COMPRESSIBLE AERODYNAMICS USING A PARALLEL ADAPTIVE TIME–DISCONTINUOUS GALERKIN LEAST–SQUARES FINITE ELEMENT METHOD*

Carlo L. BOTTASSO[†], Hugues L. De COUGNY[‡], Mustafà DINDAR[§], Joseph E. FLAHERTY[¶], Can ÖZTURAN[§], Zvi RUSAK[||], Mark S. SHEPHARD[**]

Rensselaer Rotorcraft Technology Center, and
Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180, USA

Abstract

A parallel automated adaptive methodology for the analysis of steady and unsteady compressible fluid flows on distributed memory computers is presented. The technique developed here relies on a stabilized implicit space–time finite element formulation, on a domain decomposition based strategy for achieving parallelism while efficiently exploiting data locality, on an edge based adaptive scheme that combines refinement, derefinement and triangulation optimization. An element migration and load balancing procedure is used for reducing the load imbalance caused by the adaptation process. The message passing paradigm is employed in each stage of the analysis, realizing a uniform software environment that shares a common data structure. Some preliminary results relative to the parallel adaptive analysis of compressible flow problems are presented at the end of the paper.

Introduction

The accurate simulation of the evolution of steady and unsteady compressible aerodynamic flows is a challenging problem that requires the ability to resolve the varying length and time scales of the flow. The present status of knowledge indicates that adaptivity is a viable way for effectively and accurately analyzing complex physical phenomena like compressible flows. A typical adaptive methodology should be able to efficiently realize the following steps: *(i)* effectively discretize the flow domain, *(ii)* analyze the problem using an appropriate solver on the generated discretization, *(iii)* estimate which regions of the flow are not providing sufficient accuracy, or are over refined, *(iv)* adaptively modify the discretization so that the characteristics of the flow are efficiently resolved to the desired accuracy. These steps can be accomplished in the framework of an automated adaptive environment. However, the appropriate choice of the building blocks of such an environment is crucial for the success of the methodology.

In our contribution, we try to analyze the various characteristics of the methods available in numerical mathematics, mesh generation and adaptation, and computer science, in order to select those that allow to obtain the most efficient solution of the simulation problem. This interdisciplinary approach allows the development of a homogeneous software environment, where the various tools closely cooperate sharing the same data structure and avoiding the creation of bottlenecks during the simulation process. The final goal of our research effort is to develope a reliable tool for the automated analysis of the complex unsteady flow fields that characterize helicopter rotors. This paper presents the parallel adaptive solver that represents the kernel of our future simulation tool. In the following, we will briefly discuss the main motivations behind the choice of the components of the proposed integrated methodology implemented.

The choice of an appropriate computer environment is a major concern, since the computational cost and the memory requirements of large–scale fluid dynamic simulations is prohibitive on classical scalar computers, while vector computers do not seem to keep up with the demands of today's CFD applications. The problem is exacerbated when unsteady simulations are attempted, since large and highly non–linear problems must be solved at each time step. Distributed memory parallel computers have recently been successfully employed for large–scale analysis of fluid flows [8][10]. These computers seem to offer the potential for satis-

*Copyright ©1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.
[†] Post–Doctoral Research Associate.
[‡] Research Engineer.
[§] Graduate Research Assistant.
[¶] Amos Eaton Professor of Computer Science.
[||] Assistant Professor, Mechanical Engineering, Aeronautical Engineering and Mechanics, Senior Member AIAA.
[**] Director, Scientific Computation Research Center, Johnson Chair of Engineering, Associate Fellow AIAA.

fying the demands of high performance as well as providing large memories.

Another distinguishing feature of the proposed methodology is the use of the finite element method for discretizing the governing Navier–Stokes/Euler equations. Historically, finite difference methods have been the traditional approach for numerically solving fluid flow problems. Moreover, finite difference algorithms are particularly well suited for developing parallel implementations on distributed memory computers, due to the regular nature of their data sets. Yet, the structured discretizations needed by finite difference techniques represent a major drawback when considering the automatic grid generation for complex geometries and the adaptive modification of the computational grid. Recognition of this inherent drawback has led to an increased interest in the application of finite volume and finite element methods to computational fluid dynamic problems. These methods, being able to use unstructured discretizations of the computational domain, can be coupled with automated adaptive meshing techniques that modify the grid based upon the local discretization error of the solution or an estimate of that error. The implemented finite element formulation is the Time–Discontinuous Galerkin Least–Squares method [18][19]. This method is very well suited for the incorporation in an automated adaptive environment. It has a firm mathematical foundation, with its stability, convergence and accuracy having been rigorously established. Moreover, it possesses the ability to naturally handle moving boundary problems by means of space–time deformed elements [23].

A key issue in the implementation of finite element solution techniques on distributed memory parallel computers is the partition of data among the processing nodes, given the irregular communication patterns that characterize unstructured meshes. Domain decomposition strategies are used for partitioning the computational domain in sub–domains, each sub–domain being assigned to a processing node. The sub–domains then exchange data with one another through the sub–domain boundaries. A number of algorithms have been developed for partitioning the computational domain while satisfying the load balance and low communication requirements, like Recursive Bisection (RB, together with Orthogonal RB and Moment RB) [2], which repeatedly splits the mesh into two sub–meshes, or the Spectral Recursive Bisection (SRB), which makes use of the properties of the Laplacian matrix of the mesh connectivity graph [15].

The irregular evolving behavior of adaptive unstructured meshes poses additional difficulties in a distributed memory parallel environment. In order to minimize the load imbalance created by the adaptation process, mesh redistribution techniques must be developed. Global techniques that repartition the mesh pose the fundamental problem that the cost of redistribution might be higher than the cost of performing the analysis with an unbalanced load. Hence, local procedures that redistribute the load migrating elements among the sub–domains seem to offer attractive advantages.

