INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 26, 57–78 (1998)

OCTREE PARTITIONING OF HYBRID GRIDS FOR PARALLEL ADAPTIVE VISCOUS FLOW SIMULATIONS

T. MINYARD AND Y. KALLINDERIS*

Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, Austin, TX 78712, U.S.A.

SUMMARY

A parallel finite volume method for the Navier–Stokes equations with adaptive hybrid prismatic/tetrahedral grids is presented and evaluated in terms of parallel performance. A new method of domain partitioning for complex 3D hybrid meshes is also presented. It is based on orthogonal bisection of a special octree corresponding to the hybrid mesh. The octree is generated automatically and can handle any type of 3D geometry and domain connectivity. One important property of the octree-based partitioning that is exploited is the octree's ability to yield load-balanced partitions that follow the shape of the geometry. This biasing of the octree results in a reduced number of grid elements on the interpartition boundaries and thus fewer data to communicate among processors. Furthermore, the octree-based partitioning gives similar quality of partitions for very different geometries, while requiring minimal user interaction and little computational time. The partitioning method is evaluated in terms of quality of the subdomains as well as execution time. Viscous flow simulations for different geometries are employed to examine the effectiveness of the octree-based partitioning and to test the scalability of parallel execution of the Navier–Stokes solver and hybrid grid adapter on two different parallel systems, the Intel Paragon and the IBM SP2. © 1998 John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **26**: 57–78 (1998)

KEY WORDS: parallel processing; domain partitioning; hybrid grids; grid adaptation

1. INTRODUCTION

Recent trends in computational fluid dynamics (CFD) have shown a large increase in the development of parallel algorithms for large-scale CFD simulations. Such parallel algorithms may achieve computational speeds which surpass modern vector supercomputers by dividing the computational domain onto a number of processors which perform computations independently.[1,2] Part of the popularity of the parallel algorithms can be attributed to the ease with which a parallel environment can be achieved. With the recent standardization of parallel communication schemes a simple collection of workstations networked together can behave as a parallel machine and can in fact run the same codes as those developed for state-of-the-art parallel architectures.[3]

Implementing CFD simulations in a parallel environment introduces a number of new difficulties

Correspondence to: Y. Kallinderis, Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, Austin, TX 78712, U.S.A.
Contract grant sponsor: NSF; Contract grant number: ASC-9357677
Contract sponsor: Texas Advanced Techanology Program; Contract grant Number: #003658-413

CCC 0271–2091/98/010057–22 $17.50

Received 2 February 1996
Revised 21 January 1997

not encountered with serial algorithms. First, to achieve optimum scalability, parallel simulations require computational domains to be partitioned equally so as to have identical loads on all processors. These domains can have very complex geometries. The partitioning of the computational domain should be automatic and efficient and should also seek to minimize the communication requirements along interpartition boundaries. Furthermore, the partitioning algorithm should work effectively for a variety of different geometries and mesh topologies.

Several approaches to partitioning of complex computational domains have been developed.[4–8] Two of the more popular techniques are orthogonal recursive bisection and eigenvalue recursive bisection. Orthogonal recursive bisection uses cutting planes to partition the grid based on the centroidal co-ordinates of the cells. This approach is fast, but the number of elements on the partition interfaces can be large. Moreover, the method cannot handle complex 3D grids easily. Eigenvalue recursive bisection requires solution of eigenvalue problems and is quite expensive, but it reduces the number of elements on partition interfaces.[6,9] One of the most effective eigenvalue techniques is recursive spectral bisection (RSB) which partitions on the basis of graph connectivity. This technique has been used for partitioning of unstructured meshes to obtain high-quality subdomains.[7,8]

The accuracy and stability of a numerical method are almost always a function of the size and shape of the grid elements that fill the computational domain. Therefore obtaining a mesh that can follow complex surface geometries and fluid flow patterns is of primary importance. Tetrahedra can provide this flexibility in 3D, since they can cover complicated topologies more easily than can hexahedral meshes.[10] However, employment of tetrahedral cells for some regions of the fluid flow, such as boundary layers, is quite expensive. In these regions, strong solution gradients usually occur in the direction normal to the surface, so large-aspect-ratio cells are commonly employed to resolve the boundary layer. Structured grids are superior in capturing the directionality of the flow field in viscous regions, since they can be aligned with the boundary layer. A compromise between the two different types of meshes is the use of prismatic grids. Prismatic cells can have very high aspect ratio, while providing geometric flexibility in covering complex surfaces.[11] The semi-structured nature of the prismatic grid results in a reduced amount of memory for storing the grid data structures.[12] However, prismatic cells cannot cover multiply connected domains. Thus using prismatic elements around the surface of the body and tetrahedra to fill in the rest of the domain results in a hybrid grid.[13,14]

The initial meshes used for simulation of viscous flows may not always yield the desired accuracy, and improvement of the mesh may be necessary. Adaptive grid methods have evolved as an efficient tool to obtain accurate numerical solutions without *a priori* knowledge of the grid resolution necessary to resolve the flow features. These algorithms detect the regions that have prominent flow features and increase the grid resolution locally in such areas.[12,15–18] These adaptive grid algorithms have been developed for sequential execution and have reached a level of maturity. However, the area of parallel adaptive algorithms remains relatively unexplored.[19–22] Other work in the general area of parallel solvers for unstructured grids has been reported in References 23–25.

A key issue for parallel implementation of a flow solver and grid adapter is that the algorithm should be scalable. In other words, as the number of processors increases, the amount of execution time should decrease proportionally. The factors which affect the scalability are the balance of work among the processors and the time for communicating information from one processor to another. Therefore scheduling the flow of information among processors becomes crucial for parallel performance. The communication pattern should be able to handle any arrangement and number of partitions. Furthermore, it should not result in appreciable increase in communication time as the number of processors grows, since this would render parallel execution not scalable.

The current work presents a parallel Navier–Stokes solver and grid adapter with hybrid prismatic/tetrahedral grids. The parallel implementation of both algorithms is discussed and the scalability results are examined for two different parallel systems, the Intel Paragon and the IBM SP2.

A new partitioning scheme is developed in which the orthogonal bisection approach is applied to a special octree corresponding to the hybrid mesh. The resulting subdomains based on the octree have fewer elements on the partition interface than if bisection of the grid cells were performed. The generality of the octree-based partitioning is explored with respect to very different geometries. Qualities of the resulting partitions are compared with those obtained by the recursive spectral bisection method. Scaling of execution times of the octree partitioner with increasing number of partitions is also examined.

2. ADAPTIVE NUMERICAL METHOD

2.1. Viscous flow solver

The Navier–Stokes equations for viscous flow are employed in the integral form

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \mathbf{U}\,\mathrm{d}\Omega + \int_\Omega \left(\frac{\partial \mathbf{F}_\mathrm{I}}{\partial x} + \frac{\partial \mathbf{G}_\mathrm{I}}{\partial y} + \frac{\partial \mathbf{H}_\mathrm{I}}{\partial z}\right)\mathrm{d}\Omega = \int_\Omega \left(\frac{\partial \mathbf{F}_\mathrm{V}}{\partial x} + \frac{\partial \mathbf{G}_\mathrm{V}}{\partial y} + \frac{\partial \mathbf{H}_\mathrm{V}}{\partial z}\right)\mathrm{d}\Omega, \tag{1}$$

where $\mathbf{U}^\mathrm{T} = (\rho, \rho u, \rho v, \rho w, E)$ is the state vector of unknown variables, $\mathbf{F}_\mathrm{I}$, $\mathbf{G}_\mathrm{I}$ and $\mathbf{H}_\mathrm{I}$ are the inviscid flux vectors and $\mathbf{F}_\mathrm{V}$, $\mathbf{G}_\mathrm{V}$ and $\mathbf{H}_\mathrm{V}$ correspond to the viscous terms.

The volume integral containing the spatial derivatives in equation (1) is equivalent to a surface integral via the divergence theorem. For example,

$$\int_\Omega \left(\frac{\partial \mathbf{F}_\mathrm{I}}{\partial x} + \frac{\partial \mathbf{G}_\mathrm{I}}{\partial y} + \frac{\partial \mathbf{H}_\mathrm{I}}{\partial z}\right)\mathrm{d}\Omega = \int_{\partial\Omega} (\mathbf{F}_\mathrm{I} n_x + \mathbf{G}_\mathrm{I} n_y + \mathbf{H}_\mathrm{I} n_z)\,\mathrm{d}S, \tag{2}$$

where n_x, n_y and n_z are the components of the unit vector normal to the area element $\mathrm{d}S$ on the boundary surface $\partial\Omega$.

