COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING
Commun. Numer. Meth. Engng 2005; **21**:107–118
Published online 14 December 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cnm.731

An efficient global matrix free finite element algorithm for 3D flow problems

K. Murugesan[‡], D. C. Lo[§] and D. L. Young[*,†,¶]

Department of Civil Engineering and Hydrotech Research Institute, National Taiwan University, Taipei, Taiwan

SUMMARY

This paper describes a finite element solution algorithm for the numerical solution of large size three-dimensional flow problems on a personal computer. To demonstrate the algorithm, the Stokes equations in velocity–vorticity form are solved for a lid-driven cubical cavity problem. The Galerkin's weighted residual form of the governing equations is evaluated for all the elements of the computational domain and kept as element-matrices and element-vectors. This results in a set of simultaneous equations corresponding to the global nodes of each element. Those elements that contain the boundary nodes are modified to incorporate the Dirichlet boundary conditions. A conjugate gradient iterative scheme is employed to solve the simultaneous equations in element form to get the solution at the global nodes. The matrix–vector products used in the conjugate gradient iterative solver are performed in element level, assembling only the element-level vectors to form the global vectors. Since the element-level computation has eluded the formation of global matrices, the numerical solution of three-dimensional Stokes equations using a mesh of size as high as 51^3 could be achieved on a personal computer. The algorithm is validated by comparing the results for a three-dimensional transient diffusion problem and Stokes flow in a lid-driven cubical cavity. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: finite element method; element-level storage; velocity–vorticity formulation; three-dimensional Stokes flow

1. INTRODUCTION

The application of the finite element (FE) method for the solution of three-dimensional flow problems results in sparse and diagonally dominant global coefficient matrices. The bi-conjugate gradient iterative scheme [1] is one of the widely used methods to solve large-size

*Correspondence to: D. L. Young, Department of Civil Engineering and Hydrotech Research Institute, National Taiwan University, Taipei, Taiwan.
†E-mail: dlyoung@hy.ntu.edu.tw
‡Post-doctoral Research Fellow.
§Research Scholar.
¶Professor.

Contract/grant sponsor: National Science Council of Taiwan

Received 7 November 2003
Accepted 25 August 2004

sparse matrices by storing only the non-zero entries of the matrices in compact vector storage form [2]. While solving three-dimensional flow problems using finer meshes, even the compact storage schemes require huge memory space because the number of non-zero entries increases significantly. Hence the size of the vectors storing the non-zero entries also increases, occupying a huge RAM of a computer. Depending on the mesh size, this will be in tens of multiples of the total number of grid points. For solving such large-size problems, Irons [3] developed the frontal solver, which does not form the global matrices. In the element-based frontal solver, the process of accumulating the contributions for a given degree of freedom results in a frontal width. The frontal solver performs effectively only when this frontal width is minimized by proper global element node numbering of the domain using some special algorithms. Hughes *et al.* [4] proposed an element-by-element solution scheme based on an idea of operator-splitting. For a diffusion problem their method uses the inverse square root of the capacitance matrix. Hence it is easy to implement this method only when the capacitance matrix is diagonal, which is not always the case with flow problems. Hughes *et al.* [5] extended their concept developed in Reference [4] for structural and solid mechanics problems by converting elliptical equations into parabolic equations by using artificial time derivatives. Conjugate gradient methods are highly attractive schemes to solve a large sparse system of equations because the solution scheme depends only on vectors obtained as products of coefficient matrices and vectors [6]. When these products are performed at an element level, a significant saving in computational time and effort can be achieved. Sheu *et al.* [7] implemented the BICGSTAB iterative solver in an element-by-element format to achieve computational efficiency in parallel computation of three-dimensional Navier–Stokes equations using the FE method. Thiagarajan and Aravamuthan [8] proposed a pre-conditioner for the conjugate gradient method used along with an element-by-element solution scheme for parallel computation. Phoon [9] developed a generalized Jacobi (diagonal) preconditioning approach to implement the conjugate gradient iterative solver using an element-by-element strategy.