Another issue related to the implementation of adaptive finite element procedures on distributed memory parallel computers is the choice of the programming style. The two programming models actually available are: *data parallel languages*, such as High Performance Fortran (HPF), and *message passing protocols*. Data parallel languages provide a number of compiler directives to control the distribution of data to the memory of the processing nodes. On the other hand, message passing environments provide the programmer with a collection of routines for a detailed management of the inter–processor communications. In principle, both styles are well suited for the development of parallel finite element procedures on fixed discretizations. In practice, the data parallel paradigm has been the method of choice in recent publications [13][8][10]. However, when dealing with adaptive strategies and mesh modification techniques like mesh refinement and derefinement, retriangulation and migration, the software development is more easily accomplished in a message passing programming model. With the idea of developing a uniform software environment, we decided to make use of portable message passing protocols in each stage of the analysis. The implementation has been carried out on a IBM SP–1 system using the message passing library Chamaleon [7], which is a light weight, highly portable library.

All the software tools that make up the implemented integrated adaptive environment are realized using the C language and share a novel mesh data structure [1], designed on the philosophy of object oriented programming.

Another distinctive characteristic of the present methodology lies in the specification of the physical attribute information required to support the analysis (as for example, the specification of boundary conditions). This information is tied to the geometric model definition and it is defined in a general tensor order form [20]. This is in contrast with the common procedure of defining the physical attribute information directly in terms of the discrete model. This approach offers distinct advantages in an automated analysis environment like the one described here.

The following sections describe the major issues related to this work. The finite element formulation implemented, as well as the iterative method used to solve large scale non–symmetric linear systems on distributed memory parallel computers, a simple error sensing technique for driving the adaptation procedure, the mesh partitioning procedures, element migration and load balancing techniques developed are given. A section is devoted also to the parallel mesh refinement and derefinement. In the last section some preliminary numerical test problems involving the adaptive analysis of compressible flows are presented.

Finite Element Formulation

This work is concerned with the transient response of a compressible flow. The initial/boundary value problem can be expressed by means of the Navier–Stokes/Euler equations as

$$\operatorname{div} \boldsymbol{F} + \mathcal{F} = 0 \tag{1}$$

plus well posed initial and boundary conditions. In equation (1), div is the divergence operator in $n_{sd}+1$ space–time dimensions, being n_{sd} the number of space dimensions, while $\boldsymbol{F} = (\boldsymbol{U}, \boldsymbol{F}_i - \boldsymbol{F}_i^d)$, where $\boldsymbol{U} = \rho(1, u_1, u_2, u_3, e)$ are the conservative variables, $\boldsymbol{F}_i = \rho\, u_i(1, u_1, u_2, u_3, e) + p(0, \delta_{1i}, \delta_{2i}, \delta_{3i}, u_i)$ is the Euler flux, $\boldsymbol{F}_i^d = (0, \tau_{1i}, \tau_{2i}, \tau_{3i}, \tau_{ij} u_j) + (0,0,0,0,-q_i)$ is the diffusive flux, and $\mathcal{F} = \rho(0, b_1, b_2, b_3, b_i u_i + r)$ is the source vector. In the previous expressions, ρ is the density, $\boldsymbol{u} = (u_1, u_2, u_3)^T$ is the velocity vector, e is the total energy, p is the pressure, δ_{ij} is the Kronecker delta, $\boldsymbol{\tau} = [\tau_{ij}]$ is the viscous stress tensor, $\boldsymbol{q} = (q_1, q_2, q_3)^T$ is the heat–flux vector, $\boldsymbol{b} = (b_1, b_2, b_3)^T$ is the body force vector per unit mass and r is the heat supply per unit mass.

The Time–Discontinuous Galerkin Least–Squares (TDG/LS) finite element method is used in this effort [18][19]. This method is developed starting from the symmetric form of the Navier–Stokes/Euler equations expressed in terms of entropy variables and it is based upon the simultaneous discretization of the space–time computational domain. A least–squares operator and a discontinuity capturing term are added to the formulation for improving stability without sacrificing accuracy. The TDG/LS finite element method takes the form

$$\begin{aligned}
&\int_{Q_n} \left(-\boldsymbol{W}^h_{,i} \cdot \boldsymbol{F}(\boldsymbol{V}^h) + \boldsymbol{W}^h \cdot \tilde{\boldsymbol{C}} \boldsymbol{V}^h\right) \mathrm{d}Q \\
&+ \int_{\Omega(t_{n+1})} \boldsymbol{W}^{h^-} \cdot \boldsymbol{U}(\boldsymbol{V}^{h^-})\, \mathrm{d}\Omega - \int_{\Omega(t_n)} \boldsymbol{W}^{h^+} \cdot \boldsymbol{U}(\boldsymbol{V}^{h^-})\, \mathrm{d}\Omega \\
&+ \int_{P_n} \boldsymbol{W}^h \boldsymbol{F} \cdot \mathrm{d}\boldsymbol{P} \\
&+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \left(\mathcal{L}\boldsymbol{W}^h\right) \cdot \tau \left(\mathcal{L}\boldsymbol{V}^h\right) \mathrm{d}Q \\
&+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \nu^h \hat{\nabla}_\xi \boldsymbol{W}^h \cdot \operatorname{diag}[\tilde{\boldsymbol{A}}_0] \hat{\nabla}_\xi \boldsymbol{V}^h\, \mathrm{d}Q = 0.
\end{aligned} \tag{2}$$

Integration is performed over the space–time slab Q_n, the evolving spatial domain $\Omega(t)$ of boundary $\Gamma(t)$ and the surface P_n described by $\Gamma(t)$ as it traverses the time interval $I_n =]t_n, t_{n+1}[$. $\boldsymbol{W}^h$ and $\boldsymbol{V}^h$ are suitable spaces for test and trial functions, while τ and ν^h are appropriate stabilization parameters. Refer to [18] for additional details on the TDG/LS finite element formulation.