Evaluation of the above integral employs a special cells that surround each node (dual cells). Figure 1(a) shows a 2D example of the dual areas for the nodes in a hybrid mesh. The faces of these dual volumes pass through the midpoints of the edges that share the node. As a result, the summation over all the dual mesh faces that constitute the boundary of the control volume around a node reduces to summation over all edges of the mesh.[12] Specifically, equation (1) is discretized as

$$\left(\frac{\partial \mathbf{U}}{\partial t}\right)_N \Omega_N + \sum_e \left[(\mathbf{F}_\mathrm{I} - \mathbf{F}_\mathrm{V})S_x + (\mathbf{G}_\mathrm{I} - \mathbf{G}_\mathrm{V})S_y + (\mathbf{H}_\mathrm{I} - \mathbf{H}_\mathrm{V})S_z\right]_e = 0, \tag{3}$$

where the summation is over the edges that share node N, Ω_N is the volume of the dual cell associated with node N, S_x, S_y and S_z are the areas of the dual faces projected on the planes yz, xz and xy respectively. The flux vectors are considered at the midpoints of the edges. The viscous terms $\mathbf{F}_\mathrm{V}$, $\mathbf{G}_\mathrm{V}$ and $\mathbf{H}_\mathrm{V}$ consist of first-order derivatives of the state variables, which need to be evaluated at the middle of the edges. Evaluation of the viscous terms employs special cells that surround each edge (edge dual volumes). An example edge dual area for a 2D hybrid mesh is shown in Figure 1(b). Details of the method are given in Reference 12.

The state vector $\mathbf{U}$ is stored at the vertices (nodes) of the hybrid mesh. The solution at any particular node N at time level $n+1$ can expressed in terms of the solution at time level n using a Taylor series expansion given by

$$\mathbf{U}_N^{n+1} = \mathbf{U}_N^n + \delta\mathbf{U}_N^n,$$

$$\delta\mathbf{U}_N^n = \left(\frac{\partial \mathbf{U}}{\partial t}\right)_N^n \Delta t + \left(\frac{\partial^2 \mathbf{U}}{\partial t^2}\right)_N^n \frac{(\Delta t)^2}{2} + O((\Delta t)^3). \tag{4}$$

The temporal derivatives in the above expression are evaluated in terms of the spatial derivatives using the governing equations according to the Lax–Wendroff approach, except that the viscous

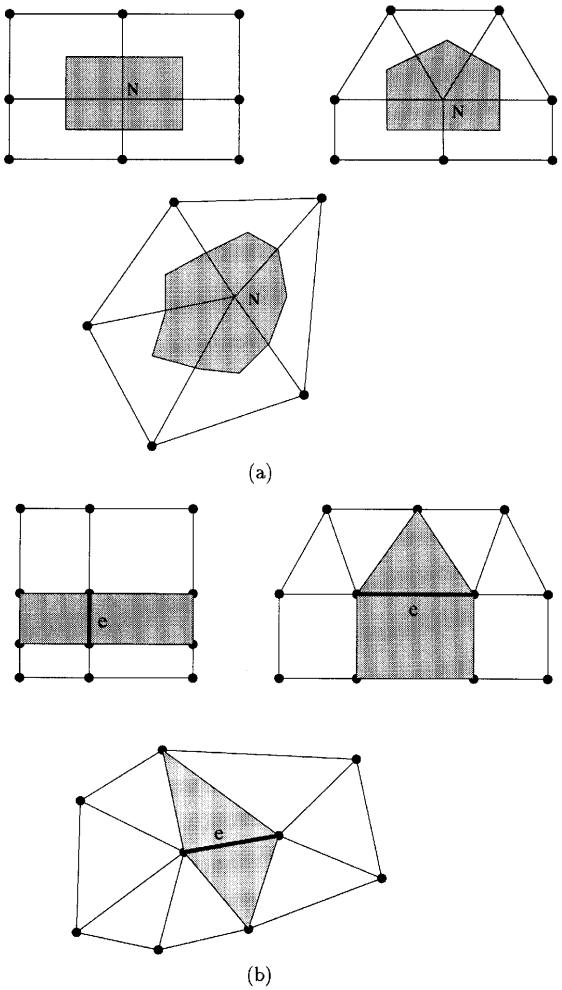


Figure 1. Examples of (a) node and (b) edge dual areas for 2D hybrid grids

terms are not included in the calculation of $\partial^2\mathbf{U}/\partial t^2$. Additional fourth-order smoothing is applied to prevent oscillations in most regions of the flow, except near shocks where second-order smoothing is used. Local time steps are employed to advance the solution in time to steady state.

Three-dimensional Navier–Stokes computations usually require a large amount of memory. In the present work the structure of the prismatic grid in one of the directions is exploited in order to reduce storage to the amount required for a two-dimensional Navier–Stokes with triangles.[12] All pointers that are employed refer to the triangular faces of the first prisms on the body surface (*base* faces), with all prisms above the same *base* face forming a *stack*. A simple index (typical of structured grid indexing) is sufficient to refer to any prism cell belonging to the same stack. The *base* faces pointers connect the faces to their corresponding edges and nodes.

2.2. Hybrid grid adaptation

Two different types of grid adaptation are employed in order to provide an optimum mesh for viscous flow simulations. The prims are locally refined in the directions parallel to the surface, while the tetrahedra can be refined either isotropically or directionally.[12]

A special type of adaptive refinement of prisms is applied in the present work in order to preserve the structure of the mesh along the *normal-to-surface* direction. The detected triangular faces on the surface are divided, then all prisms above these faces are directionally divided along the lateral directions. In other words, the triangular faces are divided but the quadrilateral faces are not. As a consequence, the cells are not divided along the third direction that is normal to the surface. In this way, grid interfaces within the prisms region are avoided and the structure of the grid along the *normal-to-surface* direction is preserved. Therefore adaptation of the prims reduces to adaptation of the triangular grid on the surface. This results in a simpler and less expensive algorithm in terms of storage and CPU time compared with a 3D adaptation algorithm.

Two types of division are applied. The first divides the triangular faces of the prisms into four smaller triangles, while the second divides them into two, resulting in creation of four and two subcells respectively. If two edges of the triangle are to be refined, the third is also refined to avoid stretching. *Transition* cells may exist at the interface between different embedded regions that contain *hanging* nodes in the middle of some of their edges owing to refinement of neighbouring cells. These cells are also divided.

In order to avoid stretched meshes created owing to several refinements of the same cells, some rules are defined as follows.

1. Only one level of refinement/coarsening is allowed during each adaptation.
2. If the *parent* cell is refined according to one-edge division, then it is divided according to the three-edge division at the next refinement.
3. If the maximum adaptation level difference of neighbouring surface triangles around a node is more than one, the coarsest triangles will be refined according to the three-edge division.

To satisfy all the above rules for grid smoothing, some *iterations* are required. Typically, the number of iterations is less than five.

The adaptation procedure for tetrahedra is very similar to that for prisms. The feature detector flags edges to be refined. The following three types of tetrahedra cell division are considered: (i) one edge is refined, (ii) three edges on the same face are refined and (iii) all six edges are refined.

After all edges are flagged, each tetrahedral cell is visited and the flagged edges are counted. Then the cell is flagged for division according to the above three types. In all cases that are different from the three cases above, the cell is divided according to the third type of division. If two edges on the same face are flagged, the third edge of that face is also marked. In order to avoid a stretched mesh, the previous rules applied to prism adaptation are employed. The method of tetrahedral cell adaptation employed in the present work is discussed in detail in Reference 12.

2.3. Implications for parallel implementation

The parallel implementation of the solver and adapter requires the addition of communication steps so that the individual subdomains distributed among the processors can communicate information along the interpartition boundaries. Two types of communication are required by the solver and adapter: (i) node and (ii) edge communications. For the solver the majority of communications are performed based on accumulation of residuals at the nodes. For each marching step of the solver, four nodal communications must be made, while only one edge communication is necessary. The first nodal communication occurs during the calculation of the first-order changes in time of the state

vector at the nodes (equation (4)). These changes are initially calculated at the cell centre and then distributed to the nodes. The accumulation of the changes at the nodes requires a communication before further calculations can be processed. Evaluation of the viscous stresses at the middle of the edges requiring an accumulation of the contribution from the surrounding cells at each edge. This results in a communication step for the edges. The second and third nodal communications occur during calculations for the artificial damping. The edge based operations to determine the pressure switch for shock smoothing and to calculate the fourth-order smoothing terms are distributed to the nodes and accumulated, thus necessitating two separate nodal communications. The final communication step transfers the total accumulated change in the state vector at the nodes among the processors that share nodes. Then all processors can update the state vector at each of the nodes in their respective subdomains, and calculations for the next marching step can proceed.

The hybrid grid adapter operates on edges of the mesh in order to flag the ones for division as well as to impose the constraints on allowable embedding level differences that were discussed in Section 2.2. The updating of the interface pointers is done through the interface edges that are divided, which involves an exchange of information among neighbouring processors. It should be emphasized that the length of the buffer that is sent across the partitions is equal to the number of those interface edges that are divided and not to the total number of interpartition edges.

The most dominant communications are the nodal communications in the parallel solver. The present work partitions the cells rather than nodes, which results in fewer grid points on the interpartition boundaries.