The implementation of the element-by-element iterative solution procedure to solve large-scale problems on a personal computer is not straight forward though the scheme has been efficiently exploited in parallel computations [7–9]. The main reason for this restriction is the requirement of huge computer memory to store the large-size global matrices. Even a compact vector storage scheme requires a memory space of 1 336 694 to store only the non-zero entries for a 3D flow problem with a mesh of size 31^3. Hence the necessity of storing such huge-size vectors restricts the use of the conjugate gradient iterative solvers such as the BICG iterative solvers [1] on personal computers. In this context, the present work proposes an alternate scheme to store and solve large-size problems involving a few hundred thousands grid points on a personal computer. The proposed algorithm makes use of the concept of element-wise approximation of the governing equations by the FEM and vector computations associated with the conjugate gradient iterative method. The present method can be used to solve both the steady state and the transient field problems without using the pseudo-time derivative for steady state problems as proposed by Hughes *et al.* [4]. The algorithm is verified by solving three-dimensional problems, one on transient heat diffusion and the other on steady state Stokes flow. The efficiency of the algorithm to solve a large number of equations on a personal computer has been demonstrated by solving the Stokes flow in a lid-driven cubical cavity using a mesh of size as high as 51^3. The proposed algorithm is explained by its application to the numerical solution of three-dimensional Stokes flow equations.

2. GOVERNING EQUATIONS AND FE SOLUTION

The governing equations for a three-dimensional steady state Stokes flow can be represented in velocity–vorticity form without involving the pressure term as [10]:

Vorticity transport equation:

$$\nabla^2\boldsymbol{\omega} = 0 \tag{1}$$

Velocity Poisson equation:

$$\nabla^2\mathbf{V} = -\nabla\times\boldsymbol{\omega} \tag{2}$$

where the vorticity is defined as

$$\boldsymbol{\omega} = \nabla\times\mathbf{V} \tag{3}$$

No-slip velocity boundary conditions are assumed on all the boundary surfaces except the top-moving lid where unit velocity is assumed. A second-order accurate Taylor's series scheme proposed by Wong and Baker [11] has been employed to compute the boundary vorticity values using the vorticity definition given as

$$\boldsymbol{\omega}_\mathrm{b} = \nabla\times\mathbf{V}_\mathrm{b} \tag{4}$$

The application of the Galerkin's weighted residual method to the governing equations (1) and (2) results in the following integral equations:

$$\int_\Omega N^\mathrm{T}(\nabla^2\boldsymbol{\omega})\,\mathrm{d}\Omega = 0 \tag{5}$$

$$\int_\Omega N^\mathrm{T}(\nabla^2\mathbf{V} + \nabla\times\boldsymbol{\omega})\,\mathrm{d}\Omega = 0 \tag{6}$$

The above integrals are evaluated for all the elements and the various coefficient matrices and the vectors are stored as element-matrices and element-vectors without forming the global matrices. The element-wise vorticity transport equations in matrix form can be written as

$$[K_{ijk}]\{\xi_{ik}\} = 0 \tag{7a}$$

$$[K_{ijk}]\{\eta_{ik}\} = 0 \tag{7b}$$

$$[K_{ijk}]\{\varsigma_{ik}\} = 0 \tag{7c}$$

Similarly, the element-wise velocity Poisson equations in matrix form can be represented as

$$[K_{ijk}]\{u_{ik}\} = [Z_{ijk}]\{\eta_{ik}\} - [Y_{ijk}]\{\varsigma_{ik}\} \tag{8a}$$

$$[K_{ijk}]\{v_{ik}\} = [X_{ijk}]\{\varsigma_{ik}\} - [Z_{ijk}]\{\xi_{ik}\} \tag{8b}$$

$$[K_{ijk}]\{w_{ik}\} = [Y_{ijk}]\{\xi_{ik}\} - [X_{ijk}]\{\eta_{ik}\} \tag{8c}$$

where i, j and k represent the element, row and column indices, respectively. The computational domain is discretized using eight-node tri-linear elements through isoparametric formulation and the integration is performed by the Gaussian quadrature. A Cartesian co-ordinate system with x–y co-ordinates on the horizontal plane and z co-ordinate in the vertical direction is assumed.