We have implemented two different three dimensional space–time finite elements. The first is based on a constant in time interpolation, and, having low order of time accuracy but good stability properties, it is well suited for solving steady problems. The second makes use of linear–in–time basis functions and, exhibiting a higher order temporal accuracy, it is well suited for addressing unsteady problems. As pointed out by Tezduyar et al. [23], this latter element naturally allows direct treatment of moving boundary problems. In fact the motion of boundaries or interfaces is automatically included in the Jacobian that relates the physical space–time coordinates with the local finite element space–time coordinates. The only difficulty in this case arises in three dimensional applications when one has to compute the space–time boundary integral. This term appears as a consequence of the integration by parts performed in the four dimensional domain, and represents the flux that traverses the three dimensional space–time boundary. This problem can be solved using the concept of differential forms and the General Stokes' Theorem (see for example Corwin and Szczarba [4]).

Discretization of the weak form implied by the TDG/LS method leads to a non-linear discrete problem, which is solved iteratively using a quasi–Newton approach. At each Newton iteration, a non–symmetric linear system of equations is solved using the GMRES algorithm [16].

Parallel Implementation

The message passing paradigm is employed in the implementation of the finite element method. The domain decomposition is used for mapping the finite element data to the processors in order to efficiently exploit data locality. Two main processing phases naturally emerge in the finite element method: the *form phase*, where the local finite element arrays at the sub–domain level are generated, and the *solve phase*, where the global problem is solved. Implementation of the form phase is straightforward, in the sense that it can be performed in parallel with no communication among the processing nodes.

The solve phase is realized in this work by means of the preconditioned matrix–free GMRES algorithm [8]. This algorithm approximates the matrix–vector products with a finite difference stencil with the advantage of avoiding the storage of the tangent matrix, thus realizing a substantial saving of computer memory at the cost of additional on–processor computations. Preconditioning is achieved by means of a nodal block–diagonal scaling transformation. Given the non–symmetric linear system

$$\boldsymbol{T} \cdot \boldsymbol{p} = -\boldsymbol{R}, \tag{3}$$

the GMRES algorithm attempts to find the approximate solution $\boldsymbol{p}_0 + \boldsymbol{z}$ to (3), $\boldsymbol{z}$ being in the Krylov space $\mathcal{K} = (\boldsymbol{r}_0, \boldsymbol{T} \cdot \boldsymbol{r}_0, \ldots, \boldsymbol{T}^{k-1} \cdot \boldsymbol{r}_0)$ and $\boldsymbol{r}_0 = -\boldsymbol{R} - \boldsymbol{T} \cdot \boldsymbol{p}_0$. $\boldsymbol{z}$ is the solution of the minimization problem $\min_{\boldsymbol{z} \in \mathcal{K}} \| -\boldsymbol{R} - \boldsymbol{T} \cdot (\boldsymbol{p}_0 + \boldsymbol{z}) \|$, which is solved by means of the QR algorithm. The GMRES algorithm obtains an orthonormal basis of $\mathcal{K}$ by means of a Gram–Schmidt procedure, that involves matrix–vector multiplications and dot products. These operations represent the computer intensive part of the algorithm.

The matrix–vector multiplications are realized in parallel and necessitate the exchange of data through the inter–processor boundaries. In the matrix–free version of the algorithm, matrix–vector multiplication of the form $\boldsymbol{T}(\boldsymbol{v}) \cdot \boldsymbol{u}$ are approximated by means of a finite difference stencil as

$$\boldsymbol{T}(\boldsymbol{v}) \cdot \boldsymbol{u} = \frac{\boldsymbol{R}(\boldsymbol{v} + \varepsilon \boldsymbol{u}) - \boldsymbol{R}(\boldsymbol{v})}{\varepsilon}, \tag{4}$$

where ε is a perturbation parameter which is computed minimizing the truncation error, which results from truncating the Taylor expansion, and the cancellation error, which is a consequence of operating in finite precision arithmetic. In order to overlap communication and computation for efficiency reasons, these operations are realized following a four step procedure on each processing node: *(i)* non–blocking send data relative to the inter-processor boundaries to each neighboring processor, *(ii)* perform computations involving only data relative to nodes that lie within the internal volume of the partition, *(iii)* blocking receive data relative to the inter–processor boundaries from all the neighbors, *(iv)* perform computations involving only data relative to nodes lying on the inter–processor boundaries.

For the implementation of the dot product operations, nodes that lie on the inter–processor boundaries are randomly split, so that two partitions that share an internal boundary are assigned only a subset of the nodes of that internal boundary. Each processing node performs then the dot product involving nodes contained in its internal volume and its subset of nodes on the partition boundaries. Global sum of the local results at the processor level yields the global dot product result.

