INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng. 2000; **48**:267–287

Application of the finite volume method and unstructured meshes to linear elasticity

H. Jasak[1,*,†] and H. G. Weller[2]

[1] *Computational Dynamics Ltd, Hythe House, 200 Shepherds Bush Road, London W6 7NY, U.K.*
[2] *Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, Exhibition Road, London SW7 2BX, U.K.*

SUMMARY

A recent emergence of the finite volume method (FVM) in structural analysis promises a viable alternative to the well-established finite element solvers. In this paper, the linear stress analysis problem is discretized using the practices usually associated with the FVM in fluid flows. These include the second-order accurate discretization on control volumes of arbitrary polyhedral shape; segregated solution procedure, in which the displacement components are solved consecutively and iterative solvers for the systems of linear algebraic equations. Special attention is given to the optimization of the discretization practice in order to provide rapid convergence for the segregated solution procedure. The solver is set-up to work efficiently on parallel distributed memory computer architectures, allowing a fast turn-around for the mesh sizes expected in an industrial environment. The methodology is validated on two test cases: stress concentration around a circular hole and transient wave propagation in a bar. Finally, the steady and transient stress analysis of a Diesel injector valve seat in 3-D is presented, together with the set of parallel speed-up results. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: finite volume; unstructured meshes; linear elasticity; steady state; transient; parallelism

1. INTRODUCTION

The field of the computational continuum mechanics (CCM) has generally been split between the finite element (FE) solvers, which seem to be unchallenged in the area of stress analysis, and the finite volume (FV) method, widely popular in fluid flows. Two numerical methods are usually associated with some distinct practices: the FE method is based on the variational principle, uses pre-defined shape functions dependent on the topology of the element, easily extends to higher order discretization, produces large block-matrices, usually with high condition numbers, and as a consequence relies on direct solvers. The FV method, on the other hand, is usually

*Correspondence to: H. Jasak, Computational Dynamics Ltd, Hythe House, 200 Shepherds Bush Road, London W6 7NY, U.K.
†E-mail: h.jasak@cd.co.uk

Received 19 February 1999
Revised 12 July 1999

second-order accurate, based on the integral form of the governing equation, uses a segregated solution procedure, where the coupling and non-linearity is treated in an iterative way, and creates diagonally dominant matrices well suited for iterative solvers. Although they are inherently similar, the two sets of practices have their advantages and disadvantages, which make them better suited for certain classes of problems. However, the situation is not as clear-cut as it might seem: for example, we cannot tell in advance whether the block solution associated with the FEM gives an *a priori* advantage over the segregated FV solver even for a simple linear elastic problem: this is a question of the trade-off between the high expense of the direct solver for a large matrix and the cheaper iterative solvers with the necessary iteration over the explicit cross-component coupling.

Although the FV discretization may be thought to be inferior to the FEM in linear elasticity, it is still compelling to examine its qualities. The reason for this may be the fact that the FVM is inherently good at treating complicated, coupled and non-linear differential equations, widely present in fluid flows. By extension, as the mathematical model becomes more complex, the FVM should become a more interesting alternative to the FEM.

Another reason to consider the use of the FVM in structural analysis is its efficiency. In recent years industrial computational fluid dynamics (CFD) has been dealing with the meshes of the order of 500 000 up to 100 million cells, which are necessary to produce accurate results for complex mathematical models and full-size geometries (e.g. car body aerodynamics, internal combustion engines, complete train, nuclear reactor assembly, etc.). This, in turn, has instigated remarkable improvements in the performance of the method in order to keep the computation time within acceptable limits. Modern FV solvers both vectorize and parallelize and it is not unusual to use massively parallel distributed memory computers with up to a thousand CPUs.

While the advent of FE methods in fluid flow dates back more than 20 years [1, 2], the opposite trend is of a much later date [3–6]. Several examples of the FV discretization in linear stress analysis can already be found in the literature [4–8] but they regularly employ multigrid solvers to speed-up convergence. In this paper, we shall examine the performance of a FV-type solver on the steady and transient linear stress analysis problem as the necessary first step before the extension to the more complicated constitutive relations. If successful, the door to further extension of the FV method to the variety of structural mechanics problems is open.

This paper describes a FV linear stress analysis solver of reasonable efficiency without multigrid acceleration, applicable to both steady-state and transient problems. It will be shown that the decomposition of the stress term into the shear and pure rotation contributions in the discretized form results in smooth and rapid convergence. The algorithm will also be adapted for parallel distributed memory computer architectures in order to achieve fast convergence on large meshes.

The rest of the text will be structured as follows: the mathematical model for a linear elastic solid will be described in Section 2. Sections 3–5 review the basics of the finite volume method, describe the details of the solution procedure and address the parallelization issues. The new solution method will then be tested on two simple problems in Sections 6.1 and 6.2, in order to illustrate its accuracy and convergence. Finally, in Section 6.3, the method is applied on a realistic geometry and a series of meshes going up to 360 000 CVs, or in FE terms 1.2 million degrees of freedom in both steady-state and transient mode. A set of parallel performance results including the real execution times is also provided. The paper is completed with a summary in Section 7.

2. MATHEMATICAL MODEL

For the purpose of this paper, we shall limit ourselves to the simplest mathematical model: a linear elastic solid. The model can be summarized as follows:

The force balance for the solid body element in its differential form states:

$$\frac{\partial^2(\rho\mathbf{u})}{\partial t^2} - \nabla . \sigma = \rho\mathbf{f} \tag{1}$$

where $\mathbf{u}$ is the displacement vector, ρ is the density, $\mathbf{f}$ is the body force and σ is the stress tensor.

The strain tensor ε is defined in terms of $\mathbf{u}$:

$$\varepsilon = \tfrac{1}{2}[\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathrm{T}}] \tag{2}$$

The Hooke's law, relating the stress and strain tensors, closes the system of equations:

$$\sigma = 2\mu\varepsilon + \lambda\,\mathrm{tr}(\varepsilon)\,\mathbf{I} \tag{3}$$

where $\mathbf{I}$ is the unit tensor and μ and λ are Lamé's coefficients, relating to Young's modulus of elasticity E and Poisson's ratio v as

$$\mu = \frac{E}{2(1+v)} \tag{4}$$

and

$$\lambda = \begin{cases} \dfrac{vE}{(1+v)(1-v)} & \text{for plane stress} \\ \dfrac{vE}{(1+v)(1-2v)} & \text{for plane strain and 3-D} \end{cases} \tag{5}$$

Using the above, the governing equation can be rewritten with the displacement vector $\mathbf{u}$ as the primitive variable:

$$\frac{\partial^2(\rho\mathbf{u})}{\partial t^2} - \nabla . [\mu\nabla\mathbf{u} + \mu(\nabla\mathbf{u})^{\mathrm{T}} + \lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u})] = \rho\mathbf{f} \tag{6}$$

The specification of the problem is completed with the definition of the solution domain in space and time and the initial and boundary conditions. The initial condition consists of the specified distribution of $\mathbf{u}$ and $\partial\mathbf{u}/\partial t$ at time zero. The boundary conditions, either constant or time varying, can be of the following type:

(i) fixed displacement,
(ii) planes of symmetry,
(iii) fixed pressure,
(iv) fixed traction and
(v) free surfaces (zero traction).

The problem is considered to be solved when the displacement is calculated; this can consequently be used to calculate the strain/stress distribution using Equations (2) and (3), or any other variables of interest.

3. FINITE VOLUME DISCRETIZATION AND SOLUTION ALGORITHM

The FV discretization is based on the integral form of the equation over the control volume (CV). The discretization procedure is separated in two parts: *discretization of the computational domain* and *equation discretization*.

