ELSEVIER

Comput. Methods Appl. Mech. Engrg. 161 (1998) 215–228

Computer methods in applied mechanics and engineering

A dynamic adaption algorithm for grid, time and satellite processors for nodal finite element formulations of Navier–Stokes equations

S.Ø. Wille*, D. Skipitaris

Faculty of Engineering, Oslo College, Cort Adelersgate 30, N-0254 Oslo, Norway

Received 3 September 1997; accepted 10 September 1997

Abstract

Two parallel nodal finite element algorithms for solving the Navier–Stokes equations are developed. The algorithms are based on operator splitting of the original Navier–Stokes equations. Linear basis functions are applied in the finite element formulation of the equation system. The finite element grid is generated by the Tri–Tree grid generation algorithm. The grid is adapted to the solution according to the element Reynolds number. The length of each time step is computed from the Courant number. The linearized equation system is solved iteratively by conjugate gradient algorithms. The most time-consuming part of the conjugate gradient algorithm, the generation of the right-hand side of the equation system and the matrix–vector multiplication, is distributed to satellite processors. The number of processors which can be used by the parallel algorithm is only limited by the number of finite elements or the number of processors available. The parallel algorithm can either select a fixed number of satellite processors or the number of satellite processors can be adapted to the amount of work performed during the computations.

1. Introduction

The algorithm developed is based on the Tri–Tree adapted grid generator [1,2]. This grid generator uses triangles in two dimensions and tetrahedra in three dimensions as tree leaves. The Tri–Tree grid generator has the same properties as the oct tree [3,4], with the exception that the oct tree uses rectangles in two dimensions and rectangular boxes in three dimensions as basic elements.

The two numerical solution algorithms use operator splitting. The first algorithm splits the Navier–Stokes equations in three equations, the velocity equation, the pressure equation and the mass equation [5,6]. The second operator splitting algorithm splits the Navier–Stokes equations in four equations, the diffusion equation, the convection equation, the pressure equation and the mass equation [7]. Both these algorithms are nodal algorithms which require no storage of the equation matrices. The velocity equation, the diffusion equation and the pressure equation are solved with diagonal preconditioned conjugate gradient algorithms. The convection equation and the mass equation are solved by lumping the mass matrix and the inversion of the diagonal lumped matrix.

In the finite element algorithm developed, the generation of the right-hand sides in the equation system and the matrix–vector multiplication in the conjugate gradient algorithms, reveal good parallel properties. The most time-consuming part of the algorithm, the conjugate gradient solver, is parallelized. The right-hand side of the equation system and matrix–vector multiplication are performed by distributing a number of finite elements to satellite processors which compute the corresponding partial right-hand side and matrix–vector multiplication.

* Corresponding author.

0045-7825/98/$19.00
PII: S0045-7825(97)00316-2

The results of the right-hand sides and multiplications from each satellite processor are then assembled in the main computer.

The idea behind parallelizing is to have N processors to cooperate on a task. Ideally, if it takes one processor T timesteps to do a job, then N processors can do the same job in T/N timesteps. Another motivation by parallelizing is that it may be possible to run larger problems by distributing the problem on several processors rather than to run the problem on one single processor.

Up to some level the speed will increase as more processors are added. However, the cost of administrating multiple processors and the bandwidth of the network will limit the speed. At some point the cost of parallelizing will be higher than the gain, and the speed will be reduced.

The efficiency of parallelization, however, mainly depends on the parallel properties of the numerical algorithm. A very efficient parallelization algorithm have been developed by Stagg et al. [8]. These authors use multiple instruction, multiple data (MIMD) computers and obtain a fixed solution time as the computational work is scaled with the number of processors in use.

The main computer and the satellite processors may be ordinary pentium processors connected with an ordinary ethernet. The parallel computing environment is therefore standard in most research facilities and the limitations of the parallel processing system are the number of processors available and the communication speed of the ethernet.

2. The Navier–Stokes equations

The nonlinear Navier–Stokes equations are given by

$$\begin{aligned} &\rho \frac{\partial \boldsymbol{v}}{\partial t} - \mu \nabla^2 \boldsymbol{v} + \rho \boldsymbol{v} \cdot \nabla p = 0 \quad \text{in } \Omega \\ &-\nabla \cdot \boldsymbol{v} = 0 \qquad\qquad\qquad\quad \text{in } \Omega \end{aligned} \tag{1}$$

where $\boldsymbol{v}$ is the velocity vector, p is the pressure, μ is the viscosity coefficient and ρ is the density. The first equation is the equation of motion which contains time, diffusion, convection and pressure terms. The second equation is the equation of continuity.

2.1. The pressure split algorithm

The original Navier–Stokes equation can be reformulated into three equations which describe the fluid flow. In the reformulated version below, there is one excessive equation, which simplify the numerical solution algorithm,

$$\begin{aligned} &\rho \frac{\partial \boldsymbol{v}}{\partial t} - \mu \nabla^2 \boldsymbol{v} + \rho \boldsymbol{v} \cdot \nabla \boldsymbol{v} + \nabla p = 0 \quad \text{in } \Omega \\ &\nabla^2 p - \rho \nabla \cdot \frac{\partial \boldsymbol{v}}{\partial t} = 0 \qquad\qquad\quad \text{in } \Omega \\ &\rho \frac{\partial \boldsymbol{v}}{\partial t} + \nabla p = 0 \qquad\qquad\qquad\quad \text{in } \Omega \end{aligned} \tag{2}$$

The finite element formulation of the pressure split equations become

$$\begin{aligned} \boldsymbol{F}_{\mathrm{v}} &= \int_{\Omega} \left[\rho L_i \frac{\partial \boldsymbol{v}}{\partial t} + \mu \, \nabla L_i \cdot \nabla \boldsymbol{v} + \rho L_i \boldsymbol{v} \cdot \nabla \boldsymbol{v} - \nabla L_i p \right] \mathrm{d}\Omega - \int_{\delta\Omega} \mu L_i \frac{\partial \boldsymbol{v}}{\partial n} \, \mathrm{d}\delta\Omega = 0 \\ \boldsymbol{F}_{\mathrm{p}} &= \int_{\Omega} \left[\nabla L_i \nabla p + \rho L_i \nabla \cdot \frac{\partial \boldsymbol{v}}{\partial t} \right] \mathrm{d}\Omega - \int_{\delta\Omega} L_i \frac{\partial p}{\partial n} \, \mathrm{d}\delta\Omega = 0 \\ \boldsymbol{F}_{\mathrm{m}} &= \int_{\Omega} \left[\rho L_i \frac{\partial \boldsymbol{v}}{\partial t} + \nabla p \right] \mathrm{d}\Omega = 0 \end{aligned} \tag{3}$$

The three equations, $\boldsymbol{F}_{\mathrm{v}}$, $\boldsymbol{F}_{\mathrm{p}}$ and $\boldsymbol{F}_{\mathrm{m}}$ are then solved sequentially by the nonlinear Newton algorithm,

$$\frac{\partial \boldsymbol{F}_{\mathrm{v}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{v}^{n+1} = -\boldsymbol{F}_{\mathrm{v}}^{n} \qquad \frac{\partial \boldsymbol{F}_{\mathrm{p}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{p}^{n+1} = -\boldsymbol{F}_{\mathrm{p}}^{n} \qquad \frac{\partial \boldsymbol{F}_{\mathrm{m}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{v}^{n+1} = -\boldsymbol{F}_{\mathrm{m}}^{n} \tag{4}$$