2.4. Communication among processors

There are several ways the processors can exchange information. Two general ways are explored in the present work. The first will be termed the *pipeline* exchange, while the second will be called the *parallel* exchange. Both methods are independent of the number and arrangement of the partitions allocated to the processors. The *pipeline* exchange is just a simple exchange method in which the communications proceed in ascending processor number order. In other words, the first processor communicates with the second, then with the third if necessary, then the fourth and so on. At the same time the second processor waits for the first and then has to communicate to the third after the first is done. This type of exchange is simplistic, without must care taken to minimize the number of communications.

The *parallel* exchange strategy differs from the *pipeline* method in that the communication occurs simultaneously between many pairs of processors. The first step is to determine which partitions will have to communicate with one another. The partitions are then coloured so that no neighbouring partitions share the same colour. The colours are then paired with one another to form the communication pattern. In this way, all processors can be active during a communication step (either sending, receiving or both) and the minimum number of communication steps is needed. The number of required steps in order to complete the exchange of information is equal to the maximum number of partitions that are neighbours (i.e. share at least a point). In order to organize the flow of messages sent and received, three lists are constructed for both the shared nodes and edges. The first stores the *id*s of the neighbouring processors that each processor has to send information to, while the second stores the *id*s of the processors that each processor will receive from. These two arrays have dimensions of *neib_max*, which is the maximum number of neighbouring partitions over the domain. This is the same as the number of communication steps needed. The third list stores the *id*s of the shared elements on the partition interfaces.

3. OCTREE-BASED PARTITIONING OF HYBRID GRIDS

A special octree decomposition of the computational domain is constructed for partitioning of an unstructured mesh. The octree is generated by recursive subdivision of the master octant, which encompasses the entire computational domain, into successively smaller octants. A sweep over the cells in the domain is performed and the cell is placed in the octant in which its centroid lies. When the number of cells in an octant exceeds a specified amount, typically 20, the octant is refined into eight smaller octants and the cells that were in the parent octant are placed in the appropriate child octant. This process continues until all cells in the domain are placed in their respective octants. The resulting octree has significantly fewer octants than the total number of grid cells. An example of the octree for a tetrahedral mesh around a sphere is shown in Figure 2. A cut of the octree at the equatorial plane is depicted.

Two advantages of the octree-based partitioning method are apparent when compared with previous approaches for partitioning grid cells. First, the octree results in a structure that follows the geometry of interest as shown in Figure 2. The computational cells are clustered around the body. As a result the octants are refined more in this region, while the octants near the far field remain relatively large. This biasing of the octants to the grid geometry results in partitions that have a lower surface-to-volume ratio, with fewer grid elements on the interpartition boundaries. The second advantage of the special octree is a reduced amount of computational time for partitioning. Generation of the octree is a fast process that requires only a small percentage of the total time for partitioning of the grid, and once an octree is generated for a hybrid grid, it can be used for any number of partitionings. Since the number of octants is only about 10% of the total grid cells, far fewer calculations are needed for partitioning and a reduced amount of computational time is realized.

The computational grid is divided into as many subgrids as processors using a partitioning algorithm which consists of the following steps: (i) co-ordinate-based grouping of octants, (ii)

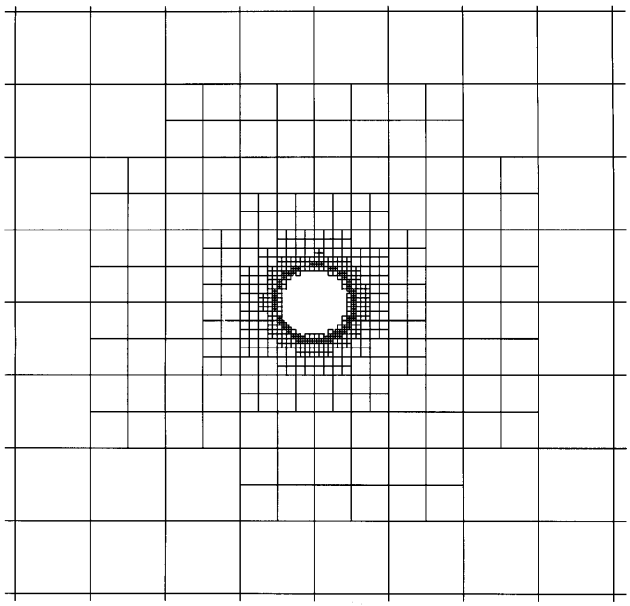


Figure 2. Two-dimensional view of special octree generated for a tetrahedral mesh around a sphere. The size and distribution of the octants follow the geometry and grid cell distribution

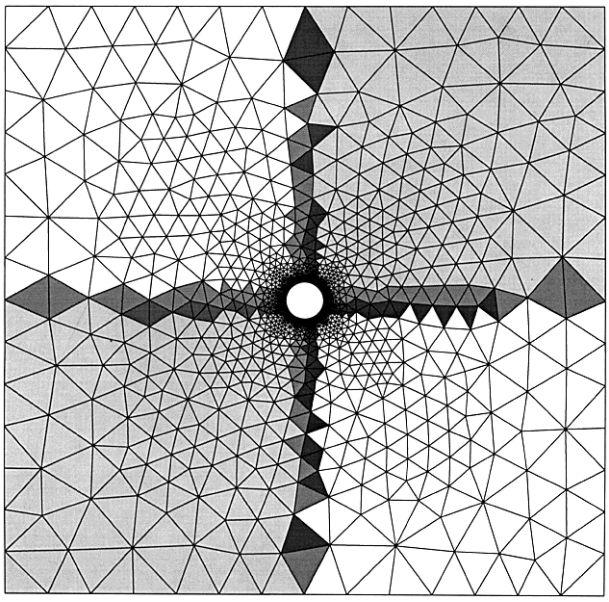


Figure 3. Partitioning of a tetrahedral mesh about a sphere using co-ordinate-based cutting planes results in long and thin subdomains with a high percentage of nodes and edges on the boundary and disconnected cells. A view of the 16 partitions on the symmetry plane of the domain is shown

smoothing of partition boundaries and (iii) mapping of global grid data structures to the local partitions. The following sections will present each of the steps using a tetrahedral grid around a sphere as an example case.

3.1. Co-ordinate-based grouping of octants

The grid is partitioned by dividing up the corresponding octree and assigning the cells in an octant to the appropriate subdomain. The octants are divided into groups based upon their centroidal co-ordinates by cutting planes that are determined by the partitioner for the number of subdomains desired. The co-ordinate-based cutting planes are better suited for division of an octree than for partitioning of the computational cells. For example, if the cutting planes were used to partition the unstructured 3D mesh about a sphere, then partitioning shown in Figure 3 would result. The future shows the footprint of the partitions on the symmetry plane for a 16-processor case. Several of the resulting subdomains are long and thin, with a high percentage of nodes and edges on interpartition boundaries and cells that are disconnected. Figure 4 shows the partitioning of the octree corresponding to the same tetrahedral grid for a sphere. The footprint of the resulting tetrahedral subdomains on the symmetry plane is shown in Figure 5(a). The partitions are not as long and thin as before and no disconnected cells are present. The percentage of nodes on the boundary has been reduced for the octree partitioning. Furthermore, the computational time for subdivision of the octree is much smaller than for division of the entire computational grid. In this case the grid contains over a 100,000 tetrahedra, but the corresponding octree has only 10,274 octants.

The structure of the prisms in the *normal-to-surface* direction is exploited. The prisms are defined by their corresponding base faces on the surface. As a result, the stacks of prisms are partitioned by simply partitioning the triangular surface mesh. All cells within each prism stack are assigned to the same partition. In this way the data structure operations for partitioning, solving and adapting of the

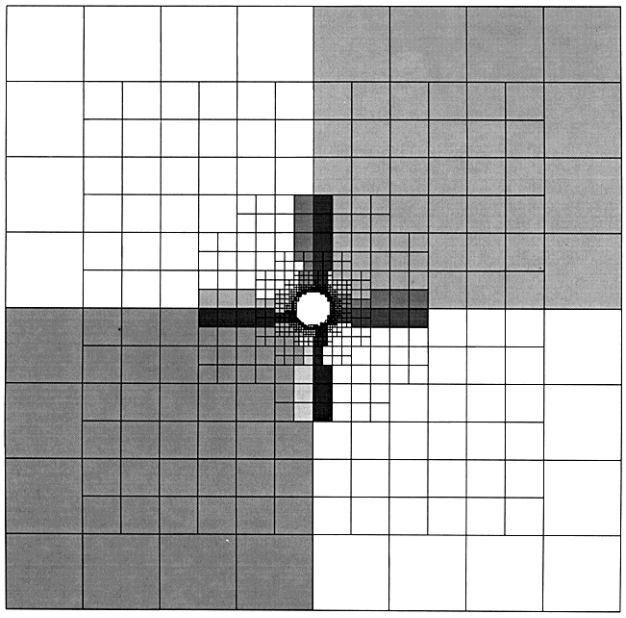


Figure 4. Partitioning of special octree corresponding to tetrahedral mesh using co-ordinate-based cutting planes. A view of the octant partitions is shown on the symmetry plane corresponding to the 16-partition case