3.1. Discretization of the computational domain

Discretization of the computational domain consists of discretization of the time interval and discretization of space. Since time is a parabolic co-ordinate, it is sufficient to specify the size of the time-step for transient calculations (for steady-state problems, the time-step is effectively set to infinity). The space discretization subdivides the spatial domain into a number of polyhedral CVs that do not overlap and completely fill the domain. Every internal face is shared by two CVs. A typical CV, with the computational point P in its centroid, is shown in Figure 1. The face f and the centroid N of the neighbouring CV sharing that face are also marked. This type of the computational mesh is termed to be arbitrarily unstructured [8–10] and offers considerable freedom in mesh generation.

Unlike the FEM, the FV discretization allows us to assemble a second-order accurate discretization irrespective of the shape of the CV, as there is no need to *a priori* postulate a topology-dependent shape function. In other words, different cell shapes can be 'mixed and matched' at will.

3.2. Equation discretization

The FV method of discretization uses the integral form of Equation (6) over the CV around point P with the volume V_P. Using the Gauss' theorem it follows:

$$\int_{V_P} \frac{\partial^2(\rho \mathbf{u})}{\partial t^2}\,\mathrm{d}V - \oint_{\partial V_P} \mathrm{d}\mathbf{s}.[\mu \nabla \mathbf{u} + \mu(\nabla \mathbf{u})^{\mathrm{T}} + \lambda \mathbf{I}\,\mathrm{tr}(\nabla \mathbf{u})] = \int_{V_P} \rho \mathbf{f}\,\mathrm{d}V \tag{7}$$

Consistent with the practices usually applied in the FVM for fluid flow, the above equation will be discretized in a *segregated manner*, where each component of the displacement vector is solved separately and the inter-component coupling is treated explicitly. The result of this approach will be well-structured diagonally dominant sparse matrices ideally suited for iterative solvers. A considerable saving in the computer memory will also be achieved: instead of one large matrix covering all three components of displacement usually seen in the FEM [11], we will have three smaller matrices, solved consecutively. Also, the iterative solver used in this study preserves the sparseness pattern of the original matrix, causing no additional memory requirement. This kind of practice allows us to solve linear elasticity problems on meshes of the order of 500 000 CVs (or 1.5 million of degrees of freedom) on a relatively small workstation. The drawback of the above method lies in the fact that it is now necessary to iterate over the inter-component coupling but, as will be shown later, careful discretization results in fast and reliable convergence.

Let us now present the discretization of the above equation on a term-by-term basis.

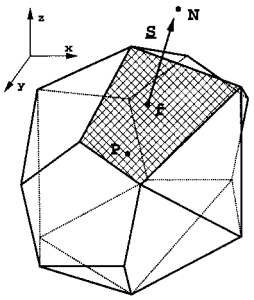

Figure 1. Control volume.

The temporal derivative is calculated using two old-time levels of $\mathbf{u}$:

$$\frac{\partial^2 \mathbf{u}}{\partial t^2} = \frac{\mathbf{u}^n - 2\mathbf{u}^o + \mathbf{u}^{oo}}{\Delta t^2} \tag{8}$$

where $\mathbf{u}^n = \mathbf{u}(t + \Delta t)$, $\mathbf{u}^o = \mathbf{u}(t)$ and $\mathbf{u}^{oo} = \mathbf{u}(t - \Delta t)$. This form of discretization is bounded but only first-order accurate in time and causes a certain amount of numerical dissipation, dependent of the Co number (based on the speed of sound). One can also construct a second-order accurate form of $\partial^2 \mathbf{u}/\partial t^2$ using three 'old-time' levels ($\mathbf{u}^{ooo} = \mathbf{u}(t - 2\,\Delta t)$):

$$\frac{\partial^2 \mathbf{u}}{\partial t^2} = \frac{2\mathbf{u}^n - 5\mathbf{u}^o + 4\mathbf{u}^{oo} - \mathbf{u}^{ooo}}{\Delta t^2} \tag{9}$$

Although Equation (9) is nominally more accurate than Equation (8), it does not preserve the boundedness of the differential form of the operator. In practice, this potentially causes unphysical stress peaks or even solution instability. For this reason, the first-order accurate temporal discretization, Equation (8), is preferred.

A second-order accurate approximation in space is obtained by assuming a linear variation of $\mathbf{u}$ over the CV:

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_P + (\mathbf{x} - \mathbf{x}_P).(\nabla \mathbf{u})_P \tag{10}$$

The volume integrals are evaluated using the mid-point rule

$$\int_{V_P} \phi \, \mathrm{d}V = \phi_P \, V_P \tag{11}$$

The surface integrals in Equation (7) are split into the sum of integrals over the cell faces and also evaluated using the mid-point rule. Let us first examine the discretization of the Div-Grad term:

$$\int_{V_P} \nabla.(\mu \nabla \mathbf{u}) \, \mathrm{d}V = \oint_{\partial V_P} \mathrm{d}\mathbf{s}.(\mu \nabla \mathbf{u}) = \sum_f \mu_f \mathbf{s}.(\nabla \mathbf{u})_f \tag{12}$$

We shall recognise two types of discretization:

(i) The *implicit discretization*. The term will be discretized assuming that the face area vector $\mathbf{s}$ and vector $\mathbf{d}_N = \overline{PN}$ (Figure 1) are parallel. It follows:

$$\mathbf{s}.(\nabla \mathbf{u})_f = |\mathbf{s}| \frac{\mathbf{u}_N - \mathbf{u}_P}{|\mathbf{d}_N|} \tag{13}$$

Equation (13) allows us to create an algebraic equation in which the value of $\nabla \cdot \nabla \mathbf{u}_P$ depends only on the values in P and the nearest neighbours of P:

$$\oint_{\partial V_P} \mathrm{d}\mathbf{s}.(\mu \nabla \mathbf{u}) = a_P \mathbf{u}_P + \sum_N a_N \mathbf{u}_N \tag{14}$$

where

$$a_N = \mu_f \frac{|\mathbf{s}|}{|\mathbf{d}_N|} \tag{15}$$

and

$$a_P = \sum_N -a_N \tag{16}$$

If the vectors $\mathbf{s}$ and $\mathbf{d}_N$ are not parallel, a 'non-orthogonal correction' is added. For the details of different non-orthogonality treatments the reader is referred to Reference [9]. For constant material properties μ_f is simply equal to μ.

(ii) The *explicit discretization*. Here, the term is discretized using Equation (12) and the interpolated gradients:

$$(\nabla \mathbf{u})_f = f_x (\nabla \mathbf{u})_P + (1 - f_x)(\nabla \mathbf{u})_N \tag{17}$$

where f_x is the interpolation coefficient. Unlike the implicit formulation, the term is now evaluated from the current values of $\nabla \mathbf{u}$ (i.e. from the available distribution of $\mathbf{u}$).

Other terms in Equation (6), namely $\nabla.[\mu(\nabla \mathbf{u})^{\mathrm{T}}]$ and $\nabla.[\lambda \mathbf{I}\,\mathrm{tr}(\nabla \mathbf{u})]$ are discretized in an explicit manner, as they contain the inter-component coupling.[‡]

Evaluation of the gradient: The cell centre gradient is calculated using the least-square fit [8] in the following manner: consider the cell P and the set of its nearest neighbours N. Assuming a linear variation of a general variable ϕ, the error at N is

$$e_N = \phi_N - (\phi_P + \mathbf{d}_N.(\nabla \phi)_P) \tag{18}$$

Minimizing the $e_P^2 = \sum_N (w_N e_N)^2$ ($w_N = 1/|\mathbf{d}_N|$ is the weighting function) leads to the following expression:

$$(\nabla \phi)_P = \sum_N w_N^2 \, \mathbf{G}^{-1}.\mathbf{d}_N (\phi_N - \phi_P) \tag{19}$$

where $\mathbf{G}$ is a 3×3 symmetric matrix

$$\mathbf{G} = \sum_N w_N^2 \, \mathbf{d}_N \mathbf{d}_N \tag{20}$$

[‡]In fact, a part of these terms could also, under certain conditions, be made implicit. More details of such a practice will be given in Section 4.