By replacing the pressure by $d^n = p^n - p^{n-1}$ [9], divergence free flow is obtained. The Newton formulation of the pressure split equations then becomes

$$\begin{aligned}
&\int_{\Omega} \left[\rho L_i \frac{L_j}{\Delta t} + \mu \nabla L_i \nabla L_j + \rho L_i (\nabla \boldsymbol{v}^n L_j + \boldsymbol{v}^n \nabla L_j) \right] \mathrm{d}\boldsymbol{v}\, \Delta \boldsymbol{v}^{n+1} \\
&\quad = -\int_{\Omega} [\mu \nabla L_i \cdot \nabla \boldsymbol{v}^n + \rho L_i \boldsymbol{v}^n \cdot \nabla \boldsymbol{v}^n + \nabla L_i p^n] \, \mathrm{d}\Omega \\
&\int_{\Omega} \nabla L_i \nabla L_j \, \mathrm{d}\Omega \, \Delta p^{n+1} = -\int_{\Omega} \left[\rho L_i \frac{\nabla \cdot \boldsymbol{v}^n}{\Delta t} + \nabla L_i \nabla d^n \right] \mathrm{d}\Omega \\
&\int_{\Omega} \rho L_i \frac{Lj}{\Delta t} \, \mathrm{d}\Omega \, \Delta \boldsymbol{v}^{n+1} = -\int_{\Omega} \nabla d^n \, \mathrm{d}\Omega
\end{aligned} \tag{5}$$

The above equations are solved sequentially and only one Newton iteration is applied to the nonlinear velocity equation $\boldsymbol{F}_{\mathrm{v}}$.

2.2. The velocity–pressure split algorithm

The original Navier–Stokes equation is reformulated into four equations,

$$\begin{aligned}
&\rho \frac{\partial \boldsymbol{v}}{\partial t} - \mu \nabla^2 \boldsymbol{v} + \nabla p = 0 \quad \text{in } \Omega \\
&\rho \frac{\partial \boldsymbol{v}}{\partial t} - \rho \boldsymbol{v} \cdot \nabla \boldsymbol{v} = 0 \qquad \text{in } \Omega \\
&\partial \nabla^2 p - \rho \nabla \cdot \frac{\partial \boldsymbol{v}}{\partial t} = 0 \qquad \text{in } \Omega \\
&\rho \frac{\partial \boldsymbol{v}}{\partial t} + \nabla p = 0 \qquad \text{in } \Omega
\end{aligned} \tag{6}$$

The finite element formulation of the velocity–pressure split equations becomes

$$\begin{aligned}
\boldsymbol{F}_{\mathrm{d}} &= \int_{\Omega} \left[\rho L_i \frac{\partial \boldsymbol{v}}{\partial t} + \mu \nabla L_i \cdot \nabla \boldsymbol{v} - \nabla L_i p \right] \mathrm{d}\Omega - \int_{\partial \Omega} \mu L_i \frac{\partial \boldsymbol{v}}{\partial n} \, \mathrm{d}\delta\Omega = 0 \\
\boldsymbol{F}_{\mathrm{c}} &= \int_{\Omega} \left[\rho L_i \frac{\partial \boldsymbol{v}}{\partial t} + \rho L_i \boldsymbol{v} \cdot \nabla \boldsymbol{v} \right] \mathrm{d}\Omega = 0 \\
\boldsymbol{F}_{\mathrm{p}} &= \int_{\Omega} \left[\rho L_i \nabla \cdot \frac{\partial \boldsymbol{v}}{\Delta t} + \nabla L_i \nabla p \right] \mathrm{d}\Omega - \int_{\delta \Omega} L_i \frac{\partial p}{\partial n} \, \mathrm{d}\delta\Omega = 0 \\
\boldsymbol{F}_{\mathrm{m}} &= \int_{\Omega} \left[\rho L_i \frac{\partial \boldsymbol{v}}{\partial t} + \nabla p \right] \mathrm{d}\Omega = 0
\end{aligned} \tag{7}$$

The four equations, $\boldsymbol{F}_{\mathrm{d}}$, $\boldsymbol{F}_{\mathrm{c}}$, $\boldsymbol{F}_{\mathrm{p}}$ and $\boldsymbol{F}_{\mathrm{m}}$, which are linear, are solved by the nonlinear Newton algorithm,

$$\begin{aligned}
&\frac{\partial \boldsymbol{F}_{\mathrm{d}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{v}^{n+1} = -\boldsymbol{F}_{\mathrm{d}}^{n} \qquad \frac{\partial \boldsymbol{F}_{\mathrm{c}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{v}^{n+1} = -\boldsymbol{F}_{\mathrm{c}}^{n} \\
&\frac{\partial \boldsymbol{F}_{\mathrm{p}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{p}^{n+1} = -\boldsymbol{F}_{\mathrm{p}}^{n} \qquad \frac{\partial \boldsymbol{F}_{\mathrm{m}}^{n}}{\partial \boldsymbol{v}} \Delta \boldsymbol{v}^{n+1} = -\boldsymbol{F}_{\mathrm{m}}^{n}
\end{aligned} \tag{8}$$

Divergence free flow is achieved by replacing the pressure by $d^n = p^n - p^{n-1}$ [9]. The Newton formulation of the pressure split equations is

$$\int_\Omega \left[\rho L_i \frac{L_j}{\Delta t} + \mu \nabla L_i \nabla L_j\right] \mathrm{d}\Omega\, \Delta \boldsymbol{v} = -\int_\Omega \left[\mu \nabla L_i \cdot \nabla \boldsymbol{v}^n + \nabla L_i p^n\right] \mathrm{d}\Omega$$

$$\int_\Omega \rho L_i \frac{L_j}{\Delta t} \mathrm{d}\Omega\, \Delta \boldsymbol{v} = -\int_\Omega \rho L_i \boldsymbol{v}^n \cdot \nabla \boldsymbol{v}^n \, \mathrm{d}\Omega$$

$$\int_\Omega \nabla L_i \, \nabla L_j \, \mathrm{d}\Omega\, \Delta p = -\int_\Omega \left[\rho L_i \frac{\nabla \cdot \boldsymbol{v}^n}{\Delta t} + \nabla L_i \nabla d^n\right] \mathrm{d}\Omega \tag{9}$$

$$\int_\Omega \rho L_i \frac{Lj}{\Delta t} \mathrm{d}\Omega\, \Delta \boldsymbol{v} = -\int_\Omega \nabla d^n \, \mathrm{d}\Omega$$

Let n_d be the spatial dimension, then the exact integrals above can easily be computed both in two and three dimensions by the formula [2,10]

$$\int_\Omega L_i^\alpha L_j^\beta L_k^\gamma = \frac{\alpha!\beta!\gamma!}{(\alpha+\beta+\gamma+n_d)!}\, n_d! \Omega \tag{10}$$

The boundary conditions are either the velocity, $\boldsymbol{v}$, specified or $\partial \boldsymbol{v}/\partial n = 0$. The boundary integrals $\int_{\delta\Omega} \mu L_i (\partial \boldsymbol{v}^n/\partial n)\, \mathrm{d}\delta\Omega$ in (7) at external boundaries are then zero. Since the pressure in Eq. (7) is replaced by $d^n = p^n - p^{n-1}$, the normal derivative $\partial d/\partial n = 0$ and the boundary integral $\int_{\delta\Omega} L_i (\partial d^n/\partial n)\, \mathrm{d}\delta\Omega$ is zero.