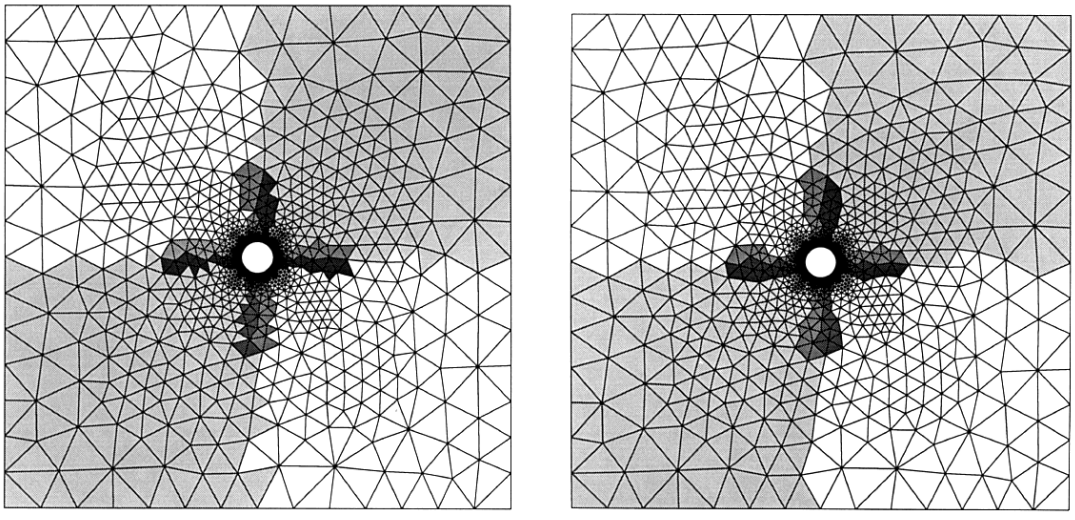


Figure 5. Resulting partitions using octre-based co-ordinate division (a) before and (b) after smoothing of partition interfaces. Views of the 16 tetrahedral partitions on the symmetry plane are shown. The partition boundaries are no longer jagged and the percentage of nodes and edges on the boundary has been reduced

prismatic grid refer to the triangular surface mesh. This results in savings in both memory and execution time.

3.2. Smoothing of interpretation boundaries

Owing to the unstructured nature of the grid, interpartition boundaries may be jagged, with a large number of nodes and edges on partition interface. These boundaries can be improved by applying a smoothing technique to the cells on the partition interfaces. At this stage of the partitioner the octree-based co-ordinate subdivision has generated balanced subdomains and each cell has been assigned a partition number. The smoothing process begins by determining which cells in the computational domain are candidates for smoothing. A cell is flagged for smoothing if all its nodes are shared between two neighbouring partitions. After flagging all the candidate cells, the partition interfaces are altered by assigning half of the flagged cells to one partition and the other half to the neighbouring partition. This process is repeated until almost all the flagged cells on the interfaces have been smoothed. A typical number of such iterations is five. The smoothing procedure also maintains the load balance among the partitions. Figure 5(b) shows the previous octree-based partitioning of the sphere tetrahedral grid after smoothing has been applied. Comparing Figure 5(b) with Figure 5(a), it is observed that the irregular and jagged boundaries have now been improved, with a lower percentage of nodes and edges on the boundary.

3.3. Mapping global data structures to local partitions

The 3D unstructured hybrid grid is uniquely defined as a collection of nodes, edges, faces and cells which are collectively referred to as 'elements'. The data structures defining the grid are split into P subsets, where P is the number of desired processors. Each processor is assigned the subsets that completely specify the subdomain allocated to it.

Within a subdomain an element of a particular type (cell/face/edge/node) is assigned a unique integer *id* between one and the total number of elements of that type in that subdomain. This simplifies the storage mechanism, since the correlation between elements of different types can now be expressed in terms of these *id*s. Thus an edge is a pair of integers (n_1, n_2), where n_1 and n_2 are *id*s of two nodes within the subdomain. These *id*s are completely local to a subdomain. There is no notion of global numbering of elements across all processors. This is of particular significance in the case of a dynamically varying computational grid. In a global numbering scheme, all processors have to communicate with each other whenever a global number is to be assigned to a newly created element. This entails additional communication overhead and can prove to be a bottleneck. Furthermore, the algorithms for dynamic updating of global numbers are cumbersome and not amenable to parallelization.

An element that is shared between several processors has multiple copies of itself, one on each processor that shares it. Furthermore, for every shared element within its partition a processor maintains a list of processors that share it, as well as the *id* of that element on each of those processors. This enables two processors that share a given element to communicate with each other when data are to be transferred to that element.

In the present work, cells are partitioned among processors according to the co-ordinates of the octant in which they lie. As a result, the boundaries between neighbouring partitions consist of edges and points. These edges and points are duplicated across the boundaries. Therefore two main arrays are employed in order to connect the duplicate elements. The pointer *nint_nod*(*i, intnod*), $i = 1, 2$, stores the *id*s of the pairs of nodes at the interfaces, while the array *nint_edg*(*i*, *intedg*), $i = 1, 2$, lists the *id*s of the pairs of edges.

4. PERFORMANCE OF HYBRID GRID PARTITIONER

The most important property of an effective grid partitioner is that it should give balanced subdomains with a minimum number of elements on partition interfaces for all types of grid elements and any complex geometry. Furthermore, the method should be as automated as possible and the amount of time for partitioning should not increase drastically as the number of partitions increases. This section presents the effectiveness of the hybrid grid partitioner by examining partition qualities and timings obtained using the octree-based method. The partition qualities are compared with those obtained from an established grid-partitioning method, namely recursive spectral bisection (RSB). Several geometries are examined, including the sphere geometry and an aircraft configuration with and without engines. It should be noted that the octree-based partitioner is automated with minimal user interaction. All the grids discussed in this paper were generated using the methods presented in Reference 13.

For the current work the quality of a partition is defined as the percentage of grid points in a subdomain that are on interpretation boundaries. This percentage relates the amount of communication required for the parallel solver to the number of computations performed within a

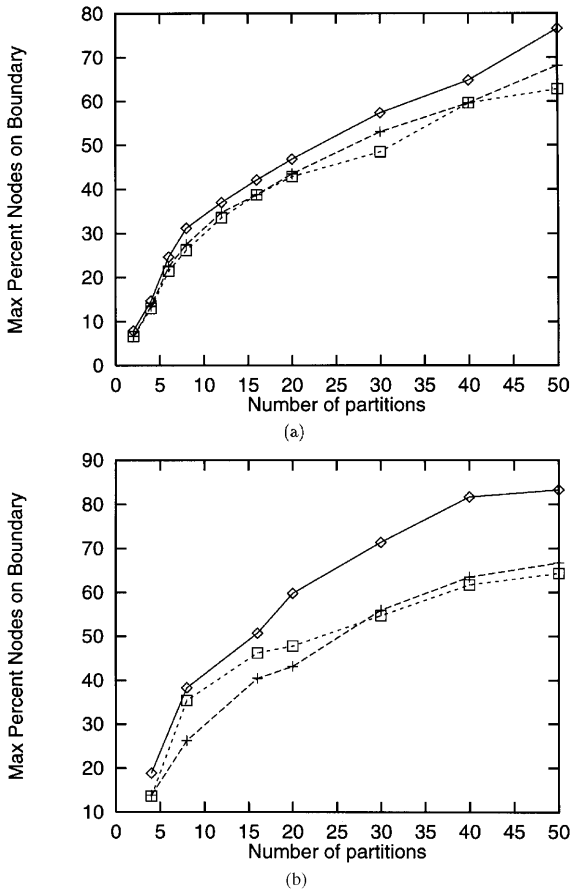


Figure 6. Comparison of scaling of maximum percentage of nodes on interface boundaries for varying number of partitions. Cases of (a) *prismatic* and (b) *tetrahedral* meshes around sphere: ◇, before smoothing; +, after smoothing; □, RSB

