VECTORIZABLE ALGORITHMS FOR ADAPTIVE SCHEMES FOR RAPID ANALYSIS OF SSME FLOWS

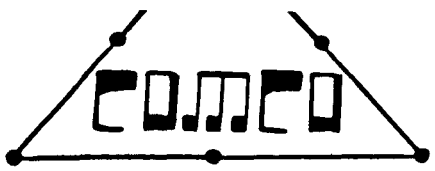
FINAL REPORT: P.O. H-85080-B

to the George C. Marshall Space Flight Center
National Aeronautics and Space Administration

TR-87-03

by
J. Tinsley Oden

April, 1987



THE COMPUTATIONAL MECHANICS COMPANY, INC.

Computational Mechanics Co.,Inc.
4804 Avenue H
Austin, Texas 78751

TABLE OF CONTENTS

I. INTRODUCTION

This report describes an initial study into vectorizable algorithms for use in adaptive schemes for various types of boundary-value problems. The project described here focused on two key aspects of adaptive computational methods which are crucial in the use of such methods for complex flow simulations such as those in the SSME:

1) The adaptive scheme itself, i.e., the significant data-management requirements for schemes on which grid-point labels and cell numbers dynamically change during the solution process, and

2) The applicability of element-by-element matrix computations in an vectorizable format for rapid calculations in adaptive mesh procedures.

In the first area, we explore versions of a rapid, h-type scheme for mesh refinement on unstructured meshes in two-dimensional domains. Schemes such as this were originally proposed by the authors in a series of papers on adaptive finite element methods (see, e.g., [1-5]). The present version of the scheme is described in Chapter 2 of this report, and exhibits the following features:

a) The scheme is designed for unstructured meshes of quadrilateral elements (although an experimental code employing triangular elements is operational and under study.)

b) The scheme does **not** employ the notion of hierarchial families of shape functions (and thus is distinctly different from and many times faster than the adaptive schemes of Babuska, Zienkiewicz, Gago, and others; see [6]).

c) The scheme does **not** rely on a tree structure (and hence is different than the adaptive schemes of Rheinboldt and Mesztenyi [7] and Ewing [8]).

d) The scheme is portable to other program structures, provided they do not require a fully structured mesh; and

e) The scheme can be extended to three-dimensional domains.

With regard to the issue of fast, element-by-element matrix calculations, we exploit the new element-by-element algorithms which have gained much popularity in recent times. In particular, the Hayes-Devloo algorithm for fast matrix-vector multiplication [9] emerged as our choice for the basis of a rapid processing of equations in an adaptive code. A discussion of the principal ideas in such schemes and in how they are used to produce vectorizable matrix assembly procedures is given in Chapter 3 of this report.

It should also be mentioned that, while this report deals with elliptic cases, all of the methods described here have been successfully applied to parabolic problems and hyperbolic systems. The adaptive scheme itself is independent of the equation type under consideration, and the matrix schemes are applicable to any situation calling for fast numerical linear algebra procedures.

II. A FAST, ELEMENT BASED, ADAPTIVE REFINEMENT STRATEGY

2.1 Error Indicators. All adaptive finite element schemes employ some local measure of solution quality as a basis for changing the local structure of the approximation (the mesh size, locatio of grid points, degree of the polynomial shape functions, etc.). In many cases, the classical interpolation estimates of finite element theory (e.g., [10,11]) have provided a convenient basis for computing local error indicators. For example, let

u = the exact solution of a boundary-value problem in IR^n

u^h = a finite element approximation of u

K = a typical finite element in the mesh supporting u^h

h_K = the diameter of K (the local mesh size)

ρ_K = the diameter of the largest sphere (disk) that can be inscribed in K

k = the degree of the complete polynomial appearing in the element shape functions

C = a constant, independent of h_K, r_K, and u

p,q = real numbers; $1 \le p, q \le \infty$

n = the dimension of the domain of u (generally $n = 1,2,$ or 3)

$|\phi|_{m,q,K}$ = the Sobolev semi-norm of order m,q of a function ϕ defined on K

In one dimension,

$$|\phi|_{m,q,K} = \left\{ \int_K \left[|\phi|^q + |\phi'|^q + |\phi''|^q + \cdots + |d^m\phi/dx^m|^q \right] dx \right\}^{1/p}$$

Then the general, local, finite element interpolation error estimate is of the form,

$$| u - u^h |_{m,q,K} \leq C_K h^{n/q - n/p} \frac{h_K^{k+1}}{\rho_K^m} | u |_{k+1,p,K} \qquad (2.1)$$

where $\bar{u}^h$ is an *interpolant* of u, i.e., $\bar{u}^h$ and various of its derivatives coincide with u and its derivatives at the nodes.

If quasi-uniform refinements are used (which is the case with the adaptive process to be described), we have

$$\frac{h_k}{\rho_K} \leq \sigma_0 = \text{const.}$$

One can arrive at a local error estimate by replacing $u - \bar{u}^h$ with $e^h = u - u^h$, u by u^h on the right-hand side, and assuming quasi-uniform refinements:

$$| e^h |_{m,q,K} \leq C h_K^{\mu} | u^h |_{k+1,p,K}$$

$$\mu = \frac{n}{q} - \frac{n}{p} + k + 1 - m \qquad (2.2)$$

For a mesh with E(h) elements, we can define as the error indicator θ_e for elements e(=K),

$$\theta_e = C h_e^{\mu} | u^h |_{k+1,p,e} \qquad (2.3)$$

For example, if

$$m = 0, \quad q = p = 2, \quad k = 1$$

we have

$$\theta_e = C h_e^2 | u^h |_{2,2,e}$$

with $h_e = h_k$ and $|u^h|_{2,2,e} = \{\int_K (u^h_{,11}|^2 + |u^h_{,12}|^2 + |u^h_{,22}|^2 \, dx_1 dx_2\}^{\frac{1}{2}}$

and θ_e represents an approximation of the L^2 - error over an element.

If

$$m = 0, \quad q = 1, \quad p = \infty, \; k = 0,$$

we have

$$\theta_e = Ch_e \, |u^h|_{1,\infty}$$

and

$$|u^h|_{1,\infty} = \max_{x \in K} |\nabla u^h(x)|$$

Then θ_e is an approximation of the average of $|e^h|$ over an element.

To handle the constant C, one can either set C = 1 (in which case the indicators can only hope to measure relative error between successive meshes) or one can estimate C. One approach is as follows: consider the case of a bilinear Q1 - element (k = 1) and let the element K be the image of a master unit square $\hat{K}$ under an affine invertible map T_K.