This produces a second-order accurate gradient irrespective of the arrangement of the neighbouring points. Moreover, the matrix **G** can be inverted only once and stored to increase the computational efficiency.

3.3. Boundary conditions

The boundary condition types mentioned in Section 2 can be divided into:

(i) Condition which specify the value of **u** on the boundary face. The necessary face gradient is then computed using the cell centre value in the neighbouring cell and taken into account in an appropriate manner.
(ii) The discretization on the *plane of symmetry* is constructed by imagining a CV on the other side of the boundary as a mirror image of the CV next to the symmetry plane.
(iii) The *traction* boundary condition (fixed pressure and free surfaces are also included here) specifies the force on the boundary face

$$\mathbf{g}_b = |\mathbf{s}_b|\,\mathbf{t} - \mathbf{s}_b p \tag{21}$$

where $\mathbf{s}_b$ is the outward-pointing boundary face area vector, t is the specified traction and p the pressure. The governing equation, Equation (7), actually represents the force balance for the CV: $\mathbf{g}_b$ is therefore directly added into the balance.

3.4. Solution procedure

Assembling Equation 7 using Equations (8), (11), (12), (14) (17) and (19) produces the following:

$$a_P \mathbf{u}_P + \sum_N a_N \mathbf{u}_N = \mathbf{r}_P \tag{22}$$

with one equation assembled for each CV. a_P and $\mathbf{r}_P$ now also include the contributions from the temporal term and the boundary conditions. Here, $\mathbf{u}_P$ depends on the values in the neighbouring cells, thus creating a system of algebraic equations

$$[A][\mathbf{u}] = [\mathbf{r}] \tag{23}$$

where $[A]$ is the sparse matrix, with coefficients a_P on the diagonal and a_N off the diagonal, $[\mathbf{u}]$ is the vector of **u**s for all CVs and $[\mathbf{r}]$ is the right-hand side vector. The above system will be solved consecutively for the three components of **u**.

The matrix $[A]$ from Equation (23) is symmetric and diagonally dominant even in the absence of the transient term, which is important for steady-state calculations. The system of equations will be solved using the incomplete Cholesky conjugate gradient solver (ICCG) [12, 13].

The discretized system described above includes some explicit terms, depending on the displacement from the previous iteration. It would therefore be unnecessary to converge the solution of Equation (23) to a very tight tolerance, as the new solution will only be used to update the explicit terms. Only when the solution changes less than some pre-defined tolerance the system is considered to be solved. In transient calculations, this will be done for every time-step, using the previously available solution as the initial guess.

4. NUMERICAL CONSIDERATIONS

Unfortunately, the above split into the implicit part, containing the temporal derivative and the $\nabla.(\mu\nabla\mathbf{u})$ and the explicit part containing everything else, results in the discretization practice that is at best only marginally convergent. The trouble is that the explicit terms carry more information than their implicit counterparts and the convergence can be achieved only with extensive under-relaxation. This is clearly not beneficial since it considerably slows the convergence; an alternative practice is needed.

The hint on the necessary modification can be obtained from the simplified analysis. Imagine a computational mesh in which all CVs are cubes aligned with the co-ordinate system. Such an arrangement allows us to produce the implicit discretization for the part of the $\nabla.[\lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u})]$ term and, indeed the $\nabla.(\mu(\nabla\mathbf{u})^{\mathrm{T}})$ term [4]. For example, the coefficient for the x-component of $\mathbf{u}$ for the neighbour on the right would be [4]

$$a_E = (2\mu + \lambda)\frac{|\mathbf{s}|}{|\mathbf{d}_E|} \tag{24}$$

and for the neighbour above

$$a_N = \mu\frac{|\mathbf{s}|}{|\mathbf{d}_N|} \tag{25}$$

For the y-component of $\mathbf{u}$ the situation would be the opposite. This idea can be extended to arbitrarily unstructured meshes by taking into account the angle between the face area vector and the co-ordinate directions. Although this practice regularly converges, the convergence is relatively slow; it can be accelerated by multigrid acceleration techniques [5], ideally suited for this kind of problems. The undesirable feature of this method is that, unlike in Equation (22), the matrix $[A]$ is now different for each component of $\mathbf{u}$.

Here, we shall examine a different path and construct the discretization procedure which works well even without multigrid acceleration and at the same time keep the matrix $[A]$ equal for all components of $\mathbf{u}$. As a reminder, Equation (6) has been discretized in the following way:

$$\frac{\partial^2(\rho\mathbf{u})}{\partial t^2} - \underbrace{\nabla.(\mu\nabla\mathbf{u})}_{\text{implicit}} - \underbrace{\nabla.[\mu(\nabla\mathbf{u})^{\mathrm{T}} + \lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u})]}_{\text{explicit}} = \rho\mathbf{f} \tag{26}$$

Using the hint from Equations (24) and (25), we shall re-write Equation (26) as

$$\frac{\partial^2(\rho\mathbf{u})}{\partial t^2} - \underbrace{\nabla.[(2\,\mu + \lambda)\nabla\mathbf{u}]}_{\text{implicit}} - \underbrace{\nabla.[\mu(\nabla\mathbf{u})^{\mathrm{T}} + \lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u}) - (\mu + \lambda)\nabla\mathbf{u}]}_{\text{explicit}} = \rho\mathbf{f} \tag{27}$$

The matrix has now been 'over-relaxed': it includes the terms which could nominally be discretized implicitly only under mesh alignment. If this is not the case, the additional terms are taken out in an explicit manner. As will be shown later, the resulting convergence of the method is impressive. Moreover, the a_P and a_N coefficients are identical for all components of $\mathbf{u}$.

Let us extend the analysis of Equation (27) a bit further, considering $\mu = \text{const.}$ and $\lambda = \text{const.}$ In this case

$$\nabla.[\lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u})] = \lambda\nabla.(\mathbf{I}\nabla.\mathbf{u}) = \lambda\nabla(\nabla.\mathbf{u}) = \lambda\nabla.(\nabla\mathbf{u})^{\mathrm{T}} \tag{28}$$

Using the above, the explicit term from Equation (27) reads

$$\nabla.[\mu(\nabla\mathbf{u})^{\mathrm{T}} + \lambda\mathbf{I}\,\mathrm{tr}(\nabla\mathbf{u}) - (\mu+\lambda)\nabla\mathbf{u}] = -\nabla.[(\mu+\lambda)(\nabla\mathbf{u} - (\nabla\mathbf{u})^{\mathrm{T}})] \tag{29}$$

which is a pure rotation and as such implies explicit treatment in a segregated algorithm. It follows that the implicit part of Equation (27) is the maximum consistent implicit contribution to the component-wise discretization.

5. PARALLELIZATION ISSUES

The issue of convergence acceleration for the FVM based on multigrid acceleration for the type of discretization similar to above has been examined in considerable detail, both for fluid flows [14, 15] and stress analysis [5]. Here, we shall examine a different way of accelerating the calculation: parallelization of the solver.

A number of different parallelization strategies used in CFD have been described in Reference [16] and their performance and limitations are well known [17]. Here, we shall parallelize the calculation using the *domain decomposition* approach, which seems most appropriate for our circumstances. The parallelization is done by splitting the spatial domain into a number of sub-domains, each of which is assigned to one processor of a parallel (distributed memory) computer. The necessary exchange of information on inter-processor boundaries is done using one of the message-passing protocols (in this case, PVM [18]). For more details on parallel and high performance programming, the reader is referred to References [19, 20].