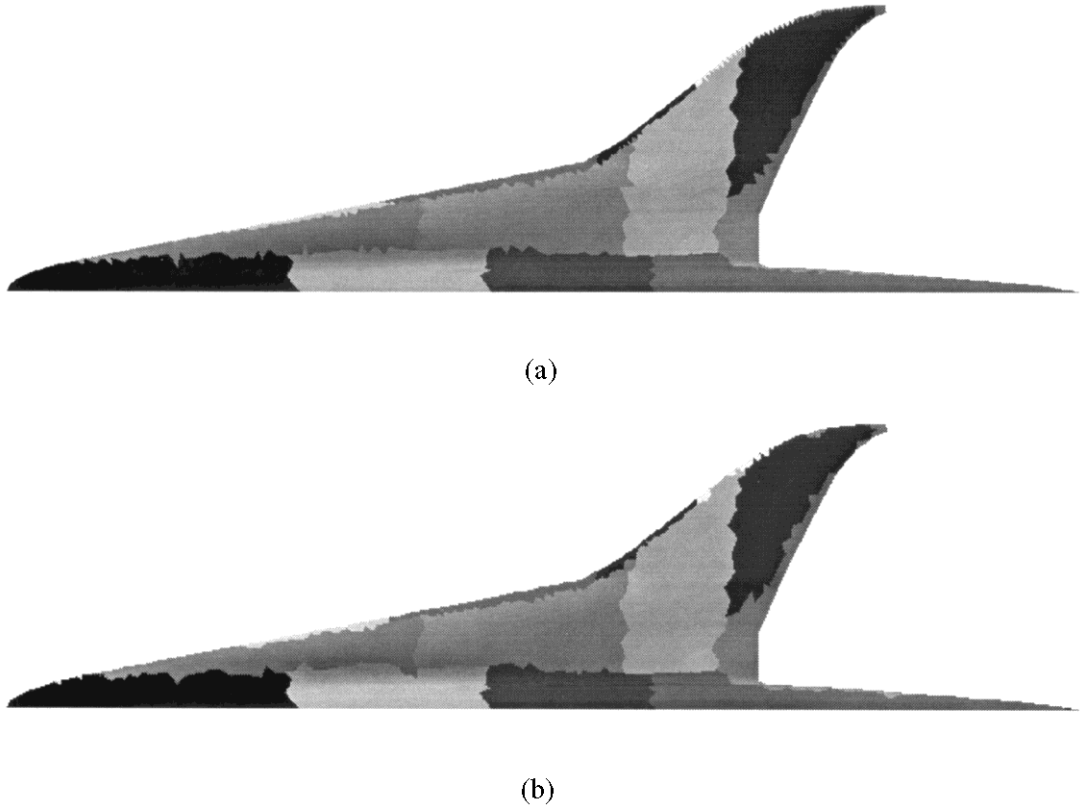


Figure 7. Signatures of 16 partitions on surface of HSCT aircraft configuration (a) before and (b) after smoothing of partition interfaces

subdomain. As this percentage increases, a larger portion of the computational time will have to be spent transferring information to neighbouring processors.

The first case tested was the hybrid grid for the sphere geometry. Figures 6(a) and 6(b) show the reduction in the maximum percentage of nodes before and after smoothing for several load-balanced prismatic and tetrahedral partitionings of the hybrid mesh about the sphere. The smoothing of partition interfaces results in a substantial reduction in the percentages of interface nodes for the tetrahedral subdomains. The figures also show how the present partitioning technique compares with the RSB approach. For most of the partitionings the present method does almost as well as RSB when comparing the maximum local percentage of nodes on partition interfaces. The average percentages of nodes on the boundaries show the same trend as the maximum percentages when compared with the RSB method. The percentages of edges in a partition that are on the interpartition boundaries also follow the same trend as the node percentages, so only the node percentages will be examined for the sake of brevity. It is also noted that the maximum number of neighbours for the octree-based partitioning was about the same as the maximum number of neighbours for the RSB partitioning. The maximum number of neighbouring partitions governs the number of communications, while the percentage of nodes on interfaces corresponds to the amount of data for communication. Even though RSB does slightly better for the static meshes, implementation of RSB in a parallel environment with dynamic meshes is much more difficult than using a load balancer with octree-based partitioning

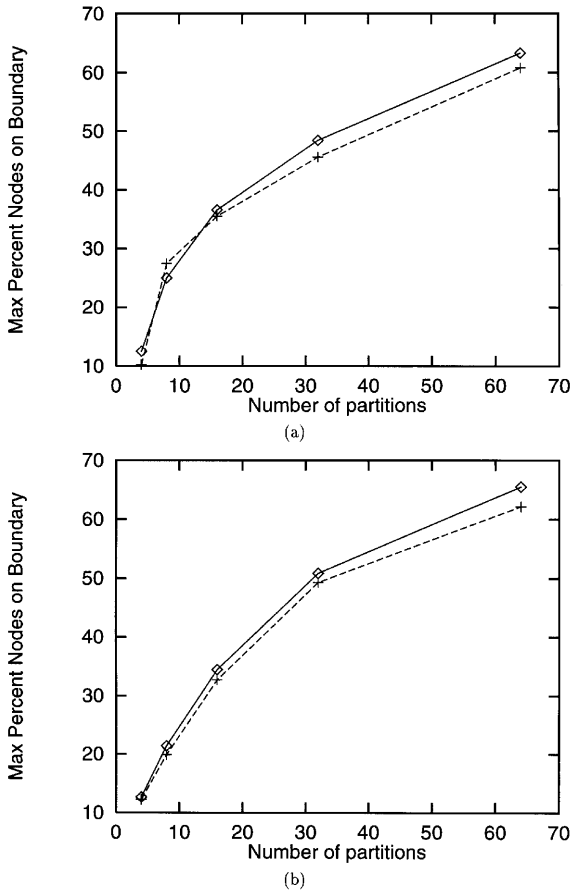


Figure 8. Comparison of scaling of maximum percentage of nodes on interface boundaries for varying number of partitions. Cases of (a) *prismatic* and (b) *tetrahedral* meshes around HSCT aircraft: ◇, octree-based partitioning; +, RSB

strategies. Since RSB uses a connectivity-based partitioning approach, the partition shapes can change significantly after adaptation and a significant portion of the grid and solution data would need to be migrated among the processors.[26]

Partitioning of the hybrid mesh about a high-speed civil transport (HSCT) aircraft configuration is now considered. The initial hybrid mesh consists of approximately 120,000 nodes, 170,000 tetrahedra, 4400 surface triangles and 176,000 prisms. The corresponding octree used to partition this mesh contains just over 15,000 octants. Figure 7(a) shows the signature of 16 partitions on the upper surface of the HSCT before smoothing of the interpretation boundaries. The surface partitions after smoothing are shown in Figure 7(b). The partition interfaces after smoothing are not as jagged as they were previously and there is a lower percentage of nodes and edges on the boundaries. Figure 8(a) compares the maximum local percentage of nodes on partition interfaces for the current octree partitioning and a partitioning generated using RSB. The octree technique yielded a slightly higher percentage of nodes on the boundaries than the RSB method. The octree partitioning of the tetrahedral portion of the HSCT hybrid grid for a case with 16 partitions is shown in Figure 9(a). The figure shows the footprint of the partitions on the symmetry plane of the mesh. The octree biases the partitioning around the aircraft geometry. The footprint of the resulting tetrahedral subdomains on the

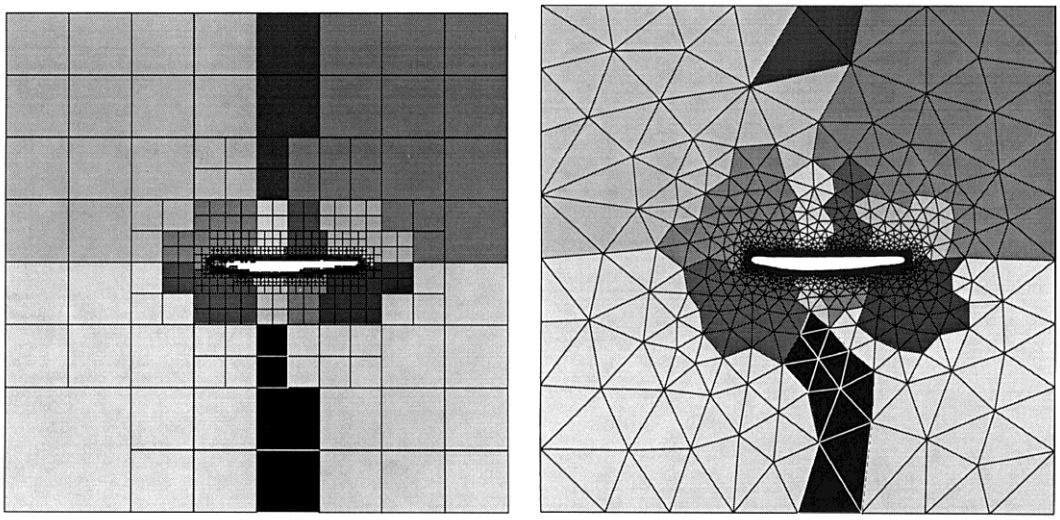


Figure 9. Partitioning of the octree corresponding to the HSCT tetrahedral mesh using co-ordinate-based cutting planes results in subdomains that are biased to the geometry. Signatures of the 16 (a) octant and (b) tetrahedral partitions are shown on the symmetry plane

symmetry plane after smoothing of the partition interfaces is shown in Figure 9(b). The partitions have smooth boundaries with no disconnected cells. A comparison of the maximum local percentage of nodes on the partition interfaces is presented in Figure 8(b). Again the percentage of nodes on interpartition boundaries is only slightly less for RSB than for the present method.