The advantage of using the nonlinear Newton formulation for solving linear equations is that the boundary conditions for the correction introduced in the equation system are always zero, while the actual boundary value is inserted in the initial solution vector.

The pressure computations are based on the continuity equation for each element. Therefore, the final pressure has to be calculated from the Poisson equation with appropriate boundary conditions. The Poisson equation is derived from the differentiation of the Navier–Stokes equations and by substitution of the continuity equation.

3. Numerical solution algorithms

In the pressure split algorithm (3), the Navier–Stokes equations are split in three equations, the velocity equation $\boldsymbol{F}_{\mathrm{v}}$, the pressure equation $\boldsymbol{F}_{\mathrm{p}}$ and the mass equation $\boldsymbol{F}_{\mathrm{m}}$. The velocity equation is solved by the diagonal preconditioned nonsymmetric CGSTAB conjugate gradient algorithm. Then, the pressure is found from the pressure equation $\boldsymbol{F}_{\mathrm{p}}$ by the diagonal preconditioned symmetric conjugate gradient algorithm. The pressure correction is performed through the mass equation $\boldsymbol{F}_{\mathrm{m}}$. The mass matrix in $\boldsymbol{F}_{\mathrm{m}}$ is lumped and the equation is solved by the inversion of the lumped diagonal mass matrix.

In the velocity–pressure split algorithm (7), the Navier–Stokes equations are split in four equations, the diffusion equation $\boldsymbol{F}_{\mathrm{d}}$, the convection equation $\boldsymbol{F}_{\mathrm{c}}$, the pressure equation $\boldsymbol{F}_{\mathrm{p}}$ and the mass equation $\boldsymbol{F}_{\mathrm{m}}$. The diffusion equation is solved by the diagonal preconditioned symmetric conjugate gradient algorithm. The convection equation is solved by lumping the mass matrix and the inversion of the lumped diagonal. The pressure is then found from the pressure equation $\boldsymbol{F}_{\mathrm{p}}$ by the diagonal preconditioned symmetric conjugate gradient algorithm. The pressure is then corrected by lumping the mass matrix in the mass equation.

The iterations in the operator split algorithms (3) and (7) are executed sequentially. When the split algorithms have converged or after a fixed number of split iterations, the grid is adapted to the solution. The grid adaptions are performed iteratively outside the split iterations.

The main computational difference in the two algorithms is that the pressure split algorithm (3) is nonlinear while the velocity–pressure split algorithm (7) is linear. The pressure split algorithm therefore requires more work in generating the matrix coefficients than the velocity–pressure split algorithm.

The time-consuming parts of the algorithms described above are the conjugate gradient equation solvers. Within the conjugate gradient solvers the computation of the right-hand sides and the matrix–vector multiplications are dominant in time consumption. Parallel computations are therefore implemented for the computations of these two arithmetic operations.

4. Parallel algorithm

The structure of a main processor with N satellite processors are shown in Fig. 1. The communication in the network is only between the main processor and each of the satellites. The data is transmitted to each satellite through the network across communication sockets. Theoretically, the number of satellite processors which can be activated is only limited by the number of finite elements present in the grid. The number of satellite processors in the computational network can be fixed and activated before the computations begin or the satellite processors can be allocated and deallocated dynamically during the computational procedure.

The parallel computation is performed at finite element level. After the grid has been adapted to the solution, the number of elements are evenly divided and the elements are distributed to each satellite processor, as shown in Fig. 2. The assignment of elements to the different processors are determined by the order which the elements

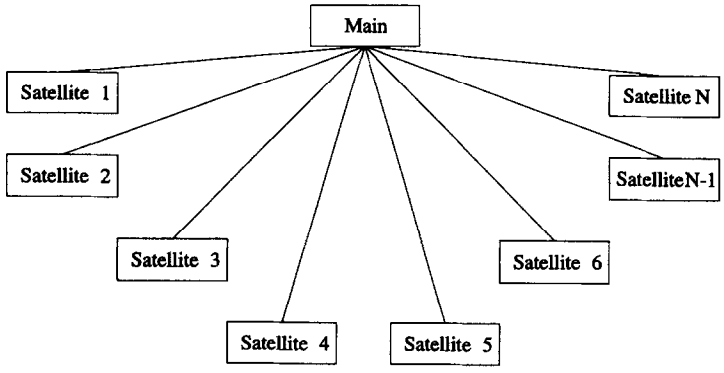


Fig. 1. This shows the structure of processors with the main processor communicating with satellite processors. The number of satellite processors can be changed during the computations.

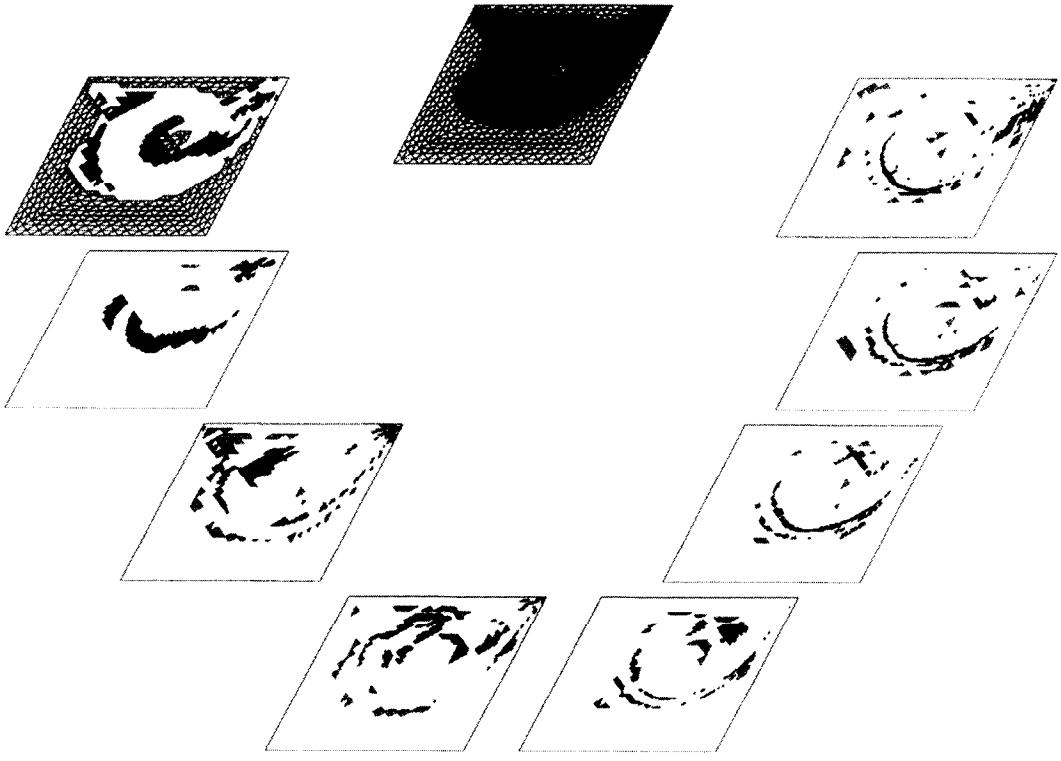


Fig. 2. This shows the distribution of finite element grids to eight satellite processors. The complete grid is shown in the middle. The total number of elements in the grid is 9426. The number of elements is 1179 in satellites 1–7 and 1173 in satellite 8.

