INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **29**: 289–309 (1999)

SIMULATION OF FREE AND FORCED CONVECTION INCOMPRESSIBLE FLOWS USING AN ADAPTIVE PARALLEL/VECTOR FINITE ELEMENT PROCEDURE

P.A.B. DE SAMPAIO[a,]* AND A.L.G.A. COUTINHO[b]

[a] *Instituto de Engenharia Nuclear, Comissão Nacional de Energia Nuclear, Rio de Janeiro-RJ, 21945-910, Brazil*
[b] *Programa de Engenharia Civil-COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro-RJ, 21945-970, Brazil*

SUMMARY

The numerical simulation of complex flows demands efficient algorithms and fast computer platforms. The use of adaptive techniques permits adjusting the discretisation according to the analysis requirements, but creates variable computational loads that are difficult to manage in a parallel/vector program. This paper describes the approach adopted to implement an adaptive finite element incompressible Navier–Stokes solver on the Cray J90 machine. Performance measurements for the simulation of free and forced convection incompressible flows indicate that the techniques employed result in a fast parallel/vector code. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: Petrov–Galerkin methods; incompressible flow; Navier–Stokes equations; parallel/vector computations; adaptive methods

1. INTRODUCTION

Analytical solutions of fluid flow problems are restricted to a few situations where simplified models and hypotheses are applicable. In order to match the engineering needs to deal with flows of industrial complexity, numerical and experimental techniques are required. Though the laboratory and the computer have been complementary tools in understanding fluid flow phenomena, one can clearly identify the growing importance of computational fluid dynamics (CFD) in the technological development of recent decades. The expanding role played by numerical analysis of flow problems is prompted by economical reasons, which are in turn connected to the development of computer technology and new solution algorithms.

This paper addresses the adaptive simulation of free and forced convection incompressible flows with special emphasis on the efficient use of the parallel/vector resources of the Cray J90 machine. The *continuum* model is based on the incompressible Navier–Stokes equations, written in primitive variables, and includes the Boussinesq approximation of thermally induced buoyancy forces. This class of flow problems is important in various branches of engineering and, in particular, in the design and safety analysis of the heat transfer equipment of nuclear power plants.

* Correspondence to: Instituto de Engenharia Nuclear, Comissão Nacional de Energia Nuclear, Rio de Janerio-RJ, 21945-910, Brazil.

CCC 0271–2091/99/030289–24$17.50
Received November 1996
Revised March 1998

Two major difficulties arise when attempting to solve the incompressible Navier–Stokes equations in primitive variables. The first arises due to the presence of convective terms that render these equations non-self-adjoint. For convective problems, the Galerkin method loses the *best approximation property* [1] and consistent Petrov–Galerkin formulations, which successfully supersede the Galerkin method, have been derived and used in the analysis of convection-dominated flows [2,3]. The second difficulty is the need to choose compatible interpolations for velocity and pressure when using the standard mixed formulation, as dictated by the Babuška–Brezzi condition [4]. Indeed, this condition rules out the use of equal-order interpolation for all variables, which would be attractive from a computational point of view. Here again, the utilisation of Petrov–Galerkin formulations is advantageous, and stable solutions can be obtained for equal-order spatial interpolation of the primitive variables [5–7]. In Section 2, the symmetric Petrov–Galerkin formulation, applied for the discretisation of the incompressible Navier–Stokes equations and for the fluid energy balance [8], is presented. The method automatically introduces *streamline upwinding* [3] and permits equal-order interpolation for velocity and pressure.

Accurate, and yet affordable, computation of complex flows demands the use of adaptive solution techniques. Adaptive solution procedures, combining mesh and time step control, are described in Sections 3 and 4. These include remeshing, based on the Zienkiewicz and Zhu error estimator [9], and the use of local time steps defined by the local velocity, physical properties and mesh size [10,11]. Such techniques are effective on adjusting the discretisation according to the physics of the underlying problem. However, they create variable computational loads that make them difficult to implement with efficiency in parallel/vector supercomputers.

The parallel/vector implementation on the Cray J90 installed at COPPE/UFRJ is described in Section 5. A dynamic mesh colouring scheme is applied to the computational meshes in order to split the finite elements into groups that can be processed in parallel/vector mode within an element-by-element (EBE) preconditioned conjugate gradient solver [12,13].

Finally, adaptive simulations of some representative free and forced convection flows are presented in Section 6. Performance measurements on the Cray J90 demonstrate the effectiveness of the numerical techniques employed.

2. THE CONTINUUM AND DISCRETE MODELS

The finite element formulation used for the simulation of incompressible viscous flows with heat transfer is presented. The problem is defined on the open-bounded domain Ω, with boundary Γ, contained in the *nsd*-dimensional Euclidean space. The continuum model comprises the incompressible Navier–Stokes equations and the transient energy balance. The Boussinesq approximation of buoyancy forces is used to model free and mixed convection flows.

The governing equations are written using the summation convention for $a = 1, \ldots, nsd$ and $b = 1, \ldots, nsd$, in Cartesian co-ordinates, as

$$\rho_0\left(\frac{\partial u_a}{\partial t} + u_b\frac{\partial u_a}{\partial x_b}\right) - \frac{\partial \tau_{ab}}{\partial x_b} + \frac{\partial p}{\partial x_a} + \rho_0\beta g_a\theta = 0, \tag{1}$$

$$\frac{\partial(\rho_0 u_a)}{\partial x_a} = 0, \tag{2}$$

$$\rho_0 c\left(\frac{\partial \theta}{\partial t}+u_b\frac{\partial \theta}{\partial x_b}\right)+\frac{\partial q_b}{\partial x_b}=0, \tag{3}$$

where the viscous stress and the heat flux are given by

$$\tau_{ab}=\mu\left(\frac{\partial u_a}{\partial x_b}+\frac{\partial u_b}{\partial x_a}\right) \tag{4}$$

and

$$q_b=-\kappa\frac{\partial \theta}{\partial x_b} \tag{5}$$

respectively.

In the above equations, u_a, p and θ denote the velocity, pressure and temperature fields. The fluid density at $\theta=0$ is represented by ρ_0. Note that c, μ, κ, β are respectively, the fluid specific heat, viscosity, thermal conductivity and thermal expansion coefficient. The gravity field Cartesian components are denoted by g_a.

The model is completed by introducing boundary conditions and initial velocity and temperature fields. Velocity and traction boundary conditions are prescribed on the partitions Γ_{ua} and Γ_{ta}, such that $\Gamma_{ua}\cup\Gamma_{ta}=\Gamma$ and $\Gamma_{ua}\cap\Gamma_{ta}=\emptyset$, as

$$u_a=\bar{u}_a(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_{ua}, \tag{6}$$

$$(-p\delta_{ab}+\tau_{ab})\boldsymbol{n}_b=\bar{t}_a(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_{ta}, \tag{7}$$

where δ_{ab} is the Kronecker delta and $\boldsymbol{n}_b$ denotes the outward normal vector at the boundary.

Temperature and heat flux boundary conditions are prescribed on the partitions Γ_θ and Γ_q, such that $\Gamma_\theta\cup\Gamma_q=\Gamma$ and $\Gamma_\theta\cap\Gamma_q=\emptyset$, as

$$\theta=\bar{\theta}(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_\theta, \tag{8}$$

$$q_b\boldsymbol{n}_b=\bar{q}(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_q. \tag{9}$$

Pressure and mass flux boundary conditions are associated with the mass balance. They are prescribed as $\bar{p}$ and $\bar{G}$ on boundary partitions Γ_p and Γ_G such that $\Gamma_p\cup\Gamma_G=\Gamma$ and $\Gamma_p\cap\Gamma_G=\emptyset$, according to

$$p=\bar{p}(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_p, \tag{10}$$

$$\rho_0 u_b\boldsymbol{n}_b=\bar{G}(\mathbf{x},t) \quad \mathbf{x}\in\Gamma_G. \tag{11}$$

Note that the incompressible flow model (Equations (1)–(3)) involves pressure gradients but not pressure itself. Thus, at least one reference pressure value must be prescribed in order to define a unique pressure field.

The model presented in this section describes laminar and turbulent incompressible flows of Newtonian fluids. It includes thermally induced buoyancy forces that render it suitable for the analysis of free and mixed convection problems. As far as turbulent flows are concerned, the difficulty is related to the characteristic time and length scales of turbulence. Although these are large enough for the *continuum* hypothesis to remain valid, they are too small to be *resolved* by an affordable discretisation. Therefore, a turbulent analysis requires recasting the model through the use of Reynolds-averaged conservation laws and *pseudo* constitutive equations (the so-called closure models). This work shall not deal with such flows, but it is anticipated that the discretisation and solution techniques to be described next apply to most Reynolds-averaged turbulent models and large eddy simulation procedures currently used.

2.1. The symmetric Petrov–Galerkin formulation

The continuum model is discretised using standard C_0 finite elements of equal-order to approximate velocity, pressure and temperature. Such a choice of interpolating spaces is not acceptable within the mixed formulation framework, as it violates the Babuška–Brezi condition [4]. However, the Petrov–Galerkin formulation presented avoids this difficulty through the introduction of extra stabilising terms [5–7]. The formulation also leads to adequate approximations of convection-dominated flows, for it generates *streamline upwinding* [3].