Analysis of the computer code shows that all the operations are naturally parallelizable, with the exception of the incomplete Cholesky preconditioning. Here, we have a choice: we can either resort to the simpler (and parallelizable) diagonal preconditioning [13] without any degradation of the parallel solver performance, or use the (recursive) incomplete Cholesky preconditioning separately on each of the sub-domains, thus avoiding the parallelization problem. The second practice causes some solver degradation, depending on the domain decomposition and the number of processors but, on balance, it still converges faster in real time than the diagonally preconditioned CG solver.

6. TEST CASES

The discretization method described in Section 3 has been implemented in the field operation and manipulation (FOAM) C++ library [21] developed by the authors and co-workers at Imperial College. In this section, we shall apply the code on three test cases. The first two, Sections 6.1, and 6.2 are aimed at validating the accuracy of the method on a simple steady-state and transient case, as well as examining its convergence properties. The third case (Section 6.3) illustrates the application of the method on a real-life engineering problem: a 3-D calculation of the stress distribution in a Diesel injector valve seat. Here, apart from the stress distribution under a constant and variable load, we will also present the performance of the parallel algorithm. This problem will be solved on three meshes going up to 360 000 CVs, with the aim to produce a fine-mesh solution of appropriate accuracy in the 1 h time frame.

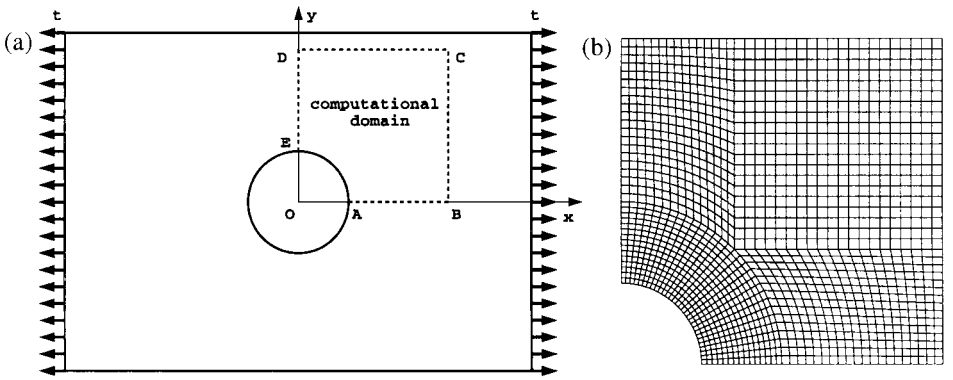

Figure 2. Stress concentration around a circular hole: (a) test setup; and (b) computational mesh.

6.1. Stress concentration around a circular hole

This, well publicized test case [4, 5], consists of an infinitely large thin plate with a circular hole loaded by uniform tension in one direction (Figure 2(a)).

The analytical solution for this problem can be found in Reference [22]. Due to the symmetry of the problem, only one quarter of the plate is modelled. Also, the plane stress condition is imposed. Following [4], the exact solution corresponding to $t = 10^4$ Pa is prescribed on the *BCD* boundary to remove the effects of the finite size of the computational domain. The symmetry plane condition is applied on *AB* and *DE*; *AE* is a zero-traction boundary. The mesh consists of 1 450 CVs and is shown in Figure 2(b). The material properties used are that of steel:

$$\rho = 7854\,\mathrm{kg/m^3}$$
$$E = 2.0 \times 10^{11}\,\mathrm{Pa}$$
$$v = 0.3 \tag{30}$$

Experience shows that the solution can be considered appropriate once the global residual (for the segregated system of equations) reaches 5×10^{-5}, but the calculation will be continued until the machine tolerance (10^{-8}) is reached. The iteration tolerance, prescribing the ratio of residuals before and after the component matrix solution is set to $r = 0.2$.

The comparison between the analytical and numerical stress distribution is shown in Figure 3, with the maximum error in σ_{xx} of 0.62 per cent. The convergence tolerance of 5×10^{-5} has been reached in 52 iterations, which on a Silicon Graphics 100 MHz R 4000 workstation took 55.7 s.

Although the above calculation shows remarkable accuracy, it does not represent a realistic test case, as the exact boundary condition has been prescribed on all boundaries. We shall now somewhat modify the test set-up in order to objectively examine the convergence: the fixed constant traction of 10^4 Pa will be applied on *BC* and zero traction on *CD*; the effects of the finite geometry now come into action.

The residual history for the calculation is given in Figure 4, showing rapid and smooth convergence. The solution has been reached in 59 iterations (or 62.3 s), much the same as before. The stress concentration now equals to 3.28, in line with expectations.

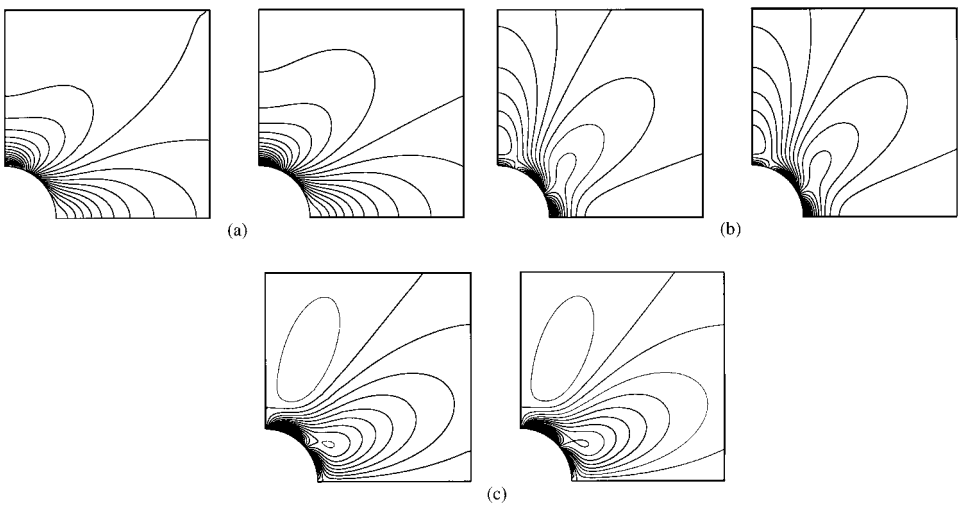

Figure 3. Stress concentration around a circular hole: comparison of the numerical solution (left) and the analytical solution (right): (a) σ_{xx} contours between 0 and 30 000 Pa; (b) σ_{yy} contours between $-10\,000$ and 6000 Pa; and (c) σ_{xy} contours between $-10\,000$ and 2000 Pa.

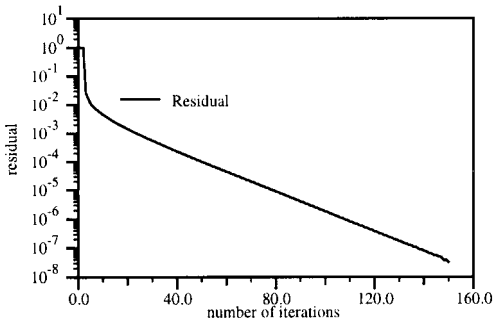

Figure 4. Circular hole: convergence history.

The stress distribution shown in Figure 3 reveals that the solution is very smooth away from the hole, implying that the mesh is too fine relative to the local error. It is therefore expected that an adaptive mesh refinement technique based on an *a posteriori* error estimate similar to the one in Reference [9] should produce great savings both in computation time and the number of CVs for a given accuracy, but this is beyond the scope of this paper.

6.2. Transient wave propagation in a bar

We shall now present an example transient calculation. The test set-up consists of a bar 10 m long and 1 m wide. The properties of steel, Equation (30) are used. The bar is considered thin and a plane stress assumption is used.