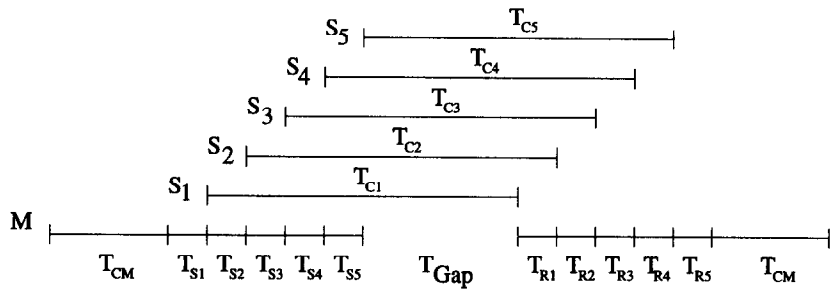


Fig. 3. This shows the time schedule for computations in the main processor, sending data to the satellites, computations in the satellite processors and reception of data from the satellite processors. M denotes the main processor and S the satellite processors. T_C is the computation time, T_S is the data sending time and T_R is the data receiving time. The time Gap indicates that the main processor is waiting for the satellite processors to finish their computations.

are generated by the Tri–Tree grid generator. In the distribution procedure, the nodes are renumbered and only those nodes needed in each satellite processor is communicated. The element data is sent to the satellite processors only once for each grid, after the grid adaption is performed.

The initial step of the conjugate gradient solver is to compute the right-hand side of the equation system. During the conjugate gradient algorithm, the diagonal of the equation matrix and the lumped mass matrix are needed. These vectors are also computed together with the right-hand side. In the transmission of data to each satellite processor, only the data associated with the elements belonging to the satellite is sent. When the result of the computations are returned to the main processor, the vectors from the satellites are assembled in the correct positions in the corresponding global vectors by the main processor. The data needed by the satellite processor in the matrix–vector multiplication is also the solution vector and the vector which is to be multiplied by the corresponding matrix.

Fig. 3 shows the time events and the time intervals for the computations and communications between the main and the satellite processors. The data to each of the satellite processors is distributed sequentially. Each satellite processor starts its computations simultaneously with the distribution of data to the next satellite processor. The amount of CPU time used by the first processor should therefore not be greater than the total time for transmitting data to the subsequent processors. The main processor will then be able to receive the results from the first satellite processor immediately after finishing the distribution of data to the last satellite processor. If the satellite processors use longer time, the main processor will be idle a certain time interval in waiting for the results from the satellite processors.

5. Adaption algorithms

There are three important parameters in the parallel solution algorithm for the Navier–Stokes equations. These parameters are the Reynolds number, the Courant number and the number of satellite processors,

$$\mathrm{Re} = \frac{\rho\|\boldsymbol{v}\cdot\nabla\boldsymbol{v}\|}{\mu\|\nabla^2\boldsymbol{v}\|} \qquad \mathrm{Co} = \frac{\rho\|\boldsymbol{v}\cdot\nabla\boldsymbol{v}\|}{\rho\left|\dfrac{\nabla\boldsymbol{v}}{\partial t}\right|}N_S \tag{11}$$

The Reynolds number is defined as the ratio of convection to diffusion. The Courant number is defined as the ratio of convection to acceleration. The Reynolds number reflects the degree of nonlinearity in the equation system. The Courant number indicates the degree of hyperbolicity in the equation system. The Reynolds number and the Courant number are computed for each element.

5.1. Grid adaption

The element Reynolds number Re_e is computed for each Tri–Tree element from the expression given below. Let L_i^c be the linear basis function evaluated at the geometrical center of the element. Then, the different parameters become

$$\mathrm{Re}_e = \frac{\sum_i L_i^c \left\| \rho \int_\Omega N_i \boldsymbol{v} \cdot \nabla \boldsymbol{v} \, \mathrm{d}\Omega \right\|}{\sum_i L_i^c \left\| \mu \int_\Omega \nabla L_i \cdot \nabla \boldsymbol{v} \, \mathrm{d}\Omega \right\|} < \epsilon_{\mathrm{Re}} \tag{12}$$

Numerical experiments, Wille [10,11], has shown that $\epsilon_{\mathrm{Re}} < 10$ in two spatial dimensions and $\epsilon_{\mathrm{Re}} < 30$ in three dimensions in order to obtain a converged solution for the Navier–Stokes equations. In the present work, the element Reynolds number limit is $\epsilon_{\mathrm{Re}} = 1$ for refinement and recoarsement of the grid.

5.2. Time adaption

The element Courant number Co_e is computed for each Tri–Tree element

$$\mathrm{Co}_e = \frac{\sum_i L_i^c \left\| \rho \int_\Omega L_i \boldsymbol{v} \cdot \nabla \boldsymbol{v} \, \mathrm{d}\Omega \right\|}{\sum_i L_i^c \left\| \rho \int_\Omega L_i \frac{\partial \boldsymbol{v}}{\partial t} \, \mathrm{d}\Omega \right\|} < \epsilon_{\mathrm{Co}} \tag{13}$$

For explicit time schemes, it has been shown theoretically that the time marching scheme remains stable if $\epsilon_{\mathrm{Co}} < 1$ [12]. These values have been derived for whole geometries where characteristic length and mean velocity are applied in the derivations. The element Courant number has experimentally [7] been found to be in the range of 1.0 to 2.5 when divergence occurs. In the present work, the Courant number limit is chosen to be 0.3. The length of the timestep is computed from

$$\Delta t < 0.2 \, \Delta t_0 / \mathrm{Max}(\mathrm{Co}_e) \tag{14}$$

where Δt_0 is the timestep in the computation of the element Courant Number, Co_e.

