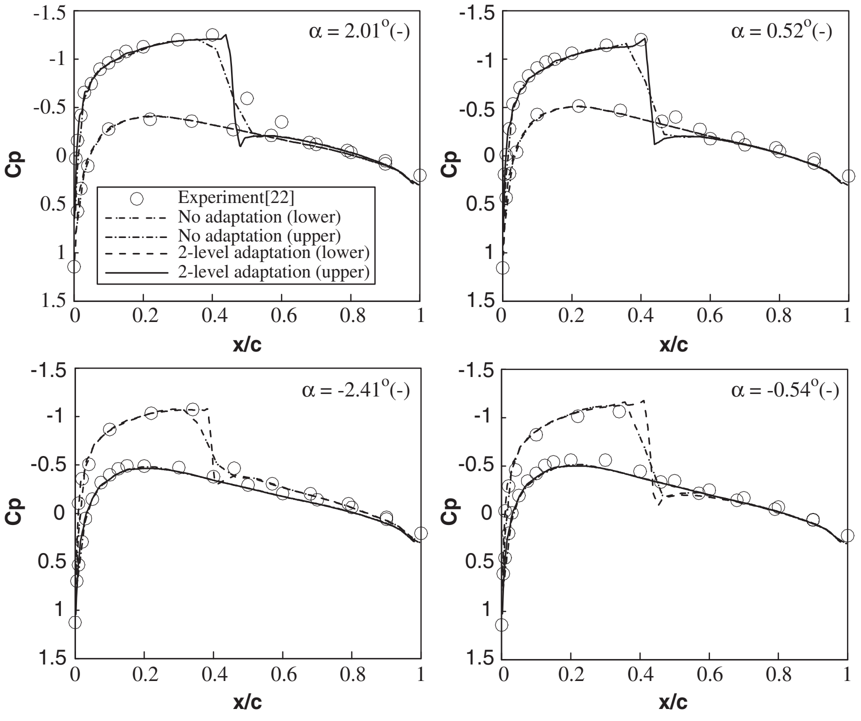

Figure 17. Instantaneous pressure distributions on the NACA0012 wing surface.

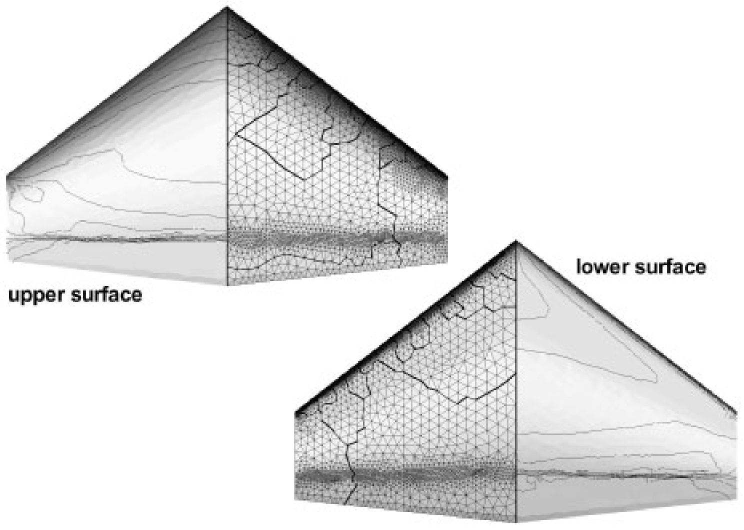

Figure 18. Surface mesh and density contours for the F-5 wing.