The minimization problem can be written in terms of an upper Hessenberg matrix, whose entries are essentially the results of the dot products performed during the orthogonalization procedure. At the end of the Gram–Schmidt procedure, each processing node has then complete knowledge of the upper Hessenberg matrix and it is therefore able to perform the solution of the minimization problem independently with no communication. It should be remarked that the size of the Hessenberg matrix is the size of the Krylov space employed, typical values for the applications here considered being around 5–30. The computer intensive SAXPY operations ($\boldsymbol{y} = \boldsymbol{y} + a \cdot \boldsymbol{x}$) needed in order to update the solution of the linear system are consequently performed in parallel with no communication. Once convergence is achieved in the iterative linear solver, each processing node has complete knowledge of the incremental solution at the current Newton step, and it is therefore able to update the current state completely independently, without any inter–processor communication.

Error Sensing

The discontinuous nature of the solution of the Navier–Stokes/Euler equations makes the development of rigorous a posteriori error estimates difficult. Some progress towards this result has been recently reported [9]. Note that this result was accomplished within the framework of the TDG/LS finite element method. What has been traditionally done to overcome the lack of sharp error estimators, is to employ error indicators. Although error indicators do not measure the real discretization error, they typically "indicate" regions of the computational domain where errors are high. Possible implementations of error indicators are based on the monitoring of the change of certain flow quantities or the use of appropriately modified error estimates for smooth problems. These approaches do suffer from the fact that the error indicated at discontinuities does not approach zero as the mesh is refined and can have difficulty in detecting weak flow features in the presence of strong flow features. Therefore, the successful use of error indicators in an adaptive solution system does require the knowledgeable selection of the indicator used and the initial mesh analyzed. This of course does partly detract from reliability of the automated system and it is an important area requiring further development.

In this work we have implemented an error indicator [12] for elements making use of linear basis functions which takes the basic form

$$e_i = \frac{h^2 \mid \text{Second Derivative of } \Psi \mid}{h \mid \text{First Derivative of } \Psi \mid + \varepsilon \mid \text{Mean Value of } \Psi \mid}, \quad (5)$$

where e_i is the error indicated at node i, h is a mesh size parameter, Ψ is the solution variable being monitored, ε is a tuning parameter. The second derivative of Ψ is computed using a variational recovery technique.

Equation (5) is used for estimating the error at the finite element nodes. Different key variables are used for detecting different features of the fluid flow. In particular, the Mach number is employed for identifying shocks and stagnation points, while entropy is used for targeting for refinement vortical regions. The edge values of the error indicator are computed by averaging the corresponding two nodal values. These edgewise error indicator values are then used for driving the mesh adaptation procedure. Appropriate thresholds are supplied for the error values, so that the edge is refined if the error is higher than the maximum threshold, while the edge is collapsed if the error is the less than the minimum threshold. Parallel implementation of equation (5) is straightforward.

Data Structures, Mesh Partitioning, Migration and Load Balancing

The data structures used in a parallel adaptive finite element solver must provide fast query and update of partition boundary information. Queries commonly needed include: *(i)* adjacency information for entities located on more than one partition, *(ii)* number and list of adjacent processors given an entity type adjacency, *(iii)* list of entities on a partition boundary given an adjacent processor and entity type, *(iv)* lists of scatter and gather maps of nodes on the partition boundary.

Besides the queries, update procedures must be available to the refinement/coarsening and element migration/load balancing components of the parallel finite element solver. Efficient computation requires updated entities be inserted into or deleted from the partition boundary within constant time, or at most time proportional to the number of adjacent processors.

To implement these fast query and update routines, a topological *entity hierarchy* data structure [1], which provides a two-way link between the mesh entities of consecutive order, i.e. regions, faces, edges and vertices, is used. From this hierarchy, any entity adjacency relationship can be derived by local traversals. The entities on the partition boundary are augmented with *links* which point to the location of the corresponding entity on the neighboring processor. These inter-processor links are then maintained in a doubly linked list with a processor *id* node as the header. From these structures, partition boundary entity insertion–deletion can be made in constant time. The entities neighboring a processor can also be traversed by starting at the header node given by the processor and following the doubly link list. Figure (1) summarizes the data structures used.

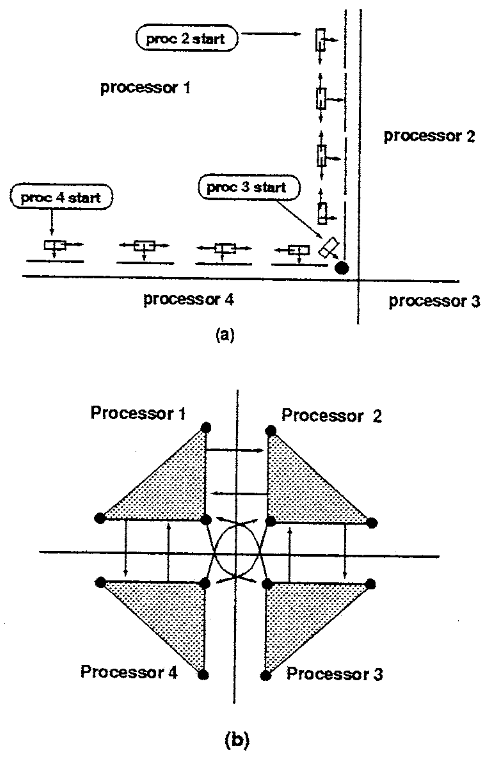

Figure 1: Data structures illustrated in 2D: *(a)* doubly linked list of inter–processor links, *(b)* inter–processor links pointing to locations of duplicate entities.

Each partition boundary entity can have attached to it either the *complete* or the *minimal* set of inter-processor links. In the complete set, all the boundary entities store the location of the corresponding duplicate entity. Since the lower entities inherit the higher level entity adjacency, it is possible to eliminate the inter-processor links whose adjacency can be derived from higher level entities. This minimal link representation has the advantage of reducing the storage needed to maintain the partition boundary entities. However, the minimal representation has the disadvantage of complicating the link update procedures when element migrations are performed. Therefore, a switching mechanism is used to allow both representations to be used disjointly.