The amount of execution time to partition three different hybrid grids is plotted versus increasing number of partitions in Figure 10. The execution times increase linearly with increasing number of partitions. These timings correspond to 10 smoothing iterations on the interpartition boundaries. The largest amount of time is spent in smoothing the tetrahedral portions of the partitioning. Only 25% of the total execution time is required for generating, sorting and dividing the octree. Approximately

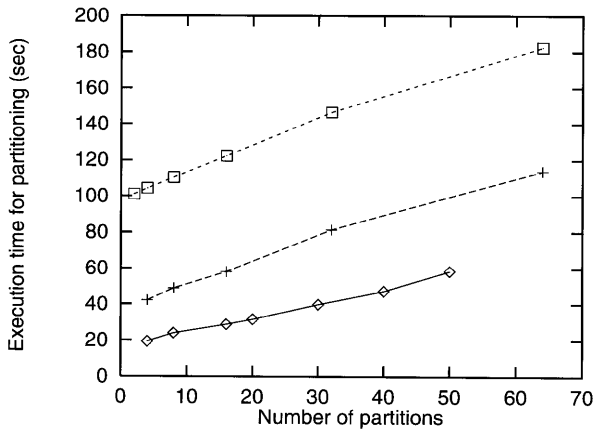


Figure 10. Scaling of execution time for partitioning of three different hybrid grids with number of partitions: ◇, sphere geometry; +, HSCT aircraft configuration without engines; □, HSCT aircraft configuration with engines

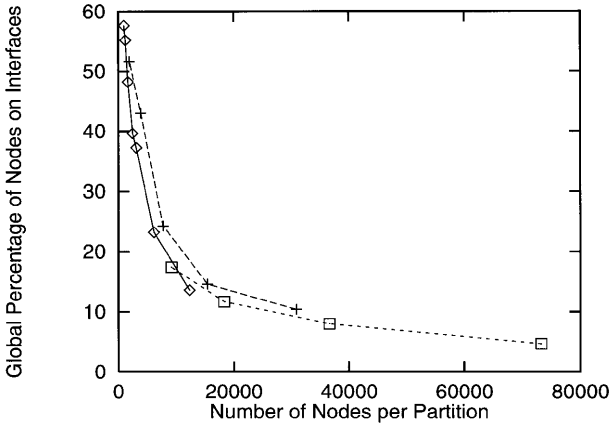


Figure 11. Geometry independence of octree-based partitioning. Global percentage of nodes on partition interface versus number of nodes per partition for sphere, HSCT without engines and HSCT with engines tetrahedral grids: ◇, sphere geometry, +, HSCT without engines; □, HSCT with engines

10% of the time is for smoothing the prism boundaries and the rest of the time is spent in smoothing the tetrahedra interfaces. The total execution time could be reduced by performing fewer iterations, but the quality of the partitions will be slightly worse. However, most of the improvement in the partition boundaries is accomplished in the first few smoothing iterations, while the remaining iterations improve the partition qualities only slightly.

Using the octree of a hybrid unstructured grid to partition the grid results in balanced subdomains that have better qualities than standard co-ordinate bisection. Since the octree is biased by the geometry and the distribution of the cells in the grid, the partitions tend to follow the same biasing and the resulting partitions have a better surface-to-volume ratio. Figure 11 shows the global percentage of nodes on interpartition boundaries plotted against the number of nodes per partition for the sphere, HSCT without engines and HSCT with engines hybrid grid. The global percentage is calculated by dividing the total number of nodes on interpartition boundaries by the total number of nodes in the grids. The curves are very similar even though the grids have different distributions of cells and nodes. This similarity shows that the current partitioning method yields approximately the same partition qualities for entirely different grids and geometries.

5. PARALLEL PERFORMANCE OF ADAPTIVE METHOD

A partitioned memory multiple-instruction/multiple-data (MIMD) architecture typically consists of a collection of multiple processors connected together by a high-speed interconnection network. Each processor has the freedom of executing its own set of instructions on its data. There is no notion of shared memory and the only way that processors can interact is through the connecting network. The Intel Paragon and the IBM SP2 are examples of such an architecture.

The same user programme is executed on all the processors, each with its own set of data. Co-ordination among processors is achieved through message passing, for which 'send' and 'receive' primitives are provided. The programming paradigm is essentially that of any ordinary sequential machine. That is, the actual structure of any programme written for a parallel machine has basically a sequential form with additional calls to the message-passing routines for synchronization among the processors. The message-passing libraries used for this work are based on the message-passing interface (MPI). The MPI provides a standard so that parallel applications are portable among

different parallel machines and clusters of workstations.[3] To examine the scalability of a parallel code, measurements are made of the execution times and communication times for an increasing number of processors. An ideal parallel code would require exactly half as much execution time for double the number of processors. Also, the amount of time spent in communication should not increase drastically with additional processors.

The following subsections present the results obtained for parallel simulation of viscous flow and parallel hybrid grid adaptation. The first two subsections examine the scalability of the solver and adapter when run on the Intel Paragon. Cases involving a duct and sphere geometry are explored by examining the execution and communication times as the number of processors increases. The final subsection discusses results for the parallel solver on the Intel Paragon and the IBM SP2 for the HSCT aircraft configuration with and without engines.

5.1. Parallel performance of Navier–Stokes solver

Two cases are considered for simulation of viscous flow on the Intel Paragon. The first case is viscous flow over a cylindrical bump in a channel. Supersonic flow with Mach number $M_\infty = 1{\cdot}4$ and a Reynolds number of 16,000 is computed. This has been a standard test case for Navier–Stokes solvers. The hybrid grid for this case consists of 18,000 prisms and 162,000 tetrahedra. The second case tested on the Intel Paragon is the case of a sphere in a supersonic flow with Mach number $M_\infty = 1{\cdot}4$ and a Reynolds number of 1000. The hybrid grid used for this case consists of 57,000 prisms and 104,000 tetrahedra. Investigation of accuracy of the solver for various cases is reported in Reference 12.

Figure 12(a) shows the reduction in execution times per time step of the parallel solver as a function of the number of processors. The scale on the axes is logarithmic with base two. It is observed that the slope of the curves is close to the ideal reduction of the execution time with increasing number of processors. The deviation from the ideal reduction in time is attributed to a slight load imbalance, since the prisms and tetrahedra are divided into separate partitions. This slight imbalance results in a small increase in execution time from the one- to two-processor channel cases. For the one-processor case the entire hybrid grid is used, but for the two-processor case one partition contains all the prisms and the other has all the tetrahedra, so a slight load imbalance arises. However, the imbalance between the prism and tetrahedral partitions does not affect the overall scaling of the parallel solver.

Figure 12(b) shows the communication times per time step of the solver as a function of the number of processors. It is observed that the *pipeline* exchange becomes quite expensive as the number of processors increases. In contrast, the *parallel* exchange time does not increase as the number of partitions increases. The *parallel* exchange strategy results in communication times that remain relatively constant as the number of processors increases for the channel and sphere cases.

5.2. Parallel performance of hybrid grid adapter

Adaptive local refinement of a hybrid prismatic/tetrahedral grid is now considered for the cases of supersonic channel flow over a bump and supersonic flow around a sphere. In order to examine the scalability of the adapter, the same hybrid grid used by the solver is adapted once by dividing a certain percentage of cells within each processor partition. For the cases considered, 10% of the cells are refined in each subdomain. The total execution time for this step is measured on each processor. The total time for execution of the algorithm is taken to be the maximum of all the individual processor timings.

Scalability of parallel execution times for the grid adapter using the channel and sphere geometries is shown in Figure 13(a). The scale on the axes is logarithmic with base two. It is observed that the

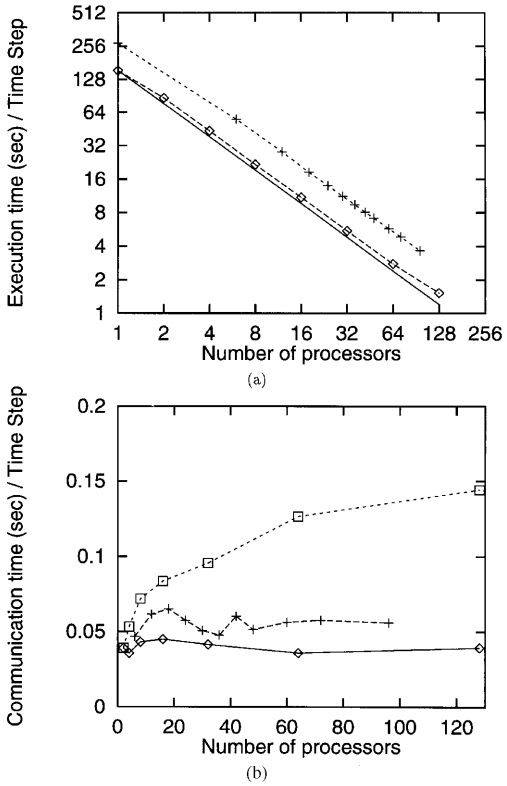


Figure 12. Scaling of parallel solver (a) execution and (b) communication times with number of processors on Intel Paragon. Cases of hybrid grids in a channel (18,000 prisms, 162,000 tetrahedra) and around a sphere (57,000 prisms, 104,000 tetrahedra): ◇, channel; +, sphere; —, ideal slope for proportional reduction in time

reduction in execution time is close to the ideal slope. The deviation of the actual times from the ideal times for the larger number of processors is caused by the larger relative communication times and the reduction in work for each processor. As the number of processors increases, the amount of work decreases and in this case the processors do not have enough calculations to keep them busy. One other item to note is that the calculations for the adapter are dominated by the tetrahedral adaptation. The prismatic adaptation reduces to a 2D adaptation on the surface, which is much faster than the 3D adaptation for the tetrahedra. The communication times for the adapter with increasing number of processors are presented in Figure 13(b). The *parallel* exchange strategy is employed. The figure shows that the communication times remain relatively the same with increasing number of processors. Also, the communication times are small compared with the amount of execution time for adaptation.