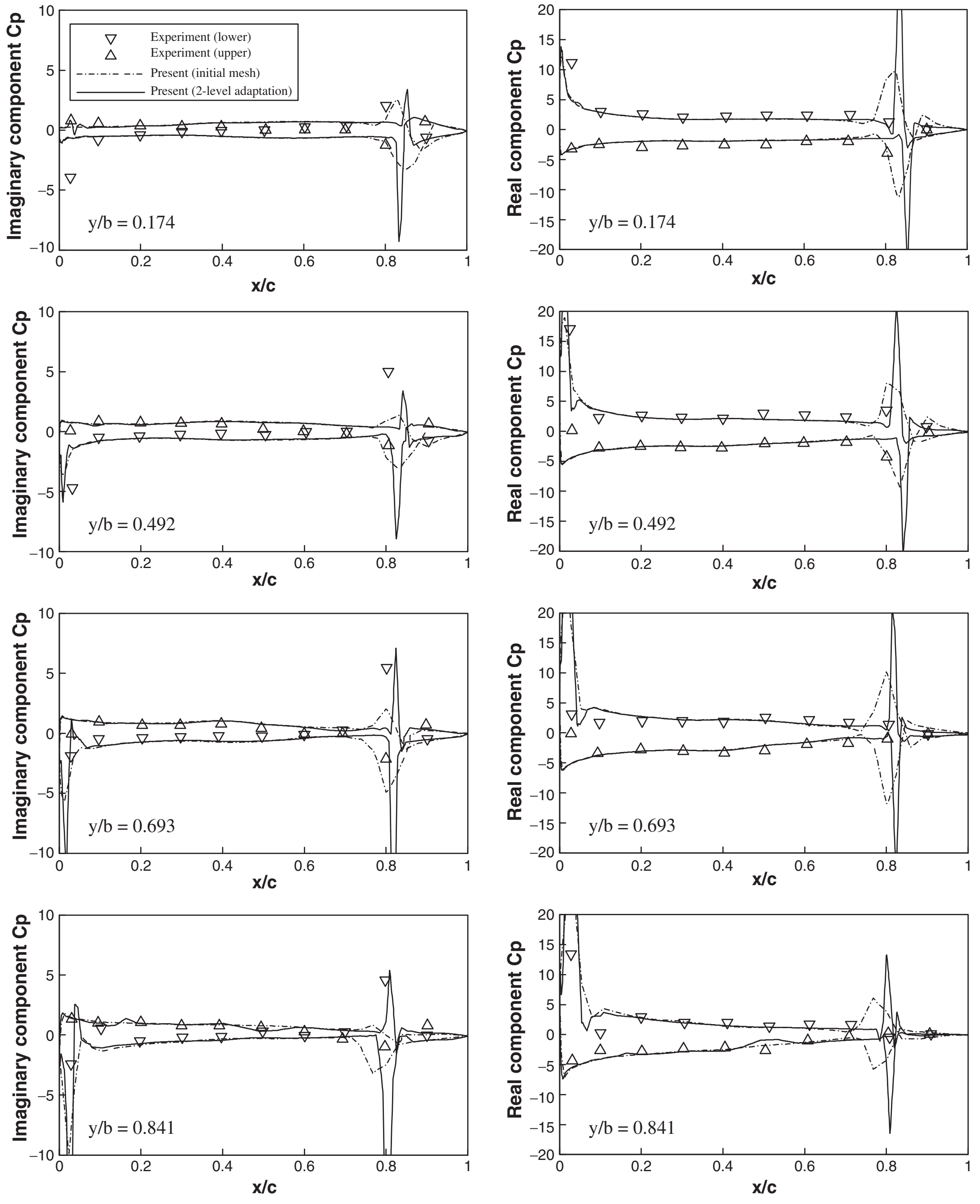

Figure 19. Imaginary and real components of the pressure coefficient for the F-5 wing under a harmonic oscillation.

component of the pressure was dominant for the present high-frequency motion. The oscillatory movement of the shock wave appeared between 80 and 90% of the chord as a sharp spike. The pressure peaks near the leading edge and at the moving shock wave were well captured, demonstrating the validity and the accuracy of the present dynamic mesh adaptation procedure.

4. CONCLUSION

An unstructured dynamic mesh adaptation and load balancing algorithm has been developed for the efficient simulation of three-dimensional unsteady inviscid flows on parallel machines. The flow solver was based on the Roe's flux-difference splitting and the Gauss–Seidel implicit time integration. A new tree data structure was developed for the efficient treatment of the multi-level data transfer between processors caused by the repeated application of the dynamic mesh adaptation. The parallel performance and the effect of the mesh adaptation were tested for a steady transonic flow around an ONERA M6 wing. The unsteady dynamic mesh adaptation and the load balancing procedure was validated by calculating a shock-tube problem and the flows around oscillating NACA0012 wing and F-5 wing. It was shown that the present method is efficient and accurate for solving three-dimensional unsteady flows under parallel environment.