Denote $\hat{\phi} = \phi \circ T_K$ for an arbitrary function ϕ and suppose that

$$|\hat{e^h}|_{0,2,\hat{K}} \leq C \, |\hat{u}|_{2,2,\hat{K}}$$

We denote by $\hat{P}\hat{u}$ the projection of $\hat{u} = u \circ T_K$ onto the quadratic polynomials $P_2(\hat{K})$ defined on $\hat{K}$.

Let u_2 be the bilinear interpolant of $\hat{u}$ over $\hat{K}$. Then choose as an approximation of C the constant

$$C^* = \text{infinum}_{\hat{f} \in P_2(\hat{K})} \frac{|f - f_2|_{0,2,K}}{|f|_{2,2,K}}$$

The use of such error indicators rests on several sweeping assumptions:

1) The local approximation error is bounded by the local interpolation error.

2) The error in replacing u by u_h in the semi-norm on the right-hand-side of (2.1) is negligible for h sufficiently small.

Numerical experiments suggest that these assumptions are frequently valid, while the validity of 2) depends on how this semi-norm of u^h is computed and on super convergence properties of u^h. Eriksson and Johnson [12] have recently presented rigorous proofs of a-posteriori estimates similar to (2.2) for a class of elliptic boundary-value problems.

2.2 The Adaptive Scheme Let us now suppose that error indicators θ_e can be calculated for each cell $K = \Omega_e$, $1 \le e \le E$, in a mesh of Q1 - quadrilaterals modeling a bounded two-dimensional domain Ω.

We begin with an initial coarse mesh containing only enough elements to model the basic geometrical features of the domain. This mesh defines a *mesh level,* denoted 0. Finer meshes are obtained by successive mesh bisections yielding mesh levels 1,2,3, $\cdots$ (Fig. 1). The size of the O-h level elements is to be that of the largest elements in the mesh. The largest level, of course, corresponds to the finest (local) mesh size. The calculation is initiated at some reasonably coarse mesh, say level 3 or 4, and thereafter different refinement levels are attained at different regions of the mesh.

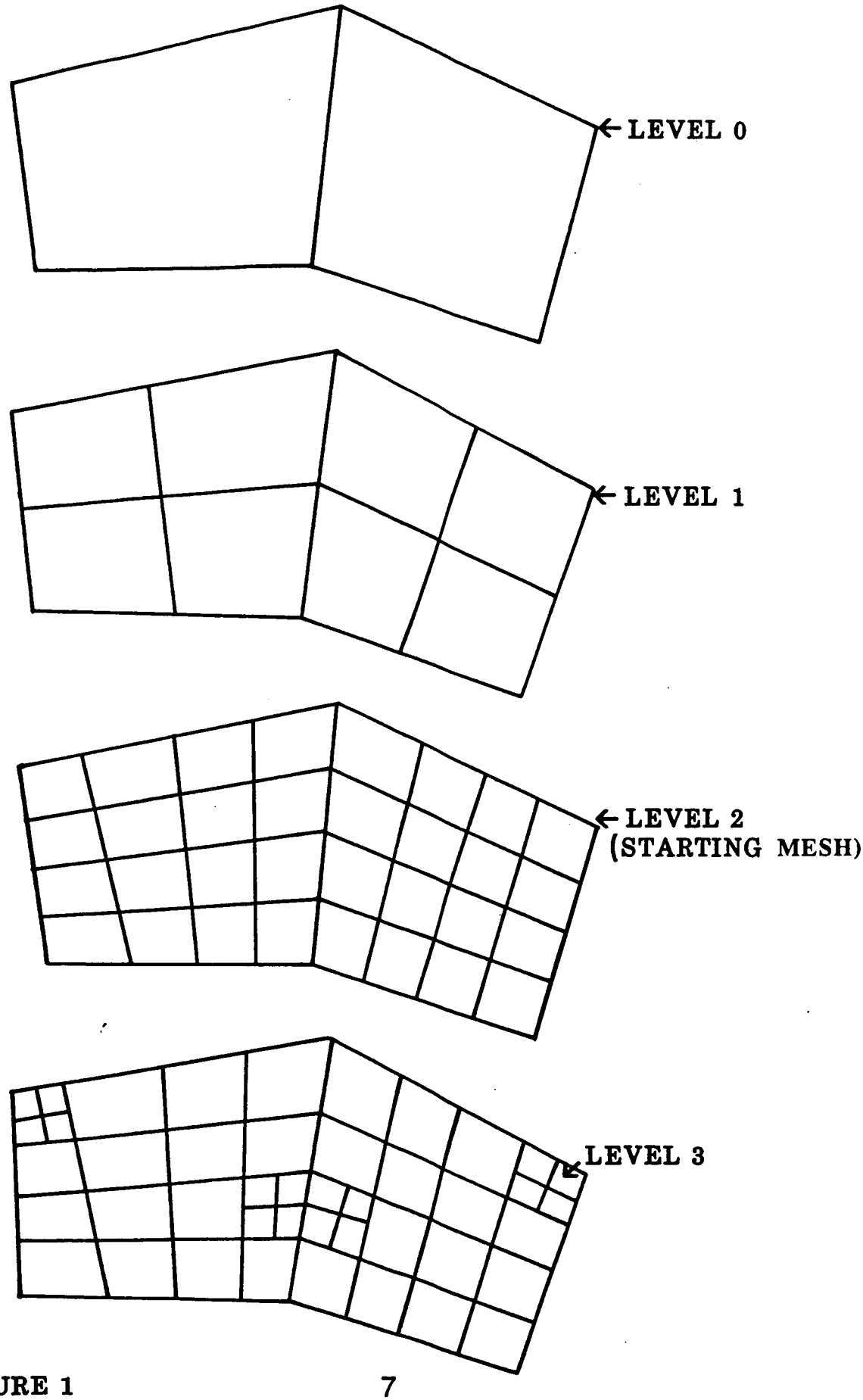


FIGURE 1

The idea of bisection to a starting mesh sets the stage for *unrefinements* in groups of four elements (i.e., the collapse of four elements into one element of lower level.)

We then proceed as follows:

1. After a set number of bisections of an initial mesh (producing 4-group clusters), a starting mesh is identified and an initial numerical solution of the problem under study is obtained on the starting mesh.

2. The adaptive procedure is initiated by computing solution quality indicators θ_e over all M elements in the grid. Let

$$\theta_{MAX} = \max_{1 \le e \le M} \theta_e .$$

3. Next, scan groups of a fixed number P of elements and compute

$$\theta^k_{GROUP} = \sum_{m=1}^{P} \theta_{k_m}$$

where θ_{k_m} denotes the m^{th} element number for group k, P=4 for 2-D grids and P=8 for 3-D grids.

