CHAPTER 34

Adaptive Implicit-Explicit and Parallel Element-By-Element Iteration Schemes*

T. E. Tezduyar†
J. Liou†
T. Nguyen††
S. Poole††

Abstract

Adaptive implicit-explicit (AIE) and grouped element-by-element (GEBE) iteration schemes are presented for the finite element solution of large-scale problems in computational mechanics and physics. The AIE approach is based on the dynamic arrangement of the elements into differently treated groups. The GEBE procedure, which is a way of rewriting the EBE formulation to make its parallel processing potential and implementation more clear, is based on the static arrangement of the elements into groups with no inter-element coupling within each group. Various numerical tests performed demonstrate the savings in the CPU time and memory.

1. Introduction

For large-scale problems in computational mechanics and physics, solution (by direct methods) of the linear equation systems involving massive global matrices and the storage of such matrices generate substantial demand on the computational resources in terms of the CPU time and memory. For most problems of practical interest, especially in three dimensions, this demand becomes too heavy to accommodate even for today's most generous computers. Matrices of this volume usually arise from the implicit time-integration of spatially discretized time-dependent problems or from spatial discretization of equations with no (real or pseudo) temporal derivatives. For example, the incompressible Navier-Stokes equations in the vorticity-stream function formulation involve a time-dependent convection-diffusion equation for the vorticity and a Poisson's equation for the stream function. In this paper we describe, in the context of the vorticity - stream function formulation, alternative solution techniques which reduce or eliminate the need for the storage and factorization of large global matrices.

The implicit-explicit algorithm proposed by Hughes and Liu [6,7] (for solid mechanics and heat conduction problems) involves the static allocation of the implicit and

* This research was sponsored by NASA-Johnson Space Center under contract NAS-9-17892 and by NSF under grant MSM-8796352.

† Department of Aerospace Engineering and Mechanics, and Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN 55455.

‡ IBM Corporation, Houston, TX 77056.

explicit elements based on the stability limit of the explicit algorithm. The adaptive implicit-explicit (AIE) scheme was first presented, in its preliminary stage, in Tezduyar and Liou [13]. It is based on the dynamic grouping of the elements into the implicit and explicit subsets as dictated by the element level stability and accuracy considerations. For this purpose the algorithm monitors the element level Courant number and some measure of the dimensionless wave number. The element level Courant number is based on the local convection and diffusion transport rates whereas the dimensionless wave number reflects the local smoothness of the "previous" solution and the spatial discretization. The "previous" solution can come from the previous time step, or nonlinear iteration step, or pseudo-time iteration step, whichever is appropriate. In this approach we can have "implicit refinement" where it is needed. Elsewhere in the domain, computations are performed explicitly thus resulting in substantial memory and CPU time savings. The savings can be further increased by performing, as often as desired, an equation renumbering at the implicit zones to obtain optimal bandwidths. Compared to the adaptive schemes based on grid-moving or element-subdividing the AIE method involves minimal bookkeeping and no geometric constrains.

It is important to note that the dynamic grouping of the AIE scheme does not necessarily have to be with respect to implicit and explicit elements. The letter "I" and "E" in "AIE" could be referring to any two procedures. In fact, there is no reason for the number of possible choices to be limited to two. For a given element, the rationale for favoring one procedure over another one could be based on any factor, such as the cost efficiency, the type of spatial or temporal discretization, the differential equations used for modelling, etc. In this paper, as an example, we implement the AIE scheme also in conjunction with the flux corrected transport (FCT) method [9,15].

It has been more than a decade since the first CRAY-1 computer was delivered to Los Alamos National Laboratories. The evolution of vector architecture in the past few years is evident. With the inception of new vectorizing compiler technology, the job of the application developers has become less architecture specific but rather more numerical and algorithmic in nature. Since the introduction of the CRAY-XMP, a multi-processor vector computer, in 1982, parallel processors and parallel processing have become more popular among the computer vendors in the Engineering and Scientific market sector. The fact that IBM introduced the Vector Facility for the 3090 multi-processors series is the ultimate acknowledgement that vector processing is an accepted standard mode of operation and parallel processing is eminent in the conventional camp.

The element-by-element (EBE) factorization scheme is very well suited for vectorization and parallel processing. It was proposed by Hughes, Levit, and Winget [5] for problems in transient heat conduction, solid mechanics, and structural mechanics. The EBE implementations in computational fluid dynamics were first reported, in the context of compressible Euler equations, by Hughes, Levit, Winget, and Tezduyar [8]. Applications to convection-diffusion equations and the incompressible flow problems can be seen in Tezduyar and Liou [13]. The EBE scheme selects a sequential product of the element level matrices as the preconditioning matrix to be used with the iterative solution of large linear equations systems arising from the finite element formulation. For symmetric and positive-definite matrices this preconditioner can be used with the conjugate-gradient method [3]. In the EBE approach the need for the formation, storage, and factorization of bulky global matrices is eliminated. The element level matrices can be either recomputed or stored. In the case the element level matrices are stored, the storage needed is still only linearly proportional to the number of elements.

The grouped element-by-element (GEBE) scheme is a variation of the regular EBE scheme. It has some common grounds with the operator splitting and domain decomposition methods [3,4]. The GEBE preconditioner is a sequential product of the element group matrices with the condition that no two elements in the same group can be "neighbors"; this is achieved by a simple element grouping algorithm which is applicable to arbitrary meshes. In our parallel computations, before we start processing a new group we first have to finish processing the current one. To minimize the overhead associated with this synchronization, we try to minimize the number of groups. Within each group, to increase the vector efficiency of the computations, elements are processed in chunks of

64 elements (or whatever the optimum chunk size for the given vector environment is). When parallel processing is invoked, each of the available CPUs work on one chunk at a time until all chunks are finished.

2. Problem Statement and the Finite Element Formulation

We consider the problems governed by the convection-diffusion equation and the two-dimensional incompressible Navier-Stokes equations. For the Navier-Stokes equations, we use the vorticity - stream function formulation which consists of a time-dependent transport equation for the vorticity ω, and a Poisson's equation relating the stream function ψ to the vorticity. Since the procedures for the spatial and temporal discretizations for the convection-diffusion equation are similar to those for the transport equation of the vorticity - stream function formulation, only the latter will be discussed in this section.

The field equations for the vorticity - stream function formulation of the two-dimensional incompressible Navier-Stokes equations are given as

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \bullet \nabla \omega = \nu \nabla^2 \omega, \tag{2.1}$$

$$\nabla^2 \psi = -\omega, \tag{2.2}$$

where $\mathbf{u}$ is the velocity and ν is the kinematic viscosity. The incompressibility condition is automatically satisfied by the following definition of the stream function:

$$\frac{\partial \psi}{\partial x_2} = u_1, \tag{2.3a}$$

$$\frac{\partial \psi}{\partial x_1} = -u_2. \tag{2.3b}$$

The flow domain can have several internal boundaries corresponding to possible obstacles (holes) in the flow field. In such cases the unknowns of the problem are: the value of the vorticity at all interior points (ω_*), the value of the vorticity at all boundary points where both components of the velocity are specified (ω_{Gq}) (this includes all the external and internal solid boundaries), and the value of the stream function at all interior points and on the internal boundaries (ψ_{*q}).

The differential equations given by (2.1) and (2.2), together with the initial condition for the vorticity and the suitable boundary conditions for the vorticity and the stream function, can be translated, via a proper variational formulation [11], into a set of ordinary differential equations. That is

$$\tilde{\mathbf{M}}(\mathbf{d}_{*q})\,\mathbf{a}_* + \tilde{\mathbf{C}}(\mathbf{d}_{*q})\mathbf{v}_* = \tilde{\mathbf{F}}(\mathbf{d}_{*q}, \mathbf{v}_{Gq}, \mathbf{a}_{Gq}), \tag{2.4a}$$

$$\mathbf{v}_*(0) = (\mathbf{v}_*)_0, \tag{2.4b}$$

$$\mathbf{K}\,\mathbf{d}_{*q} = \mathbf{F}_{II}(\mathbf{v}_*, \mathbf{v}_{Gq}), \tag{2.5a}$$

$$\mathbf{M}\,\mathbf{v}_{Gq} = \mathbf{F}_{III}(\mathbf{d}_{*q}, \mathbf{v}_*), \tag{2.5b}$$

where $\mathbf{d}_{*q}$, $\mathbf{v}_*$, $\mathbf{a}_*$, $\mathbf{v}_{Gq}$, and $\mathbf{a}_{Gq}$ are the vectors of nodal values of ψ_{*q}, ω_*, $\frac{\partial \omega_*}{\partial t}$, ω_{Gq}, and $\frac{\partial \omega_{Gq}}{\partial t}$, respectively.