The initial mesh is partitioned using orthogonal RB or its variant, moment of inertia RB. The whole mesh is first loaded into one processor and then recursively split in half and sent to other processors in parallel. The splitting can be done by either sorting or the faster linear time median finding algorithm. In this way, the initial partitioning by parallel RB takes $O(n)$ time where n is the number of elements. Since very little of the computation is performed on the initial mesh, concern for optimizing the initial partitions is not critical. What is more critical is what happens to the partitions as the calculation proceed.

In an adaptive parallel distributed memory environment, procedures are needed to migrate elements among the processors for the purpose of redistributing the mesh in order to achieve load balance. The migration routines are implemented in three stages: *(i)* the element mesh and its attribute data is packed into messages and sent, *(ii)* packed elements are received and unpacked, *(iii)* the inter-processor links are updated. The procedures provided allow each processor to send and receive multiple migrations of the elements.

Our load balancing scheme iteratively migrates elements from heavily loaded to less loaded processors. To decide which processors should be involved in load migration, we use a heuristic based on the Leiss and Reddy approach [11] of letting each processor request load from a heavily loaded neighbor. Leiss and Reddy, as well as heuristics based on similar load request process [24][14], calculate the amount of load that will be transferred by a local averaging of neighbor processors. The treatment of hierarchic load request as a tree [14] enables processor pairing by edge-coloring the trees formed. The edge-coloring is performed efficiently by a parallel scan operation on the tree. The pairs of processors exchange load to even out the load imbalance.

Let T denote a tree and $|T|$ denote the number of nodes in the tree. Scan operations on trees can be performed efficiently with complexity $O(log|T|)$. Details and implementation on distributed memory machines can be found in [14][22]. The total load on the whole tree can then be found and the load that should be migrated to balance the tree can be calculated. Since the whole tree is balanced rather than immediate processor neighborhood as in [11], the convergence of the iterative load balancing scheme can be expected to be faster.

Let $load\_mig_i$ denote amount of load that will be migrated into or out of a tree node i which represents a processor. Let also T_i denote the subtree with node i as the root of the subtree and $load(T_i)$ be the sum of loads of nodes in this subtree. The amount of load migration is then calculated as

$$load\_mig_i = load(T_i) - avg\_load(T) * |T_i|$$

with $avg\_load(T) = load(T)/|T|$ representing the average load on the tree when balanced. Given $load\_mig_i$, the direction of load migrations can be found as

$$load\_mig_i \quad \begin{array}{ll} = 0, & \text{do nothing with parent,} \\ < 0, & \text{get load from parent,} \\ > 0, & \text{send load to parent.} \end{array}$$

Having calculated the directions of load migration, the elements on the partition boundary are migrated slice by slice until $load\_mig_i$ of them has been transferred. Each slice of elements forms a peeling of the partition boundary and are selected by choosing elements which touch the boundary by any one of their vertices.

Figure (2) *(a)* gives an example of the tree formed when processors request load from heavily loaded neighboring processors. Since the total load on the tree is 14, the load migrations shown in *(b)* is needed to balance the nodes to average load of 2 per processor. All the calculations needed to compute the above values are performed by the scan operation on the tree.

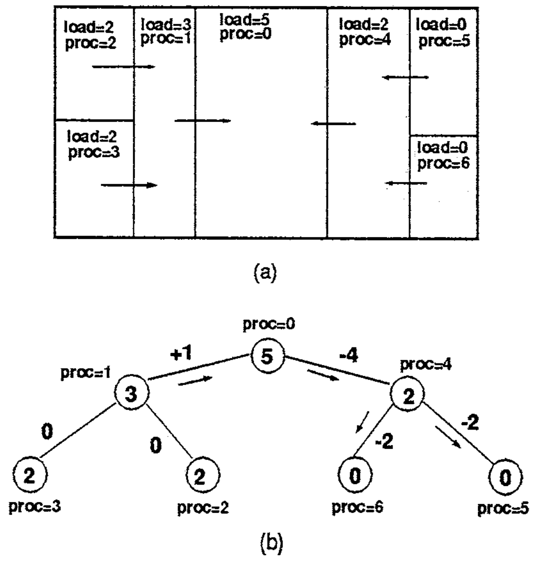

Figure 2: *(a)* unbalanced processors and load requests. *(b)* hierarchical load tree and the amounts of load to be migrated.

The procedure presented above enables a scan directed load balancing technique to be applied within a dynamically changing and irregularly connected arrangement of processors. This extends and improves some of the earlier applications of scan directed load balancing [3] which were only used within the context of static and uniformly connected processors in 1 or 2D.

Parallel Mesh Refinement and Derefinement

The mesh level adaptive scheme combines derefinement, refinement and triangulation optimization using local retriangulations [5]. The derefinement step is based on an edge collapsing technique. A mesh edge is collapsed by deleting all mesh regions connected to one end vertex and connecting the faces of the resulting polyhedral cavity to the other end vertex (see Figure (3)). Edge collapsing is not always possible for two reasons: *(i)* it may lead to local topological invalidity of the triangulation and *(ii)* it can lead to the creation of invalid elements. It does not require storage of any history information and it is therefore not dependent on the refinement procedure.