2.1. Storage of global matrices

In the classical global matrix algorithm, the coefficient matrices and the load vectors of Equations (7) and (8) are evaluated for all the elements and they are assembled to obtain the global form of simultaneous equations given as

$$[A_{ij}]\{\phi_j\} = \{f_i\} \tag{9}$$

where i and j represent the row and column indices. The global coefficient matrix $[A_{ij}]$ is sparse and banded. In order to save computer memory, only the non-zero entries of the coefficient matrix are stored either in a banded matrix form or using a compressed row/column format. This procedure becomes very significant in the context of numerical solution of three-dimensional flow problems. As far as the equation solver is concerned, the conjugate gradient iterative solvers are more efficient to solve sparse matrices compared to other solvers. The solution algorithm of a conjugate gradient method depends only on vectors obtained by the products of $N \times N$ matrices with $N \times 1$ vectors. In the BICG iterative solver [1] the non-zero entries of the coefficient matrices are stored in a column format. The size of a vector required to store the non-zero entries will be in tens of multiples of the total number of grid points in the computational domain. For example when a mesh of size 51^3 is used in a three-dimensional Stokes flow problem, the size of a vector required to store the locations and the non-zero entries of a coefficient matrix is 6 407 854. Storage of vectors of this size in various subroutines in the BICG solver occupies a huge RAM of a computer. Hence solving three-dimensional flow problems using finer meshes demands either use of powerful computers like Cray computers or parallel computation. The proposed algorithm in this article aims at providing an alternate method to store the non-zero entries so that large-size problems can be handled on a personal computer.

2.2. Global matrix free FE scheme

When all the coefficient matrices and the vectors are kept in an element-wise storage scheme, the formation of sparse global matrices and the associated compact storage procedures can be avoided. However, this demands the inclusion of the Dirichlet boundary values at the element level itself. This can be achieved by determining the number of elements contributing the Dirichlet value to a boundary node. In a three-dimensional computational domain the boundary nodes are located on corners, edges and surfaces. A boundary node at a corner is shared by only one element, a boundary node along an edge is shared by two elements and a boundary node on a surface is shared by four elements. In Figure 1, the shaded region represents a section of a boundary surface of a three-dimensional computational domain. The corner nodes 1, 3, 9 and 7 share only one element each, that is, elements 1, 2, 4 and 3, respectively. If we consider the boundary node 8 along an edge, it shares with two elements, 3 and 4. Similarly the edge node 2 shares with elements 1 and 2. For the case of a surface boundary node 5, there are four elements 1, 2, 3 and 4 common to this node and hence the node 5 is shared by all these four elements. Thus for each boundary node, the data of (i) the number of contributing elements, (ii) the element numbers and (iii) the respective rows of the elements that contain the boundary node are generated using a subroutine. Using these data the fraction of the Dirichlet boundary value contribution from each element surrounding a boundary node is computed. Then the coefficient matrices and the load vectors of those elements containing the boundary nodes are modified for the inclusion of the Dirichlet boundary values. The