Remark 2.1 Particular attention must be paid to the boundary conditions for the vorticity on the (external and internal) solid surfaces and for the stream function on the internal boundaries. Discussion of these boundary conditions and the derivation of the proper variational formulations can be found in [11].

Remark 2.2 Equation systems (2.4a) and (2.5) are derived from the variational formulations corresponding to (2.1) and (2.2), respectively. The initial condition for (2.1) is represented by (2.4b).

Remark 2.3 The matrices $\mathbf{K}$ and $\mathbf{M}$ are symmetric and positive-definite. The matrix $\mathbf{M}$ is topologically one-dimensional because it is associated with the vector $\mathbf{v}_{Gq}$ corresponding to the (unknown) value of the vorticity on the boundaries. The matrices $\tilde{\mathbf{M}}$ and $\tilde{\mathbf{C}}$ are functions of the stream function because in the variational formulation of (2.1) we employ a Petrov-Galerkin method [10] with weighting functions dependent upon the velocity field.

Remark 2.4 Equations (2.4) and (2.5) are solved by employing a predictor/multi-corrector algorithm which, in nature, is a block-iteration scheme [12]. In fact the way we have written these equations reflects the way the block-iteration scheme works. In the blocks corresponding to (2.4a),(2.5a), and (2.5b), we update $\mathbf{v}_*$, $\mathbf{d}_{*q}$, and $\mathbf{v}_{Gq}$, respectively. To move from time step n to time step n+1 we perform iterations as outlined below:

block 1:

$$\bar{\mathbf{M}}^{i}_{n+1}\ (\Delta\mathbf{a}_*)^{i}_{n+1} = \mathbf{R}^{i}_{n+1}\ , \tag{2.6a}$$

where

$$\bar{\mathbf{M}}^{i}_{n+1} = \tilde{\mathbf{M}}((\mathbf{d}_{*q})^{i}_{n+1}) + \alpha\,\Delta t\,\tilde{\mathbf{C}}((\mathbf{d}_{*q})^{i}_{n+1}), \tag{2.6b}$$

and

$$\begin{aligned}\mathbf{R}^{i}_{n+1} &= \tilde{\mathbf{F}}((\mathbf{d}_{*q})^{i}_{n+1}, (\mathbf{v}_{Gq})^{i}_{n+1}, (\mathbf{a}_{Gq})^{i}_{n+1}) \\ &\quad - (\tilde{\mathbf{M}}((\mathbf{d}_{*q})^{i}_{n+1})\,(\mathbf{a}_*)^{i}_{n+1} + \tilde{\mathbf{C}}((\mathbf{d}_{*q})^{i}_{n+1})\,(\mathbf{v}_*)^{i}_{n+1}\ ,\end{aligned} \tag{2.6c}$$

with updates

$$(\mathbf{v}_*)^{i+1}_{n+1} = (\mathbf{v}_*)^{i}_{n+1} + \alpha\,\Delta t\,(\Delta\mathbf{a}_*)^{i}_{n+1}, \tag{2.6d}$$

$$(\mathbf{a}_*)^{i+1}_{n+1} = (\mathbf{a}_*)^{i}_{n+1} + (\Delta\mathbf{a}_*)^{i}_{n+1}. \tag{2.6e}$$

Here i is the iteration count, Δt is the time step, and α is a parameter which controls the stability and accuracy of the time integration.

block 2:

$$\mathbf{K}(\mathbf{d}_{*q})^{i+1}_{n+1} = \mathbf{F}_{II}((\mathbf{v}_*)^{i+1}_{n+1}, (\mathbf{v}_{Gq})^{i}_{n+1}). \tag{2.7}$$

block 3:

$$\mathbf{M}(\mathbf{v}_{G\,q})^{i+1}_{n+1} = \mathbf{F}_{III}((\mathbf{d}_{*q})^{i+1}_{n+1}, (\mathbf{v}_*)^{i+1}_{n+1}). \tag{2.8}$$

The iterations continue until a predetermined convergence criterion is met.

3. The Adaptive Implicit-Explicit (AIE) Scheme

In our block-iterative procedure, at every iteration we need to solve three equation systems : (2.6a),(2.7), and (2.8). The cost involved in (2.8) is not major (see Remark 2.3) and therefore we solve it with a direct method. We are mainly interested in minimizing the CPU time and memory demands of equations (2.6a) and (2.7) which, after dropping all the subscripts and superscripts, can be rewritten as follows:

$$\bar{\mathbf{M}}\,\Delta\mathbf{a} = \mathbf{R}, \tag{3.1}$$

$$\mathbf{K}\,\mathbf{d} = \mathbf{F}. \tag{3.2}$$

We need to remember that $\mathbf{K}$ is symmetric and positive-definite but $\bar{\mathbf{M}}$, in general, is not. We propose to employ the AIE scheme for both of these equations.

the AIE scheme in the context of the vorticity transport equation

Let $\mathcal{E}$ be the set of all elements, e=1,2,..., n_{el}. The assembly of the global matrix $\bar{\mathbf{M}}$ in equation (3.1) can be expressed as follows:

$$\bar{\mathbf{M}} = \sum_{e\in\mathcal{E}} \bar{\mathbf{M}}^e, \tag{3.3}$$

where $\bar{\mathbf{M}}^e$ is the element contribution matrix which is obtained by permutating the element level matrix $\bar{\mathbf{m}}^e$. It should be noted that $\bar{\mathbf{M}}^e$ has the same dimensions as the global matrix $\bar{\mathbf{M}}$ but only as many number of nonzero entries as $\bar{\mathbf{m}}^e$ (e.g. 4×4 for a two-dimensional quadrilateral element with a scalar unknown).

Let $\mathcal{E}_I$ and $\mathcal{E}_E$ be the subsets of $\mathcal{E}$ corresponding to the implicit and explicit elements, respectively, such that

$$\mathcal{E} = \mathcal{E}_I \cup \mathcal{E}_E, \tag{3.4a}$$

$$\varnothing = \mathcal{E}_I \cap \mathcal{E}_E. \tag{3.4b}$$

Consequently, from equation (3.3) we get

$$\bar{\mathbf{M}} = \sum_{e\in\mathcal{E}_I} \bar{\mathbf{M}}^e + \sum_{e\in\mathcal{E}_E} \bar{\mathbf{M}}^e. \tag{3.5}$$

The AIE scheme is based on modifying $\bar{\mathbf{M}}$ by replacing $\bar{\mathbf{M}}^e$ ($\forall\, e \in \mathcal{E}_E$)with its lumped mass matrix part; that is

$$\bar{\mathbf{M}} = \sum_{e\in\mathcal{E}_I} \bar{\mathbf{M}}^e + \sum_{e\in\mathcal{E}_E} \mathbf{M}_L^e. \tag{3.6}$$

The grouping given by (3.4) is done dynamically (adaptively) based on the stability and accuracy considerations.

The stability criterion is in terms of the element Courant number $C_{\Delta t}$ which is defined as

$$C_{\Delta t} = \frac{\|\mathbf{u}\|\,\Delta t}{h}, \tag{3.7}$$

where h is the "element length" [10]. Any element with its Courant number greater than the stability limit of the explicit method should belong to the implicit group $\mathcal{E}_I$.

For the accuracy considerations we use a quantity which is meant to be a measure of the dimensionless wave number [10]. For this purpose, in [13] the "jump" in the solution across an element was defined as

$$j^e(\omega) = \max_a (\omega_a^e) - \min_a (\omega_a^e), \tag{3.8}$$

where a is the element node number and ω_a^e is the value of the dependent variable ω at node a of the element e. This definition reflects the magnitude of the variation of the solution across an element. It was proposed in [13] that the elements with jumps greater than a predetermined amount should belong to the group $\mathcal{E}_I$.

In our numerical tests involving the one-dimensional propagation of various triangular profiles we observed that the accuracy of the solution is affected not only by the jump in the solution but also by the derivative of the flux of the dependent variable. Therefore, as an alternative criterion we propose to employ the product of the jump and the derivatives of the flux; that is

$$\sigma^e = j^e(\omega) \; \left(\left| \frac{\partial(u_1\omega)}{\partial x_1} \right| + \left| \frac{\partial(u_2\omega)}{\partial x_2} \right| \right)^e. \tag{3.9}$$

A global scaling is needed for this quantity and we propose the following procedure:

$$\sigma_\varepsilon^e = \frac{\sigma^e}{\sigma_{max}^e}, \tag{3.10}$$

where

$$\sigma_{max}^e = \max_{e \in \varepsilon} \sigma^e . \tag{3.11}$$

In this case the elements with σ_ε^e greater than a predetermined value belong to group $\mathcal{E}_I$.