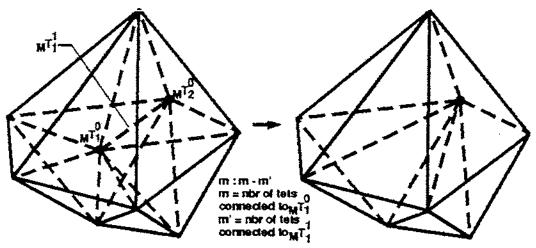
MT_1^1
MT_2^0
MT_1^0
m : m - m'
m = nbr of tets connected to MT_1^0
m' = nbr of tets connected to MT_1^1

Figure 3: Edge collapsing in three dimensions.

Refinement makes use of subdivision patterns. All possible subdivision patterns have been considered and implemented to allow for speed and annihilate possible over-refinement. If the bounding face of a mesh region to be subdivided with two and only two marked edges is already triangulated, the template for that region must be able to match the face triangulation. Since there are a priori two ways to triangulate a face with two marked edges, any pattern which has N faces with two and only two marked edges needs 2^N templates. The complete set of subdivision patterns is shown in Figure (4).

Triangulation optimization is necessary to prevent triangulation quality degradation. It is particularly important when the snapping of refinement vertices on curved model boundaries can potentially create invalid or poorly shaped elements. The idea is to iteratively consider the local retriangulation of simple and well defined polyhedra. The optimization procedure builds upon the coupling of edge removal [6] and its dual, multi-face removal. Both techniques can be seen as tools to retriangulate simple polyhedra. Edge removal consists of deleting all mesh regions connected to a mesh

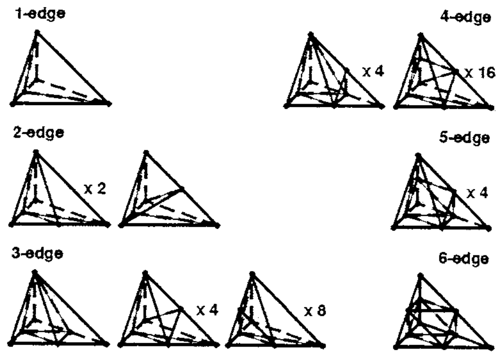

Figure 4: Subdivision patterns in three dimensions.

edge and retriangulating the space in the resulting polyhedral cavity without recreating the mesh edge to be removed (see Figure (5)). Multi-face removal is the reverse process of edge removal. An example of multi-face removal is shown in Figure (6). At this point, it is worthwhile to note that edge collapsing is also a form of local retriangulation that removes a mesh vertex. Then, derefinement and local retriangulation can be treated in a similar way.

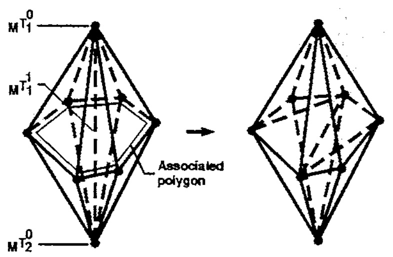

Figure 5: Edge removal.

Since refinement uses templates, its parallelization presents no difficulty. First, mesh faces on the partition boundary are triangulated. Triangulation compatibility is implicitly guaranteed by the fact that duplicate faces have the same orientation. Face level interprocessor links are set up for the child faces of the mesh faces on the partition boundary. Then, mesh regions are triangulated without communication involved. Any mesh edge carrying minimal inter-processor links transfers link information to its two child edges.

The challenge resides in the efficient parallelization of the derefinement and triangulation optimization steps.

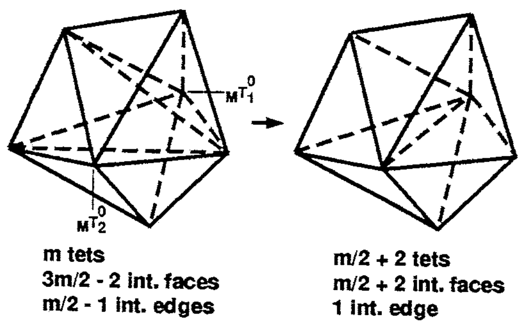

Figure 6: Multi–face removal.

An efficient way to retriangulate polyhedra in parallel can be decomposed in three steps: *(i)* retriangulate polyhedra which are fully interior to the partition, *(ii)* shift the partition boundary using element migration techniques, and *(iii)* retriangulate those polyhedra that are now fully accessible due to the shift. Since it is likely that several processors request the same off-processor region, a decision has to be made concerning which processors has priority over the others. The processor with lowest identification number is the only one which can have its request granted.

In the case of triangulation optimization which is iterative by nature, shifting the partition boundary always in the same direction will quickly create load imbalance. Therefore, after each iteration, a load balancing step is applied so that the next iteration is not penalized by the shift.

The parallelized snapping procedure begins by computing the snapped location of all refinement vertices classified on model edge or face. This step involves no communication. Each refinement vertex is then examined in turn for possible invalidity of connected elements. If an element is found invalid, local retriangulation using edge removal and multi-face removal is attempted locally to that element. If local retriangulation cannot be applied because the retriangulation polyhedron lies across more than one processor, a request will be made for migration of the missing parts of the polyhedron. After all requests have been processed by shifting the partition boundary, the vertex will eventually be reprocessed.

Numerical Examples

This section presents some numerical and performance characteristics of our parallel adaptive methodology with the help of a few preliminary example problems. The scope here is of showing the potential of the proposed approach, rather than solving complex fluid dynamic problems. With the maturation of this technique, we hope to be able in a near future to address more challenging steady and unsteady aerodynamic problems.

Oblique Shock Problem

This steady problem is characterized by a Mach two flow over a wedge at an angle of 10°, resulting in an oblique shock emanating from the leading edge of the wedge at an angle of 29.3°. All flow variables are prescribed at the inflow, while no boundary conditions are prescribed at the outflow, the flow being supersonic. This two dimensional problem was solved in a three dimensional domain, using linear tetrahedral elements. The slip condition is prescribed on the wedge and on the two symmetry planes. A four Gauss point integration rule was used on each element, together with a CFL number of 10 and a local time stepping strategy to reach convergence.