4. The solution quality bounds are defined by two real numbers, $0<\alpha$, $\beta<1$. If

$$\theta^e \ge \beta \, \theta_{MAX}$$

element θ_e is **refined.** This is done by bisecting into four new sub-elements. If

$$\theta^k_{GROUP} \le \alpha \theta_{MAX}$$

group k is **unrefined** by replacing this group with a single new element with nodes coincident with the corner nodes of the group. This is always possible because each group is itself the result of an initial bisectioning.

This general process can be followed for any choice of a solution quality indicator.

2.3 Data Structures. An important consideration in all adaptive schemes is the data structure and associated algorithms needed to handle the changing number of elements, their node locations and numbers, and the element labels.

As noted in the preceding paragraphs, the algorithm is designed to process (refine or unrefine) in groups of four elements at each local refinement / unrefinement step. Consider, for example, the case of an initial mesh of 20 quadrilateral elements shown in Figure 2. Assign to each element in this mesh an element number, NEL = 1,2,..., NELEM and to each global node a label NODE. The array, NODES(J,NEL) relates the local node number J (J = 1,2,3,4) of element NEL to the global node number NODES. In addition, the coordinates XJ, YJ of each node are also provided relative to a fixed global coordinate system. File these numbers in two arrays,

NODES (J,NEL) = the array of global node numbers assigned to node J of element NEL

XCO (JCO,NODE) = the array of JCO -- coordinates of global node NODE (JCO = 1 or 2).

If a solution quality indicator signals that an element should be refined, say element 10 in the example, some system for assigning appropriate labels to the new elements and nodes must be devised. Toward this end, a convention can be established that defines the connectivity of the specified element with its neighbors in the mesh. This information is provided by a third connectivity array,

NELCON(NC,NEL) = the NCth connection of element NEL, where NC = 1,2,...,8

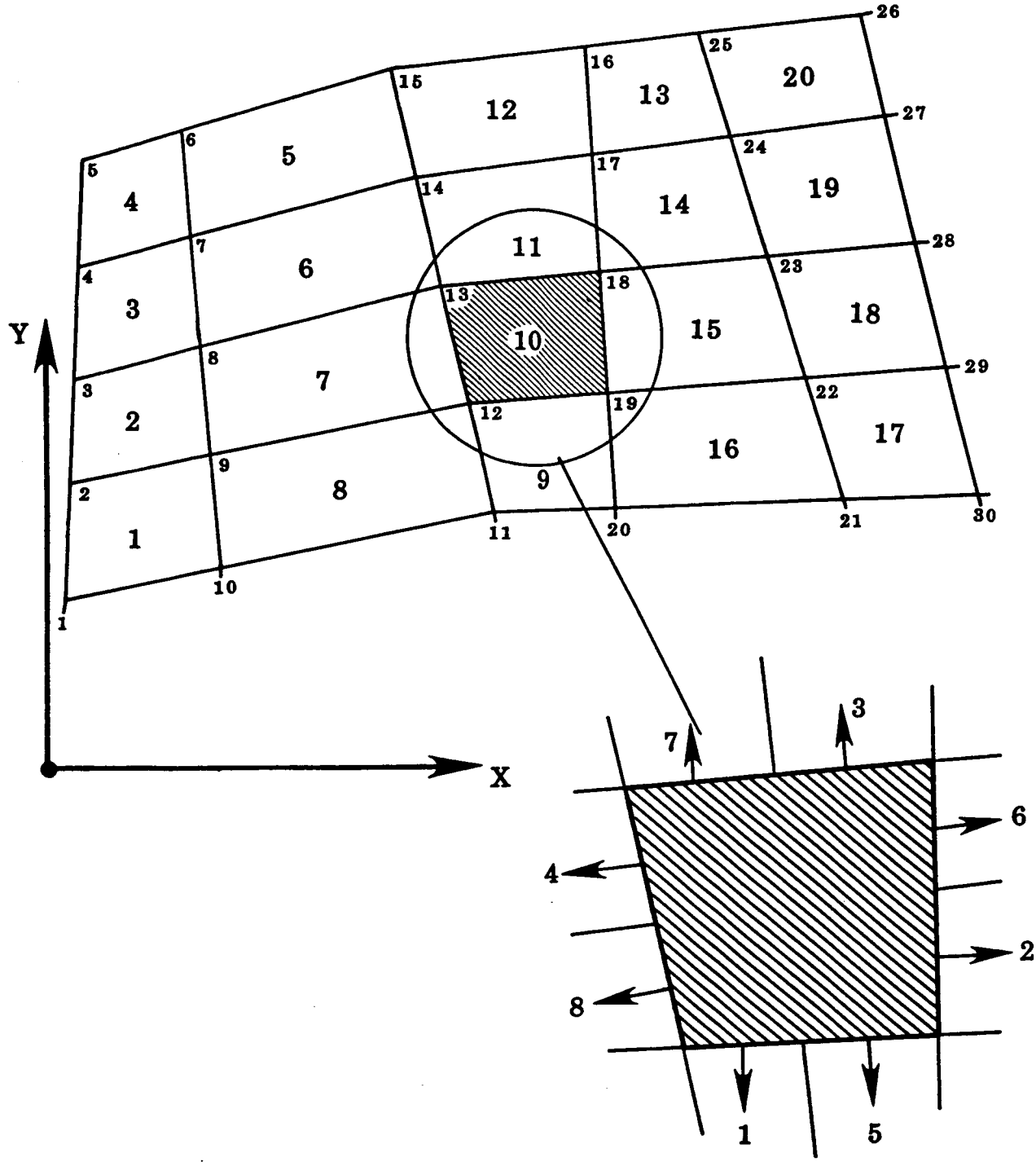

Y
X
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

FIGURE 2

As seen in Figure 2, each side of an element may be connected to two other elements so that NELCON is dimensioned accordingly;

NELCON(8,MAXEL)

with MAXEL denoting the maximum number of elements in the mesh.

The entire refinement process (or its inverse -- the unrefinement process) just described is accomplished by specifying a series of element levels. For example, the initial coarse mesh could be assigned level 0. When an element is refined, its sub-elements belong to a higher level, level 1, and when these sub-elements are refined, elements of level 2 result, and so on. In this way, if the maximum level any element in the mesh can achieve is limited, then the maximum number of elements the mesh can contain is also limited. In general, no such limit need be set.