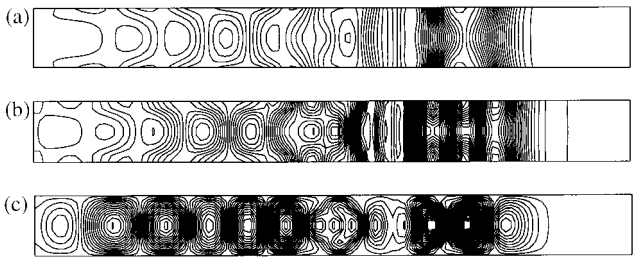

Figure 5. Transient wave propagation in a bar, $t = 1.5$ ms: (a) x-displacement contours between 0 and 1.3 mm; (b) σ_{xx} contours between -267 and 131 MPa; and (c) σ_{yy} contours between -84 and 100 MPa.

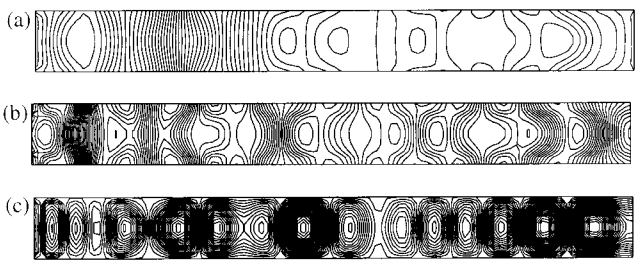

Figure 6. Transient wave propagation in a bar, $t = 4.5$ ms: (a) x-displacement contours between 0 and 1.3 mm; (b) σ_{xx} contours between -135 and 84 MPa; and (c) σ_{yy} contours between -50 and 40 MPa.

The initial condition is $\mathbf{u} = \mathbf{0}$ and $\partial \mathbf{u}/\partial t = \mathbf{0}$ everywhere. At time zero, a fixed displacement of 1 mm is prescribed at the left end of the bar; the right end is fixed. This will cause the propagation of the stress wave through the bar at the speed of sound, which for steel equals:

$$C = \sqrt{\frac{E}{\rho}} = \sqrt{\frac{2 \times 10^{11}}{7854}} = 5046.2 \text{ m/s} \tag{31}$$

When the stress wave reaches the other end of the bar it will reflect and travel backwards. A secondary effect of the transverse waves, caused by the Poisson's effect will also be visible.

The mesh consists of 100×9 CVs and the solution will be converged to 10^{-7} for each time-step. The calculation will be done on two different Courant numbers (based on the sonic velocity): $\text{Co} = 0.5$ and 0.05, giving the time-step size of 10^{-5} and 10^{-6} s, respectively. In addition, a second-order accurate temporal discretization will be presented for the larger Co number.

Figures 5 and 6 show the distribution of the x-component of the displacement and σ_{xx} for $\text{Co} = 0.05$ at $t = 1.5$ and 4.5 ms, respectively. The transverse waves in the solution can be clearly seen, as well as the fact that the wave has been reflected off the right-hand boundary between the two times presented.

The variation of $\mathbf{u}$ and σ_{xx} in time for the point located in the middle of the bar is shown in Figure 7. The wave propagation speed can now be easily checked: according to Equation (31), the wave should reach the point in question in exactly 0.991 ms, clearly seen in the time trace.

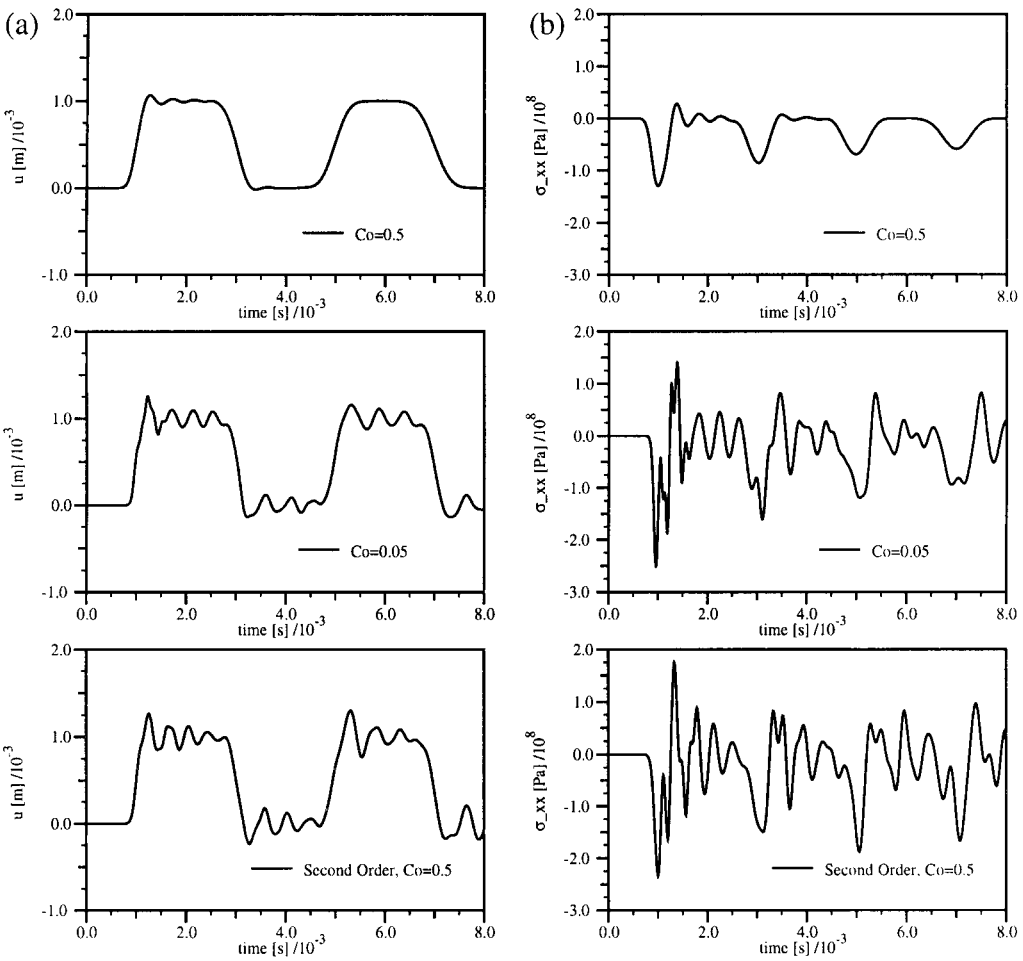

Figure 7. Variation of u and σ_{xx} in time at (5, 0.5): (a) u against time; and (b) σ_{xx} against time.

Finally, a word on temporal accuracy: Figure 7 shows that the transverse waves die out much quickly for the larger Co number in the case of first-order temporal accuracy. Also, the maximum stress in Figure 7(b) decreases in every cycle, but the situation is not as severe as with the transverse waves. This is the consequence of numerical diffusion introduced by the first-order temporal discretization [9]. The third row of graphs in Figure 7 shows the results for second-order accurate temporal scheme with $\mathrm{Co} = 0.5$. The result is of similar accuracy as the first-order solution with $\mathrm{Co} = 0.05$, at a considerably lower cost (the number of time-steps is now 10 times lower). However, careful analysis of the result reveals the tendency of the second-order temporal scheme to over- and under-shoot in the presence of steep gradients, resulting in unrealistic peaks in the calculated stress distribution. For that reason, a first-order time scheme and a small time-step are preferred.

The run times for the three runs are given in Table I below.

Table I. Transient run: CPU time for 0.0008 s simulation time.