REFERENCES

1. Lohner R. An adaptive finite element scheme for transient problems in CFD. *Computer Methods in Applied Mechanics and Engineering* 1987; **61**:323–338.
2. Lohner R, Baum JD. Adaptive H-refinement on 3-D unstructured grids for transient problems. *International Journal for Numerical Methods in Fluids* 1992; **14**:1407–1419.
3. Rivara MC. Selective refinement/derefinement algorithms for sequences of nested triangulations. *International Journal for Numerical Methods in Engineering* 1989; **28**:2889–2906.
4. Batina JT. Unsteady Euler airfoil solutions using unstructured dynamic meshes. *AIAA Paper 89-0115*, 1989.
5. Rausch RD, Batina JT. Three-dimensional time-marching aeroelastic analysis using an unstructured-grid Euler method. *AIAA Journal* 1993; **31**(9):1626–1633.
6. Kallinderis Y, Vijayan P. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA Journal* 1993; **31**(8):1440–1447.
7. Biswas R, Strawn R. A new procedure for dynamic adaptation of three-dimensional unstructured grids. *AIAA Paper 93-0672*, 1993.
8. Biswas R, Strawn R. A dynamic mesh adaptation procedure for unstructured hexahedral grids. *AIAA Paper 96-0027*, 1996.
9. Mavriplis DJ. Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes. *AIAA Paper 97-0857*, 1997.
10. Speares W, Berzins M. A 3D unstructured mesh adaptation algorithm for time-dependent shock-dominated problems. *International Journal for Numerical Methods in Engineering* 1997; **25**:81–104.
11. Nakahashi K, Deiwert GS. Self-adaptive grid method with application to airfoil flow. *AIAA Journal* 1993; **25**(8):513–520.
12. Pirzadeh SZ. An adaptive unstructured grid method by grid subdivision, local remeshing and grid movement. *AIAA Paper 99-3255*, 1999.
13. Cougny HL, Shephard MS. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering* 1999; **46**:1101–1125.
14. Biswas R, Oliker L, Sohn A. Global load balancing with parallel mesh adaptation on distributed-memory systems. *Proceedings of Supercomputing'96*, August 1996.
15. Oliker L, Biswas R, Strawn RC. Parallel implementation of an adaptive scheme for 3D unstructured grids on the SP2. *Proceedings of the Third International Workshop on Parallel Algorithms for Irregular Structured Problems*, August 1996.

16. Leyland P, Richter R. Completely parallel flow simulations using adaptive unstructured meshes. *Computer Methods in Applied Mechanics and Engineering* 2000; **184**:467–483.
17. Roe PL. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics* 1981; **43**:357–372.
18. Frink NT. Upwind scheme for solving the Euler equations on unstructured tetrahedral meshes. *AIAA Journal* 1992; **30**(1):70–77.
19. Holmes DG, Connell SD. Solution of the 2D Navier–Stokes equations on unstructured adaptive grids. *AIAA Paper 89-1932*, 1989.
20. Alonso JJ, Jameson A. Fully-implicit time-marching aeroelastic solutions. *AIAA Paper 94-0056*, 1994.
21. Karypis G, Kumar V. Analysis of multilevel graph partitioning. *Technical Report TR 95-037*, Department of Computer Science, University of Minnesota, 1995.
22. Landon RH. NACA0012 oscillatory and transient pitching. *Compendium of Unsteady Aerodynamic Measurements*, *AGARD R-702*, 1982.
23. Tijdeman H, Van Nunen JWG, Kraan AN, Persoon AJ, Poestkoke R, Roos R, Schippers P, Siebert CM. Transonic wind tunnel tests on an oscillating wing with external stores, part 2: The clean wing. *AFFDL-TR-78-194*, U.S. Air Force, 1979.