Implementation of the AIE scheme is quite straightforward. A global search is performed on the entire set of elements. With this search, based on the two criteria given by equations (3.7) and (3.10), the elements are grouped into the subsets $\mathcal{E}_I$ and $\mathcal{E}_E$. Compared to having all elements treated implicitly, this grouping results in substantial savings in the CPU time and memory.

Remark 3.1 It is important to note that the grouping does not have to be with respect to implicit and explicit elements. In the acronym "AIE", the letter "I" can refer to the "I-elements" subject to some "I-procedure" whereas the letter "E" can refer to the "E-elements" subject to some other "E-procedure". Let us suppose that we are concerned with the computational cost but we also believe that "you get what you pay for". Then we might have an "I-procedure" which is sophisticated but costly and an "E-procedure" which is less sophisticated but also less costly. We can employ the "I-procedure" where it is needed and the "E-procedure" elsewhere. Even if the computational cost is not an issue, in certain regions of the domain we just might have a reason to prefer the "I-procedure" over the "E-procedure", or vice versa. The reasons could be based on the type of interpolation functions, spatial discretization, differential equations used for modelling, etc. The AIE concept is based on making decisions at the element level, dynamically, and

according to some desired and reliable criteria. Later in this section we will give an example for an AIE scheme which is not based on the implicit-explicit grouping.

Remark 3.2 In the AIE approach one can have a high degree of refinement throughout the mesh and raise the implicit flag only for those elements which need to be treated implicitly.

Remark 3.3 The savings in the CPU time and memory can be maximized by performing, as often as desired, an equation renumbering at the "implicit zones" to obtain optimal bandwidths. Bandwidth optimizers are already available for finite element applications, especially in the area of structural mechanics.

Remark 3.4 Time-dependent convection-diffusion of a passive scalar can be treated as a special case of equation (2.1) with the velocity field **u** given. The only equation system that needs to be solved is (3.1).

the AIE scheme in the context of the flux-corrected transport (FCT) method

It was stated in Remark 3.1 that the AIE scheme does not have to be based on implicit-explicit grouping. It can be based on any element level, dynamic, and rational selection between an "I-procedure" and an "E-procedure". Here we give an example by employing the AIE scheme in conjunction with the FCT method described in [9,15]. We apply this method to the time-dependent convection-diffusion of a passive scalar (see Remark 3.4).

The FCT method given in [9,15] is based on the correction of the flux obtained with a lower-order scheme with the flux obtained with a higher-order scheme, while limiting the maximum allowable correction. The balance law is satisfied at the element level. In our case, as the lower-order scheme we use the one-pass explicit SUPG method and as the higher order scheme the multi-pass explicit SUPG method. With sufficient number of iterations (typically three) the multi-pass explicit method produces solutions indistinguishable from those obtained by the implicit method. In the utilization of our AIE scheme we define the "E-procedure" to be the one-pass explicit SUPG method alone and the "I-procedure" to be the FCT method based on the lower- and higher-order schemes described above.

Remark 3.5 It is not our intention here to make any statement about what lower- and higher-order schemes should be used with the FCT method or about whether the FCT method should be preferred over some other method. We just needed to pick an example.

For the AIE scheme we employ the "v-form" of the predictor/multi-corrector algorithm described by equation (2.6). That is

$$\mathbf{M}_L\,(\Delta \mathbf{v}_*)^i_{n+1} = \bar{\mathbf{R}}^i_{n+1}, \tag{3.12a}$$

where $\mathbf{M}_L$ is the (diagonal) lumped mass matrix version of $\tilde{\mathbf{M}}$ and

$$\bar{\mathbf{R}}^i_{n+1} = \Delta t\ \tilde{\mathbf{F}}_{n+1} - (\tilde{\mathbf{M}} + \alpha \Delta t \tilde{\mathbf{C}})\,(\mathbf{v}_*)^i_{n+1} + (\tilde{\mathbf{M}} - (1-\alpha)\Delta t\,\tilde{\mathbf{C}})\,(\mathbf{v}_*)_n. \tag{3.12b}$$

Note that these expressions are quite simpler due to the fact that this is a linear problem and that no stream function is involved.

The nodal value increment vector $(\Delta v_*)^i_{n+1}$ and the residual vector $\bar{R}^i_{n+1}$ can be written as sums of their element level contribution vectors

$$(\Delta v_*)^i_{n+1} = \sum_{e\in\varepsilon} (\Delta v^e_*)^i_{n+1}, \tag{3.13}$$

$$(\bar{R}^i_{n+1}) = \sum_{e\in\varepsilon} (\bar{R}^e)^i_{n+1}. \tag{3.14}$$

The AIE scheme employed is outlined below:

1. given $(v_*)_n$
   $i = 0$
2. $(\Delta v^e_*)^i_{n+1} = M_L^{-1} (\bar{R}^e)^i_{n+1} \quad \forall e\in \varepsilon$ (3.15)
3. $(v_*)^{i+1}_{n+1} = (v_*)^i_{n+1} + \sum_{e\in\varepsilon} (\Delta v^e_*)^i_{n+1}$ (3.16)
4. $i = i+1$
5. $(\Delta v^e_*)^i_{n+1} = M_L^{-1} (\bar{R}^e)^i_{n+1} \quad \forall e\in \varepsilon_I$ (3.17)
6. $(v_*)^{i+1}_{n+1} = (v_*)^i_{n+1} + \sum_{e\in\varepsilon_I} c^e (\Delta v^e_*)^i_{n+1}$ (3.18)
7. if $i < n_{it}$ then go to 4
8. $n = n+1$ and goto 1

where c^e is a parameter [9,15] determined by the maximum correction allowed for the element e and n_{it} is the number of iterations per time step for the higher-order solution.

Limiting the steps 5 and 6 to the group ε_I results in substantial savings in the computational cost involved.

solution of the discrete Poisson's equation

We use the preconditioned conjugate gradient method to solve the equation system given by (3.2). In conjunction with the AIE scheme employed for equation (3.1) we define our preconditioner matrix **P** to be

$$\mathbf{P} = \sum_{e\in\varepsilon_I} \mathbf{K}^e + \sum_{e\in\varepsilon_E} \mathrm{diag}(\mathbf{K}^e), \tag{3.19}$$

where $\mathbf{K}^e$ is the element contribution matrix of **K**. If $\varepsilon_E = \varnothing$ then $\mathbf{P} = \mathbf{K}$ and the solution technique becomes a direct one. If $\varepsilon_I = \varnothing$ then the method becomes a Jacobi iteration [2]. Other definitions for **P** are of course possible; the one given by (3.19) is simple to implement and reflects our belief that there should be a correlation between the solution procedures for (3.1) and (3.2).

Remark 3.6 The incompressible Navier-Stokes equations in the velocity-pressure formulation, upon spatial and temporal discretizations, can also translate to a set of equations given by (3.1) and (3.2). In this case equations (3.1) and (3.2) would

correspond, respectively, to the momentum equation and the incompressibility constraint. The latter one would consist of a discrete Poisson's equation for pressure.

Remark 3.7 Another example for the type of problems which can translate to equations (3.1) and (3.2) is the electrophoresis separation process (see [1]). In the context of a block-iteration scheme, the time-dependent transport equation for the chemical species and the equation for the electric potential would transform to equations (3.1) and (3.2), respectively.

4. The Grouped Element-by-Element (GEBE) Preconditioned Iteration Method

The linear equation systems given by (3.1) and (3.2) are both in the form

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{4.1}$$

In this section we describe our parallel GEBE-preconditioned iteration method for solving (4.1).

element grouping

We group the set of elements $\mathcal{E}$ in such a way that

$$1.\ \mathcal{E} = \bigcup_{K=1}^{N_{pg}} \mathcal{E}_K, \tag{4.2}$$

$$2.\ \emptyset = \mathcal{E}_J \cap \mathcal{E}_K \quad \text{for } J \neq K, \tag{4.3}$$

3. no two elements which belong to the same group can be neighbors (we define being neighbors as having at least one common node),

where N_{pg} is the number of such (parallelizable) groups.