Co number	No. of steps	CPU time (s)	CPU time per step (s)
0.5	800	937.8	1.172
0.05	8000	8653.2	1.082
0.5 (second order)	800	948.92	1.186

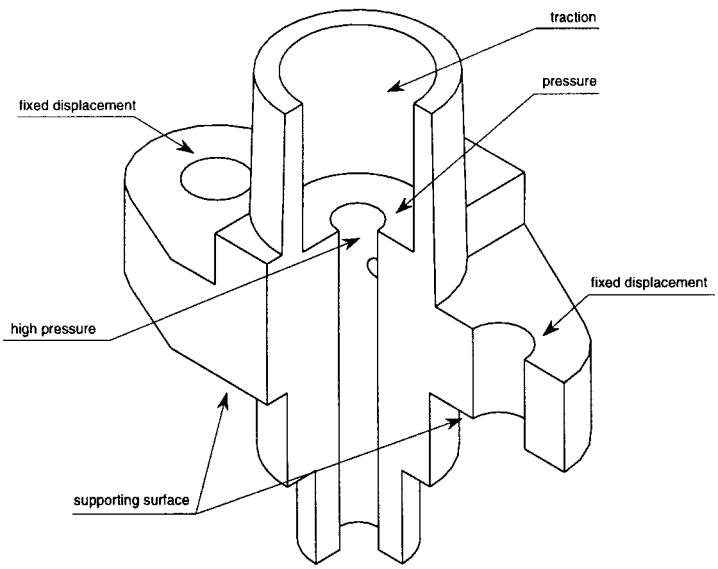

Figure 8. Valve seat: boundary conditions.

6.3. Diesel injector valve seat

The final test case illustrates the application of the method to a realistic geometry. We shall calculate the distribution of the stress in the valve seat of a Diesel injector. The boundary conditions prescribed on the valve (Figure 8), represent the working conditions of the valve seat: the bottom surface of the flange is supported in the horizontal plane and a part of the top surface is fixed. Inside, a combination of pressure boundary conditions is used: parts of the valve are subjected to the pressure of up to 1 000 bar, simulating the realistic working load. Also, graded traction of up to 0.36 MPa on the inside surface of the top part is used. The material properties of steel, Equation (30), are used again.

Although a plane of symmetry exists, the whole geometry will be meshed, as the purpose of the exercise is to examine the performance of the method on large meshes. The coarsest mesh (Figure 9), consisting of 5712 CVs has been systematically refined, to create the finest mesh with 359 616 CVs. The mesh consists of a combination of hexahedra, pyramids and prisms.

6.3.1. Steady-state calculation and parallel performance. The objective of this calculation is to examine the efficiency of the method rather than examine the stress distribution for the case in

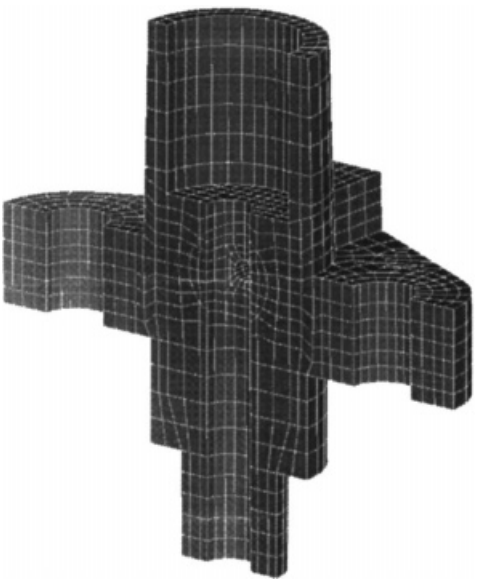

Figure 9. Valve seat: coarse mesh, 5712 CVs.

question: we shall therefore present only a very limited set of results. Figure 10 shows the σ_{zz} in four cutting planes. The actual distribution is presented as the 'deformation' of the rubber sheet away from the plane. The highest stress concentration is located at the intersection of the two internal channels, giving the peak equivalent stress of $\sigma_{\mathrm{eq}} = 264\,\mathrm{MPa}$.

Let us now examine the efficiency of the method. As mentioned before, the objective is to obtain the fine-mesh solution in less than 1 h by the use of parallel computers. The platform available for this study is a 24 CPU 195 MHz R 10 000 SGI Origin 2000 machine. We shall further complicate the matter by using the machine in a non-dedicated mode (i.e. it is being used by other users at the same time). This will somewhat invalidate the parallelization results, but will produce a more realistic picture of the code performance.

Table II presents the CPU time used to produce the solution converged to 5×10^{-5} on a single processor. The last column shows the 'slow-down' of the calculation: for example, the finest mesh has got 64 times as many CVs as the coarse mesh but takes 372 times longer to run, giving the relative slow-down of 0.172. The convergence is still smooth and monotonic: the residual history for the finest mesh is given in Figure 11. The memory requirement for the finest mesh is 333 MB in double precision, or approximately 900 bytes per CV.

It takes almost 10 h to produce a fine-mesh solution on a single CPU, way higher than our goal. This, however, is not totally unrealistic: we could produce the solution in an over-night run on a relatively modest workstation. On the other hand, if a parallel machine is available, a much higher turn-around can be achieved.

Table III presents the CPU time necessary to produce the converged solution on the parallel machine. For control, all the runs have been independently run to 600 iterations, as the speedup varies depending on the other load on the machine. A few words of comment are due:

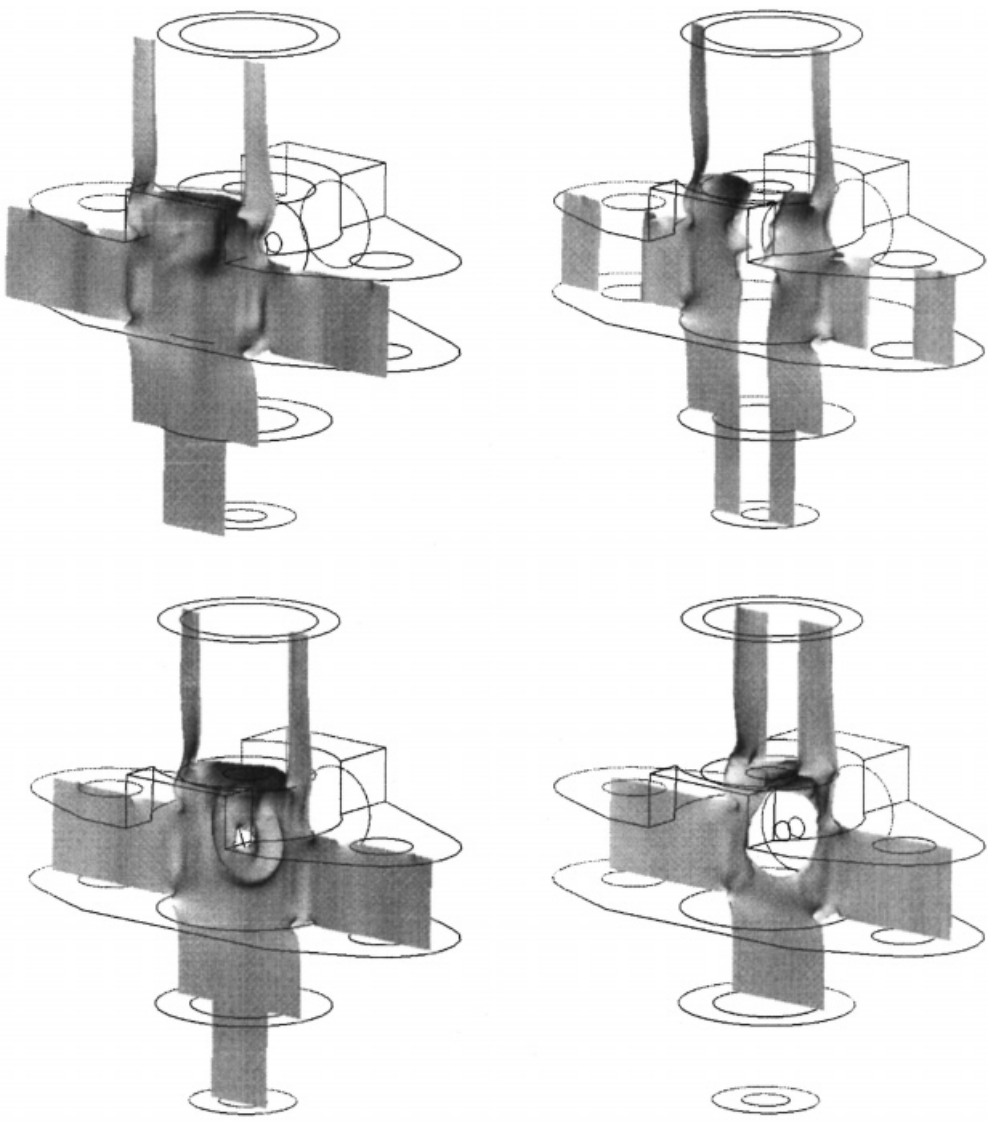

Figure 10. Diesel injector: distribution of σ_{zz} in four planes.

Table II. Single-processor CPU time to convergence.

Mesh size	CPU time (s)	CPU/CPU (coarse)	Relative slow-down
5712	95.77	1.0	1.0
44 952	2244.87	23.4	0.68
359 616	35620.43	371.9	0.172

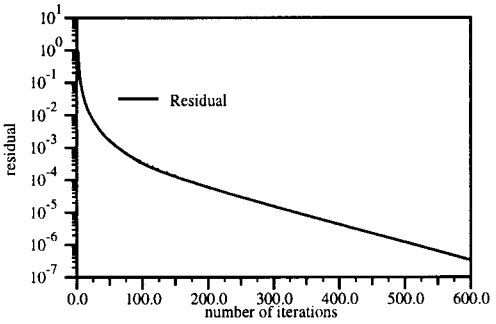

Figure 11. Diesel injector: convergence history.

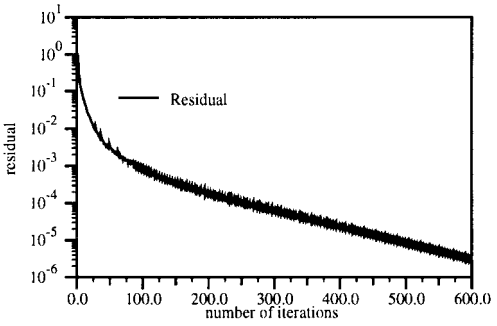

Figure 12. Diesel injector: convergence history with $r = 0.5$.

Table III. CPU time to convergence for the finest mesh on a parallel machine.

No. of CPUs	CPU time (5×10^{-5}) (s)	CPU time (600 iter.) (s)	Speed-up
1	35620.4	90785.2	1.00
2	22398.8	56605.2	1.60
4	11406.6	29244.2	3.10
8	4247.32	10218.6	8.88
16	3766.13	8498.09	10.68

(i) Parallel performance data needs to be examined in the light of the possible degradation of the incomplete Cholesky (IC) preconditioning, which ultimately depends on the initial mesh decomposition. If the diagonal preconditioning were used instead, the scaling would be close to linear. Our purpose here, however, is not to present just another set of parallel performance results, but to illustrate realistic run-times for a realistic problem. The IC preconditioning is therefore preferred: in spite of the unfavourable scaling it ultimately runs faster than the diagonal preconditioning.

(ii) A 'curious' result of super-linear scaling for 8 CPUs is actually caused by better cacheing of the data: as the part of the mesh assigned to a single CPU becomes smaller, the processor can access it faster, thus compensating for the solver degradation. Also, the 16-CPU decomposition has been marred by a slight load imbalance (10.8 per cent), but this has been considered too small to be compensated out.

(iii) The final two results (8 and 16 CPU) are influenced by other load on the machine: in order to get a better grasp on the real performance, the runs have been repeated 3 times, giving speeds-up ranging between 6.6 and 8.8 for 8 CPUs and 10.2 to 12.4 for 16 CPUs.

(iv) We have almost reached our goal: the 16 CPU run takes less than 63 min for the desired accuracy.

Although 63 min is not a bad result, we shall make one more attempt to break the 1-h limit. It has been noted that the solver uses what seems a relatively large number of sweeps in the later

Table IV. Description of the run with interpolated solutions.

Operation	No. of CPUs	CPU time (s)
Coarse mesh solution	1	95.7
Interpolation to intermediate mesh	1	33
Intermediate mesh solution	1	1231.1
Interpolation to fine mesh	1	273
Parallel decomposition of data (8 CPU)	1	186
Parallel decomposition of data (16 CPU)	1	135
Parallel fine mesh solution	8	4537.3
Parallel fine mesh solution	16	3400.7
Total (8 CPU)	8	6356.1
Total (16 CPU)	16	5168.5

stages of the calculation. The solution is now close to its final shape and changes very slowly. If the number of sweeps could be reduced even marginally, the 1-h objective could be easily reached. The number of solver sweeps can be controlled through the iteration tolerance, which will now be increased to $r = 0.5$. The run has again been executed on 16-CPUs and we can finally report success: the convergence has been reached in 1879.05 s, or slightly more than 31 min. The convergence curve (Figure 12), is not as smooth as before, but the speedup has well beaten our expectations.

An alternative way of achieving the desired speed will now be examined: inspired by the multigrid procedure, we shall first solve the problem on the coarse mesh and interpolate the solution to the intermediate mesh, and use it as the initial guess. The same will then be done with the two fine meshes. The first two meshes are run on a single CPU and the final solution is again obtained on 8 or 16 CPUs. Each of the runs will be converged to 5×10^{-5}. Table IV gives an overview of the whole procedure.

Based on the above data, two conclusions can be reached. Firstly, we did not speed up the overall solution time for the finest mesh, with the 16-CPU run taking slightly more than 86 min. However, we have achieved two desirable side-effects: instead of a single solution, we now have three solutions on systematically refined meshes which can be used to estimate the discretization error using Richardson extrapolation or a similar procedure. Secondly, we have managed to somewhat reduce the load on the parallel machine: a part of the job is now done in a single-CPU mode.

6.3.2. Transient calculation. Under working conditions, the valve seat is subjected to the load that rapidly varies in time, as the valve opens and closes. Under such conditions, a steady-state calculation may not give a complete picture of the stress distribution. We shall now present a transient calculation with time-varying load.

Figure 13 shows the prescribed variation of the pressure inside the valve in time, changing from zero to the maximum value and back in 0.1 ms, a realistic time for the piece in question. The variable boundary condition will produce travelling stress waves, similar to the one in Section 6.2. Here, we can also expect interesting interaction of the waves reflected from different boundaries, superimposed over the 'steady-state' solution. The time-step for the calculation is set to 2×10^{-8} s, allowing us to produce a result of good temporal accuracy. The total simulation time will be 0.2 ms, or 10 000 time-steps.

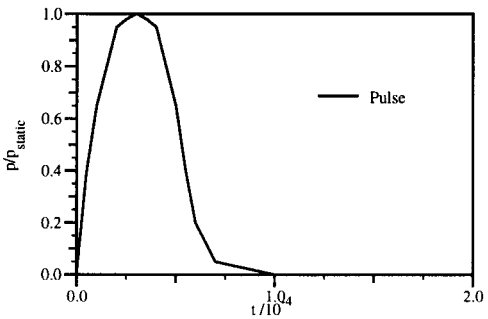

Figure 13. Transient calculation: variation of the pressure in time.