In order to derive the method, the momentum and energy-squared residuals are defined as

$$S = \int_{\Omega} \hat{F}_a \hat{F}_a \, d\Omega, \tag{12}$$

$$R = \int_{\Omega} \hat{E}^2 \, d\Omega, \tag{13}$$

where

$$\hat{F}_a = \rho_0 \left(\frac{\hat{u}_a^{n+1} - \hat{u}_a^n}{\Delta t} + \hat{u}_b^n \frac{\partial \hat{u}_a^{n+1/2}}{\partial x_b} \right) - \frac{\partial \tau_{ab}^n}{\partial x_b} + \frac{\partial \hat{p}^{n+1/2}}{\partial x_b} + \rho_0 \beta g_a \hat{\theta}^n \tag{14}$$

and

$$\hat{E} = \rho_0 c \left(\frac{\hat{\theta}^{n+1} - \hat{\theta}^n}{\Delta t} + \hat{u}_b^n \frac{\partial \hat{\theta}^{n+1/2}}{\partial x_b} \right) + \frac{\partial q_b^n}{\partial x_b}. \tag{15}$$

In the above equations, $\hat{u}_a$, $\hat{p}$ and $\hat{\theta}$ are the discretised fields, interpolated using element shape functions denoted by N_i. The velocity, pressure and temperature nodal values are represented by u_{ai}, p_i and θ_i respectively. The superscripts n, $n+1/2$ and $n+1$ indicate the time level and Δt is the time step. Note that the viscous stress and heat flux contributions are taken at level n. For the time being, no particular spatial discretisation for τ_{ab}^n and q_b^n will be specified and the viscous and heat flux contributions will be treated as source terms.

The minimisation of the squared momentum residuals (12) and (13) with respect to the free nodal variables at level $n+1$ leads to the following weighted residual equations:

$$\int_{\Omega} \left(N_i + \frac{\Delta t}{2} u_c^n \frac{\partial N_i}{\partial x_c} \right) \hat{F}_a \, d\Omega = 0 \quad \forall \text{ free } u_{ai}^{n+1}, \tag{16}$$

$$\int_{\Omega} \Delta t \frac{\partial N_i}{\partial x_a} \hat{F}_a \, d\Omega = 0 \quad \forall \text{ free } p_i^{n+1}, \tag{17}$$

$$\int_{\Omega} \left(N_i + \frac{\Delta t}{2} u_c^n \frac{\partial N_i}{\partial x_c} \right) \hat{E} \, d\Omega = 0 \quad \forall \text{ free } \theta_i^{n+1}. \tag{18}$$

The traction and heat flux boundary conditions given by Equations (7)–(9) are approximated by

$$\int_{\Gamma_{ta}} N_i (-\hat{p}^{n+1/2} \delta_{ab} + \tau_{ab}^n) \boldsymbol{n}_b \, d\Gamma = \int_{\Gamma_{ta}} N_i \bar{t}_a \, d\Gamma \tag{19}$$

and

$$\int_{\Gamma_q} N_i q_b^n \boldsymbol{n}_b \, d\Gamma = \int_{\Gamma_q} N_i \bar{q} \, d\Gamma. \tag{20}$$

Using Equations (19) and (20), Equations (16) and (18) yield respectively,

$$\begin{aligned}
&\int_\Omega \frac{\rho_0}{\Delta t}\left(N_i+\frac{\Delta t}{2}u_c^n\frac{\partial N_i}{\partial x_c}\right)\left(u_a^{n+1}+\frac{\Delta t}{2}\hat{u}_b^n\frac{\partial \hat{u}_a^{n+1}}{\partial x_b}\right)\mathrm{d}\Omega\\
&=\int_\Omega \frac{\rho_0}{\Delta t}\left(N_i+\frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\right)\left(\hat{u}_a^{n}-\frac{\Delta t}{2}\hat{u}_b^n\frac{\partial \hat{u}_a^{n}}{\partial x_b}\right)\mathrm{d}\Omega+\int_\Omega \frac{\partial N_i}{\partial x_a}\hat{p}^{n+1/2}\,\mathrm{d}\Omega\\
&\quad-\int_\Omega \frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\frac{\partial \hat{p}^{n+1/2}}{\partial x_a}\,\mathrm{d}\Omega-\int_\Omega \frac{\partial N_i}{\partial x_b}\tau_{ab}^n\,\mathrm{d}\Omega+\int_{\Gamma_{ta}} N_i\bar{t}_a\,\mathrm{d}\Gamma\\
&\quad-\int_\Omega\left(N_i+\frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\right)\rho_0\beta g_a\hat{\theta}^n\,\mathrm{d}\Omega+\int_\Omega \frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\frac{\partial \tau_{ab}^n}{\partial x_b}\,\mathrm{d}\Omega,\quad \forall \text{ free } u_{ai}^{n+1}
\end{aligned}\tag{21}$$

and

$$\begin{aligned}
&\int_\Omega \frac{\rho_0 c}{\Delta t}\left(N_i+\frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\right)\left(\hat{\theta}^{n+1}+\frac{\Delta t}{2}\hat{u}_b^n\frac{\partial \hat{\theta}^{n+1}}{\partial x_b}\right)\mathrm{d}\Omega\\
&=\int_\Omega \frac{\rho_0 c}{\Delta t}\left(N_i+\frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\right)\left(\hat{\theta}^{n}-\frac{\Delta t}{2}\hat{u}_b^n\frac{\partial \hat{\theta}^{n}}{\partial x_b}\right)\mathrm{d}\Omega+\int_\Omega \frac{\partial N_i}{\partial x_b}q_b^n\,\mathrm{d}\Omega-\int_{\Gamma_q} N_i\bar{q}\,\mathrm{d}\Gamma\\
&\quad-\int_\Omega \frac{\Delta t}{2}\hat{u}_c^n\frac{\partial N_i}{\partial x_c}\frac{\partial q_b^n}{\partial x_b}\,\mathrm{d}\Gamma,\quad \forall \text{ free } \theta_i^{n+1}.
\end{aligned}\tag{22}$$

An equation for pressure is obtained combining the mass conservation equation (2) and Equation (17), which states the minimisation of the squared momentum residuals with respect to the pressure degrees of freedom,

$$\int_\Omega \Delta t\frac{\partial N_i}{\partial x_a}\hat{F}_a\,\mathrm{d}\Omega+\int_\Omega N_i\frac{\partial(\rho_0\hat{u}_a^{n+1})}{\partial x_a}\,\mathrm{d}\Omega=0,\quad \forall \text{ free } p_i^{n+1}.\tag{23}$$

Using the mass flux boundary condition (11) and taking the convective term in $\hat{F}_a$ at the time level n, Equation (23) becomes a Poisson equation relating pressure to the mass and momentum balances,

$$\begin{aligned}
\int_\Omega \Delta t\frac{\partial N_i}{\partial x_a}\frac{\partial \hat{p}^{n+1/2}}{\partial x_a}\,\mathrm{d}\Omega=&-\int_\Omega \Delta t\frac{\partial N_i}{\partial x_a}\rho_0\hat{u}_b^n\frac{\partial \hat{u}_a^n}{\partial x_b}\,\mathrm{d}\Omega-\int_\Omega \rho_0 N_i\frac{\partial \hat{u}_a^n}{\partial x_a}\,\mathrm{d}\Omega+\int_\Omega \Delta t\frac{\partial N_i}{\partial x_a}\frac{\partial \tau_{ab}^n}{\partial x_b}\,\mathrm{d}\Omega\\
&-\int_\Omega \Delta t\frac{\partial N_i}{\partial x_a}\rho_0\beta g_a\hat{\theta}^n\,\mathrm{d}\Omega-\int_{\Gamma_G} N_i(\bar{G}^{n+1}-\bar{G}^n)\,\mathrm{d}\Gamma,\quad \forall \text{ free } p_i^{n+1}.
\end{aligned}\tag{24}$$

At this point, the spatial discretisation of the viscous and heat flux terms in Equations (21), (22) and (24) must be introduced. Based on Equations (4) and (5), these quantities are expressed in terms of the discretised velocity and temperature fields as:

$$\hat{\tau}_{ab}^n=\mu\left(\frac{\partial \hat{u}_a}{\partial x_b}+\frac{\partial \hat{u}_b}{\partial x_a}\right)^n\tag{25}$$

and

$$\hat{q}_b^n=-\kappa\frac{\partial \hat{\theta}^n}{\partial x_b}.\tag{26}$$

It is important to remark that with the above approximations, the equivalence between Equations (21), (22) and (24) and the least-squares method is lost. This is due to the occurrence, in Equations (12) and (13), of second-order spatial derivatives of the primitive variables (velocity and temperature), which can be represented inside the finite elements, but not across element interfaces. In the formulation adopted here, the viscous and heat flux contributions to Equations (21), (22) and (24) are evaluated on element interiors by Equations (25) and (26), following a procedure that has become standard in the context of Petrov–Galerkin formulations [3,14].

Note that the implementation of the least-squares method would require either recasting the problem in terms of first-order spatial differentials, with the introduction of new dependent variables, or employing C_1 shape functions to interpolate velocity and temperature. Although no longer equivalent to the least-squares method, the Petrov–Galerkin formulation presented above inherits from the former, the important mathematical properties of *symmetry* and *positive definiteness*, whilst retaining the use of simple C_0 shape functions and the primitive variables approach. In this work, the symmetric Petrov–Galerkin formulation is applied to the simulation of 2D problems, using linear triangular elements to approximate velocity, temperature and pressure.

The problem is solved using a segregated solution procedure. Pressure is computed first, then the velocity and the temperature fields are updated. Equations (21), (22) and (24) lead to symmetric positive definite matrices, allowing the use of preconditioned conjugate gradient solvers. The implementation of an EBE parallel/vector conjugate gradient procedure will be discussed in Section 5.

3. THE LOCAL TIME STEPPING SCHEME

The weighting functions applied to the momentum and energy balances, Equations (16) and (18), have the *streamline upwind Petrov–Galerkin* structure [3],

$$W_i = N_i + \frac{\Delta t}{2}\hat{u}_c^n \frac{\partial N_i}{\partial x_c}. \tag{27}$$

For linear elements, a proper amount of *streamline upwinding* is introduced in the momentum balance, choosing the time step as

$$\Delta t = \left[\coth\left(\frac{Re}{2}\right) - \frac{2}{Re}\right]\frac{h_e}{\|\mathbf{u^n}\|}, \tag{28}$$

where $\|\mathbf{u^n}\|$ is the velocity modulus and h_e is the characteristic element size (the square root of the element area). The element Reynolds number Re is formed using $\|\mathbf{u^n}\|$ and h_e.