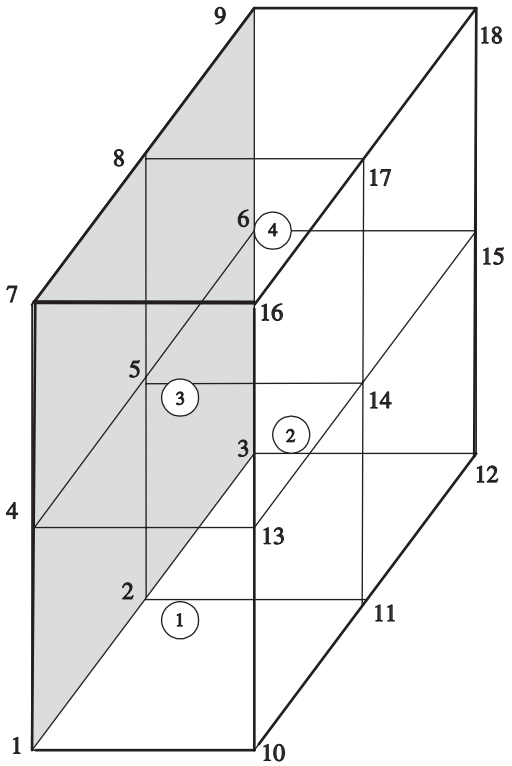

Figure 1. Schematic diagram of boundary nodes and contributing elements.

modified matrices should produce the same effect as that of the global matrices obtained as a result of assembly of all the elements. The computational effort for this task is insignificant because the number of boundary nodes is very small compared to the total number of grid points in the computational domain. Moreover, similar to the grid points' data this information is generated only once.

The procedure can be explained by considering the boundary node 8 located along the edge (Figure 1). This node is surrounded by two elements, 3 and 4. The coefficient matrices and the load vectors for the elements 3 and 4 can be represented as

$$[A_{3jk}]\{\phi_{3k}\} = \{f_{3j}\} \tag{10a}$$

$$[A_{4jk}]\{\phi_{4k}\} = \{f_{4j}\} \tag{10b}$$

where j and k refer to the row and column indices that vary from 1 to 8. The corresponding global nodal connectivity for the elements 3 and 4 is given as:

Element 3: 4, 13, 14, 5, 7, 16, 17, 8
Element 4: 5, 14, 15, 6, 8, 17, 18, 9

The boundary node 8 appears in both the elements 3 and 4 but with different row numbers. It appears in the eighth row in the element 3 whereas it appears in the fifth row in the element 4.

As discussed by Reddy [12] the coefficient matrices and the right-hand side column vectors of Equations 10(a) and 10(b) are modified for the known Dirichlet value at the boundary node 8. Since the boundary node 8 is surrounded by two elements, the Dirichlet value used to modify the coefficient matrices and the right-hand column vectors of the elements 3 and 4 will be $\frac{1}{2}$ times the Dirichlet value at the node 8. In general, if ϕ_{bn} is the Dirichlet boundary value at a boundary node 'bn', then each element surrounding this node gets a contribution equal to $(1/\mathrm{nesbn})\phi_{\mathrm{bn}}$ where 'nesbn' is the total number of elements surrounding the given boundary node. Following a similar procedure all the element-level coefficient matrices and the right-hand column vectors are modified with the known Dirichlet values. Finally, this will result in an element-wise system of simultaneous equations given as

$$[A_{ijk}]\{\phi_{ik}\} = \{f_{ij}\} \tag{11}$$

where i, j and k represent the element, row and column indices.

Since the conjugate gradient iterative procedure is associated with only the vectors obtained by the multiples of matrices and vectors, all the computations are carried out at the element level and only vector-level assembly is carried out to form the global vectors. This ensures the maximum size of a vector computed in the present algorithm just equal to the total number of grid points irrespective of the global node numbering of the computational domain. In this aspect the present approach is different from the frontal solver [3] whose efficiency depends upon the global node numbering of the elements in order to keep the front width minimum. In the proposed algorithm a simple conjugate gradient method without preconditioning is used. The simplicity in implementing the present scheme on a personal computer is demonstrated with the following three-dimensional test problems.

2.3. Test problem: 1—3D heat diffusion in a cube

Initially the global matrix free FE algorithm is tested for heat diffusion in a cube studied by Zienkiewicz and Parekh [13]. Cooling of a cube initially at a unit temperature is investigated by four-element subdivisions using linear elements. The cube is subjected to the following boundary conditions:

$T = 1$ on the regions $-a < x < a$, $-a < y < a$ and $-a < z < a$ and
$T = 0$ on the boundary

The temperature variation with time at $x = y = z = 0$ is examined by solving the transient heat conduction equation [13]. Eight-node tri-linear elements are used to discretize the computational domain through an isoparametric formulation whereas the time domain is discretized using a second-order accurate Crank–Nicolson scheme. Table I shows the comparison of the present results for linear elements with the corresponding results of Zienkiewicz and Parekh [13] for $D\Delta t/a^2 = 0.0125$, where D is the diffusion coefficient. The present results are in close agreement with the results of Zienkiewicz and Parekh [13].