The initial uniform mesh and the refined meshes along the oblique shock are shown in Figure (7), in clock–wise order from the top left portion of the picture. The sub–domains used for the computation are also shown in the same picture. The final solution was obtained by means of three adaptive steps characterized by one single enrichment level. The error indicator was computed using density and Mach number as key variables. The first partition was obtained on the initial mesh with Moment RB, giving origin to three element groups of 384 elements, one of 385, and two of 768. The other partitions are obtained as a result of the mesh adaptation and load balancing procedures. The second mesh has four partitions of 1462 elements, one of 1463 and one of 1466. The third mesh has three partitions with 5044 elements and three with 5045, while the fourth and last mesh has three partitions of 18843 elements and three of 18844.

Onera M6 Wing at Transonic Mach Number

The Onera M6 wing is a classical example for assessing the performance of three dimensional compressible flow calculations. Extensive experimental results are reported in [17], for a variety of Mach numbers and angles of attack. The wing has an aspect ratio of 3.8, a taper ratio of 0.562, a leading edge sweep angle of 30°, and a symmetric Onera D airfoil section. A steady flow calculation has been performed at a Mach number of 0.84, with an angle of attack of 3.06°.

The initial mesh for this problem was obtained with the Finite Octree automatic mesh generator [21], and it is shown in Figure (8), together with its partition in

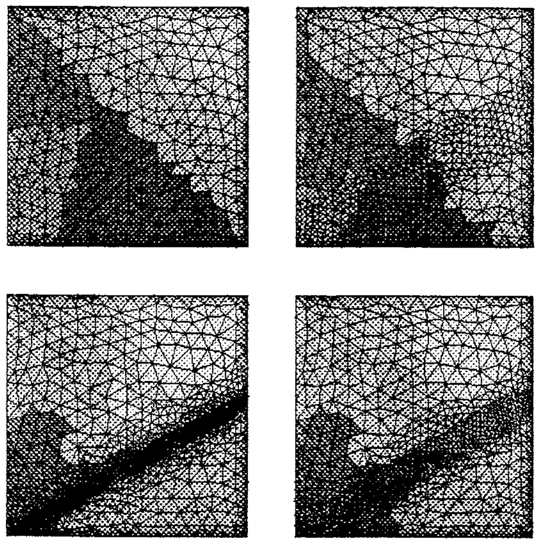

Figure 7: Oblique shock problem: initial uniform mesh and adapted meshes.

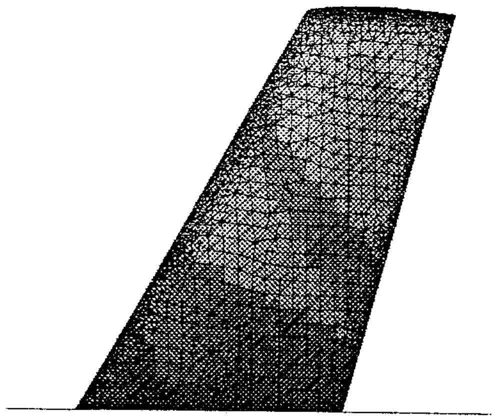

Figure 8: Onera M6 wing problem: initial mesh with partitions in eight sub–domains.

eight sub–domains. A preliminary calculation involving one single adaptive step was performed, and a zoom at the upper surface tip of the refined wing mesh is shown in Figure (9). The error indication was performed using Mach number and density as key variables. The adaptive procedure is clearly starting to refine the mesh at the tip and along one of the two inboard shocks. Clearly more adaptive iterations are needed for accurately resolving the complex front patterns on the upper surface of the wing. Performing a fully converged analysis involving more adaptive steps will represent our next research effort, together with timings of the simulation procedure for investigating its scalability.

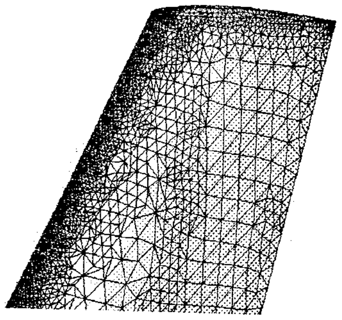

Figure 9: Onera M6 wing problem: zoom at the tip of the upper surface of the refined mesh after one adaptive step.

Concluding Remarks

This paper has presented a parallel automated adaptive finite element method for the simulation of compressible flows. The implementation has been carried out in the context of a stabilized space–time finite element formulation for compressible flows which readily allows a natural way of dealing with unsteady moving boundary problems by means of space–time deformed finite elements. This solver represents the kernel of an integrated tool for the analysis of the flow field around helicopter rotors that we are currently developing.

We have discussed possible solutions to the challenging problems posed by the steady and unsteady simulations of compressible flows. Our starting point of view implies the fact that adaptivity is the key to the efficient and accurate analysis of complex physical phenomena. In particular our efforts to date have been devoted to the development of the software tools that compose the proposed methodology in a distributed memory parallel environment. A domain decomposition based approach has been used for achieving parallelism, while the message passing paradigm has been used for simplifying the interaction among the various components: finite element flow solver, mesh partitioner, mesh adaptation and load balance procedures.

We have discussed some preliminary results relative to a few example problems involving the adaptive analysis of steady compressible flows, showing some of the basic features and numerical characteristics of our approach.