It is interesting to note that the time step given by Equation (28) is appropriate to follow the time evolution of the convection–diffusion processes resolvable in a mesh with size h_e (De Sampaio [6]). Indeed, Equation (28) gives for the pure convection limit, i.e. $Re \to \infty$,

$$\Delta t = h_e/\|\mathbf{u}^n\|, \tag{29}$$

whereas for pure diffusion ($Re = 0$), it yields

$$\Delta t = \rho h_e^2/6\mu. \tag{30}$$

The relationship between the time step given by Equation (28), also called the *intrinsic time scale*, and the modelling of the subgrid (or unresolvable) scales was investigated by Hughes [15].

In order to introduce optimal *upwinding* in the fluid energy equation, the element Reynolds number in Equation (28) must be replaced by the element Peclet number $Pe = Re\ Pr$, where $Pr = \mu c/\kappa$ is the Prandtl number. Clearly, unless $Pr = 1$, the time scales for the momentum and energy equations will differ. Furthermore, note that the time step given by Equation (28) varies spatially according to local values of velocity, physical properties and mesh size. If optimal *upwinding* is to be introduced in both momentum and energy balances, there is a need to consider two distinct, spatially varying time step distributions.

An algorithm which allows each degree of freedom to advance in time according to its own local time step, whilst interpolated results are periodically outputted at fixed times, is used [10,11]. The algorithm begins with all degrees of freedom *active* and with variables defined at time t^n. Then, it proceeds as follows:

```
(a) Set element time steps for velocity and pressure using the corresponding $Re$ and element
    time steps for temperature using the corresponding $Pe$.
(b) Project the element time step values onto mesh nodes, obtaining nodal time step
    distributions for $\hat{u}_a$, $\hat{p}$ and $\hat{\theta}$.
(c) Choose an interpolation time step $\Delta t_{\mathrm{int}}$, between the minimum ($\Delta t_{\min}$) and the maximum
    ($\Delta t_{\max}$) time scales.
(d) Define the interpolation time level $t_{\mathrm{int}} = t^n + \Delta t_{\mathrm{int}}$.
(e) Solve Equations (21), (22) and (24) for the active degrees of freedom using the respective
    nodal time step distributions.
(f) Interpolate on the time domain, the degrees of freedom whose tracked time has exceeded
    $t_{\mathrm{int}}$ and freeze their interpolated values at $t_{\mathrm{int}}$. These degrees of freedom are temporarily
    removed from the list of active variables and treated as pseudo boundary conditions for the
    problem defined in terms of the remaining active variables.
(g) Are there any active variables left?
  If yes
    (g1) Recompute the local time steps for the remaining active variables and return to
         step (e).
  Else
    (g2) Output the solution at $t_{\mathrm{int}}$.
    (g3) Release the inactive (frozen) degrees of freedom.
    (g4) Redefine $t^n = t_{\mathrm{int}}$ and return to step (a).
  End if.
```

The process continues until the required analysis time interval has been covered. Note that the extra bookkeeping needed for tracking each degree of freedom time *position* pays off in computational effort, for degrees of freedom associated with larger time steps are updated less frequently than those associated with smaller ones. The algorithm described above leads to a weighting function adaptive method, where the local time step is adjusted according to the local velocity, physical properties and mesh size, aiming to optimise the approximation on a given mesh.

4. ADAPTIVE REMESHING

Much progress has been achieved in the field of finite element mesh adaptivity. The rapid development is both a consequence of the research on error estimates [9,16] and of the availability of mesh generator routines capable of using the error data to build improved meshes [17,18].

Note that a remeshing scheme concerns only the spatial discretisation. However, when dealing with transient processes, the overall error in the solution is associated, not only to the spatial discretisation, but also to the time integration of the governing equations. Thus, some form of time step adaptation is necessary as far as a transient analysis is concerned.

In this work, the *a posteriori* error estimator proposed by Zienkiewicz and Zhu [9] is used to estimate the velocity gradient error and to guide the remeshing. The local time stepping algorithm is used in conjunction with the remeshing scheme. This permits the linking of spatial and time step refinement through Equation (28), and naturally leads to a simultaneous time–space adaptive procedure. Indeed, whenever the remeshing scheme creates some local refinement to better resolve a particular flow feature, the time step distribution is also adapted accordingly, so that the corresponding time evolution can be appropriately followed.

The velocity gradient error is estimated according to

$$\|E\|_\Omega = \left[\int_\Omega (\nabla u_a^* - \nabla \hat{u}_a)(\nabla u_a^* - \nabla \hat{u}_a)\, \mathrm{d}\Omega\right]^{1/2}, \tag{31}$$

where $\nabla \hat{u}_a$ is the discontinuous gradient computed directly from the analysis and ∇u_a^* is the smoothed gradient obtained by least-squares projection of $\nabla \hat{u}_a$ onto the C_0 finite element basis [8].

The following measures of the velocity gradient $\|S\|_\Omega$ and the relative error η can be defined as:

$$\|S\|_\Omega = \left[\int_\Omega \nabla u_a^* \nabla u_a^*\, \mathrm{d}\Omega\right]^{1/2}, \tag{32}$$

$$\eta = \frac{\|E\|_\Omega}{\|S\|_\Omega}. \tag{33}$$

If Ω_i is the subdomain associated with a typical element i, the corresponding element error is

$$\|e\|_{\Omega_i} = \left[\int_{\Omega_i} (\nabla u_a^* - \nabla \hat{u}_a)(\nabla u_a^* - \nabla \hat{u}_a)\, \mathrm{d}\Omega\right]^{1/2}. \tag{34}$$

The domain error given by Equation (31) is related to the individual element errors according to

$$\|E\|_\Omega^2 = \sum_{i=1}^{m} \|e\|_{\Omega_i}^2, \tag{35}$$

where m is the number of elements in the mesh. Thus, one can compute the average error per element $\bar{e}_m$ as

$$\bar{e}_m = \frac{\|E\|_\Omega}{\sqrt{m}} \tag{36}$$

or, in terms of the relative error η,

$$\bar{e}_m = \eta \frac{\|S\|_\Omega}{\sqrt{m}}. \tag{37}$$

The remeshing scheme is based on the concept of generating a new mesh in such a way that the error will become uniformly distributed among the new elements. This requires defining a target error per element and leads to various alternative strategies.

In a previous work [8], Zienkiewicz and Zhu [9] were followed and Equation (37) was to set the remeshing target error per element according to the user-prescribed aimed analysis quality

$\bar{\eta}$. Such an approach focused on keeping a *nearly* constant error level, while letting the number of elements vary. In the remeshing procedure presented next, the priority is somewhat reversed. In fact, the error level is allowed to be adjusted during the analysis according to a user-defined maximum allowable number of elements. This seems to be more budget-oriented than the previous implementation, where an excessive number of elements was sometimes created.

Suppose that through a remeshing scheme, starting with a coarse mesh containing m elements, one could obtain a refined mesh with m' elements, such that $m' > m$. Further assume that the refined mesh is optimal in the sense that the error is evenly distributed among the m' elements. Then, using Equation (36), the ratio between the average error per element measured on such meshes is

$$\bar{e}_{m'}/\bar{e}_m = \sqrt{m}\,\|E'\|_\Omega/\sqrt{m'}\,\|E\|_\Omega. \tag{38}$$

Noting that the domain error in the refined optimal mesh is smaller than that on the coarse mesh, it is concluded that

$$\bar{e}_{m'}/\bar{e}_m = \sqrt{m}/\sqrt{m''}, \quad \text{for some } m'' > m'. \tag{39}$$

In view of the above remarks, the following target error per element is chosen for remeshing

$$\bar{e}_\mathrm{t} = \sqrt{\frac{m}{m''}}\,\bar{e}_m, \tag{40}$$

where m'' is the user-prescribed maximum number of elements, m is the number of elements in the current mesh and $\bar{e}_m$ is the average error per element measured on the current mesh.

For the linear elements employed in the computation, it is assumed the individual element errors are proportional to the corresponding element sizes. Thus, the new element size distribution required to attain the target error $\bar{e}_\mathrm{t}$ can be defined on every element of the new mesh.

The new element size distribution can be expressed in terms of the element sizes and errors on the current mesh and of the uniform target error $\bar{e}_\mathrm{t}$ aimed for the new mesh as

$$h_i^{k+1} = h_i^k \frac{\bar{e}_\mathrm{t}}{\|e\|_{\Omega_i}}, \quad \text{for } i = 1, \ldots, m \tag{41}$$

where $k + 1$ and k denote the new and the current mesh respectively.

In some applications it is necessary to limit the minimum acceptable element size (h_min) as the time step computed from Equation (28) may become too small for an affordable computation.

Equation (41) gives a piecewise continuous element size distribution for the new mesh, which is defined on the current mesh. The mesh generator used in this work requires the element size distribution to be continuous and defined on the nodes of a specified background mesh. Therefore, the size distribution given by Equation (41) is projected to the nodes of the current mesh via least-squares smoothing, and then transferred to the background mesh nodes.

A Delaunay mesh generator is employed to construct meshes composed of triangular elements. The problem is defined on a initial mesh containing boundary and initial condition data. This initial mesh is used as the background mesh, from which all subsequent meshes will be generated by the insertion of new points and the application of Bowyer's algorithm [19] to connect the newly created points in a Delaunay triangulation. In the remeshing procedure adopted in this work, the background mesh is indeed a computational mesh, which is actually used to start the computation.

The automatic remeshing is controlled by the variation of the relative error given by Equation (33). A new mesh is constructed whenever the relative error varies more than 1% during the transient analysis.

4.1. Transferring field data to a newly created mesh

An important aspect in a remeshing procedure concerns the interpolation of field data between two consecutive computational meshes. The problem of transferring data between meshes is illustrated in Figure 1, where the determination of field values at a node on the new mesh requires the interpolation of variable values that are only available on the old mesh.

Referring to Figure 1, let u be a generic variable. Then, the simplest way to compute u at point **P** is through a linear interpolation based on the old mesh element shape functions, i.e.

$$\hat{u}_p = N_k u_k, \tag{42}$$

where u_k are nodal values on the old mesh and N_k are the corresponding linear shape functions evaluated at point **P**.