2.4. Test problem: 2—Stokes flow in a lid-driven cubical cavity

In order to test the present algorithm for flow problems, we consider the steady state Stokes flow in a lid-driven cubical cavity. The top lid of the cube is assumed to move with a unit velocity in the horizontal x direction. Initially a mesh-dependence study is carried out to establish a benchmark solution. For comparison purpose we use the numerical results obtained

Table I. Comparison of results for 3D heat diffusion problem with $D\Delta t/a^2 = 0.0125$ [13].

	Element length $= a$		Element length $= a/2$	
$\frac{Dt}{a^2}$	Present	Reference [13]	Present	Reference [13]
0.0	1.0000	1.0000	1.0000	1.0000
0.1	0.4066	0.4062	0.7847	0.7856
0.2	0.1653	0.1650	0.3698	0.3694
0.3	0.0672	0.0670	0.1699	0.1696
0.4	0.0273	0.0272	0.0780	0.0778
0.5	0.0111	0.0111	0.0358	0.0357
0.6	0.0045	0.0045	0.0164	0.0164
0.7	0.0018	0.0018	0.0075	0.0075
0.8	0.0007	0.0007	0.0035	0.0034
0.9	0.0003	0.0003	0.0016	0.0016
1.0	0.0001	0.0001	0.0007	0.0007

Table II. Results of DQ method.

Mesh	$u_{\min}$	$w_{\min}$	$w_{\max}$
7^3	−0.22517	−0.17593	0.17593
11^3	−0.22084	−0.17972	0.17972
13^3	−0.22086	−0.18012	0.18012

Table III. Results for mesh sensitivity study.

Mesh	$u_{\min}$	$w_{\min}$	$w_{\max}$
11^3	−0.21316	−0.17071	0.17071
21^3	−0.21884	−0.18036	0.18036
31^3	−0.21958	−0.18026	0.18026
51^3	−0.22235	−0.18063	0.18063

by the differential quadrature (DQ) method [14], which approximates the derivatives using higher order polynomials. The minimum value of u velocity along the central vertical plane and the minimum and maximum value of w velocity along the central horizontal plane are considered as the parameters of comparison for the mesh sensitivity study. In order to establish the numerical accuracy of the DQ method, the Stokes flow results are obtained using three different meshes of size 7^3, 11^3 and 13^3 and are shown in Table II. Since the flow results computed using the meshes 11^3 and 13^3 are closer to each other as observed in the Table II, the results obtained with the mesh 11^3 are used for the comparison purpose. Then the Stokes flow results are obtained by the global matrix free FE algorithm using non-uniform meshes of size 11^3, 21^3 and 31^3. In order to demonstrate the ability of the present scheme to handle large-size problems, computations also have been carried out using a mesh of size 51^3 for the solution of the Stokes flow equations. Table III shows the comparison of the flow