Corresponding to this grouping, the matrix A can be written as

$$\mathbf{A} = \sum_{K=1}^{n_{pg}} \mathbf{A}_K, \tag{4.4}$$

where the "group matrices" are given as

$$\mathbf{A}_K = \sum_{e \in \mathcal{E}_K} \mathbf{A}^e\ ,\ K = 1,2,\ldots,\ N_{pg}. \tag{4.5}$$

Remark 4.1 Within each group, since there is no inter-element coupling, computations performed in an element-by-element fashion (for example such operations performed on a group matrix) do not depend on the ordering of the elements.

Remark 4.2 In our parallel computations, before we start with a new group we first have to finish with the current one. To minimize the overhead associated with this synchronization, we try to minimize the number of groups. For example, in a two-dimensional problem with rectangular domain and 100×100 mesh, there will be 4 groups with each group having 2500 elements and arranged in a checkerboard style. A simple element grouping algorithm for arbitrary meshes is described below:

initialization:

$e=1$	(start with the first element)
$N_{pg} = 1$	(create the first group)
$e \in \mathcal{E}_{Npg}$	(put the element in that group)

next element:
e = e+1 (take the next element)
if e > n_{el} then return (if there is no such element return)
K = 0 (otherwise get ready to search for an eligible group among the existing groups, starting with the first existing group)
next existing group:
K = K+1 (take the next existing group)
if e has no neighbors in $\mathcal{E}_K$ (check if the element can be put in that group)
then
$e \in \mathcal{E}_K$ (if so, do it)
go to the next element
else if K< N_{pg} (otherwise check if there is a next existing group)
then
go to next existing group (if so, try that next group)
end if
create a new group:
$N_{pg} = N_{pg} + 1$ (otherwise create a new group)
$e \in \mathcal{E}_{Npg}$ (and put the element in that group)
go to the next element

Remark 4.3 Within each group, to increase the vector efficiency of the computations performed, elements are processed in chunks of 64 elements (or whatever the optimum chunk size is for the given vector environment). For the example of Remark 4.2 each group would have 40 chunks (with an assumed chunk size of 64 elements).

GEBE-preconditioned iterative solution

Equation (4.1) is solved iteratively as follows:

$$\mathbf{P}\,\Delta\mathbf{y}_m = \mathbf{r}_m, \tag{4.6}$$

where the residual vector is defined as

$$\mathbf{r}_m = \mathbf{b} - \mathbf{A}\,\mathbf{x}_m, \tag{4.7}$$

and the two-pass GEBE-preconditioning matrix is given as

$$\mathbf{P} = \mathbf{D}^{-1} \left(\prod_{K=1}^{N_{pg}} \mathbf{E}_K \prod_{K=N_{pg}}^{1} \mathbf{E}_K \right) \mathbf{D}^{-1}, \tag{4.8}$$

with the scaling matrix

$$\mathbf{D} = \frac{(\Delta\theta)^{1/2}}{\mathbf{W}^{1/2}} . \tag{4.9}$$

Here $\Delta\theta$ is a free parameter and for $\mathbf{W}$, depending on the properties of $\mathbf{A}$, we can pick one of the following two choices:

$$\mathbf{W} = \text{diag}\,\mathbf{A}\,, \qquad ([5]) \tag{4.10}$$

and

$$\mathbf{W} = \mathbf{M}_L\,. \qquad ([8]) \tag{4.11}$$

Note that equation (4.9) needs $\mathbf{W}$ to be positive-definite. While the choice of (4.10) does not guarantee this when $\bar{\mathbf{M}}$ is not positive-definite, the alternative choice of (4.11) does. For problems in fluid mechanics $\bar{\mathbf{M}}$ is not in general positive-definite. A certain type of streamline-upwind/Petrov-Galerkin method, however, in some cases result in $\bar{\mathbf{M}}$ being symmetric and positive-definite.

The matrix $\mathbf{E}_K$ is defined as

$$\mathbf{E}_K = (\mathbf{I} + \frac{1}{2}\mathbf{D}\mathbf{B}_K\mathbf{D}) , \qquad K=1,2,...,N_{pg}, \tag{4.12}$$

where $\mathbf{I}$ is the identity matrix. Considering that there is no inter-element coupling within each group, $\mathbf{E}_K$ can also be written as follows:

$$\mathbf{E}_K = \prod_{e\in\varepsilon_k} (\mathbf{I} + \frac{1}{2}\mathbf{D}\mathbf{B}^e\mathbf{D}) , \qquad K=1,2,...,N_{pg}. \tag{4.13}$$

For the element matrix $\mathbf{B}^e$ we consider the following two choices:

$$\mathbf{B}^e = \mathbf{A}^e, \tag{4.14}$$

and

$$\mathbf{B}^e = \mathbf{A}^e - \frac{\mathbf{W}^e}{\Delta\theta} \qquad \text{(Winget regularization [2]).} \tag{4.15}$$

Remark 4.4 The grouped EBE approach is just a way of rewriting the regular EBE formulation. The rewriting is based on arranging the elements into parallelizable groups. We can say that the GEBE formulation is "parallel-ready".

Remark 4.5 The expression given by (4.8) depends on the ordering of the groups. This is the only expression that depends on any ordering. The expression given by (4.13) does not depend on the element ordering. We can therefore conclude that the EBE schemes depend on the ordering of the groups but not on the ordering of the elements in a strict sense. The corollary is that the number of possible ordering combinations is $(N_{pg})!$ but not $(n_{el})!$. There may be some degree of non-uniqueness in the grouping of the elements but we assume this non-uniqueness to be a minor one (nonexistent for structured meshes).

The updated value of $\mathbf{x}$ is computed according to the following formula:

$$\mathbf{x}_{m+1} = \mathbf{x}_m + s\,\Delta\mathbf{y}_m, \tag{4.16}$$

where s is the search parameter determined, depending on the properties of $\mathbf{A}$, as follows:

$$s = \frac{\Delta\mathbf{y}_m \bullet \mathbf{r}_m}{\Delta\mathbf{y}_m \bullet \mathbf{A}\Delta\mathbf{y}_m} \quad \text{(if } \mathbf{A} \text{ is symmetric positive-definite),} \tag{4.17a}$$

or

$$s = \frac{(\mathbf{A}\Delta\mathbf{y}_m)\bullet\mathbf{r}_m}{\| \mathbf{A}\Delta\mathbf{y}_m\|^2} \quad \text{(if } \mathbf{A} \text{ is not symmetric positive-definite).} \tag{4.17b}$$

The latter expression for s is obtained by minimizing $\|\mathbf{r}_{m+1}\|^2$ with respect to s.

Remark 4.6 If **A** is symmetric positive-definite, the preconditioned conjugate-gradient algorithm can be used with the preconditioner given by equation (4.8). This is what we do to solve equation (3.2).

5. Remarks on the Vectorization and Parallel Processing

In a multi-processor vector machine each processor is an individual vector processor. The single vector processor can gain high performance through overlapping multiple operations using segmented functional units. When a program is restructured or vectorized to facilitate this overlapping, the speed can be improved by an order of magnitude. Performance improvements from vectorization can be augmented through the use of parallel processing in a multi-processor machine. Parallel processing allows a single program to use more than one CPU to do the required work. This results in a reduction of the elapsed time needed to do the job. The amount of reduction is proportional to the percentage of the program that can be executed independently. In our implementation we have, so far, focus on the vectorization and parallel processing of element level matrix/vector calculations, factorization and back substitution, and global residual calculations.

All the above calculations have been vectorized in this work. The structure of the vectorized code contains an inner loop over elements that belong to particular chunks of elements with identical calculations. An important implementation point is the restructuring of the code to enable compiler optimization. A specific example is the partitioning of the element level matrix/vector calculations into multiple loops. Of course this introduces additional storage requirements as we need temporary arrays to hold the intermediate computational results across loops. Another example is the deployment of CVMxx, CRAY compiler extension for vectorizing IF construct.

Additionally, all calculations are parallel processed using Microtasking, a compiler directive driven preprocessor for parallel processing on the CRAY. Microtasking is implemented with comment card directives. Before compiling the code, a preprocessor expands the directives into Fortran code which includes calls to the Microtasking library. It is important to note that if the preprocessor is not used, the code is portable to other machines since the Microtasking directives appear as Fortran comment cards. The code is currently being used for benchmarking on additional architectures (e.g., IBM 3090/VF). This is the subject of a future report.

If there are n_{CPU} CPUs available for parallel processing, we would like to distribute the work among all CPUs in such a way that large task granularity, good load balance and synchronization are satisfactorily achieved. The parallel performance gain must be considered against the deterioration of the vectorization performance.