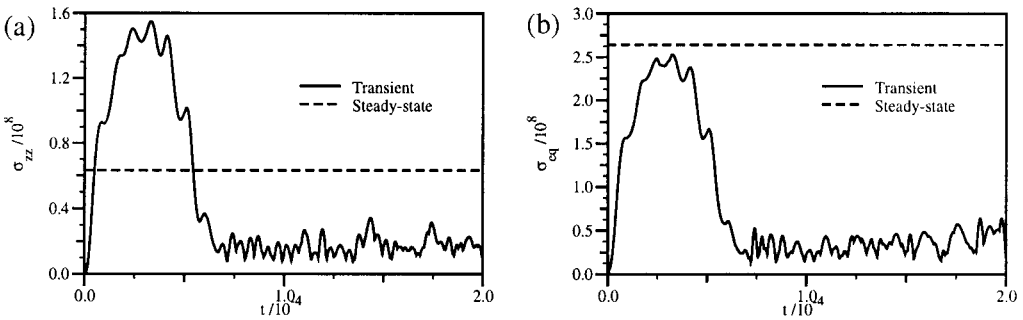

Figure 14. Transient calculation: variation of the maximum stress in time; (a) Max. σ_{zz} against time; and (b) max. σ_{eq} against time.

Before we present the results, the reader should be reminded that the amount of data that will be created will be extremely high, consisting of 10 000 fields of displacement and stress. Obviously, it would be impractical to store such amounts of data and extract the necessary information after the run is finished; it is much more practical to deal with the data 'on the fly' and only store the parameters of engineering interest. In our case, we shall follow the variation of the maximum stresses in time and compare it with the results of the steady-state analysis.

Figure 14 shows the variation of the maximum σ_{eq} and σ_{zz} in time caused by the pressure pulse on the boundary. The dashed line also gives the maximum stress in steady state. The dynamic effects play a significant role in the load: maximum σ_{zz} in the transient run is about twice as high as the steady-state value. In the case of σ_{eq}, the predicted peak is marginally lower than its steady-state equivalent. The propagation of the secondary stress waves and their high-frequency reflections can also be seen.

The computational requirement for the transient calculation is considerably higher than for the steady state. Here, we effectively have to solve the problem 10 000 times, somewhat assisted by the reasonable 'initial guess': the solution from the previous time-step. Typically, it takes between 5 and 20 iterations to reach convergence for the time-step. The complete calculation with the necessary 'on the fly' post-processing took 373 430 s or 37.3 s per time-step on 16 CPUs in parallel. This computation time (slightly more than 4 days) should be seen in the light of

the huge amount of data that is being produced. However, without the use of parallel computer architectures the calculation of this type would take more than a month, definitively not practical in a real engineering environment.

7. CONCLUSIONS

This paper describes the application of the second-order FV discretization to the linear stress analysis problem. The combination of arbitrarily unstructured meshing, segregated approach and parallelism results in a fast and memory-efficient solution algorithm. The discretization stems directly from the integral form of the governing equation over the CV and is therefore simple to understand and extend to non-constant material properties or non-linear constitutive relations [7]. Furthermore, the non-linearity can be treated naturally with only a modest increase in computational cost, as the solution procedure already iterates over the system of equations.

Careful discretization allowed us to produce an efficient and easily parallelizable solution procedure even without the use of multigrid acceleration. The use of parallel computer allows us to produce accurate solutions for complex geometries on large meshes in realistic time scales for an engineering environment. With such efficiency, the use of meshes with up to 1 million CVs can become a routine. Also, if the problem in question requires it, a transient solution of good quality can be obtained in a realistic time with appropriate computer resources. Moreover, further efficiency improvements can be achieved by means of parallel multigrid [15], compatible with the procedure described here. All these points encourage further research in the finite volume stress analysis with the objective to provide fast and reliable solvers that could compete with the well-established finite element method.

REFERENCES

1. Zienkiewicz OC, Taylor RL. *The Finite Element Method*, *Solid and Fluid Mechanics. Dynamics and Non-Linearity* (4th edn). vol. 2. McGraw-Hill: New York, 1989.
2. Girault V, Raviart P-A. *Finite Element methods for Navier–Stokes equations*, Springer Series in Computational Mathematics, vol. 5. Springer: Berlin, 1986.
3. Demirdžić I, Ivanković A, Martinović D. Numerical simulation of thermal deformation in welded workpiece. *Zavarivanje* 1988; **31**:209–219 (in Croatian).
4. Demirdžić I, Muzaferija S. Finite volume method for stress analysis in complex domains. *International Journal for Numerical Methods in Engineering* 1994; **37**:3751–3766.
5. Demirdžić I, Muzaferija S, Perić M. Benchmark solutions of some structural analysis problems using finite-volume method and multigrid acceleration. *International Journal for Numerical Methods in Engineering* 1997; **40**(10): 1893–1908.
6. Demirdžić I, Muzaferija S, Perić M. Advances in computation of heat transfer, fluid flow and solid body deformation using finite volume approaches. In *Advances in Numerical Heat Transfer*, Minkowycz WJ, Sparrow EM (eds). vol. 1, Chapter 2. Taylor & Francis: London, 1997.
7. Demirdžić I, Martinović D. Finite volume method for thermo-elasto-plastic stress analysis. *Computer Methods in Applied Mechanics and Engineering* 1993; **109**:331–349.
8. Demirdžić I, Muzaferija S. Numerical method for coupled fluid flow, heat transfer and stress analysis using unstructured moving meshes with cells of arbitrary topology. *Computer Methods in Applied Mechancis and Engineering* 1995; **125**(1–4):235–255.
9. Jasak H. Error analysis and estimation in the Finite Volume method with applications to fluid flows, *Ph.D. Thesis*, Imperial College, University of London, 1996.
10. Gosman, AD. Developments in industrial computational fluid dynamics. *Chemical Engineering Research and Design* 1998; **76**(A2):153–161.
11. Zienkiewicz OC, Taylor RL. *The Finite Element Method. Basic Formulation and Linear Problems*, (4th edn). vol. 1. McGraw-Hill: New York, 1989.

12. Jacobs DAH. Preconditioned conjugate gradient methods for solving systems of algebraic equations. Central Electricity Research Laboratories, 1980.
13. Hestens MR, Steifel EL. Method of conjugate gradients for solving linear systems, *Journal of Research* 1952; **29**: 409–436.
14. Hortmann M, Perić M, Scheurer G. Finite volume multigrid prediction of laminar natural convection: bench-mark solutions. *International Journal for Numerical Methods in Fluids* 1990; **11**:189–207.
15. Schreck E, Perić M. Computation of fluid flow with a parallel multigrid solver. *International Journal for Numerical Methods in Fluids* 1993; **16**:303–327.
16. Ecer A, Hauser J, Leca P, Periaux J (eds). *Parallel Computational Fluid Dynamics. New Trends and Advances*. N-H Elsevier: Amsterdam, 1995.
17. Simon HD, Dagum L. Experience in using SIMD and MIMD parallelism for computational fluid dynamics. *Technical Report RNR-91-014*, NAS Applied Research Branch (RNR), 1991.
18. Beguelin AL, Dongarra JJ, Geist GA, Jiang WC, Manchek RJ, Moore BK, Sunderam VS. PVM version 3.3: Parallel Virtual Machine System: http://www.epm.ornl.gov/pvm/pvm_home.html, 1992.
19. Lester BP. *The Art of Parallel Programming*. Prentice-Hall: Englewood Cliffs, NJ, 1993.
20. Dowd K. *High Performance Computing*. O'Reilly & Associates: Sebastopol, 1993.
21. Weller HG, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object orientated techniques. *Computers in Physics* 1998; **12**(6):620–631.
22. Timoshenko SP, Goodier, JN. *Theory of Elasticity* (3rd edn). McGraw-Hill: London, 1970.