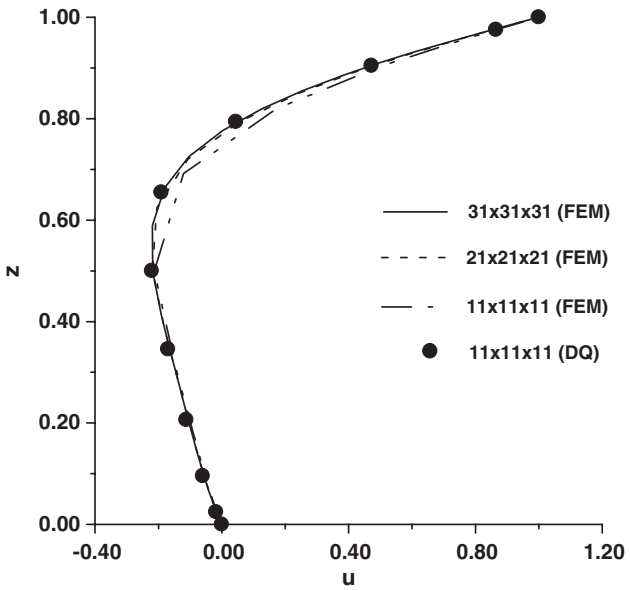

Figure 2. Comparison of u–z plot for different meshes and the DQ method.

parameters obtained by the present algorithm for the mesh sensitivity study. Flow results shown in Table III indicate that the increase in the numerical accuracy of the present algorithm is consistent with the refinement of the mesh. Besides, the results obtained by the present algorithm using the 21^3mesh are closer to the results computed by the DQ method using the 11^3 mesh.

The u-velocity profiles along the vertical centre line are shown in Figure 2 for the 11^3, 21^3 and 31^3 meshes considered in the global matrix free FE algorithm. For the purpose of comparison, the results obtained by the DQ method using the 11^3 mesh are also shown as symbols in Figure 2. A similar plot is shown in Figure 3 for the variation of w velocity along the horizontal centre line. The results depicted in Figures 2 and 3 highlight the increase in the numerical accuracy of the present algorithm with the refinement of the mesh. Besides, an excellent agreement of the present results with the results of the DQ method is clearly seen in the above figures. The velocity vectors and the y direction vorticity distributions on the x–z plane at $y = 0.5$ obtained using the 31^3 mesh are depicted in Figures 4 and 5, respectively. Both the figures exhibit the expected flow behaviour in the cavity.

The computations for the 3D Stokes flow have been carried out on a Pentium-IV personal computer. In order to demonstrate the efficiency of the present algorithm to handle large-size problems on a personal computer, the details about the size of the memory storage used in the present algorithm and the memory space required to store only the non-zero entries of the global coefficient matrix are computed for the meshes 11^3, 21^3, 31^3 and 51^3 and are shown in Table IV. The present algorithm stores the coefficient matrices at the element level as indicated in the fifth row of Table IV. A comparison of the entries in the fourth and the fifth rows indicates that the storage used in the present scheme is slightly higher than the storage used for storing only the non-zero entries in a vector format. But the additional

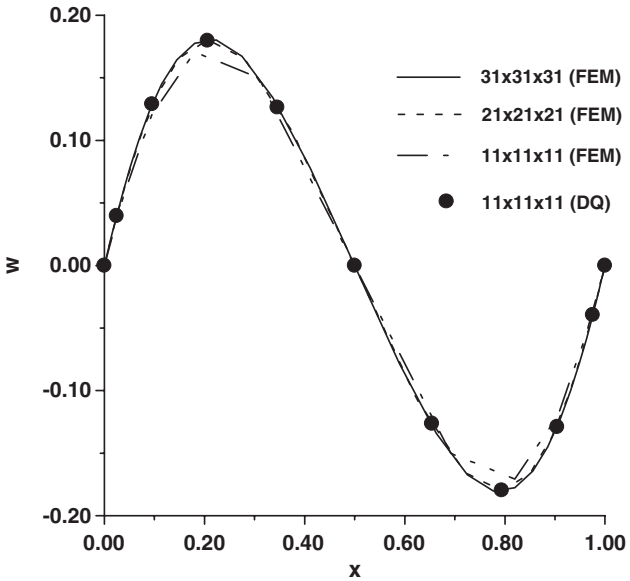

Figure 3. Comparison of x–w plot for different meshes and the DQ method.