5.3. Parallel adaptive simulation of viscous flow around the HSCT aircraft

Simulation of viscous flow around the HSCT aircraft configuration is now considered. Figure 14 shows the entropy contours at three stations when simulating a subsonic ($M_\infty = 0{\cdot}3$), high-angle-of-attack ($\alpha = 17{\cdot}8^\circ$) viscous flow with $Re = 10^6$ around the HSCT aircraft without engines.[14] The growth of the primary vortex and the development of the secondary vortex are clearly visible. Several of the partitions obtained using the octree method were run on the Intel Paragon and the IBM SP2.

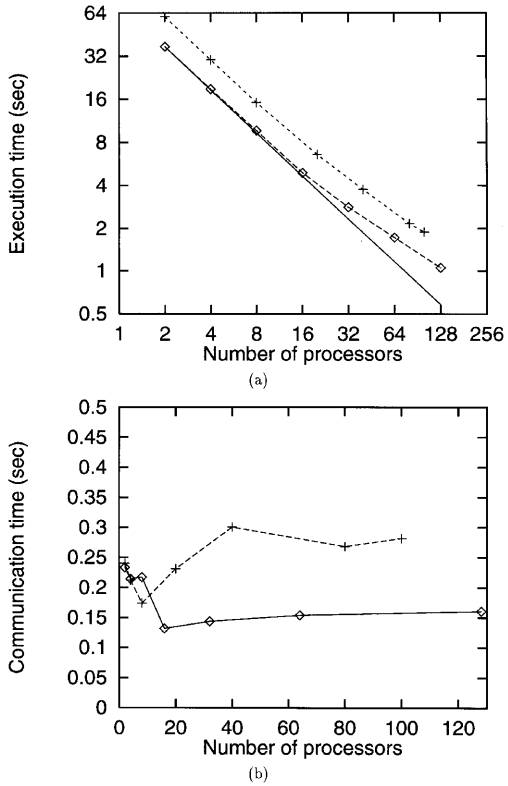


Figure 13. Scaling of (a) execution and (b) communication times with number of processors for parallel adaptation on Paragon. No appreciable increase in communication times on Paragon using *parallel* exchange strategy. Cases of hybrid grids in a channel (18,000 prisms, 162,000 tetrahedra) and around a sphere (57,000 prisms, ◇, channel; +, sphere, —, ideal slope for proportional reduction in time

The prismatic and tetrahedral regions of the grid are divided into the same number of subdomains. These subdomains are then combined to form the hybrid partitions, resulting in a better load balance. Figure 15 presents the scalability results for parallel execution of the solver for the HSCT with and without engines. The scale on the axes is logarithmic with base two. The execution times per time step for increasing number of processors exhibit a proportional reduction with only a slight deviation from the ideal reduction in time. The deviation from the ideal for the cases run on the IBM SP2 is attributed to the larger relative communication times and not enough work to keep the processors busy. It should be noted that approximately 10 processors on the IBM SP2 take about the same amount of execution time as one processor on the Cray C90. The communication times per time step for this case are shown in Figure 16. The communication times remain relatively constant with increasing number of processors. Furthermore, the times are small when compared with the execution time of the solver.

After the initial solution was obtained, the mesh was adapted, resulting in a mesh with approximately 512,000 prisms and 366,000 tetrahedra. Figure 17 shows the surface triangulation for the prisms before and after adaptation for 16-processor case. Most of the adaptation has occurred on the upper surface of the aircraft, especially near the wing–fuselage junction, owing to the large vortex coming from the wing leading edge near the front of the aircraft for this angle of attack. The

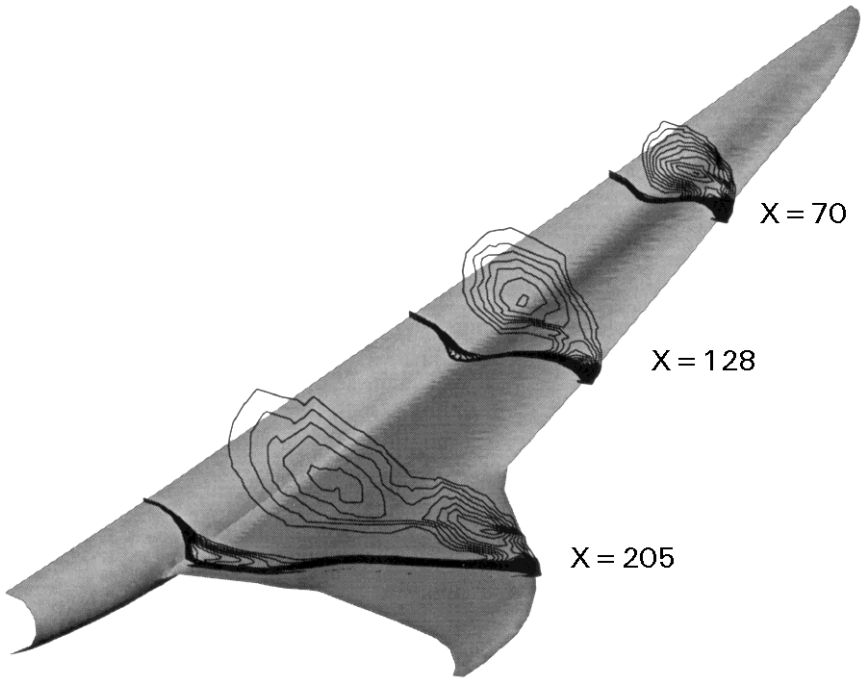


Figure 14. Entropy contours at three stations for subsonic, high-angle-of-attack viscous flow simulation ($M_\infty = 0{\cdot}3$, $\alpha = 17{\cdot}8^\circ$) about HSCT aircraft. Growth of the primary vortex and development of the secondary vortex are clearly visible. Case of hybrid grid with 176,000 prisms and 170,000 tetrahedra

overlapping grid lines seen in the figure near the wing leading and trailing edges are due to the plotting package showing the lower surface grid. The total time required to adapt this 16-processor case was 18·3 s on the IBM SP2. For this case the adaptation is not uniform in all the subdomains, so now an imbalance exists among the processors. However, the octree-based partitioning approach discussed earlier could be used in parallel to dynamically rebalance the load.[26,27]

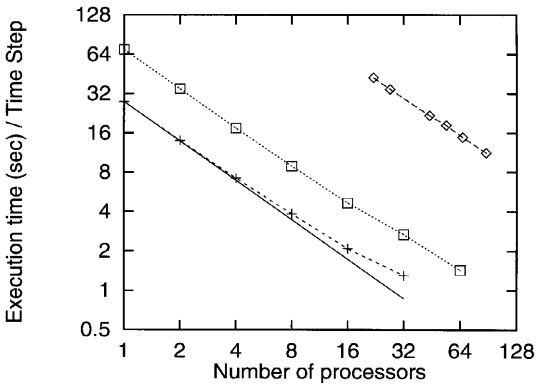


Figure 15. Scalability results for parallel execution of solver on Paragon and SP2 for HSCT aircraft configurations. Cases of hybrid grids around HSCT without engines (176,000 prisms, 170,000 tetrahedra) and HSCT with engines (395,000 prisms, 488,000 tetrahedra): ◇, HSCT without engines on Paragon; +, HSCT without engines on SP2; □, HSCT with engines on SP2; —, ideal slope for proportional reduction in time