5.3. Satellite adaption

The optimal choice of the number of satellite processors will be to adapt exactly the number of satellite processors which make $T_{\mathrm{Gap}} = 0$ in Fig. 3. Assume that the sending and receiving time are equal, $T_{Si} = T_{Ri}$. Let T_{Ci} be the CPU time used by satellite processor i, $T_S = \Sigma T_{Si}$ be the total time for sending data to the satellite processors and N_S the number of satellite processors. Then, a robust adaption criterion will be

$$\frac{\Sigma T_{Ci}}{N_S} < T_S \frac{N_S - 1}{N_S} \quad \text{or} \quad N_S = \frac{\Sigma T_{Ci}}{\Sigma T_{Si}} + 1 \tag{15}$$

The satellite adaption is performed after the grid adaption. The number of satellites needed in the computations is computed from Eq. (15). The number of satellite processors can be both increased and reduced as the number of finite elements varies during the computations.

6. Experiments

The numerical algorithms are tested for the driven cavity flow. The density of the fluid is $\rho = 1000$ and the viscosity is $\mu = 0.001$.

The real time for solving a computational problem when applying a multiuser processor system depends on several factors, as for example the external computer load and the varying load of the operating system. Such factors are beyond the control of a single user. In the test experiments, the main processor was running on one computer and all the satellite processors were running on other computers. Since only a limited number of computers were available, several satellite processors were running on the same computer.

Experiments show that the CPU time used in each satellite processor and the communication time between the main and the satellite processors, varies slightly even if the number of element data is the same in each

satellite processor. The sending times and receiving time of the data between processors varies only slightly and they are therefore considered equal. A robust measure for satellite CPU time and communication time are therefore based on the average CPU time in all satellite processors and the average sending and receiving time.

6.1. Fixed number of satellite processors

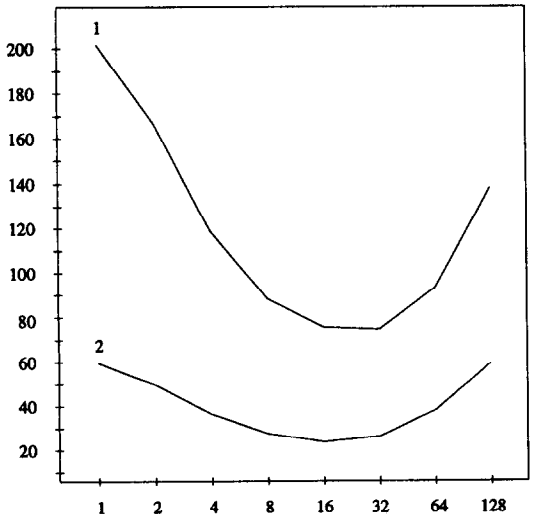
The real computational CPU time is calculated from Fig. 3. The computational CPU time, T_{tot}, must be considered in two cases

$$\begin{aligned} &\frac{\Sigma T_{Ci}}{N_S} > \frac{N_S - 1}{N_S}\Sigma T_{Si} \qquad T_{\text{tot}} = T_{CM} + \frac{\Sigma T_{Ci}}{N_S} + \frac{N_S + 1}{N_S}\Sigma T_{Si} \\ &\frac{\Sigma T_{Ci}}{N_S} < \frac{N_S - 1}{N_S}\Sigma T_{Si} \qquad T_{\text{tot}} = T_{CM} + 2 \cdot \Sigma T_{Si} \end{aligned} \tag{16}$$

When processing time in the satellite processors is greater than the total data sending time, the total CPU time is the sum of the CPU time in the main processor, the average satellite processor time and sending time for all satellite processors. When processing time in the satellite processors is less than or equal to the total data sending time, the total CPU time is the sum of the CPU time in the main processor, the sending time and the receiving time. However, the sending and receiving times are considered equal so that the total time is the CPU time in the main processor and twice the sending time.

The purpose of the following experiments is to measure the different time intervals. The measurements are performed for Reynolds number 400 and Reynolds number 800. The number of all iterations are fixed to make the results comparable. The number of linear iterations $N_{\text{L}} = 50$, the number of operator split iterations $N_{\text{O}} = 10$ and the number of grid iterations $N_{\text{G}} = 2$. The grid consists of 1693 elements in the first and 1967 elements in the second grid iteration for Reynolds number 400. The grid consists of 4835 elements in the first and 6300 elements in the second grid iteration for Reynolds number 800. The average number of elements is 1840 for Reynolds number 400 and 5568 for Reynolds number 800. The number of elements for Reynolds number 800 is 2.9 times the number of elements for Reynolds number 400.

Fig. 4 shows the estimated real CPU time for the velocity–pressure split algorithm for Reynolds number 400



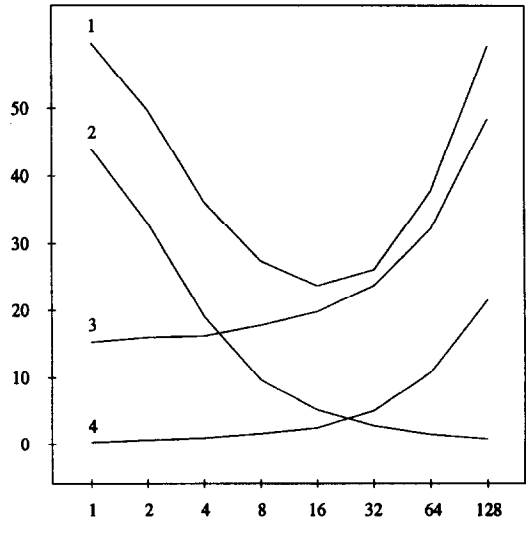


Fig. 4. This shows the estimate of the total real time in seconds with the velocity–pressure algorithm as a function of number of satellite processors. Curve 1 shows the estimated real time for Reynolds number 800. Curve 2 shows the estimated real time for Reynolds number 400.

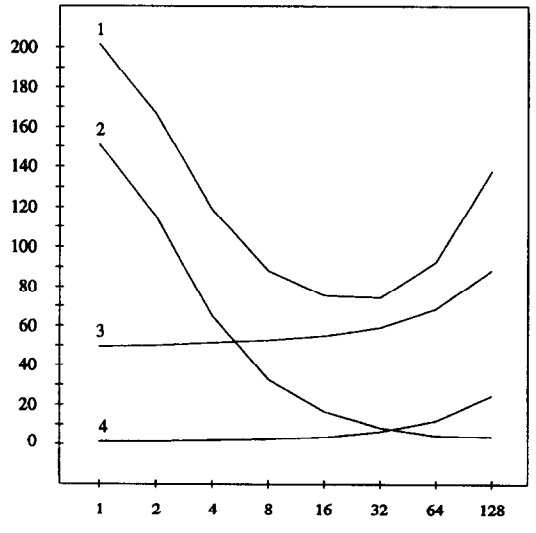
Fig. 5. This shows the times in seconds for the velocity-split algorithm as a function of number of satellite processors for Reynolds number 400. Curve 1 shows the estimated real time, curve 2 shows mean satellite CPU time, curve 3 shows the main processor CPU time and curve 4 shows the sending time for data transmission to the satellites.

and Reynolds number 800. The minimum CPU time for Reynolds number 400 is 25 s and Reynolds number 800 is 75 s. As the number of elements for Reynolds 800 is 2.9 times the number of elements for Reynolds number 400, these results indicate a linear increase in CPU time as a function of number of elements. The minimum CPU time for Reynolds number 400 occurs for 16 satellite processors and the minimum CPU time occurs for 32 satellites processors for Reynolds number 800. Since the number of elements for Reynolds number 800 is 2.9 times the number of element for Reynolds number 400, the increase in number of satellite processors is less than 2.9 times the number of satellite processors for the minimum CPU time for Reynolds number 400. This fortunate effect can be explained by relatively large overhead cost in CPU time when the number of elements is small.