In the implementation of the parallel GEBE procedure, within each group, elements are sub-grouped into n_{CHUNK} chunks with each chunk containing n_{VR} (e.g. 64) elements. The last chunk in each group may contain less than n_{VR} elements. We choose n_{CHUNK} and n_{VR} in such a way that n_{CHUNK} is a multiple of n_{CPU}. When parallel processing is invoked (with a compiler directive in the outer loop over n_{CHUNK}), each of the available n_{CPU} CPUs will work on one chunk at a time until all chunks are finished. Because each chunk contains the same amount of work (except for the last chunk) and the overall number of chunks can evenly be distributed to n_{CPU} CPUs, load balancing is ensured.

6. Numerical Tests

We have tested the AIE and GEBE methods on problems governed by the convection-diffusion and the two-dimensional incompressible Navier-Stokes equations.

one-dimensional pure advection of a cosine wave

The purpose of this simple test is to provide a conceptual illustration of the way the AIE scheme works. A cosine wave of unit amplitude is being advected from the left to the right with an advection velocity of 1.0. A uniform mesh is being used and the element Courant number is 0.8.

The AIE scheme is based on the selection between the implicit and the one-pass explicit versions of the streamline-upwind/Petrov-Galerkin (SUPG) procedure. The threshold value for the criterion given by equation (3.8) is set to 1%. Figure 1 shows the solution and the corresponding distribution of the implicit elements at time steps 0, 50, and 100. Compared to the implicit method, the AIE scheme results in a 79% reduction in the CPU time and 81% in the memory needed for the global matrix; the solutions obtained by the implicit and the AIE schemes are indistinguishable.

one-dimensional pure advection of a discontinuity

The conditions in this case are nearly identical to those in the previous case. This time the profile which is being advected is a discontinuity which initially spans four elements. The AIE scheme is based on the selection between the one-pass explicit SUPG method and the FCT method implemented as described in Section 3. The threshold value given by (3.8) is set to 0.01%. Figure 2 shows the solution and the corresponding distribution of the FCT elements, at time steps 0, 50, and 100. Compared to the "full" FCT approach the savings in the CPU time is 42%.

two-dimensional pure advection of a plateau

This test is for evaluating the performance of the AIE scheme for two-dimensional problems with moving sharp fronts. A 40×40 mesh is chosen in a 1.0×1.0 computational domain. All the boundary conditions are the Dirichlet type. The advection velocity is in the diagonal direction and has unit magnitude. The time step is 0.01.

The AIE scheme is based on the selection between the implicit and the one-pass explicit versions of the discontinuity-capturing SUPG procedure [14]. The threshold value for the criterion given by (3.8) is 1%. Figure 3 shows the solution and the distribution of the implicit elements at time steps 0, 20, and 40. Compared to the implicit method, the AIE scheme results in a 48% reduction in the CPU time and 79% in the memory needed for the global matrix.

flow past a circular cylinder at Reynolds number 100

In this problem both the AIE and the GEBE schemes were tested. We used a mesh with 1940 elements and 2037 nodal points. The dimensions of the computational domain, normalized by the cylinder diameter, are 16 and 8 in the flow and the cross flow directions, respectively. Figure 4 shows the finite element mesh and its element-grouped version ("GEBE mesh") for the parallel computations. The free-stream velocity is 0.125 and the initial condition for the vorticity is zero everywhere in the domain. The time step is 1.0.

The AIE scheme employed to solve the vorticity transport equation and the Poisson's equation is exactly as described in Section 3. For the vorticity transport equation, both the implicit and the explicit methods are SUPG based and involve several iterations due to the nonlinearity of the equation. In this problem the Courant number for some elements is above the stability limit of the one-pass explicit algorithm (we estimate the maximum Courant number to be somewhere around 1.5-1.6). Therefore the stability criterion given by equation (3.7) also becomes active in the implicit-explicit grouping. In fact our numerical experiments show that the computations do not converge when the entire set of elements belong to the explicit group. For the accuracy considerations, the threshold value of the criterion given by (3.10) is set to 1%. For solution of the discrete Poisson's equation, the iterations continue until the residual norm goes below 10^{-8} (we realize that this is rather an overkill).

Both the AIE and the GEBE methods give the expected solution for this problem: initially a symmetric flow pattern with two attached eddies growing in the wake of the cylinder, then the symmetry breaks, and as time goes by the vortices are formed alternately at the upper and lower downstream vicinity of the cylinder and carried along the Karman vortex street. Figures 5,6, and 7 show the solution obtained by the AIE scheme and the corresponding distributions of the implicit elements at time 680, 1500, and 2000.

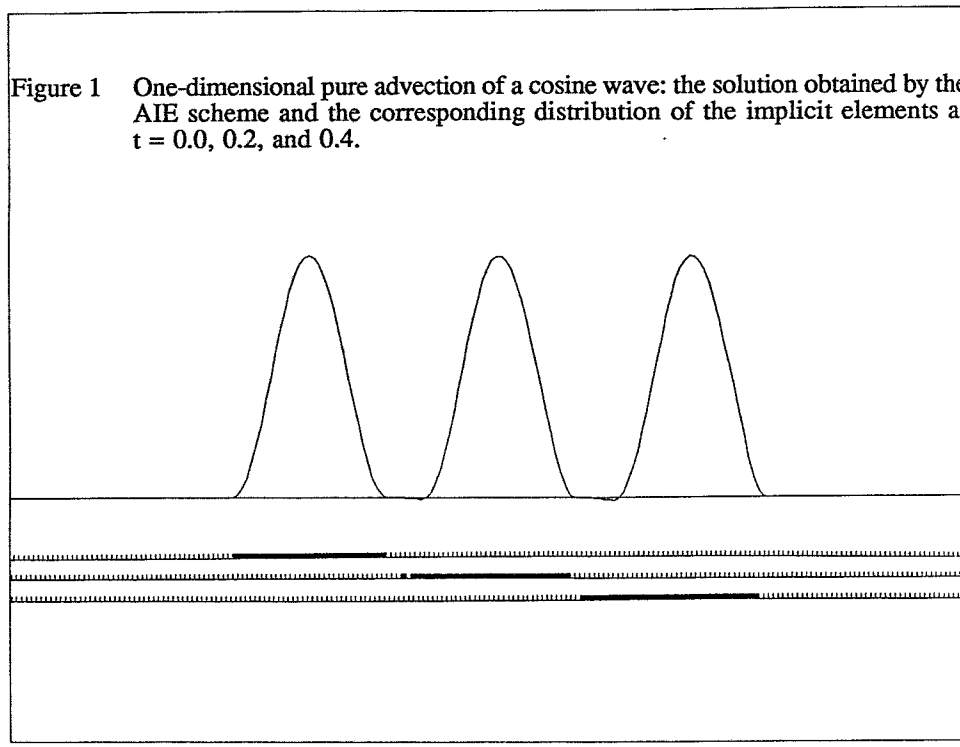

Figure 1 One-dimensional pure advection of a cosine wave: the solution obtained by the AIE scheme and the corresponding distribution of the implicit elements at t = 0.0, 0.2, and 0.4.

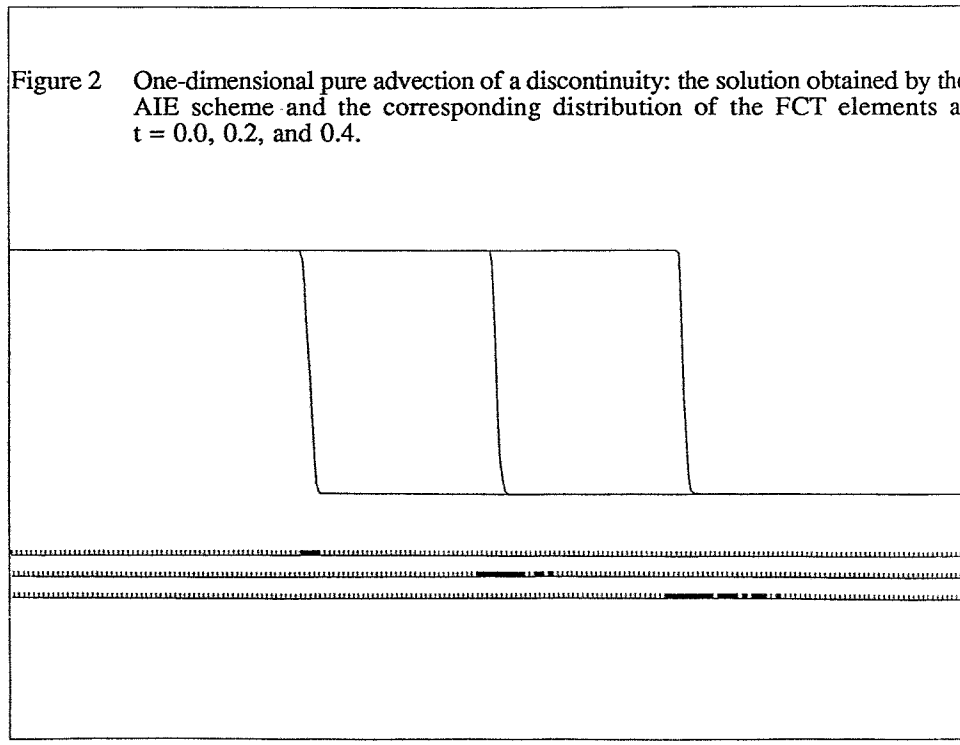

Figure 2 One-dimensional pure advection of a discontinuity: the solution obtained by the AIE scheme and the corresponding distribution of the FCT elements at t = 0.0, 0.2, and 0.4.