However, the use of the above equation to transfer field data is inadequate for transient problems involving a large number of computational meshes. Indeed, the repeated use of Equation (42) introduces spurious dissipation in the numerical analysis [8]. Instead, a low-/high-order interpolation scheme, similar to that presented by De Sampaio *et al.* [8], is adopted.

A second-order correction to Equation (42) can be constructed using the smoothed gradients of u, obtained by least-squares projection of the piecewise constant gradients available from the finite element analysis.

The smoothed gradients are linear functions defined on each old mesh element according to

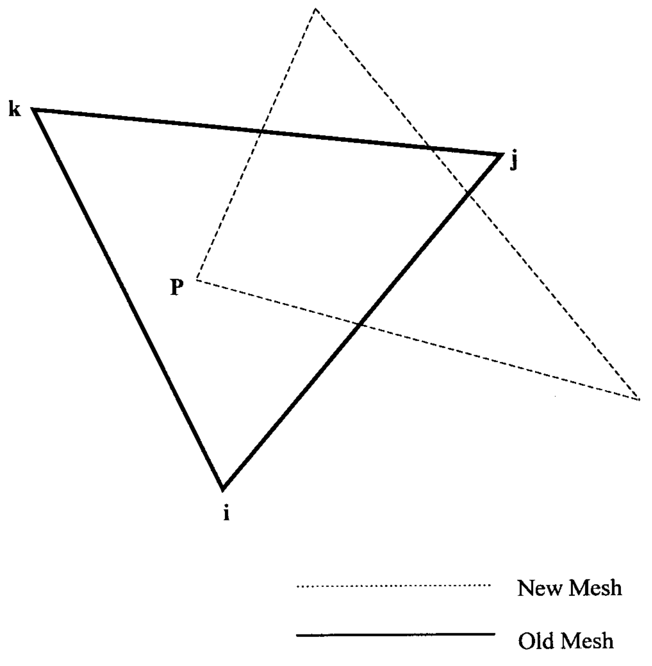


Figure 1. Field values at the new mesh node **P** are interpolated using data stored on the old mesh nodes **i**, **j**, **k**.

$$\frac{\partial u^*}{\partial x} = N_j\left(\frac{\partial u^*}{\partial x}\right)_j, \tag{43}$$

$$\frac{\partial u^*}{\partial y} = N_j\left(\frac{\partial u^*}{\partial y}\right)_j, \tag{44}$$

where $(\partial u^*/\partial x)_j$ and $(\partial u^*/\partial y)_j$ are smoothed gradients nodal values.

From the above equations, the following second-order derivatives of u, which are constant within each old mesh element, can be defined:

$$\left[\frac{\partial^2 u^*}{\partial x^2}\right]_e = \frac{\partial N_j}{\partial x}\left(\frac{\partial u^*}{\partial x}\right)_j, \tag{45}$$

$$\left[\frac{\partial^2 u^*}{\partial x\,\partial y}\right]_e = \left[\frac{\partial^2 u^*}{\partial y\,\partial x}\right]_e = \frac{1}{2}\left\{\frac{\partial N_j}{\partial x}\left(\frac{\partial u^*}{\partial y}\right)_j + \frac{\partial N_j}{\partial y}\left(\frac{\partial u^*}{\partial x}\right)_j\right\}, \tag{46}$$

$$\left[\frac{\partial^2 u^*}{\partial y^2}\right]_e = \frac{\partial N_j}{\partial y}\left(\frac{\partial u^*}{\partial y}\right)_j. \tag{47}$$

A second-order interpolation scheme is obtained, adding a correction term to Equation (42). The correction vanishes if point **P** coincides with an old mesh node. The resulting interpolation satisfies the above second-order derivatives within the old mesh element and is given by

$$\tilde{u}_p = \hat{u}_p + \Delta u_p, \tag{48}$$

where

$$\Delta u_p = f(x_p, y_p) - \sum_{j=1}^{j=3} N_j(x_p, y_p) f(x_j, y_j), \tag{49}$$

$$f(x, y) = \frac{1}{2}\left\{\left[\frac{\partial^2 u^*}{\partial x^2}\right]_e x^2 + 2\left[\frac{\partial^2 u^*}{\partial x\,\partial y}\right]_e xy + \left[\frac{\partial^2 u^*}{\partial y^2}\right]_e y^2\right\}, \tag{50}$$

and x_p, y_p are the Cartesian co-ordinates of point **P**.

In some instances, it may be convenient to leave some numerical dissipation in the interpolation procedure by not fully introducing the correction term. The low-/high-order interpolation scheme is written as:

$$\tilde{u}_p = \hat{u}_p + \varepsilon\,\Delta u_p, \tag{51}$$

where ε varies from 0, the linear case, to the fully corrected scheme for $\varepsilon = 1$.

Recall that the element sizes in any mesh generated, are constrained by the prescribed minimum size $h_{\min}$. The linear scheme is used when interpolating data within such tiny elements. This is done in order to damp the contributions from spatial scales that cannot be resolved on the computational meshes used during the analysis. On the other hand, the quadratic procedure is employed when interpolating within elements larger than the prescribed minimum size.

4.2. Searching procedures

Many operations described above depend on finding the element that contains a point given by its co-ordinates. If not carefully programmed, these search operations become quite CPU consuming, and may severely undermine the overall code performance.

The neighbour-to-neighbour search algorithm, as shown in Figure 2, is used in this work [17]. According to Löhner [20], provided a close enough starting element is given to initiate the

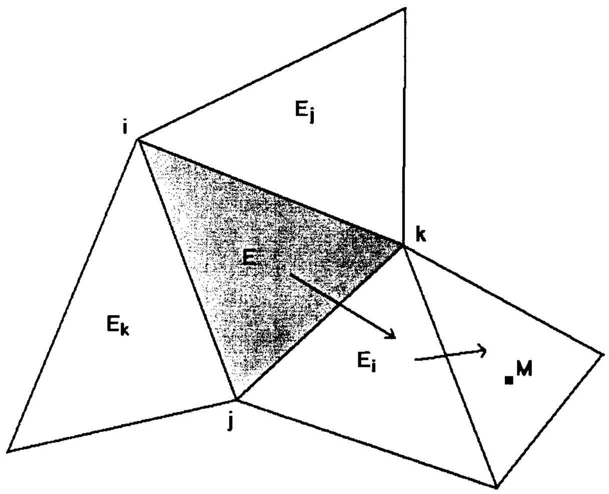


Figure 2. The neighbour-to-neighbour search algorithm. The area co-ordinates L_i, L_j and L_k of element E are evaluated at the searched point M. Here, $L_i(M) = \min(L_i(M), L_j(M), L_k(M))$ and the next element to be checked is E_i. The algorithm is repeated until the element containing point M is found.

search, this algorithm, though scalar, out-performs other vectorised procedures. A simple data structure has been devised to guarantee that the searches required in the computations always start from a close element. Because all meshes produced are obtained by refinement of the same background mesh, the elements of the background mesh can be used as regions to which all points and elements are referred. Thus, if the point being searched is associated with a given background region, one must ensure that the starting element for the search belongs to the same region. The neighbour-to-neighbour algorithm works well for convex regions but may eventually fail otherwise. Thus, a vectorised search on all elements has been programmed as a fall-back position. This is activated whenever the neighbour-to-neighbour scheme fails to find the desired point after visiting 20 elements.

5. THE PARALLEL/VECTOR ITERATIVE SOLVER

Provided the mesh generator and the searching routines are programmed efficiently, the updating of variables is the most time consuming operation in the whole analysis. The updating of the velocity, pressure and temperature fields demands the cyclic solution of four symmetric positive definite systems of equations. These systems are solved using a Jacobi preconditioned EBE conjugate gradient (CG) method [13].

Note that sparse matrix–vector multiplications are required during the CG computation. Due to the local nature of finite element approximation, the sparse matrix–vector product can be recast as

$$\mathbf{Ax} = \sum_{e=1}^{nel} \mathbf{A}_e \mathbf{x}_e, \tag{52}$$

where *nel* is the current number of elements in the mesh. Thus, the computations involve only the element matrices, without the burden of assembling and handling large global sparse matrices.

The implementation of the EBE matrix–vector multiplication follows the three-step algorithm:

```
for each element e do

    GATHER          $\mathbf{x}_e$ from $\mathbf{x}$
    COMPUTE         $\mathbf{v}_e=\mathbf{A}_e\mathbf{x}_e$
    SCATTER+ADD     $\mathbf{v}=\mathbf{v}+\mathbf{v}_e$

end do.
```

The above algorithm has a great potential for vectorisation and parallelisation since the GATHER and COMPUTE steps can be performed independently from the other elements. However, the SCATTER + ADD step involves a write operation on a global array. For adjacent elements, write operations are performed on shared addresses in $\mathbf{v}$. Therefore, the SCATTER + ADD operations can only be performed concurrently within a group of non-adjacent elements. Groups of non-adjacent elements can be constructed, reordering the elements before starting the computations by a mesh colouring algorithm, producing a pattern like the one shown in Figure 3. The element grouping requires a small fraction of CPU time and is performed before starting computations on the current mesh. Note, though, that the local time stepping scheme used in this work implies variable lists of active elements and degrees of freedom. Therefore, prior to each CG solution, the groups are edited to extract the current active elements. The EBE computations within a group (colour) can be readily vectorised and distributed for multiple processors using the *autotasking* facilities available in Cray machines. This approach yields vector lengths of the size of the current number of elements in a group. Consequently, the colouring algorithm should attempt to produce groups with approximately the same number of elements. It is important to note that storage requirements in the EBE scheme are

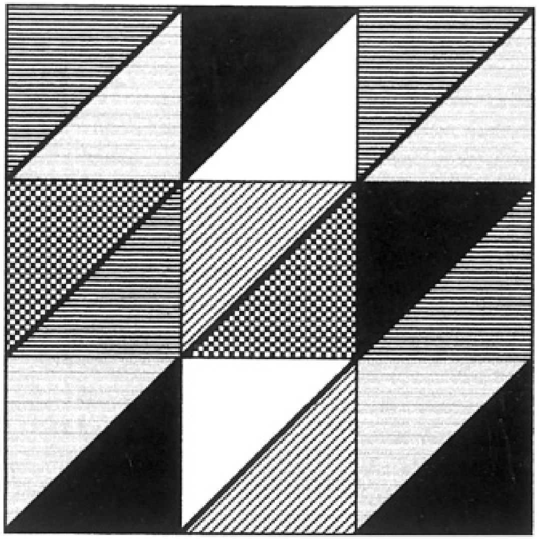


Figure 3. Multicolour grouping of finite elements.

directly proportional to the number of elements in the mesh. Further, one does not evaluate and store the element matrices. Actually, the action of the element is computed on the product. Thus, geometric factors, shape functions etc., are recomputed as they are needed. The resulting solution strategy keeps storage at a minimum and is a parallel/vector, matrix-free method.