Fig. 5 shows curves for the estimated CPU time for the computations, the mean satellite CPU time, the main processor CPU time and the sending time for data transmission for Reynolds number 400. As expected, the results show that the mean satellite CPU time is decreasing with an increasing number of satellite processors. The main processor CPU time is approximately constant until a certain number of satellite processors is reached. Then, the main processor CPU time increases with the increase in number of satellite processors due to more work in assembling, disassembling of the grid and communicating vectors. The sending time of data is also approximately constant until a certain number of satellite processors is reached. Then, the sending time is increasing due to the increase in transmission overhead for communicating with each satellite processor. The satellite CPU time curve and the sending time curve are crossing each other close to the minimum of the estimated total CPU time.

Fig. 6 shows curves for the estimated CPU time for the computations, the mean satellite CPU time, the main processor CPU time and the sending time for data transmission for Reynolds number 800. These time vs. number of satellite processor curves have similar properties as those for Reynolds number 400. The mean satellite CPU time is decreasing with an increasing number of satellite processors. The main processor CPU time is constant until a certain number of satellite processors is reached, and the main processor CPU time increases with the increase in number of satellite processor. The first part of the sending time of data curve is constant and starts to increase when the optimal number of satellite processors is reached. The satellite CPU time and the sending time curves are crossing is confirmed to be close to the minimum of the estimated total CPU time.

Fig. 7 displays the simulation results for the pressure split algorithm for Reynolds number 800. The minimum estimated CPU time for the pressure-split algorithm presented by Goda [5] is approximately 100 s. The



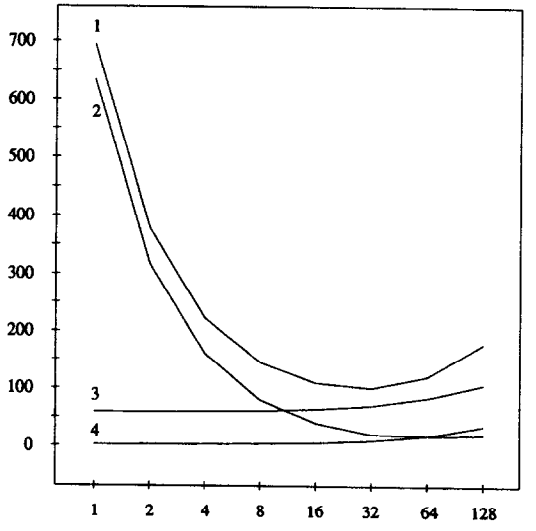


Fig. 6. This shows the times in seconds for the velocity-split algorithm as a function of number of satellite processors for Reynolds number 800. Curve 1 shows the estimated real time, curve 2 shows mean satellite CPU time, curve 3 shows the main processor CPU time and curve 4 shows the sending time for data transmission to the satellites.

Fig. 7. This shows the times in seconds for the pressure-split algorithm as a function of number of satellite processors for Reynolds number 800. Curve 1 shows the estimated real time, curve 2 shows mean satellite CPU time, curve 3 shows the main processor CPU time and curve 4 shows the sending time for data transmission to the satellites.

minimum estimated CPU time for the velocity–pressure split algorithm presented in Fig. 6 is 75 s. As the level of convergence for the two algorithms with fixed number of iterations is of the same level [7], the estimated CPU time of the velocity–pressure algorithm is 75% of the pressure split algorithm. Otherwise, the time vs. number of satellite processor curves for the pressure split algorithms reveals similar properties as for the velocity pressure split algorithm.

6.2. Adapted number of satellite processors

The convergence criteria for the different iterations in the dynamic adaption of the number of satellite processors are chosen as follows. The linear conjugate gradient iterations is $\epsilon_L = 0.0001$. The limiting number of linear conjugate gradient iterations is $N_{\text{Lmin}} = 5$ and $N_{\text{Lmax}} = 50$. The maximum number of operator split iterations is always $N_{\text{Omax}} = 100$. There are therefore two possibilities for exiting the linear and nonlinear iterations. The iterations are either stopped when the convergence criterion is reached or when the number of iteration limits is reached. The number of grid adaption iterations N_G at each Reynolds number is 10.

$$N_S = \frac{\Sigma T_{Ci}}{\Sigma T_{Si}} + 1 \tag{17}$$

The size of the simulations is shown in Fig. 8. An increase in the Reynolds number with a factor two also increases the number of elements and nodes by the same factor. At Reynolds number 3200 the number of elements is 179 001 and the number of nodes is 89 563. The number of degrees of freedom for Reynolds number 3200 is 268 689.

Fig. 9 shows the number of elements, the number of Tri–Tree refinements and recoarsements for each grid iteration. The number of refinements and recoarsements show peaks at the increase in Reynolds number. The increase in number of refinements and recoarsements simultaneously indicate that the concentration of refined elements moves towards the cavity boundary with increasing Reynolds number [2].

In the first grid iteration after an increase in the Reynolds number by a factor 2, there are no estimates for the mean satellite CPU time and sending time of data to the satellite processors for the new Reynolds number. Thus, in the first grid iteration, the number of satellite processors used is the same as for the previous Reynolds number. For the second grid iteration at each Reynolds number the appropriate times for estimating the number

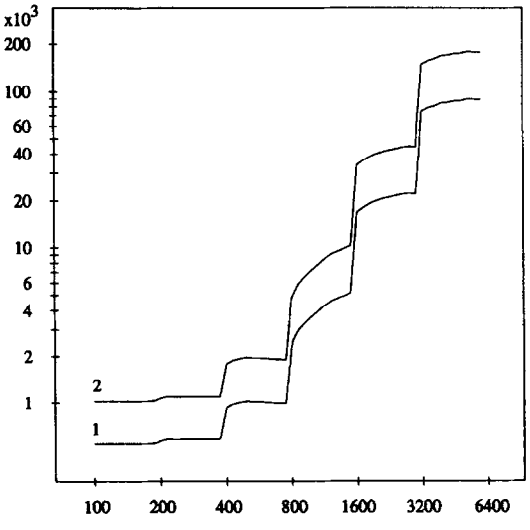


Fig. 8. This shows the number of elements, curve 1, and the number of nodes, curve 2, for the different grid iterations at each Reynolds number.