Thus, the bookkeeping element and node numbers evolved in a refinement process is monitored by the arrays NODES (. , .) , XCO, NELCON (. , .) and an array LEVEL (NEL) which assigns a level number to element NEL. Initially, the same level can be assigned to all elements, and this level is an arbitrary parameter prescribed in advance by the user. Thus, provisions are now in hand for an arbitrary, dynamic renumbering of elements and nodes.

2.4 Adaptation Rules. Several rules must be established to successfully implement the refinement or coarsening of the mesh. The following "element" rules are employed:

1) An element may be refined only if its neighbors are at the same refinement level or higher.

2) If a "neighbor" element of an element to be refined is at a lower level of refinement, it must be refined first;

3) Refinement of an element results in creation of 8 sub-elements for 3-D meshes and 4 sub-elements for 2-D meshes.

4) To be eligible for coarsening a group of elements must not contain another group of elements and each element of the group to be coarsened must not be connected to a "neighbor" element of a higher level.

For example if element 10, in Figure 2, is to be refined, we proceed through the following steps:

1) Loop over the neighbors of element 10 (which is made possible with the NELCON array) and check the level of the neighboring elements relative to the level of element 10;

2) If any neighboring element has a level lower than 10, then the element cannot be refined at this stage;

3) If 10 can be refined (as is the case in Fig. 2), we generate new element numbers (thus changing NELEM and new node numbers for unconstrained nodes);

4) Compute the connectivity matrix NELCON for the new elements;

5) Adapt the connectivity matrices for the neighboring elements (since the refinement of 10 has now changed this connectivity).

In addition to the element rules, a set of nodal rules must be used. Nodes which exist along the edge of an embedded domain must be given special treatment by the finite element code used. The "constrained" nodes will appear and disappear as a mesh is refined or coarsened. This necessitates the following "node" rules:

1) An intermediate node is common to two members of a group, only.

2) An intermediate node that is created along a domain boundary cannot be constrained.

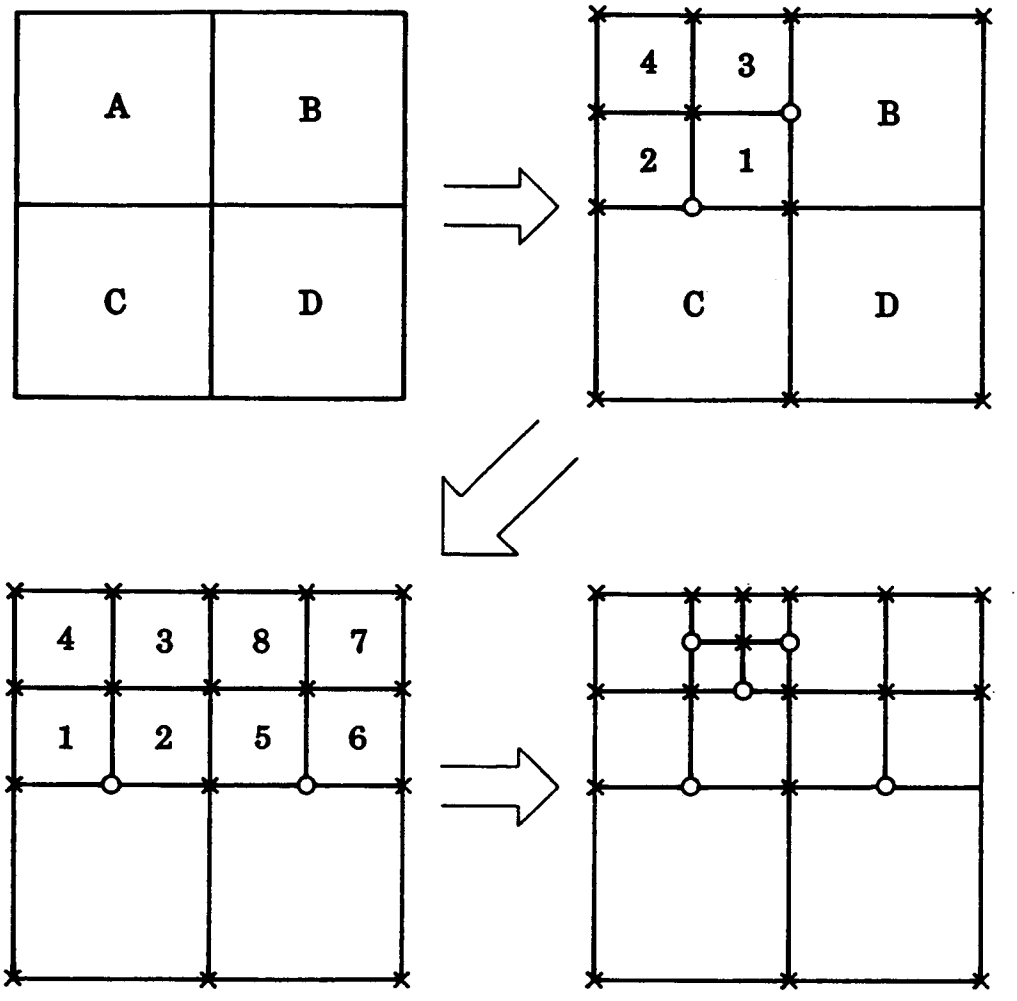
3) If an element and its neighbor both of which are at the same level are connected to a third element at a lower level, then the intermediate node which exists along the edge common with the third element is constrained.

4) If a group of elements is eligible for coarsening, then the intermediate constrained node along the edge common to an element which is not a member of the group will be eliminated

5) If a group of elements is eligible for coarsening, then the node along the edge common to this group and its neighbor group will become constrained.

6) If a group of elements is eligible for coarsening, then the intermediate node along a domain boundary edge is eliminated.

Use of the above rules can be illustrated by considering the uniform grid of four elements shown in Figure 3a. Suppose element A is marked for refinement. By



× -ACTIVE NODE

o -CONSTRAINED NODE

FIGURE 3

applying element rules 1 and 3 element A is divided into sub-elements, 1, 2, 3, 4 as shown. Application of node rules 1 and 2 dictates that the nodes marked by circles be constrained. Nodes marked X with the symbol are unconstrained.

Next, let element 3 be chosen for further refinement. Element 3 cannot be refined since one of its neighbors, B is at a lower level. Refinement of element 3 before element B would violate element rule 1. Therefore, element B is refined as shown in Figure 3b. Note that node β is no longer constrained, since node rule 2 no longer is satisfied. Node C1 remains constrained.

Now that element B has been divided into elements 5, 6, 7, 8, element rule 1 can be applied. Figure 3c illustrates this division.