6. NUMERICAL EXAMPLES

The techniques described above have been applied to the analysis of free and forced convection incompressible flows. Free convection examples include the problem of thermal stratification inside a square cavity and the simulation of the external flow generated around a hot horizontal pipe. Also presented is the simulation of a confined forced convection flow through a tube bank, with special emphasis on assessing the program parallel/vector performance on the Cray J90.

6.1. Free convection examples

For the sake of convenience, the governing equations in non-dimensional form are considered. The relationship between the dimensional and the non-dimensional variables is defined as $x'_a = x_a/L$; $t' = \mu t/\rho_0 L^2$; $u'_a = \rho_0 u_a L/\mu$; $\theta' = \theta/\Delta\theta$ and $p' = \rho_0 p L^2/\mu^2$, where $\Delta\theta$ and L are the reference temperature difference and the reference length scale for the problem considered.

The governing equations can thus be recast in the non-dimensional form

$$\frac{\partial u'_a}{\partial t'} + u'_b \frac{\partial u'_a}{\partial x'_b} - \frac{\partial}{\partial x'_b}\left(\frac{\partial u'_a}{\partial x'_b} + \frac{\partial u'_b}{\partial x'_a}\right) + \frac{\partial p'}{\partial x'_a} + Gr\,\gamma_{\rm a}\theta' = 0, \tag{53}$$

$$\frac{\partial u'_a}{\partial x'_a} = 0, \tag{54}$$

$$Pr\left(\frac{\partial \theta'}{\partial t'} + u'_b \frac{\partial \theta'}{\partial x'_b}\right) - \frac{\partial}{\partial x'_b}\left(\frac{\partial \theta'}{\partial x'_b}\right) = 0. \tag{55}$$

In the above equations, $\gamma_a = g_a/\|\mathbf{g}\|$ indicates the gravity field direction and Gr and Pr are respectively, the Grashof and the Prandtl numbers

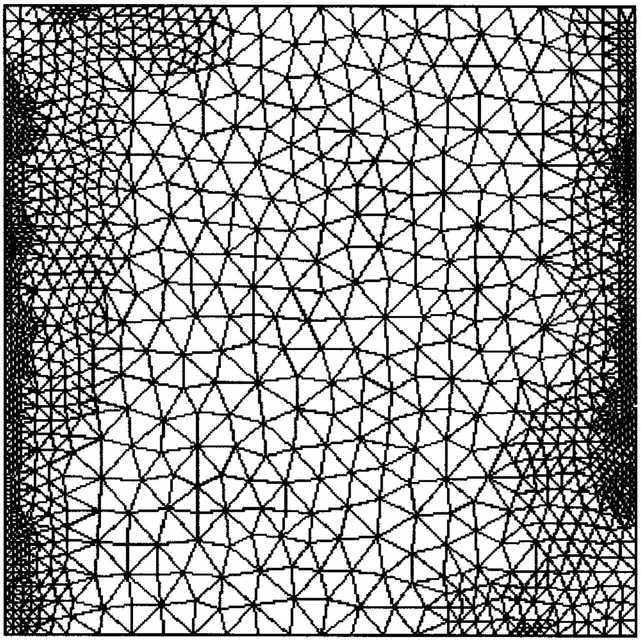
$$Gr = \frac{\rho_0^2 \beta\,\|\mathbf{g}\|\Delta\theta L^3}{\mu^2}, \tag{56}$$

$$Pr = \frac{c\mu}{\kappa}. \tag{57}$$

6.1.1. Thermal stratification in a square cavity. The fluid contained in a square cavity is initially at rest and in thermal equilibrium when a sudden temperature difference $\Delta\theta$ is applied and maintained between the vertical walls. The temperature at the left wall is increased by $0.5\Delta\theta$, whilst the temperature at the right wall is decreased by the same amount. The resulting buoyancy forces initiate a internal free convection flow, which leads to thermal stratification within the cavity.

$\Delta\theta$ and the height of the cavity, L, have been chosen as the characteristic scales for non-dimensionalising the analysis. Plate 1 and Figure 4 illustrate the transient simulation, covering the time interval from $t' = 0$ to $t' = 0.5$, for $Gr\,Pr = 10^6$ and $Pr = 0.72$.

The initial mesh comprised 441 nodal points and 800 elements. The transient analysis required 3128 time steps to reach the final time. A total number of 144 meshes have been



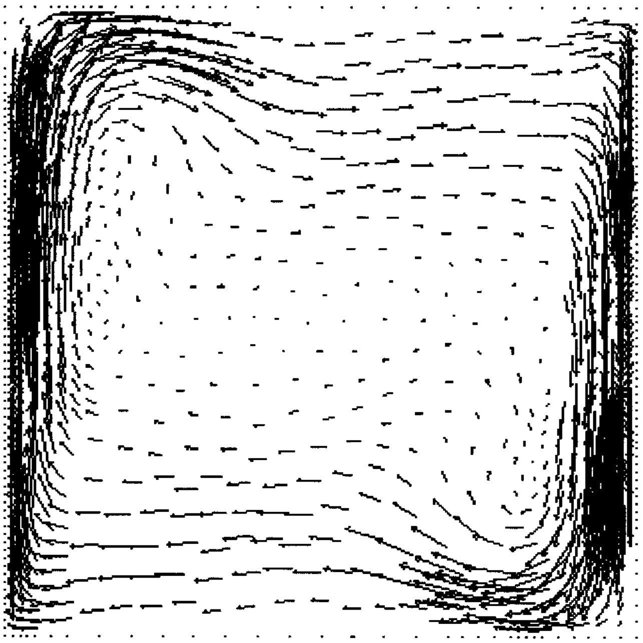


Figure 4. Free convection inside a square cavity: mesh and velocity field at $t' = 0.5$.

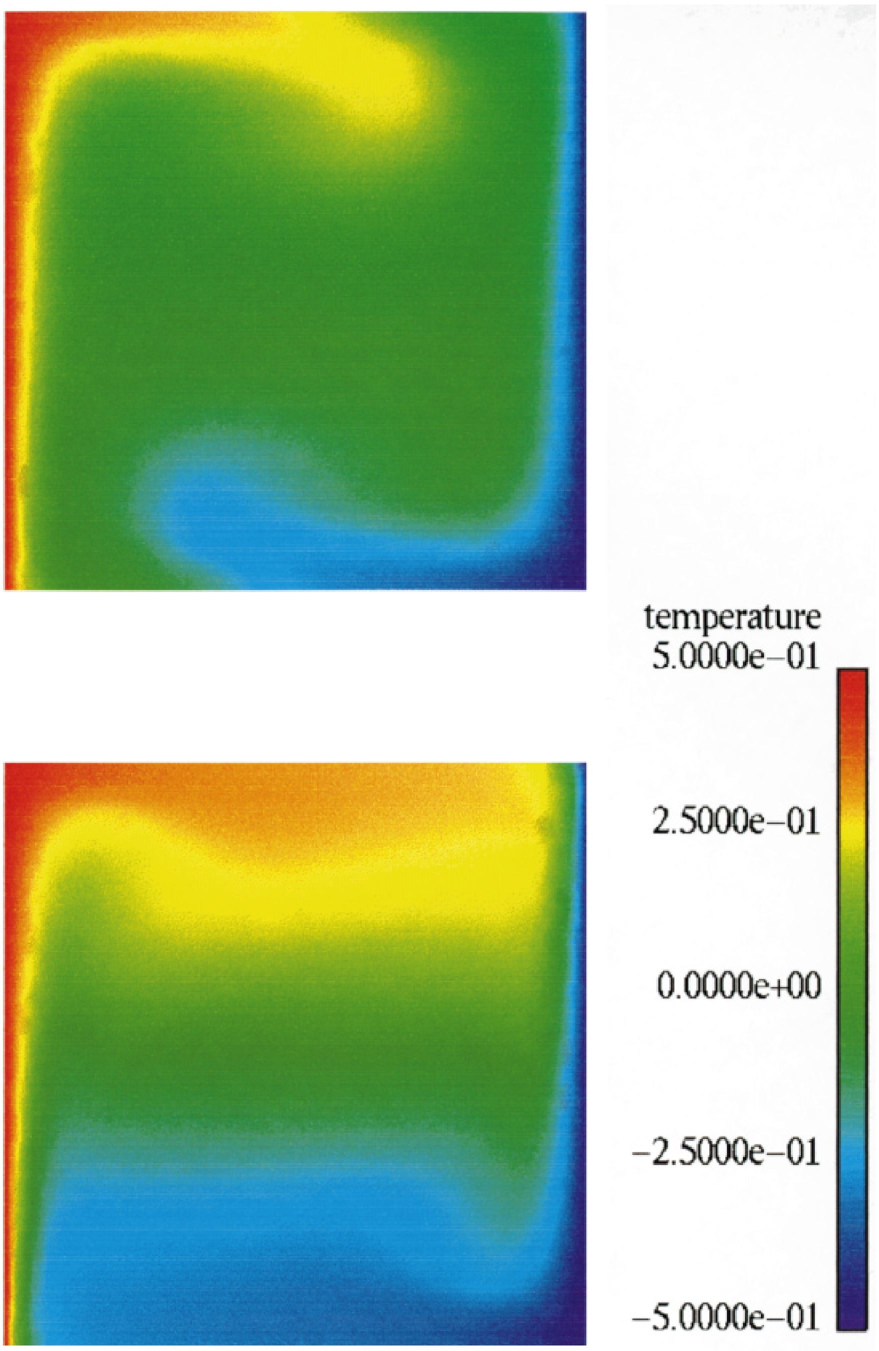


Plate 1. Free convection inside a square cavity: non-dimensional temperature field at $t' = 0.05$ (top) and $t' = 0.5$ (bottom).