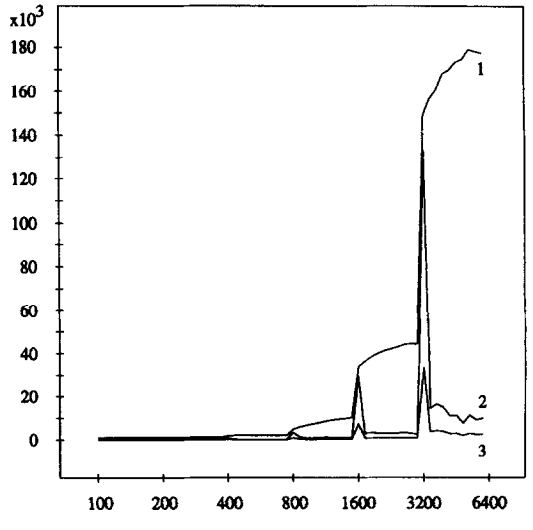


Fig. 9. This shows the number of elements, curve 1, the number of Tri–Tree refinements, curve 2, and the number of Tri–Tree recoarsements, curve 3, for each grid iteration.

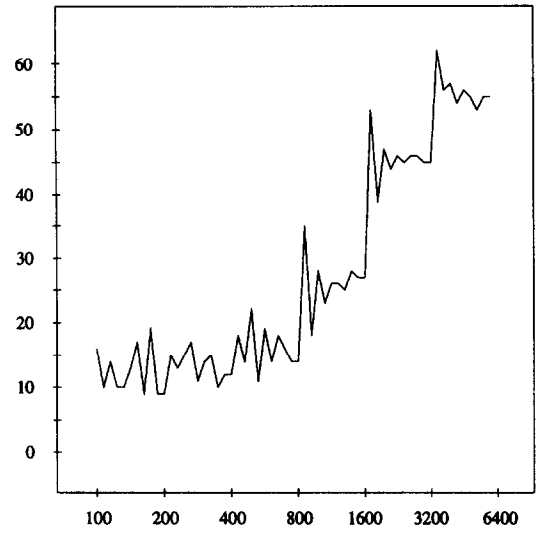


Fig. 10. This shows the number of satellite processors used at each grid iteration.

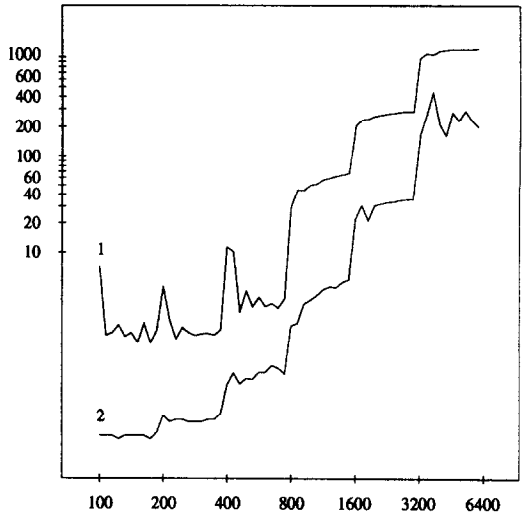


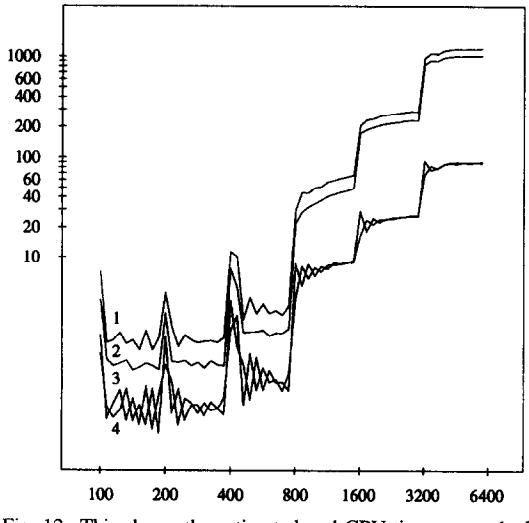
Fig. 11. This shows the CPU time used in the main processor, curve 1, for computations and the CPU time used for grid adaption, curve 2.

of satellite processors were measured and the selection of the number of satellite processors is given by Eq. (15).

Fig. 10 shows the number of satellite processors at each grid iteration. The number of satellite processors demonstrates oscillations after an increase in Reynolds number. For the highest Reynolds numbers, these oscillations disappear after some grid iterations. In the transition from one Reynolds number to the next, the same number of satellite processors is used in the first grid iteration at the new Reynolds number. However, a better estimate for the optimal number of satellite processors for the initial iteration at each Reynolds number will be sought.

Fig. 11 shows the comparison of CPU time used in the numerical solution algorithm and the CPU time used in the grid adaption algorithm. The figure shows that the grid adaption algorithm is faster than the numerical algorithm. However, parallelization of the grid adaption algorithm will increase the efficiency of the total algorithm.

Fig. 12 shows the estimated real CPU time, the effective computational CPU time in the main processor, the average satellite CPU time and the total data sending time. The figure indicates that the estimated real CPU time



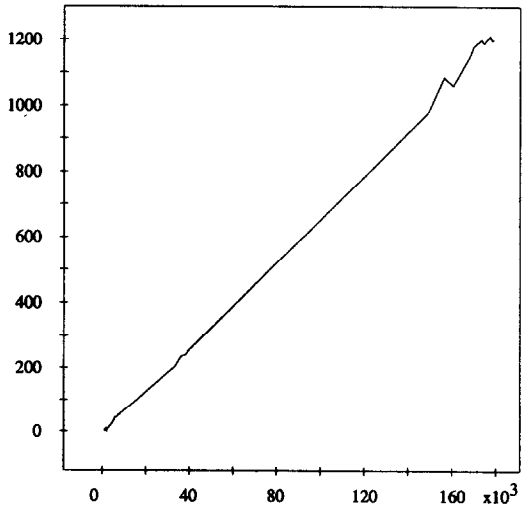


Fig. 12. This shows the estimated real CPU time, curve 1, the effective CPU time in the main processor, curve 2, the average satellite CPU time, curve 3, and the total sending time, curve 4.

Fig. 13. This shows the estimated real time as a function of the number of elements.

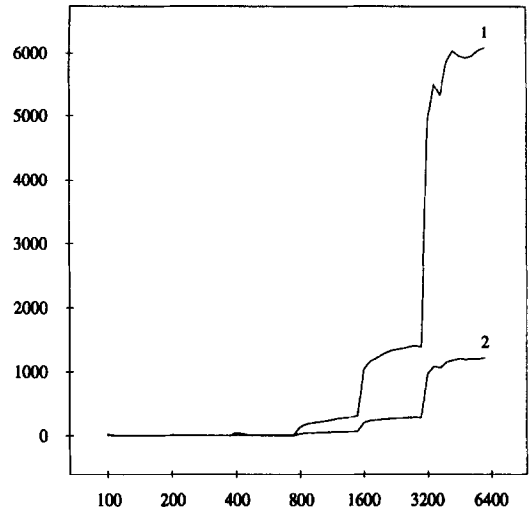


Fig. 14. This shows the estimated real time for a sequential algorithm, curve 1, and the estimated real time for the parallel algorithm, curve 2.

is only a little longer than the effective main processor CPU time. This means most of the CPU time in the main processor is used in the communication between the main processor and the satellite processors. The parallelization is therefore very efficient. The satellite processor sending time and mean satellite processor CPU time are almost equal. The small difference is due to the number of satellite processors computed from the previous time measuring period.