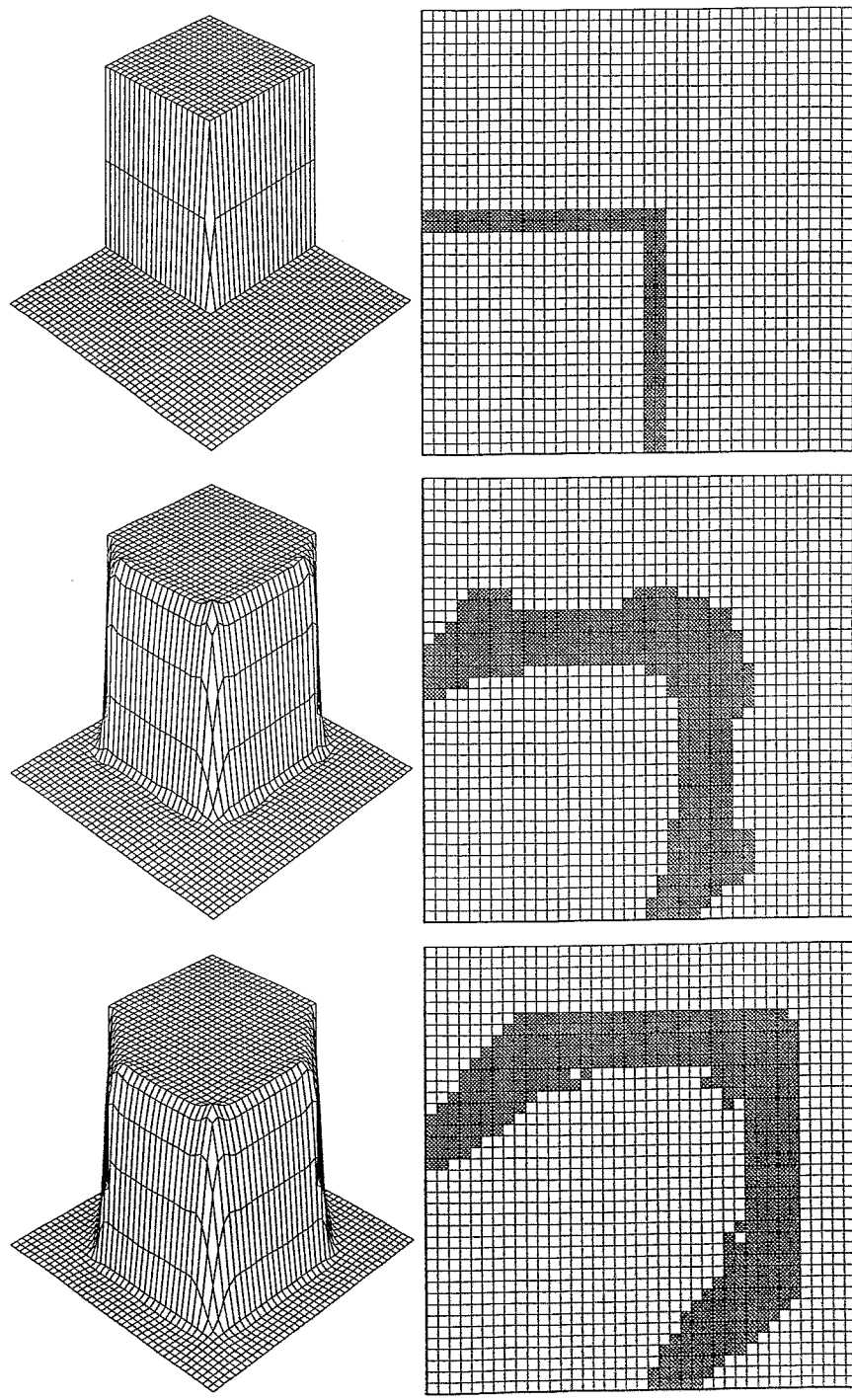

Figure 3 Two-dimensional pure advection of a plateau: the solution obtained by the AIE scheme and the corresponding distribution of the implicit elements at t = 0.0, 0.2, and 0.4.

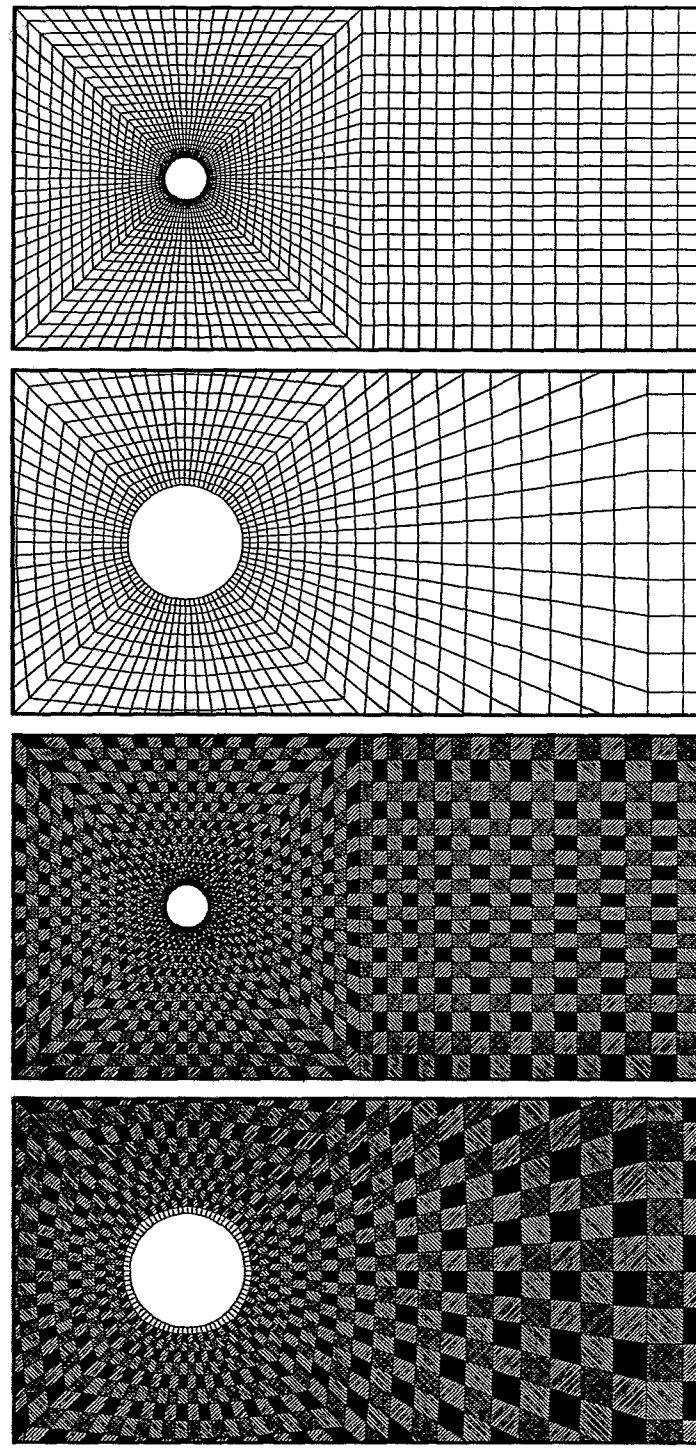

Figure 4 Flow past a circular cylinder at Reynolds number 100: the finite element mesh and its element grouped version ("GEBE mesh") for the parallel computations.