Suppose the group of elements 5, 6, 7, 8 shown in Figure 3c is marked for coarsening. This group is not eligible for coarsening until the group of elements, α, β, γ, δ has been coarsened. Element 7 has neighbors β and δ which are of a higher level. This violates element rule 4.

Now let the group of elements, α, β, γ, δ be marked for coarsening. Element rule 4 is satisfied and elements α, β, γ, δ are replaced by element 3 . The intermediate constrained nodes associated with elements α, β, γ, δ are eliminated through use of node rule 4. The intermediate node along the upper domain boundary is eliminated using node rule 6.

III. THE VECTORIZED HAYES - DEVLOO SCHEME FOR ELEMENT BY ELEMENT MATRIX PROCESSING

We shall now describe a version of an algorithm structure introduced by Hayes and Devloo [9] which provides for a new, fast, vectorizable method for multiplying and assembling stiffness matrices in a fine element environment which has the following features:

1) The method focuses on the element by element assembly process common to finite element techniques; it is an "element-by-element" strategy.

2) The process is independent of (global) numbering of grid points or elements.

3) The process is applicable to unsymmetric matrices as well as symmetric matrices.

4) The process exploits parallelism in vector machines by "stacking" element stiffness rather than using the usual assembly process.

5) The introduction of new elements in a mesh merely results in the corresponding new stiffness being added to the stiffness matrix stack and does not interfere with the structure of the existing element matrices.

Obviously, these properties are critically important in adaptive procedures of the type discussed in Chapter 2.

For a given finite element mesh, it is standard procedure to compute local element stiffness matrices k_e, solution vectors u_e, and load vectors b_e for each element e, e = 1,2, ..., E, and to assemble the elements so as to obtain the global stiffness relation

$$\mathbf{K u = b}$$

By appropriate node and entry numbering, this process can be written

$$\mathbf{K} = \sum_e \mathbf{A}_e^T \mathbf{k}_e \mathbf{A}_e$$

$$\mathbf{A}_e \mathbf{u} = \mathbf{u}_e$$

$$\sum_e \mathbf{A}_e^T \mathbf{b}_e = \mathbf{b}$$

where $\mathbf{A}_e$ are Boolean matrices.

Alternatively, if

$$\mathbf{K}_e = \mathbf{A}_e^T \mathbf{k}_e \mathbf{A}_e$$

we can write

$$\mathbf{K}\,\mathbf{u} = \left(\sum_e \mathbf{K}_e \right) \mathbf{u} = \sum_e \mathbf{k}_e \mathbf{u}_e = \sum_e \mathbf{b}_e = \mathbf{b}$$

so that the assembly process is accomplished by a finite sum after or during a sequence of matrix multiplications. When using traditional sequential computing, the assembly is performed first and then the multiplication is performed. However, on a vector computer it makes more sense to use elementwise multiplication. Then the assembly process deals with the element vectors $\mathbf{b}_e$.

3.1 The Matrix Multiplication $k_e u_e$. The single matrix multiply of a 4 x 4 matrix is done in a diagonal form as follows:

$$\begin{vmatrix} K^e_1 & K^e_5 & K^e_9 & K^e_{13} \\ K^e_{14} & K^e_2 & K^e_6 & K^e_{10} \\ K^e_{11} & K^e_{15} & K^e_3 & K^e_7 \\ K^e_8 & K^e_{12} & K^e_{16} & K^e_4 \end{vmatrix} \begin{vmatrix} u^e_1 \\ u^e_2 \\ u^e_3 \\ u^e_4 \end{vmatrix} =$$

$$\begin{vmatrix} K^e_1 \\ K^e_2 \\ K^e_3 \\ K^e_4 \end{vmatrix} \begin{vmatrix} u^e_1 \\ u^e_2 \\ u^e_3 \\ u^e_4 \end{vmatrix} + \begin{vmatrix} K^e_5 \\ K^e_6 \\ K^e_7 \\ K^e_8 \end{vmatrix} \begin{vmatrix} u^e_2 \\ u^e_3 \\ u^e_4 \\ u^e_1 \end{vmatrix} + \begin{vmatrix} K^e_9 \\ K^e_{10} \\ K^e_{11} \\ K^e_{12} \end{vmatrix} \begin{vmatrix} u^e_3 \\ u^e_4 \\ u^e_1 \\ u^e_2 \end{vmatrix}$$

$$\begin{vmatrix} K^e_{13} \\ K^e_{14} \\ K^e_{15} \\ K^e_{16} \end{vmatrix} \begin{vmatrix} u^e_4 \\ u^e_1 \\ u^e_2 \\ u^e_3 \end{vmatrix} = \begin{vmatrix} b^e_1 \\ b^e_2 \\ b^e_3 \\ b^e_4 \end{vmatrix}$$

and the elements of matrix K_e are stored by diagonals. Calculations are done in four vector multiplications and two vector additions. To increase the length of each vector instruction, element matrices are stacked as shown in Figure 4. The data structure which is used in the stacked form of the algorithm is

$$\begin{vmatrix} K_1 \\ K_2 \\ K3 \\ K_4 \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{vmatrix} + \begin{vmatrix} K_5 \\ K_6 \\ K_7 \\ K_8 \end{vmatrix} \begin{vmatrix} u_2 \\ u_3 \\ u_4 \\ u_1 \end{vmatrix} + \begin{vmatrix} K_9 \\ K_{10} \\ K_{11} \\ K_{12} \end{vmatrix} \begin{vmatrix} u_3 \\ u_4 \\ u_1 \\ u_2 \end{vmatrix}$$

$$+ \begin{vmatrix} K_{13} \\ K_{14} \\ K_{15} \\ K_{16} \end{vmatrix} \begin{vmatrix} u_4 \\ u_1 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{vmatrix}$$

where $\mathbf{K}_i$,$\mathbf{u}_i$ and $\mathbf{b}_i$ are vectors whose lengths are the number of elements in the grids. Conceptually, the K^i entries are multiplied with the u^i entries for all the elements in the grid, then the K_2 entry is multiplied by the u_2 entry for all the elements in the grid, etc., so that multiplication can be performed in four vector multiply operations and two matrix addition instruction whose lengths are four times the number of elements in the grid.

3.2 Assembly Process. Here we follow Ref. [9]. According to Hayes and Devloo, once the elementwise multiplication has been performed, it is necessary to assemble or add the results into the final form. This corresponds to adding to node i contributions from all of the elements which contain that node. The following algorithm was designed so that:

1) only one rearrangement of the data $\mathbf{b}_e$ is needed in the assembly;
2) a minimum number of operations are done on zero data; and
3) a minimum number of vector operations are performed.