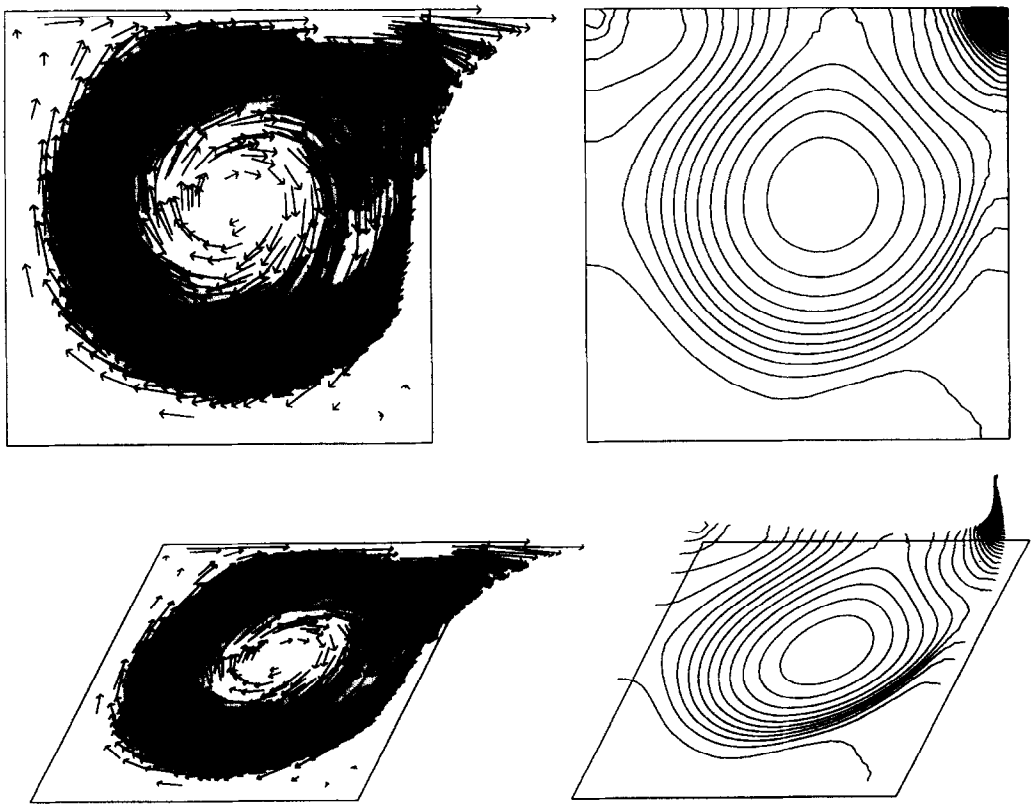


Fig. 15. This shows the velocity vectors (left) and the pressure isobars (right) for Reynolds number 1600. The solution is computed by the nodal velocity–pressure split algorithm.

Fig. 13 shows the estimated real CPU time as a function of the number of elements in the grid. The figure shows an advantageous linear relation, the estimated real CPU time increases linearly with the number of elements.

Fig. 14 shows the comparison of estimated CPU time for a serial and a parallel computation. The serial computation time is obtained by adding the satellite processors CPU time and the CPU time for the main processor. The parallel computation time is obtained from Eq. (16). The figure shows that the parallel computations are approximately six times faster than the serial computations for the highest Reynolds number.

Fig. 15 shows the solution of the cavity flow problem in terms of velocity vectors and pressure isobars for Reynolds number 1600.

The gain in parallelization can be improved both by improving the parallel numerical algorithms and by improving the speed of the processors and communication network. As seen from Eq. (16) the efficient CPU real time is strongly dependent on the CPU time in the main processor and the time for sending and receiving data. If the communication speed of the network can be improved, more satellite processors can be attached for increased efficiency.

7. Discussions

In the present work a nodal finite element algorithm has been presented. The nodal algorithm saves the storage space which is usually needed in implicit algorithms. The saving in computer storage by the nodal algorithm makes it possible to solve problems which can be several orders of magnitude larger in size than for implicit algorithms where the equation matrices are stored.

The nodal algorithm has been implemented for parallel processing. The parallel processing is performed on ordinary pentium processors and does not require the power of expensive supercomputers. The communication between the main processor and the satellite processors are done on standard ethernet. For extremely large problems, processors which are geographically located elsewhere can be adopted in the parallel processing.

The experiments show that the estimated computational time as a function of the number of satellite processors has a minimum. The number of satellite processors for which this minimum occur can be predicted and dynamically adjusted from measurements of the satellites CPU time and data transmission times.

Although the parallelization of the numerical algorithms play an important role in gaining efficiency in the computations, the efficiency depends on both computational and communication network speed.

Acknowledgments

The authors are grateful to Trygve Svoldal for valuable suggestions and corrections of the manuscript. The project has been supported by The Norwegian Research Council, grant no. NN2461K., in partially financing the computer runtime expenses.

References

[1] S.Ø. Wille, A structured Tri–Tree search method for generation of optimal unstructured finite element grids in two and three dimensions, Int. J. Numer. Methods Fluids 14 (1992) 861–881.

[2] S.Ø. Wille, The prolonged adaptive multigrid method for finite element Navier–Stokes equations, Comput. Methods Appl. Mech. Engrg. 138 (1996) 227–271.

[3] Y. Kallinderis, Adaptive hybrid prismatic/tetrahedral grids, Int. J. Numer. Methods Fluids 20 (1992) 1023–1037.

[4] K.F.C. Yiu, D.M. Greaves, S. Cruz, A. Saalehi and A.G.L. Borthwick, Quadtree grid generation: Information handling, boundary fitting and CFD applications, Comput. Fluids 25 (1996) 759–769.

[5] K. Goda, A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows, J. Comput. Phys. 30 (1979) 76–95.

[6] G. Ren and T. Utnes, A finite element solution of the time-dependent Navier–Stokes equations using a modified velocity correction method, Int. J. Numer. Methods Fluids 17 (1993) 349–364.

[7] S.Ø. Wille, Nodal operator splitting adaptive finite element algorithms for the Navier–Stokes equations, Int. J. Numer. Methods Fluids (1997) in press.
[8] A.K. Stagg, D.D. Cline, G.F. Carey and J.N. Shadid, Parallel, scalable parabolized Navier–Stokes solver for large-scale simulations, AIAA J. 33 (1995) 102–108.
[9] D. Goldberg, Applications de la methode des projections au calcyl par elements finis d'ecoulements tridimensionnels de fluides visqueux incompressible, These de Doctoratl'Universite Paris VI, 1994.
[10] S.Ø. Wille, The three dimensional prolonged adaptive unstructured finite element multigrid method for the Navier–Stokes equations, Int. J. Numer. Methods Fluids 25 (1996) 371–392.
[11] S.Ø. Wille, A local predictive convection–diffusion refinement indicator for the Tri–Tree adapted finite element multigrid algorithm of the Navier–Stokes equations, Comput. Methods Appl. Mech. Engrg. 134 (1996) 181–196.
1996. [12] Z.U.A. Warsi, Fluid Dynamics, Theoretical and Computational Approaches (CRS Press, London, 1993).