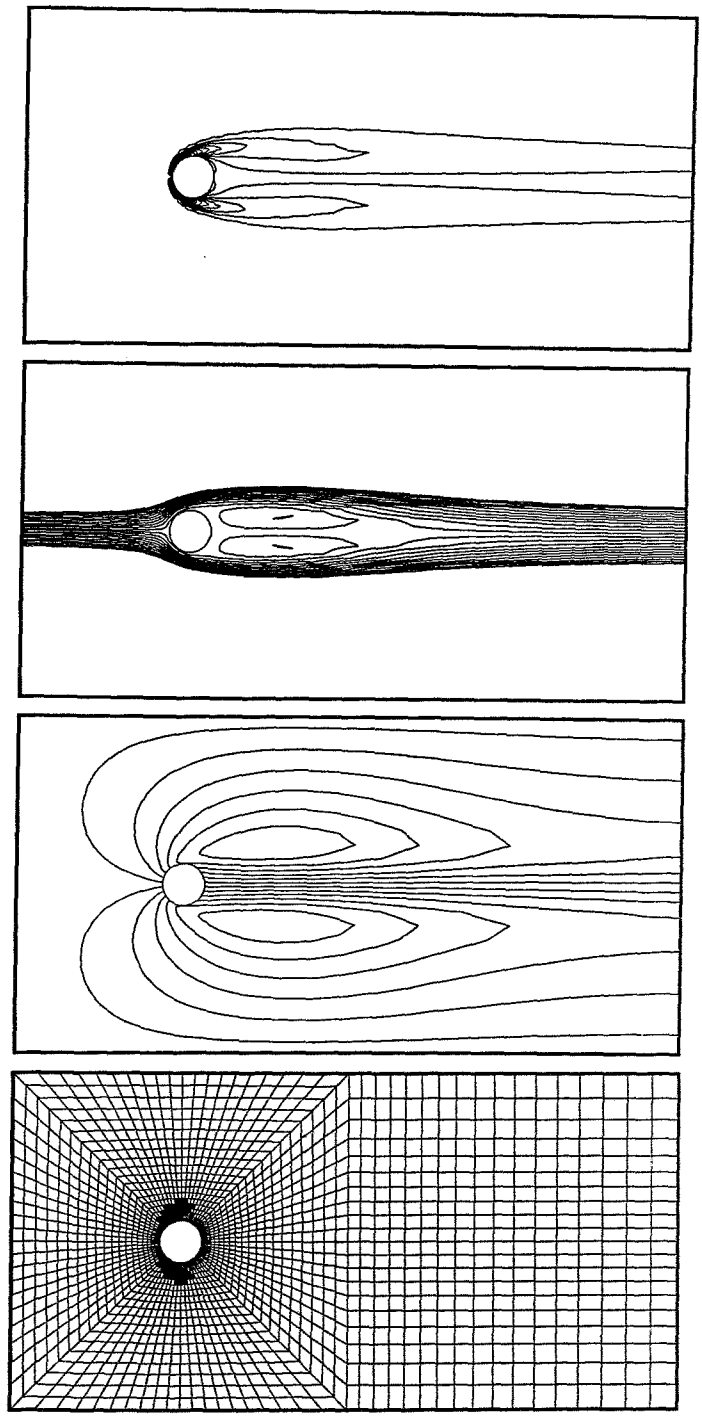

Figure 5 Flow past a circular cylinder at Reynolds number 100: the symmetrical solution obtained by the AIE scheme at t = 680; from top to bottom: vorticity, streamlines, relative streamlines, and distribution of the implicit elements.

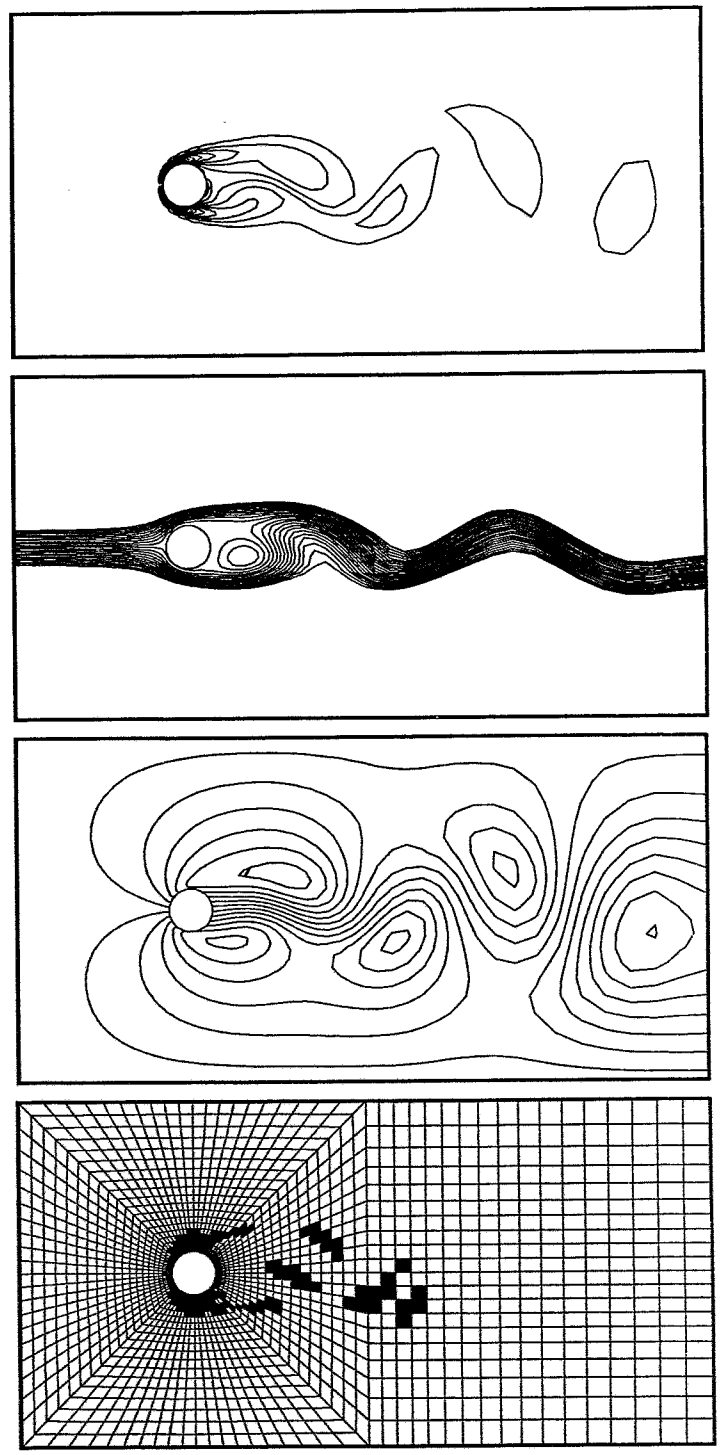

Figure 6 Flow past a circular cylinder at Reynolds number 100: the solution obtained by the AIE scheme at t = 1500; from top to bottom: vorticity, streamlines, relative streamlines, and distribution of the implicit elements.