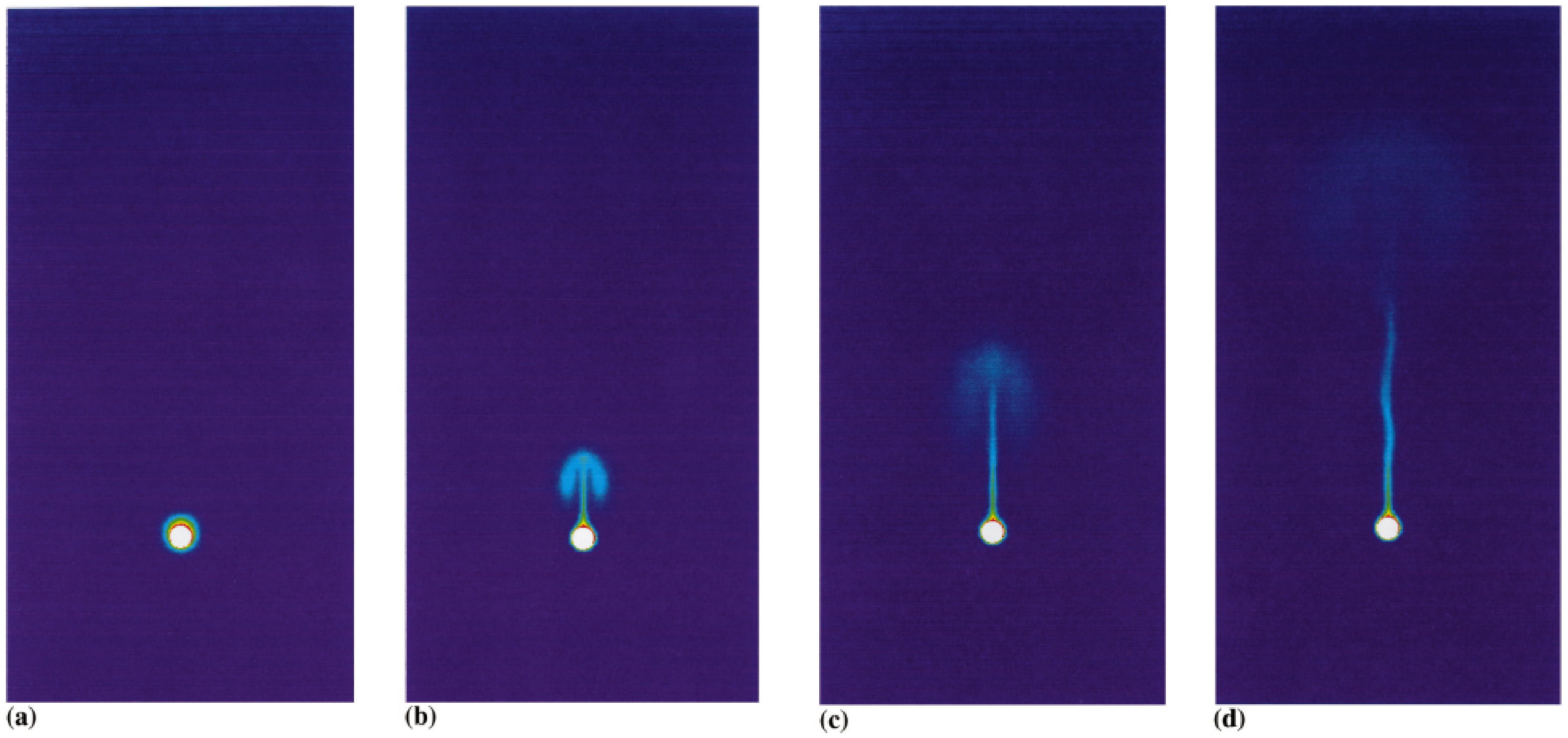
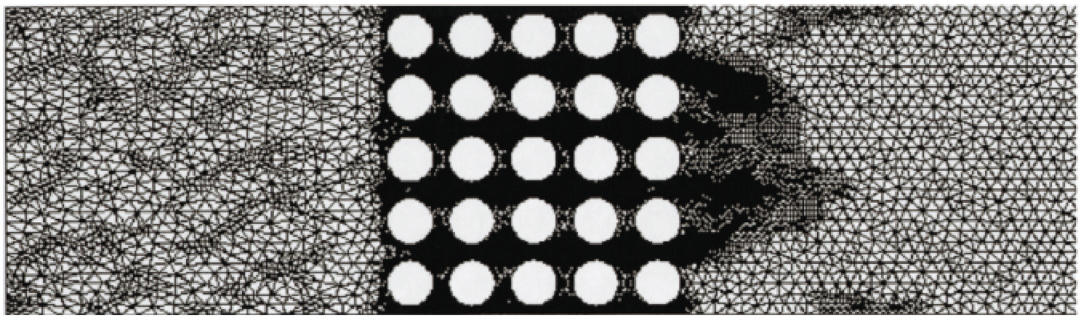


Plate 2. Free convection around a hot pipe: non-dimensional temperature field at (a) $t' = 0.004$, (b) $t' = 0.020$, (c) $t' = 0.040$, (d) $t' = 0.080$.



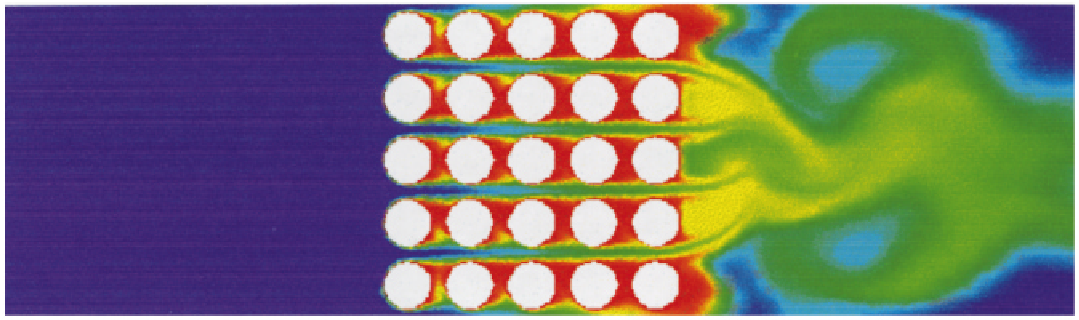


Plate 3. Simulation of a confined cross-flow past a tube bank: mesh at $t = 8d/u_0$ (top) and corresponding non-dimensionalised temperature (bottom).

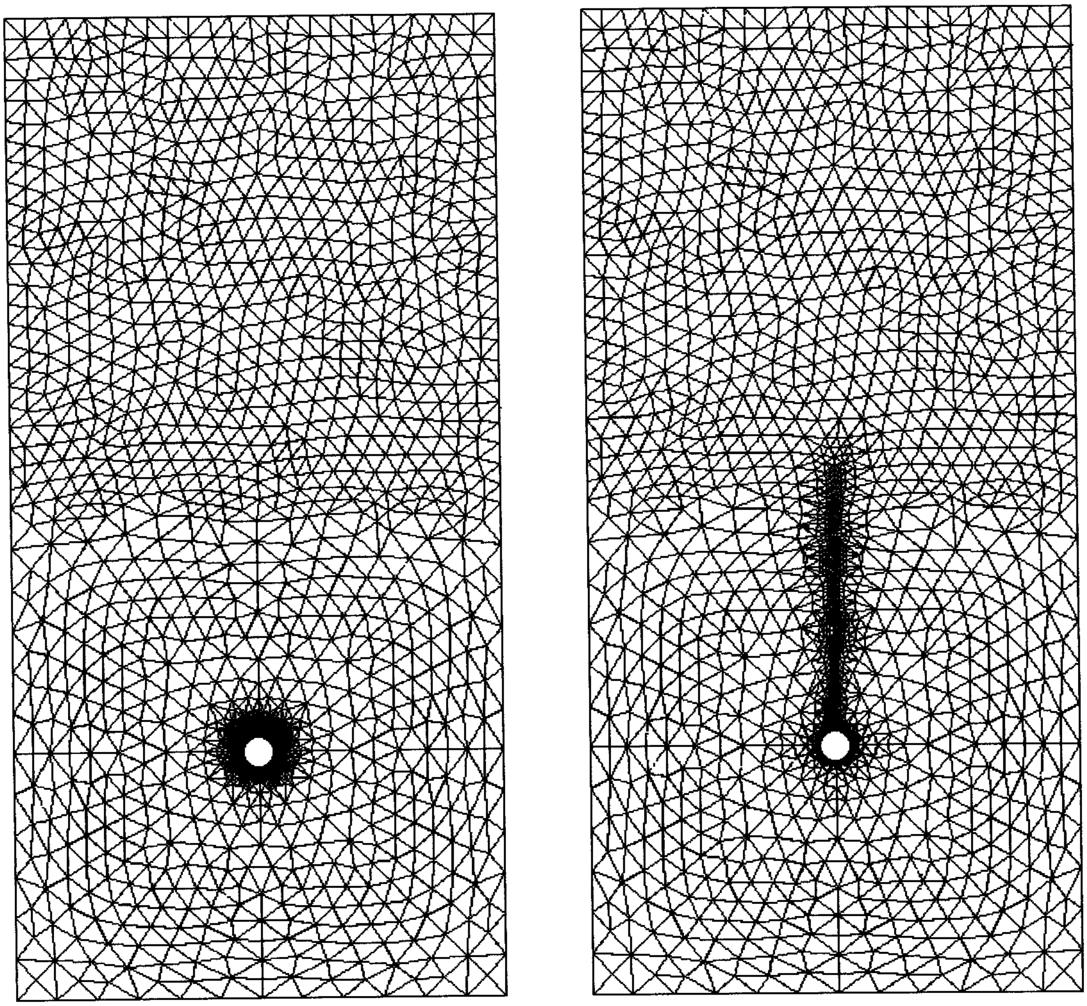


Figure 5. Free convection around of a hot pipe: mesh at $t' = 0.004$ (left) and at $t' = 0.060$ (right).

automatically constructed during the computation. The number of nodes and elements in the final mesh are 1239 and 2264 respectively.