For each node, i, the number of element connections, NEC_i, is defined to be the number of elements which contain the node i. Then define the maximum number of connections in the grid as

$$MAXNEC = \underset{1}{MAX} \ (NEC_i)$$

The values in the element resultants, $\mathbf{b}_e$, are rearranged in the following manner. The values in the small vectors be are placed into vectors $\mathbf{B}_k$, where $\mathbf{B}_k$ is a vector containing one data value from every node which has at least K element connections. If node i has four element connections, then its element contributions from the four vectors $\mathbf{b}_e$ will be placed into the four vectors $\mathbf{B}_1$ - $\mathbf{B}_4$. The $\mathbf{B}_k$ arrays will be used for the addition in the assembly process, so it does not matter what order the data values from $\mathbf{b}_e$ are placed into the $\mathbf{B}_k$ arrays. The NEC_i array will be used to create an index array that will be used for the rearrangement of data.

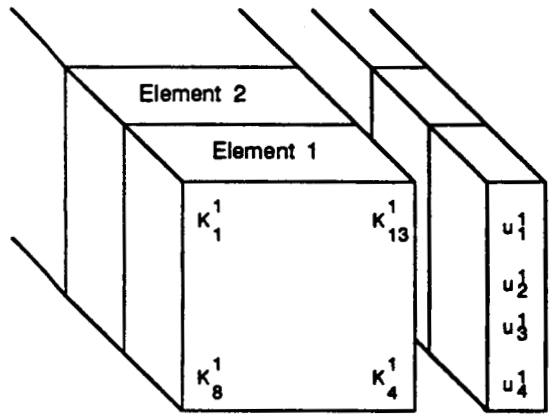
The $\mathbf{B}_k$ can then be added, and the final product $\mathbf{b}$ will be calculated as

$$\mathbf{b} = \sum_{k=1}^{MAXNEC} \mathbf{B}_k$$

The vectors $\mathbf{B}_k$ are of different lengths, and there are two options when performing the additions: 1) the vectors can be zero filled to the maximum length, and a divide-and-conquer strategy can be used, or 2) one can perform additions only on entries corresponding to the shorter vector. The following scheme is defined to maximize vectorization and to minimize calculations on artificially filled zero data. Each of the N nodes belongs to at least one element, the length of vector B_i is N. In addition, the length of $\mathbf{B}_i$ is greater than or equal to the length of $\mathbf{B}_j$ whenever i is less than or equal to j.

The number of maximum element connections hereafter is set to 8 for illustrative purposes.

1. The differences in the length of the vectors $B_1 + B_2$, $B_5 + B_6$, $B_3 + B_4$, and $B_7 + B_8$ is minimal; so we add them together in one vector operation, by arranging the arrays as shown (Cf[9]):



Thus, if $B_1 < B_2 < B_3 < \cdots < B_8$, the vector instruction length is as shown between the dotted lines below:

$$\begin{array}{|cc|} \hline B_1 & B_2 \\ B_5 & B_6 \\ B_3 & B_4 \\ B_7 & B_8 \\ \hline \end{array} \quad = \quad \begin{array}{|c|} \hline B_{12} \\ B_{56} \\ B_{34} \\ B_{78} \\ \hline \end{array}$$

2. Memory locations between the end of B_6 and the beginning of B_4 are filled with zeros. Here,

$$B_{ij} = B_i + B_j$$

3. Next, the addition $B_{12} + B_{34}$ is performed (now with no zero fill) to give

$$B_{1234} \text{ and } B_{5678}$$

and the final addition is performed by adding $\mathbf{B}_{1234}$ and $\mathbf{B}_{5678}$ with the array

$$\mathbf{B}_{12345678}$$

the final result.

Notice that although all vectors had a different length, only two vectors had to be zero-filled and the addition of the 8 arrays, $\mathbf{B}_1$ - $\mathbf{B}_8$, is accomplished in three vector additions.

To obtain an ordering of $\mathbf{B}_{12345678}$ in the order of consecutive node numbering, an addition permutation is needed only for output.

IV. FINAL COMMENTS AND FUTURE WORK

This report describes a small pilot study of algorithms for adaptive schemes that could, with additional work, form the basis for new, fast, vectorizable schemes for SSME flow analysis. Principal conclusions of this investigation are:

1. The analysis of complex elliptic problems defined on general two-dimensional computational domains can be carried out using an adaptive, vectorizable, data management scheme that enables the code to automatically refine and unrefine meshes when signaled by computed error indicators.

2. The procedure uses fully unstructured meshes, and handles the significant problem of managing a changing list of node labels, connectivities, and element labels.

3. Need for a sophisticated mesh generator is minimized or eliminated. Rather than have the analyst guess where mesh refinements are needed, the adaptive process attempts to produce an optimal mesh with an equidistributed error. The algorithm produces the mesh necessary to yield a preassigned level of accuracy.

4. The procedures described here have been tested extensively during the report period on elliptic problems, but that general procedure is independent of equation type. Results of an Euler calculation are given in Fig. 4 for comparison with those obtained on a uniform mesh.

5. Element-by-element solution schemes are ideally suited for adaptive schemes of the type described here. The new vectorized matrix-vector multiply schemes of Hayes and Devloo were found to be particularly attractive for these types of adaptive methods. Moreover, the element-by-element, by exploiting the natural vector structure of the methods used here, can produce solution schemes quite superior to more common direct solvers and iterative schemes. Comparisons of computation times experienced by Hayes' using an element-by-element solver with those obtained with an ELLPACK solver are given in Table 1 [9].