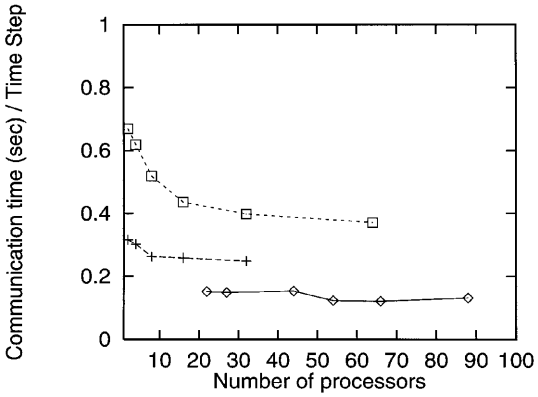
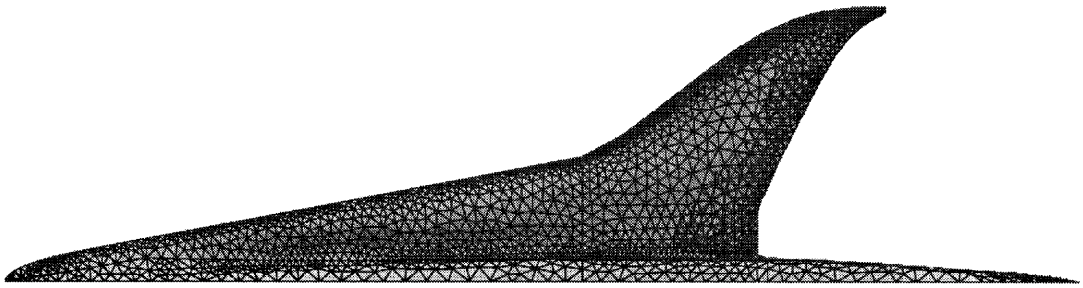
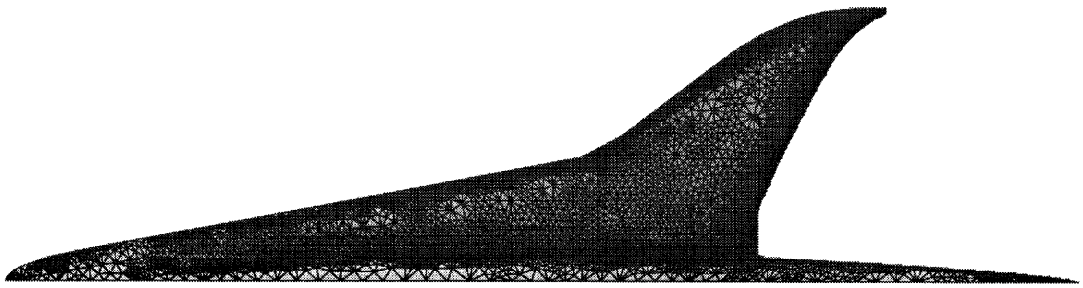


Figure 16. No appreciable increase in communication times for solver on Paragon and SP2 using *parallel* exchange strategy. Cases of hybrid grids around HSCT without engines (176,000 prims, 170,000 tetrahedra) and HSCT with engines (395,000 prisms, 488,000 tetrahedra): ◇, HSCT without engines on Paragon; +, HSCT without engines on SP2; □, HSCT with engines on SP2



(a)



(b)

Figure 17. Surface triangular of HSCT aircraft configuration without engines: (a) before adaptation (176,000 prisms, 171,000 tetrahedra); (b) after adaptation (512,000 prisms, 366,000 tetrahedra)

6. CONCLUDING REMARKS

The octree-based partitioner results in load-balanced subdomains that are biased to the computational domain. The sizes and distributions of the octants follow the shape and cell clustering of the mesh geometry. This results in better partitions compared with co-ordinate bisection of the grid cells themselves. Furthermore, the octree-based method gives similar-quality partitions for very different geometries, namely the channel, sphere and HSCT aircraft configurations. The partition qualities obtained from the octree-based partitioning are about the same as RSB, but RSB is not as effective on dynamic meshes in a parallel environment. Parallel implementation of the Navier–Stokes solver and hybrid grid adapter yields near-ideal scalability for the channel and sphere geometries on the Intel Paragon. Viscous flow simulations around the HSCT aircraft with and without engines will result in near-ideal scalability on both the Intel Paragon and the IBM SP2.

ACKNOWLEDGEMENTS

The authors would like to thank Horst Simon for providing us with the RSB partitioning code Version 2.2 written by Steve Barnard and Horst Simon. This work was supported by NSF Grant ASC-9357677 (NYI program) and Texas Advanced Technology Program (ATP) Grant #003658-413. Parallel computing time on the Intel Paragon and IBM SP2 was provided by the NAS Division of NASA Ames Research Center. Supercomputing time was provided by the High Performance Computing Facility at the University of Texas at Austin.

REFERENCES

1. T. L. Holst, M. D. Salas and R. W. Claus, 'The NASA Computational Aerosciences Program—toward teraflop computing', *AIAA Paper 92-0558*, 1992.
2. H. D. Simon, 'Seven years of parallel computing at NAS (1987–1994): what have we learned?', *AIAA Paper 95-0219*, 1995.
3. 'MPI: a message passing standard', *Int. J. Supercomput. Appl. High Perform. Comput.*, **8**, (1994).
4. M. J. Berger and S. H. Bokhari, 'A partitioning strategy for nonuniform problems on multiprocessors', *IEEE Trans. Comput.*, **C-36**, 570–580 (1987).
5. C. Farhat and M. Lesoinne, 'Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics', *Int. j. nuer. meth. engng.*, **36**, 745–764 (1993).
6. A. Posthen, H. D. Simon and K.-P. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl*, **11**, 430–452 (1990).
7. H. D. Simon, 'Partitioning of unstructured problems for parallel processing', *Tech. Rep. RNR-91-008*, NASA Ames Research Center, Moffett Field, CA, 1991.
8. G. Karypis and V. Kumar, 'A fast and high quality multilevel scheme for partitioning irregular graphs', *Tech. Rep. TR 95-035*, Department of Computer Science, University of Minnesota, 1995 (a short version appears in *Int. Conf. on Parallel Processing*, 1995).
9. E. R. Barnes, 'An algorithm for partitioning the nodes of a graph', *SIAM J. Alg. Disc. Meth.*, **3**, 541 (1982).
10. T. J. Baker, 'Developments and trends in three dimensional mesh generation', *Appl. Numer. Math.*, **5**, 275–304 (1989).
11. Y. Kallinderis and S. Ward, 'Prismatic grid generation for 3-D complex geometries', *AIAA J.*, **31**, 1850–1856 (1993).
12. V. Parthasarathy and Y. Kallinderis, 'Adaptive prismatic–tetrahedral grid refinement and redistribution for viscous flows', *AIAA J.*, **34**, 707–716 (1996).
13. Y. Kallinderis, A. Khawaja and H. McMorris, 'Hybrid prismatic/tetrahedral grid generation for viscous flows around complex geometries', *AIAA J.*, **34**, 291–298 (1996).
14. A. Khawaja, H. McMorris and Y. Kallinderis, 'Hybrid grids for viscous flows around complex 3-D geometries including multiple bodies', *AIAA Paper 95-1685-CP*, 1995.
15. R. Lohner and J. Baum, 'Numerical simulation of shock interaction with complex geometry three-dimensional structures using a new adaptive H-refinement scheme on unstructured grids', *AIAA Paper 90–0700*, 1990.
16. M. J. Aftosmis, 'Viscous flow simulations using upwind method for hexahedral adaptive meshes', *AIAA Paper 93-0772*, 1993.
17. R. Biswas and R. Strawn, 'A new procedure for dynamic adaptation of three dimensional unstructured grids', *AIAA Paper 93-0672*, 1993.

18. Y. Kallinderis, 'Grid adaptation by redistribution and local embedding', in *Lecture Notes for 27th Computational Fluid Dynamics Lecture Series*, Von Karman Institute for Fluid Dynamics, Brussels, 1996.
19. A. Vidwans and Y. Kallinderis, 'Unified parallel algorithm for grid adaptation on a multiple-instruction multiple-data architecture', *AIAA J.*, **32**, 1800–1807 (1994).
20. H. L. de Cougny, K. D. Devine, J. E. Flaherty, R. M. Loy, C. Özturan and M. S. Shepard, 'Load balancing for the parallel adaptive solution of partial differential equations', *Appl. Numer. Math.*, **16**, 157–182 (1994).
21. Z. Weinberg and L. N. Long, 'A massively parallel solution of the three dimensional Navier–Stokes equations on unstructured, adaptive grids', *AIAA Paper 94-0760*, 1994.
22. M. S. Shepard, *et al.*, 'Parallel automated adaptive procedures for unstructured meshes', *Scientific Computation Research Center Rep.* 11-1995, Renesselaer Polytechnic Institute, Troy, NY, 1995.
23. R. D. Williams, 'Supersonic fluid flow in parallel with an unstructured mesh', *Concurrency Pract. Experience*, **1**, 51–62 (1989).
24. V. Venkatakrishnan, H. D. Simon and T. J. Barth, 'A MIMD implementation of a parallel Euler solver for unstructured grids', *J. Supercomput.*, **6**, 117–137 (1992).
25. V. Venkatakrishnan, 'Parallel implicit unstructured grid Euler solvers', *AIAA Paper 94-0759*, 1994.
26. T. Minyard, Y. Kallinderis and K. Schulz, 'Parallel load balancing for dynamic execution environments', *AIAA Paper 96-0295*, 1996.
27. T. Minyard and Y. Kallinderis, 'Applications of grid partitioning and parallel dynamic load balancing', *AIAA Paper 97-0879*, 1997.