The CPU time for a single processor vector run on the Cray J90 was 2682 s. A sustained job performance of 78 Mflops has been achieved in this run, which includes the generation of files for CEA Ensight, the visualisation package available in the system.

6.1.2. Free convection around a hot pipe. This example shows the external free convection flow that develops around a hot horizontal pipe. The temperature at the pipe surface exceeds that of the surrounding fluid by $\Delta\theta$. The characteristic temperature scale for this analysis is $\Delta\theta$, whilst the pipe diameter is chosen for the length scale.

Plate 2 and Figures 5 and 6 illustrate this simulation. The transient analysis was run from $t' = 0$ to $t' = 0.08$, with $Gr\,Pr = 10^5$ and $Pr = 0.72$.

The initial mesh comprised 700 nodes and 1304 elements. The number of time steps required to reach the final time was 1979, and a total of 351 adaptive meshes have been automatically constructed during the computation. The final mesh contains 2218 nodes and 4250 elements.

The CPU time for a single processor vector run on the Cray J90 was 2418 s, with a sustained job performance of 66 Mflops.

6.2. Simulation of cross-flow past a tube bank

The simulation of cross-flow past a tube bank is presented here. The fluid is confined between two horizontal plates, which directs the flow towards a regular array of 5×5 cylinders. The pitch-to-diameter ratio is $p/d = 1.33$, and the ratio between the plates' distance l and the diameter d is $l/d = 6.65$. A uniform velocity profile u_0 is assumed at the flow entrance and free traction boundary conditions are imposed at the outflow. The Reynolds number, defined by u_0 and l, is $R_l = 665$ and the Prandtl number is $Pr = 1$. No-slip boundary conditions are imposed on all solid surfaces. The non-dimensionalised temperature at the inlet is $\theta^* = 0$, whilst the temperature at the surface of the cylinders is fixed at $\theta^* = 1$. The initial fluid temperature is $\theta^* = 0$.

The transient analysis was run from $t = 0$ to $t = 20d/u_0$. A total number of 499 adaptive meshes have been automatically constructed. The maximum number of elements prescribed for remeshing was $m'' = 40000$ and the prescribed minimum characteristic element size was $h_e = 0.02d$. The average number of elements and degrees of freedom per mesh was approxi-

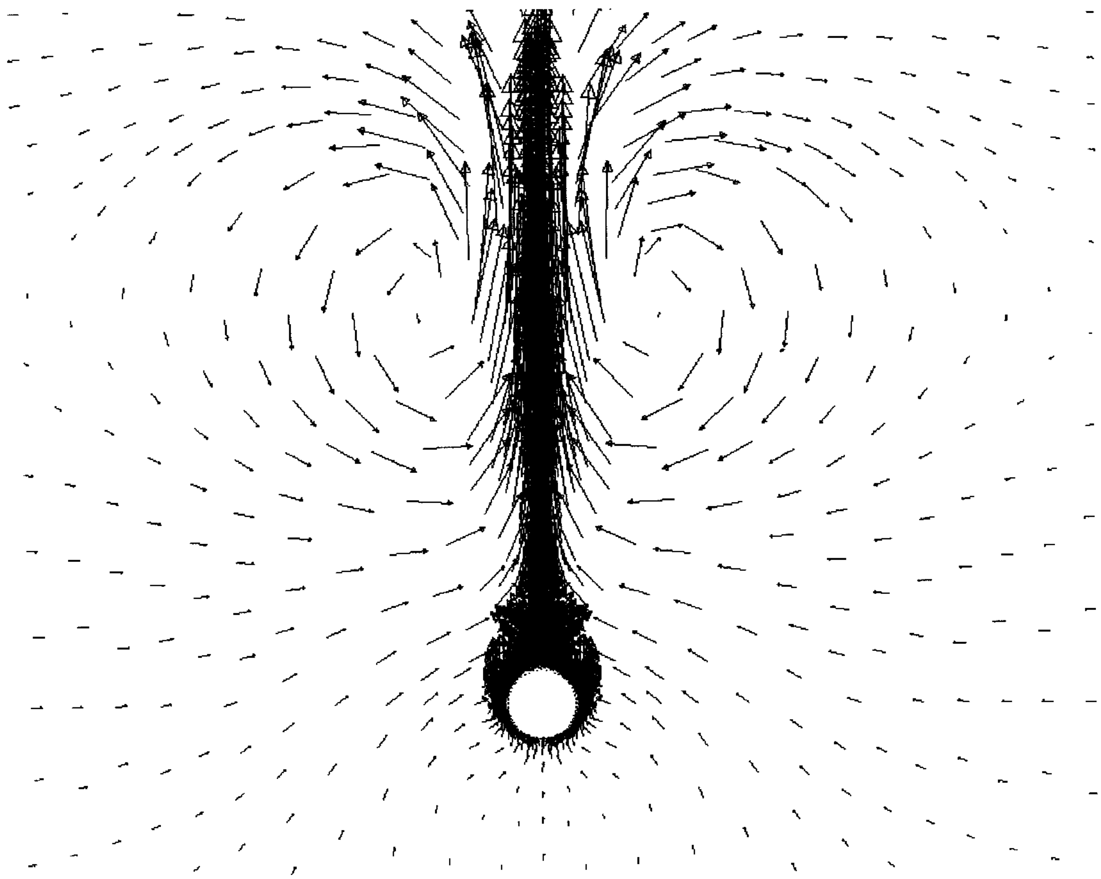


Figure 6. Free convection around a hot pipe: detail of the velocity field at $t' = 0.060$.

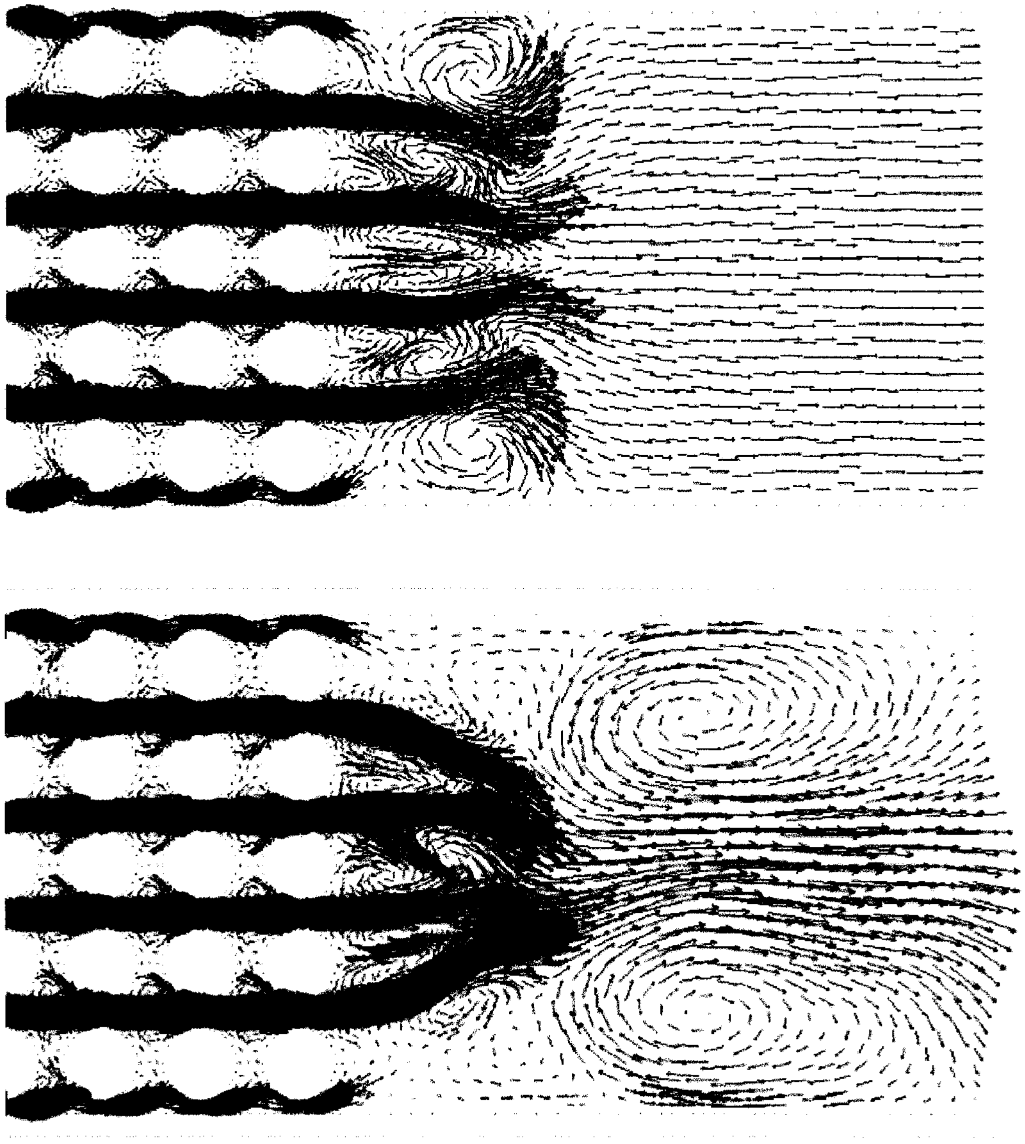


Figure 7. Simulation of a confined cross-flow past a tube bank; velocity field at $t = 2.5d/u_0$ (top) and at $t = 11d/u_0$ (bottom).

mately 25000 and 50000 respectively. The transient analysis required 11140 time interpolation steps Δt_{int}. Some results of this simulation are shown in Plate 3 and Figure 7. Figure 8 depicts the evolution of the number of elements per adapted mesh.