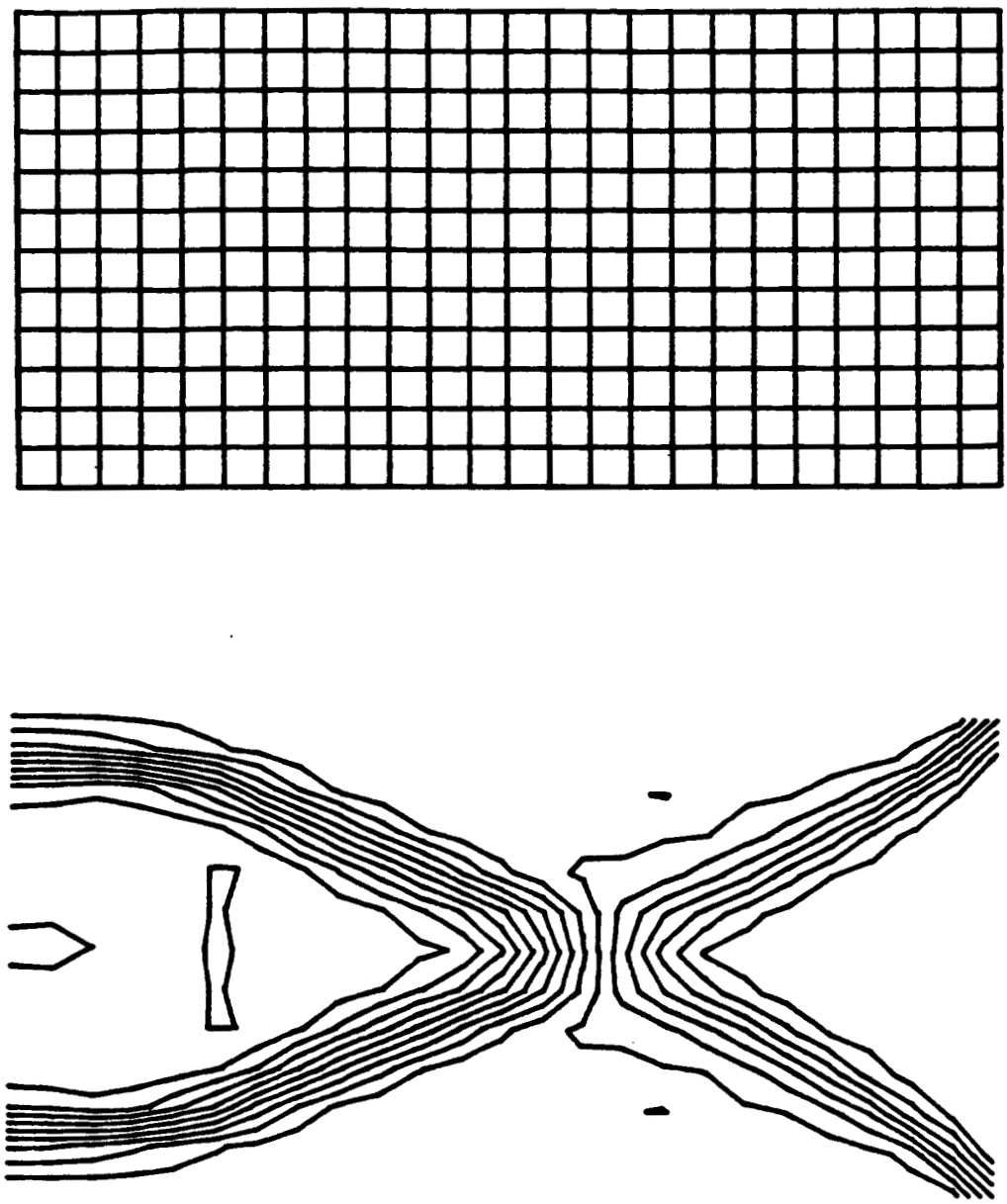
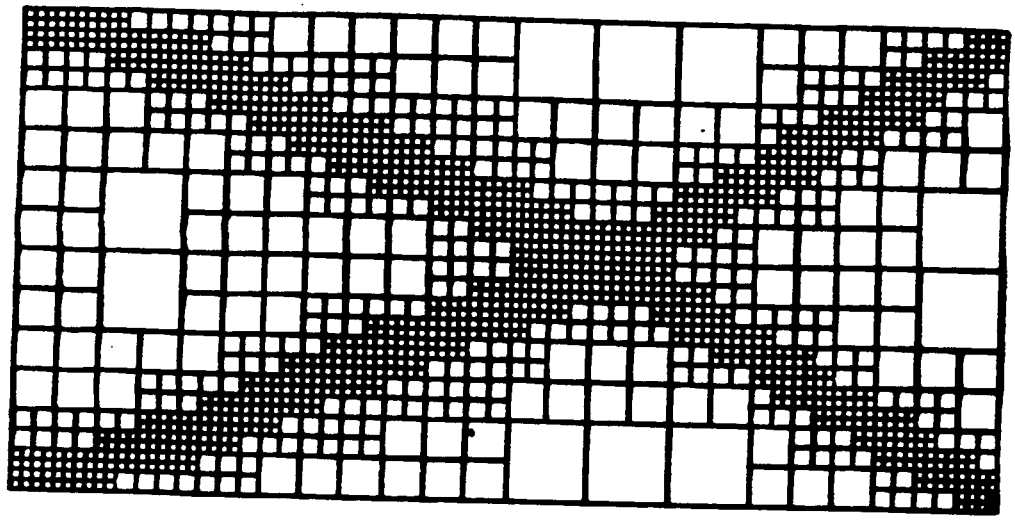
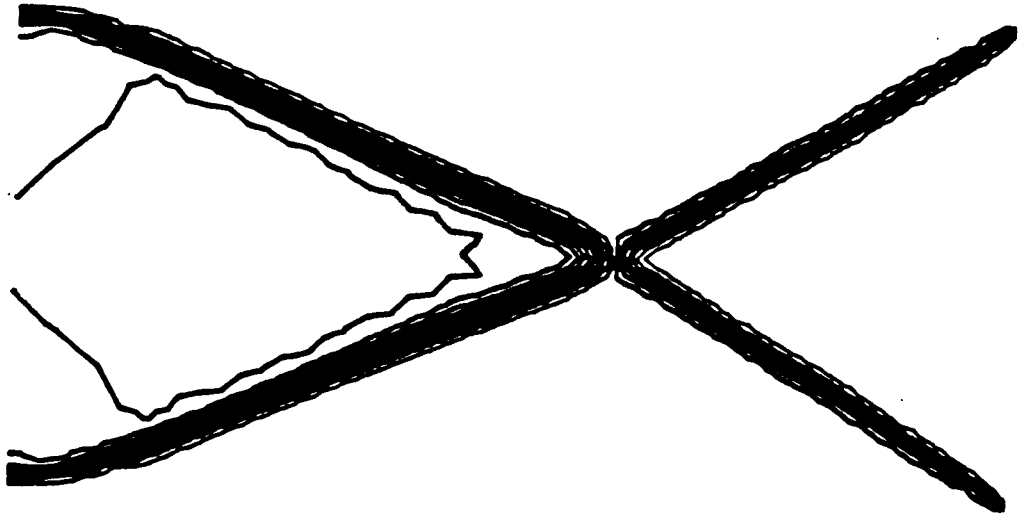


Figure 4. Representative results of an adaptive Euler calculation and comparison of density contours for uniform and adapted meshes.



c)



d)

Figure 4, continued.