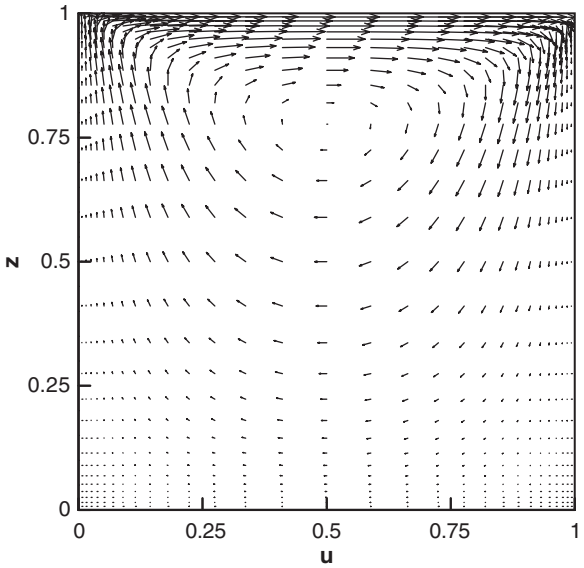

Figure 4. Velocity vectors on $y = 0.5$ plane.

storage reduces significantly as the mesh is refined. This can be understood by computing the ratio between the entries stored by the present algorithm and the storage used for the non-zero entries. As can be seen from Table IV, this ratio decreases as the mesh is made finer. But in the computer code for the BICG method [1], vectors equal to twice the total

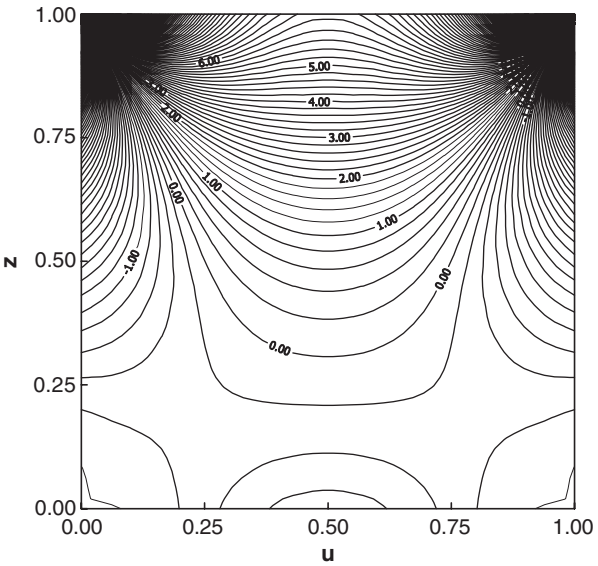

Figure 5. η vorticity distribution on $y = 0.5$ plane.

Table IV. Comparison of memory storage for different schemes.

Mesh	11^3	21^3	31^3	51^3
Number of elements	1000	8000	27 000	125 000
Number of nodes	1331	9261	29 791	132 651
Storage for non-zero entries and their locations in column format	41 534	379 114	1 336 694	6 407 854
Memory storage used in the present method	64 000 (1000,8,8)	512 000 (8000,8,8)	1 728 000 (27 000,8,8)	8 000 000 (125 000,8,8)
Present storage/storage used in column format (row 5/row 4)	1.541	1.351	1.293	1.248
Non-zero entries/number of nodes (row 4/row 3)	31.21	40.94	44.87	48.31

number of non-zero entries have to be used in many subroutines. These vectors will occupy huge computer memory for large-size problems, thus prohibiting the use of such solvers on personal computers.

The values in Table IV also indicate that the total number of non-zero entries increases as the mesh is refined. The rate of increase of the non-zero entries with mesh refinement can be determined by computing the ratio between the non-zero entries and the total number of grid points. As expected this ratio increases as the mesh is refined to a finer mesh. That means the use of a finer mesh increases the vector size used to store the non-zero entries in the BICG iterative scheme [1]. But in the present iterative solution procedure, the size of a vector never

exceeds the total number of grid points and is independent of the size of the mesh used in the computations. This characteristic of the present algorithm has made possible to execute the computer code for a mesh 51^3 on a personal computer. Thus the present scheme is proved to be highly efficient for storing and solving large number of equations obtained as a result of either multi-dimensional problems or mesh refinement. As a first attempt in implementing the global matrix free FE algorithm, a conjugate gradient iterative solver without preconditioning has been employed in the present study. Even with this basic iterative solver, the convergence for the iterations could be achieved in a number of steps little less than the total number of equations. However, the rate of convergence can be increased further by incorporating a preconditioning technique, which is under study.