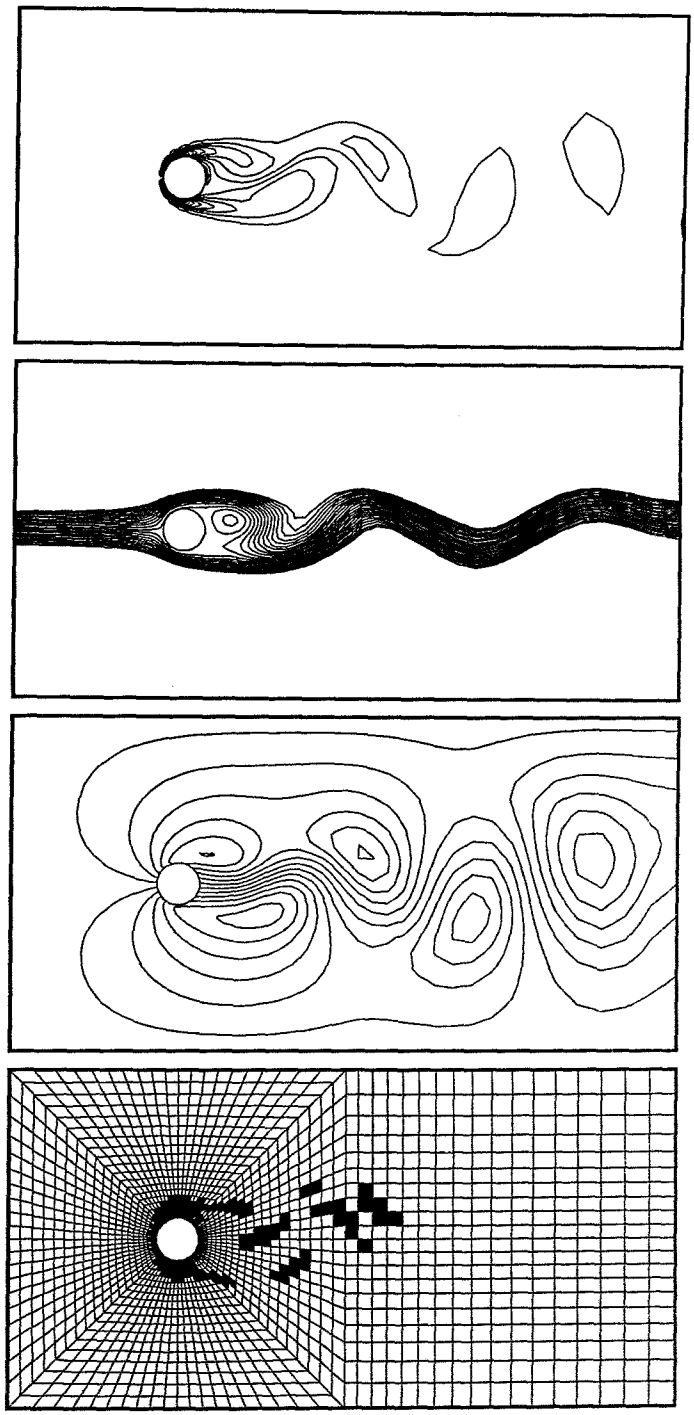

Figure 7 Flow past a circular cylinder at Reynolds number 100: the solution obtained by the AIE scheme at t = 2000; from top to bottom: vorticity, streamlines, relative streamlines, and distribution of the implicit elements.

The AIE method reduces the memory needed for the global matrices by 85%. The CPU time however is more than that of the implicit scheme. This is because, for the solution of the vorticity transport equation, the nonlinear iterations continue until a certain convergence criterion is met; this convergence criterion is the same whether it is the implicit or the AIE scheme. That is, for the solution of the vorticity transport equation, compared to the implicit method, the AIE scheme is not giving up any accuracy even in the explicit zones.

In the case of the GEBE computations, one has to note that for the first layer of elements around the cylinder computations can not be fully parallel. This is because of the way the problem is formulated to determine the unknown stream function value corresponding to this internal boundary. The tests for the GEBE computations were performed on a CRAY-XMP/416 : 4 CPUs, 8.5 ns clock, 128 Megabytes of memory, COS 1.16 operating system and products (Cray Research Mendota Heights data center).

The benchmark test lasted for 10 time steps with several non-linear iterations per time step. The vector performance reached 100 Mflops for the element level matrix/vector computations, 70 Mflops for the factorization and back substitution, and 25 Mflops for the global residual calculation. The overall program performance was at 48 Mflops. We expect to obtain higher performance rates as we vectorize the program further and filter out the initial setup/overhead cost. Parallel processing on the element level matrix/vector computations rendered speed ups of 3.5-3.8. The speed up on the factorization/back substitution and global residual calculation is around 2-3. This is due to the fact that the task granularity is small because of the relatively small size of the test problem.

7. Concluding Remarks

In this paper we have presented some efficient solution techniques for large equation systems emanating from the finite element formulation of fluid mechanics problems.

In the adaptive implicit-explicit (AIE) method the elements are dynamically grouped into implicit and explicit subsets based on the element level stability and accuracy criteria. This dynamic grouping idea can be applied in any context where there are some reasons for preferring a given procedure over another one. As an example we implemented the AIE concept in conjunction with the flux-corrected transport method.

The grouped element-by-element (GEBE) iteration scheme, which is a variation of the regular EBE scheme, is based on defining the preconditioner to be a sequential product of the element group matrices. To facilitate efficient parallel processing, the number of groups is minimized with the condition that there can be no inter-element coupling within each group.

We applied these methods to various test problems and demonstrated that substantial savings in the CPU time and memory can be achieved.

In our future studies we plan to experiment with the combinations of the AIE and GEBE schemes. For example we can use the GEBE scheme to solve the discrete Poisson's equation for the stream function (or the pressure), employ the AIE scheme to solve the time-dependent transport equation for the vorticity (or the momentum), but use, again, the GEBE scheme at the implicit zones of the implicit-explicit distribution.

Acknowledgement

We are grateful for the CRAY time provided by Cray Research Inc.

References

(1) D.K. Ganjoo, W.D. Goodrich, and T.E. Tezduyar, *A New Formulation for Numerical Simulation of Electrophoresis Separation Processes*, UMSI 88/37, April, 1988, University of Minnesota Supercomputer Institute.

(2) R. Glowinski, *Numerical Methods for Nonlinear Variational Problems*, Springer-verlag, New York, 1984.

(3) R. Glowinski, Q.V. Dinh, and J. Periaux, *Domain Decomposition Methods for Nonlinear Problems in Fluid Dynamics*, Computer Methods in Applied Mechanics and Engineering, 40 (1983), pp. 27-109.

(4) T.J.R. Hughes, R. M. Ferencz, *Fully Vectorized EBE Preconditioners for Nonlinear Solid Mechanics: Applications to Large-Scale Three-Dimensional Continuum, Shell and Contact/Impact Problems*, First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G.H. Golub, G.A. Meurant, and J. Periaux (eds.), SIAM, 1988, pp. 261-280.

(5) T.J.R. Hughes, I. Levit and J.Winget, *An Element-by-Element Solution Algorithm for Problems of Structural and Solid Mechanics*, Computer Methods in Applied Mechanics and Engineering, 36 (1983), pp. 75-110.

(6) T.J.R. Hughes and W.K. Liu, *Implicit-Explicit Finite Elements in Transient Analysis: Stability Theory*, Journal of Applied Mechanics, 45 (1978), pp. 371-374.

(7) T.J.R. Hughes and W.K. Liu, *Implicit-Explicit Finite Elements in Transient Analysis: Implementation and Numerical Examples*, Journal of Applied Mechanics, 45 (1978), pp. 375-378.

(8) T.J.R. Hughes, J. Winget, I. Levit, and T.E. Tezduyar, *New Alternating Direction Procedures in Finite Element Analysis Based upon EBE Approximation Factorizations*, in Computer Methods for Nonlinear Solids and Mechanics, S.N. Atluri, and N. Perrone (eds.), AMD Vol. 54, ASME, New York, 1983, pp. 75-110.

(9) R. Lohner, K. Morgan, J. Peraire, and M. Vahdati, *Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations*, Finite Elements in Fluids, 7 (1987), R.H. Gallagher, R. Glowinski, P.M. Gresho, J.T. Oden, and O.C. Zienkiewicz (eds.), John Wiley & Sons Ltd., pp. 105-121.

(10) T.E. Tezduyar and D.K. Ganjoo, *Petrov-Galerkin Formulations with Weighting Functions Dependent upon Spatial and Temporal Discretization: Application to Transient Convection-Diffusion Problems*, Computer Methods in Applied Mechanics and Engineering, 59 (1986), pp. 47-71.

(11) T.E. Tezduyar, R. Glowinski, and J. Liou, *Petrov-Galerkin Methods on Multiply-Connected Domains for the Vorticity - Stream Function Formulation of the Incompressible Navier-Stokes Equations*, to appear in the International Journal of Numerical Methods in Fluids.

(12) T.E. Tezduyar, R. Glowinski, J. Liou, T. Nguyen, and S. Poole, *Block-Iterative Finite Element Computations for Incompressible Flow Problems* , to appear in the Proceedings of the 1988 International Conference on Supercomputing, Saint-Malo, France, July 1988.

(13) T.E. Tezduyar and J. Liou, *Element-by-Element and Implicit-Explicit Finite Element Formulations for Computational Fluid Dynamics* , First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G.H. Golub, G.A. Meurant, and J. Periaux (eds.), SIAM, 1988, pp. 281-300.

(14) T.E. Tezduyar and Y.J. Park, *Discontinuity-Capturing Finite Element Formulations for Nonlinear Convection-Diffusion-Reaction Equations*, Computer Methods in Applied Mechanics and Engineering, 59 (1986), pp. 307-325.

(15) S.T. Zalesak, *Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids*, Journal of Computational Physics, 31 (1979), pp. 335-362.