Our future developments will include more testing and timing of the present procedure, the analysis of unsteady problems, as well as the coupling of the present CFD solver kernel with an appropriate CSD solver for addressing the typical aeroelastic interaction problems that characterize the dynamic response of rotors.

Acknowledgements

The authors would like to acknowledge the Army Research Office for funding this research through the ARO Rotorcraft Technology Center at Rensselaer Polytechnic Institute (DAAH04-93-G-0003, G. Anderson project monitor).

References

[1] BEALL, M.W. and SHEPHARD, M.S., Mesh Data Structures for Advanced Finite Element Applications, Scientific Computation Research Center, RPI, Troy, NY, in preparation for submission.

[2] BERGER, M.J. and BOKHARI, S.H., A Partitioning Strategy for Nonuniform Problems on Multiprocessors, *IEEE Trans. Comp.*, C-36(5):570-580, 1987.

[3] BIAGIONI, E.S. and PRINS, J.F., Scan Directed Load Balancing For Highly-Parallel Mesh Connected Computers, In I. P. Mehrotra, J. Saltz, and R. G. Voigt, editors, *Unstructured Scientific Computation on Scalable Multiprocessors*, pages 371-394. 1992.

[4] CORWIN, L.J. and SZCZARBA, R.H., *Calculus in Vector Spaces*, Marcel Dekker Inc., 1979.

[5] DE COUGNY, H.L. and SHEPHARD, M.S., Refinement, Derefinement and Local Retriangulations in Three-Dimensions, Scientific Computation Research Center, RPI, Troy, NY, in preparation for submission.

[6] DE L'ISLE, B.E. and GEORGE, P.L., Optimization of Tetrahedral Meshes, INRIA, Domaine de Voluceau, Rocquencourt BP 105 Le Chesnay France, 1993.

[7] GROPP, W. and SMITH, B., Users Manual for the Chamaleon Parallel Programming Tools, Argonne National Laboratory, ANL-93/23, June 1993.

[8] JOHAN, Z., Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics, Ph.D. Thesis, Stanford University, July 1992.

[9] JOHNSON, C., Finite Element Methods for Flow Problems, Tech. Rep., Neuilly Sur Seine, France, 1992, AGARD rep. 787, ref. 1.

[10] KENNEDY, J.G., BEHR, M., KALRO, V. and TEZDUYAR, T.E., Implementation of Implicit Finite Element Methods for Incompressible Flows on the CM-5, Army High-Performance Computing Research Center, University of Minnesota, Rep. 94-017, April 1994.

[11] LEISS, E. and REDDY, H., Distributed Load Balancing: Design and Performance Analysis, *W. M. Keck Research Computation Laboratory*, 5:205-270, 1989.

[12] LOHENER, R., An Adaptive Finite Element Scheme for Transient Problems in CFD, *Comp. Meth. Appl. Mech. Eng.*, 61:323-338, 1987.

[13] MATHUR, K.K. and JOHNSON, S.L., The Finite Element Method on a Data Parallel Computing System, *Int. J. High Speed Comp.*, 1:29-44, 1989.

[14] ÖZTURAN, C., DECOUGNY, H., SHEPHARD, M.S., and FLAHERTY, J.E., Parallel Adaptive Mesh Refinement and Redistribution on Distributed Memory Computers, *Comp. Meth. Appl. Mech. Eng.*, to appear, Symposium on Parallel Finite Element Computations, Oct. 1993, Univ. Minnesota Supercomputer Institute.

[15] POTHEN, A., SIMON, H.D. and LIOU, K.P., Partitioning Sparse Matrices with Eigenvectors of Graphs, *SIAM J. Matrix Anal. Appl.*, 11:430-452, 1990.

[16] SAAD, Y. and SCHULTZ, M., GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sc. Stat. Comp.*, 7:856-869, 1986.

[17] SCHMITT, V. and CHARPIN, F., Pressure Distributions on the Onera M6 Wing at Transonic Mach Numbers, AGARD R-702, pag. B1-1, B1-44, 1982.

[18] SHAKIB, F., Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations, Ph.D. Thesis, Dept. of Mech. Eng., Stanford University, CA, 1988.

[19] SHAKIB, F., HUGHES, T.J.R. and JOHAN, Z., A New Finite Element Formulation for Computational Fluid Dynamics: X. The Compressible Euler and Navier Stokes Equations, *Comp. Meth. Appl. Mech. Eng.*, 89:141-219, 1991.

[20] SHEPHARD, M.S., The Specification of Physical Attribute Information for Engineering Analysis, *Eng. with Comp.*, 4:145–155, 1988.

[21] SHEPHARD, M.S. and GEORGES M.K., Automatic Three–Dimensional Mesh Generation by the Finite Octree Technique, *Int. J. Num. Meth. Eng.*, 32:709–749, 1991.

[22] SZYMANSKI, B. K. and MINCZUK A., A Representation of a Distribution Power Network Graph, *Archiwum Elektrotechniki*, 27(2):367–380, 1978.

[23] TEZDUYAR, T.E., BEHR, M., MITTAL, S. and LIOU J., A New Strategy for Finite Element Computations Involving Moving Boundaries and Interfaces – The Deforming–Spatial–Domain/Space–Time Procedure: I. The Concept and the Preliminary Tests, *Comp. Meth. Appl. Mech. Eng.*, 94:353–371, 1992.

[24] WHEAT, S.R., DEVINE, K.D., and MACCABE, A.B., Experience with Automatic, Dynamic Load Balancing and Adaptive Finite Element Computation, In H. El-Rewini and B. D. Shriver, editors, *Proc. of 27th Hawaii International Conference on System Sciences*, volume 2, pages 463–472. IEEE Computer Society Press, January 1994.