3. CONCLUSIONS

A global matrix free finite element solution algorithm is discussed to solve large-scale three-dimensional flow problems on a personal computer. By computing the elements surrounding each boundary node, the corresponding fractional value of the Dirichlet boundary conditions are enforced on the boundary nodes at the element-level itself. This has avoided the formation of global matrices, thus resulting in a significant saving in the computer memory. The element-level equations are solved using a conjugate gradient iterative solver which uses the matrix products in vector form. Test results obtained for a three-dimensional transient heat diffusion problem and a steady state Stokes flow in a lid-driven cubical cavity show good agreements with the results obtained by other numerical schemes. Though the present algorithm uses memory storage little higher than that of the compact column storage scheme for a given coefficient matrix, the additional storage decreases significantly when the mesh is refined to a finer mesh. In addition to this, the maximum size of a vector computed in the present algorithm never exceeded that of the total number of grid points in the computational domain irrespective of the mesh size used. This feature of the present algorithm has made possible to obtain the solution of Stokes flow equations using a mesh of size as high as 51^3 on a personal computer.

ACKNOWLEDGEMENTS

The National Science Council of Taiwan is gratefully acknowledged for providing the financial support to carry out the present research.

REFERENCES

1. Press WH, Teukolshy SA, Vetterling WT, Flannery BP. *Numerical Recipes in FORTRAN 90*, (2nd edn). Cambridge University Press: New York, 1996.
2. Saad Y. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company: Boston, 1996.
3. Irons B. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering* 1970; **2**:5–32.
4. Hughes TJR, Levit I, Winget J. Element-by-element implicit algorithms for heat conduction. *Journal of Engineering Mechanics* 1983; **109**:576–585.
5. Hughes TJR, Levit I, Winget J. An element-by-element solution algorithm for problems of structural and solid mechanics. *Computer Methods in Applied Mechanics and Engineering* 1983; **36**:241–254.
6. Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 1952; **49**:409–436.

7. Sheu TWH, Wang MMT, Tsai SF. Element-by-element parallel computation of incompressible Navier–Stokes equations in three dimensions. *SIAM Journal of Scientific Computing* 2000; **21**:1387–1400.
8. Thiagarajan G, Aravamuthan V. Parallelization strategies for element-by-element preconditioned conjugate gradient solver using high-performance Fortran for unstructured finite-element applications on Linux clusters. *Journal of Computing in Civil Engineering* 2002; **16**:1–10.
9. Phoon KK. Iterative solution of large-scale consolidation and constrained finite element equations for 3D problems. *International e-Conference on Modern trends in Foundation Engineering*: *Geological Challenges and Solutions*, Indian Institute of Technology, Madras, India, 26–30 January, 2004.
10. Tsai CC. Meshless numerical methods and their engineering applications. *Ph.D. Thesis*, Department of Civil Engineering, National Taiwan University, Taipei, Taiwan, 2002.
11. Wong KL, Baker AJ. A 3D incompressible Navier–Stokes velocity–vorticity weak form of finite element algorithm. *International Journal for Numerical Methods in Fluids* 2002; **38**:99–123.
12. Reddy JN. *An introduction to the Finite Element Method*, (2nd edn). Mc-Graw Hill: Singapore, 1992.
13. Zienkiewicz OC, Parekh CJ. Transient filed problems: two-dimensional and three-dimensional analysis by isoparametric finite elements. *International Journal for Numerical Methods in Engineering* 1970; **2**:61–71.
14. Shu C, Wang L, Chew YT. Numerical computation of three-dimensional incompressible Navier–Stokes equations in primitive variable form by DQ method. *International Journal for Numerical Methods in Fluids* 2003; **43**: 345–368.