The parallel/vector performance on the Cray J90 has been analysed, trying to follow as much as possible, the guidelines from Crowl [21]. The basic variables for the performance measurements are *flops* rate and *speed-up* factor. The *flops* rates are evaluated using Cray's tools, the hardware performance monitor (*hpm*) and the profiling tool (*perfview*) on a single CPU version of the code, where vectorisation is the main issue. Parallel *speed-up* is evaluated by the *atexpert* tool during a four CPU run.

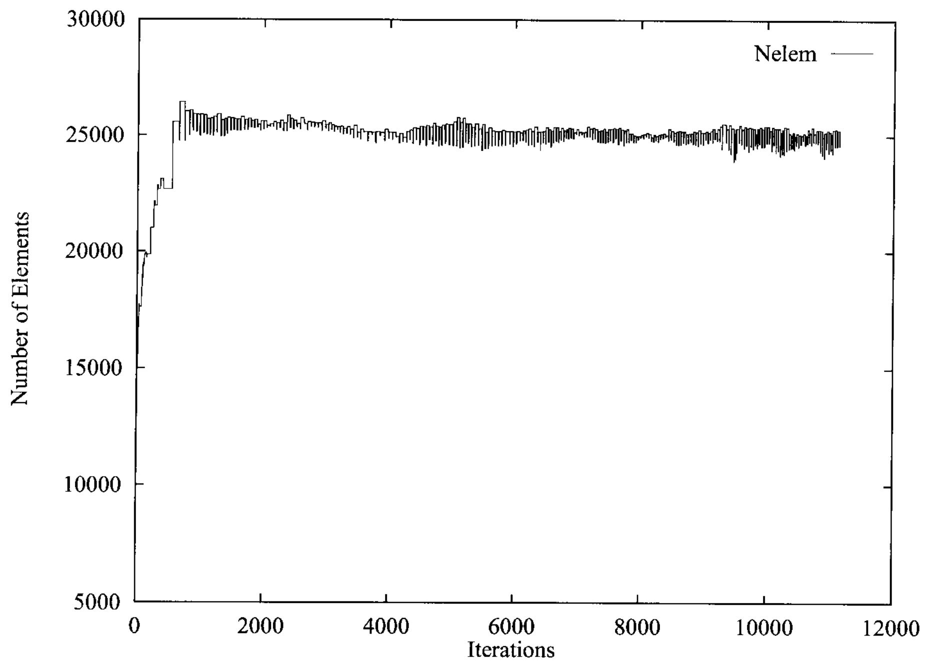


Figure 8. Evolution of the number of elements in the adapted meshes employed during the simulation of cross flow past a tube bank.

The CPU time of the vectorised single processor run was 54.5 h. The sustained job performance on this run was 109.3 Mflops, indicating a highly efficient vectorised code. A computational intensity (ratio of *flops* to memory references) of 2.23 was achieved during the analysis, a value that is considered to be very good by the *perfview* tool. Table I presents the *Mflops* rates obtained for the most relevant inner kernels during the single CPU run. In this table, *solvpr* and *solvel* represent the conjugate gradient updating of pressure and velocity respectively; *mshcol* is the mesh colouring algorithm that performs only integer operations; *newpts* is the point creation routine used in the mesh generator procedure.

Computational costs are dominated by the pressure update, which requires the solution of the discretised Poisson equation (24). The updating of velocity and temperature also involves the solution of linearised equation systems, but is less expensive. This occurs because the last available solutions are used to start the CG, whilst closely following the time scales of the momentum and energy convection–diffusion processes.

Even though a total of 499 meshes have been generated during the transient, the cost of mesh generation, mesh colouring, data transfer between meshes (including searching routines) and writing of visualisation files represent only 11% of the overall solution cost.

Table I. Performance summary (1 CPU)

Routines	% CPU time	Mflops
solvpr	76.3	91.7
mshcol	7.8	0.0
solvel	2.3	99.9
newpts	2.1	144.5

Regarding the parallel performance in four CPUs, the *atexpert* tool predicts a speed-up of 3.4 for a dedicated run.

7. CONCLUDING REMARKS

An adaptive parallel/vector finite element procedure for the simulation of incompressible viscous flows with heat transfer has been presented. The algorithm involves space and time adaptation in unstructured meshes. According to Simon [22], the class of *dynamic implicit unstructured* computation is the hardest among parallel applications. In spite of such difficulties, good performance was achieved in the simulation of free and forced convection flows on the Cray J90.

It has been observed that linear equation solving is responsible for most of the CPU time. The CG procedure is completely vectorised and parallelised through the use of the matrix-free EBE scheme. On the other hand, the sequential part of the code is very fast. In particular, the local search procedure, based on the neighbour-to-neighbour algorithm described in Section 4.2, has proven to be reliable in all simulations performed.

It is worth stressing that the numerical methods presented here for 2D laminar flows, naturally extend for the simulation of 3D problems and for the computation of turbulent flows using Reynolds-averaged equations.

ACKNOWLEDGMENTS

The authors kindly acknowledge the Centre for Parallel Computation of COPPE/UFRJ for the use of the Cray J90. This work is partially sponsored by CNPq Grant No. 520589/95-5.

REFERENCES

1. K.W. Morton, 'Generalised Galerkin methods for steady and unsteady problems', in K.W. Morton and M.J. Baines (eds.), *Numerical Methods for Fluid Dynamics*, Academic Press, New York, 1982, pp. 1–32.
2. J.C. Heinrich, P.S. Huyakorn, O.C. Zienkiewicz and A.R. Mitchell, 'An upwind finite element scheme for two-dimensional convective transport equations', *Int. J. Numer. Methods Eng.*, **11**, 131–143 (1977).
3. A. Brooks and T.J.R. Hughes, 'Streamline upwind/Petrov–Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations', *Comp. Methods Appl. Mech. Eng.*, **32**, 199–259 (1982).
4. F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*, Springer, New York, NY, 1991.
5. T.J.R. Hughes, L.P. Franca and M. Ballestra, 'A new finite element formulation for computational fluid dynamics: V. circumventing the Babuška–Brezzi condition: A stable Petrov–Galerkin formulation of the Stokes problem accommodating equal-order interpolations', *Comp. Methods Appl. Mech. Eng.*, **59**, 85–89 (1986).
6. P.A.B. de Sampaio, 'A Petrov–Galerkin formulation for the incompressible Navier–Stokes equations using equal-order interpolation for velocity and pressure', *Int. J. Numer. Methods Eng.*, **31**, 1135–1149 (1991).
7. O.C. Zienkiewicz and J. Wu, 'Incompressibility without tears—how to avoid restrictions of mixed formulation', *Int. J. Numer. Methods Eng.*, **32**, 1189–1203 (1991).
8. P.A.B. de Sampaio, P.R.M. Lyra, K. Morgan and N.P. Weatherill, 'Petrov–Galerkin solutions of the incompressible Navier–Stokes equations in primitive variables with adaptive remeshing', *Comp. Methods Appl. Mech. Eng.*, **106**, 143–178 (1993).
9. O.C. Zienkiewicz and J.Z. Zhu, 'A simple error estimator and adaptive procedure for practical engineering analysis', *Int. J. Numer. Methods Eng.*, **24**, 337–357 (1987).
10. P.A.B. de Sampaio, 'Transient solutions of the incompressible Navier–Stokes equations in primitive variables employing optimal local time stepping', in C. Taylor (ed.), *Proc. 8th Int. Conf. on Numerical Methods for Laminar and Turbulent Flow*, vol. 8, Pineridge Press, Swansea, UK, 1993, pp. 1493–1504.
11. P.A.B. de Sampaio and A.L.G.A. Coutinho, 'Parallel/vector finite element simulation of coupled flow and heat transfer', *Proc. 16th Iberian Latin American Conference on Computational Methods for Engineering*, Curitiba, Brazil, 1995, pp. 1401–1410.

12. T.J.R. Hughes, R.M. Ferencz and J.O. Hallquist, 'Large scale vectorized implicit calculations in solid mechanics on a CRAY X-MP/48 utilising EBE preconditioned conjugate gradients', *Comp. Methods Appl. Mech. Eng.*, **61**, 215–248 (1987).
13. A.L.G.A. Coutinho, J.L.D. Alves, L. Landau and N.F.F. Ebecken, 'Avaliação de estratégias computacionais pare o método dos elementos finitos em computadores vetoriais', *Rev. Int. de Met. Num. para Calculo y Diseño en Ingenieria*, **9**, 271–297 (1993).
14. T.J.R. Hughes, 'Recent progress in the development and understanding of SUPG methods with special reference to the compressible Euler and Navier–Stokes equations', *Int. J. Numer. Methods Fluids*, **7**, 1261–1275 (1987).
15. T.J.R. Hughes, 'Multiscale phenomena: Green's functions, subgrid scale models, bubbles, and the origins of stabilized methods', *Proc. 9th Int. Conf. on Finite Elements in Fluids*, Venezia, Italia, 1995, pp. 99–114.
16. I. Babuška, O.C. Zienkiewicz, J. Gago and E.R.A. Oliveira, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, Wiley, New York, 1986.
17. J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comp. Phys.*, **72**, 449–466 (1987).
18. N.P. Weatherill, 'A method for generating irregular computational grids in multiply connected planar domains', *Int. J. Numer. Methods Fluids*, **8**, 181–197 (1988).
19. A. Bowyer, 'Computing Dirichlet tessellations', *Comp. J.*, **24**, 162–166 (1981).
20. R. Löhner, 'Robust, vectorized search algorithms for interpolation on unstructured grids', *J. Comp. Phys.*, **118**, 380–387 (1995).
21. L.A. Crowl, 'How to measure, present and compare parallel performance', *IEEE Parallel Distrib. Technol.*, Spring, 9–25 (1994).
22. H.D. Simon, 'High performance computing: architecture, software and algorithms', *NAS Report RNR–93-018*, NASA Ames Research Center, 1993.