TABLE 1

Problem Size	Time (Sec)		Speedup
	Element-by-Element [9]	ELLPACK Solver	
441 equations	0.015	0.045	67%
1,681 equations	0.102	0.208	49%
11,025 equations	1.70	2.60	35%

Similar results have been obtained on comparing such schemes with a variety of other well-known linear equation solvers, including the Symmetric Yale Sparce Matrix solver and others (see [9, 13, 14].

It is important to note that the results in Table 2 were obtained for regular meshes. For irregular meshes, of the type generated by adaptive methods, speedup times of element-by-element procedures ran as high as 71% faster conventional sparce matrix solvers and iterative methods.

6. As noted earlier, the adaptive schemes described here are quite different from some others recently published in the literature:

a) The strategy does not use a tree structure, as, for example, in Rheinboldt and Mesztenyi [7], and therefore does not require the generally inefficient step of retaining data at tree roots and branches. The new scheme described here abandons labels and connectivities as they are changed and, thus, does not require additional storage for this purpose.

b) The scheme does not use hierarchial families of shape functions (such as those schemes which assign new shape functions to each new node added in the refinement as in [6]) and, thus, realizes an increase in efficiency several times that of hierarchial methods.

The results of this study point to several issues worthy of study in the future:

1. First and foremost, the implementation of the schemes developed here in large-scale codes for SSME analysis should be actively pursued. These methods are very general and can be easily incorporated in any code that is not restricted to fixed structured meshes. On the other hand, it may be possible to adapt these procedures to structured meshes, although it is likely that some of the principal advantages of these methods may be lost or hampered.

2. We believe that adaptive schemes of the type described here may offer the only hope for solving one of the classical problems in computational fluid mechanics: that of scales. How can one resolve complicated features of three-dimensional flow, such as shock structures, separations, etc., and at the same time, resolve finer features of the flow in thin boundary layers? The successful resolution of shocks generally requires a healthy portion of artificial viscosity to dampen oscillations, but such artificial viscosity can override actual viscosity effects unless the mesh is very, very fine. This paradox has led some to suggest the use of meshes with as many as 10^6 grid points, a prospect leading to serious doubts that these issues are being dealt with correctly by contemporary difference schemes.

We believe that adaptive methods can be used to resolve these classical computational fluid dynamics issues; however, it is likely that a new family of adaptive methods will have to be developed for this purpose. It is known that fewer degrees of freedom are required of spectral or p-methods than by mesh refinement

methods to yield the same accuracy. On the other hand, such methods place great demands on the data management routines. These facts, we believe, point to combined h-p or h-spectral adaptive methods as having great potential for the most complex flow problems. We conjecture that a very effective strategy for such problems would be to use a fast h-method for adaptive refinement up to a fixed level, say level 4 or 5, so that major features of the flow are identified, and that then a p- or spectral adaptive scheme could be put into action to resolve finer features such as thin boundary layers. Theoretical estimates suggest that such techniques may be able to resolve boundary layers using one- or two-orders of magnitude fewer unknowns than the ultra-fine difference grids mentioned earlier.

An equally appealing feature of such methods is that these combined methods are ideally suited for parallel computing. By exploiting both parallel architectures and algorithms and h-p adaptivity, it should be possible to produce the most powerful and efficient computational fluid dynamics strategies ever devised.

REFERENCES

1] Oden, J.T., Strouboulis, T., and Devloo, Ph., "Adaptive Finite Element Methods for the Analysis of Inviscid Compressible Flow: I. Fast Refinement/Unrefinement and Moving Mesh Methods for Unstructured Meshes," *Computer Methods in Applied Mechanics and Engineering* , 59 (3), 1986.

2) Oden, J.T., Strouboulis, T., Devloo, Ph., Robertson, S.J., and Spradley, L.W., "Adaptive Moving Mesh Finite Element Methods for Flow Interaction Problems," Proceedings, **Sixth International Conference on Finite Element Methods in Fluids,** Antibes, France, June, 1986.

3) Demkowicz, L. and Oden, J.T., "An Adaptive Characteristic Petrov-Galerkin Finite Element Method for Convection Dominated Linear and Non-Linear Parabolic Problems in One Space Variable, *Journal of Computational Physics,* Vol. 68, No. 1, pp. 188-273, November, 1986.

4) Oden, J.T., Strouboulis, T., and Devloo, Ph., "Recent Advances in Error Estimation and Adaptive Improvement of Finite Element Calculations", **Computational Mechanics - Advances and Trends**, edited by A.K. Noor, ASME , New York, 1986, pp.369-410.

5) Oden, J.T., Spradley, L.W., Strouboulis, T., Devloo, Ph., and Price, J., "An Adaptive Finite Element Strategy for Complex Flow Problems.," Paper AAIA -87-0557, **25th Aerospace Sciences Meeting**, Reno, 1987.

6) Zienkiewicz, O.C., Kelley, D.W. Gago, J.P. de S.R., and Babuska, I., " Hierarchical Finite Element Approahces, Adaptive Refinement, and Error Estimates," **The Mathematics of Finite Elements with Applications**, Ed. by J. R. Whiteman, Academic Press Ltd., London, pp. 313-346, 1982.

7) Rheinboldt, W., and Mesztenyi, C.K., "On a Data Structure for Adaptive Finite Element Refinements," TOMS, Vol. 6, pp. 166-187, 1980.

8) Ewing, R., "Adaptive Mesh Refinements in Large-Scale Fluid Flow Simulation", **Accuracy Estimates and Adaptive Refinements in Finite Element Computations**,Ed. by Babuska et al., John Wiley and Sons, Ltd., Lond, pp. 299-314, 1986.

9) Hayes, L.H., and Devloo, Ph., "A Vectorized Version of a Sparse Matrix-Vector Multiply," *International Journal of Numerical Methods in Engineering,* Vol. 23, pp. 1043-1056, 1986.

10) Oden, J.T., and Reddy, J.N., **An Introduction to the Mathematical Theory of Finite Elements**, Wiley - Interscience, New York, New York, 1976.

11) Oden, J.T., and Carey, G.F., **Finite Elements: Mathematical Aspects**, Englewood Cliffe, 1983.

12) Ericksson, K. and Johnson, C., "An Adaptive Finite Element Method for Linear Elliptic Problems," **Chalmers University of Technology, Preprint**, 1986.

13) Hayes, L.J., "A Vectorized Matrix-Vector Multiply and Overlapping Block Iterative Method," **Proceedings: Super Computer Applications Symposium**, Plenum Press, October, 1984.

14) Hayes, L.J., "Advances and Trends in Element-by-Element Techniques," **State of the Art Surveys in Computational Mechanics**, Ed. by A. K. Noor, ASME, Special Volume, New York, 1987.