INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2002; **40**:1347–1363 (DOI: 10.1002/fld.368)

Homogeneous and heterogeneous distributed cluster processing for two- and three-dimensional viscoelastic flows

A. Baloch, P. W. Grant and M. F. Webster*,†

Institute of non-Newtonian Fluid Mechanics, Department of Computer Science, University of Wales Swansea, Swansea, SA2 8PP, U.K.

SUMMARY

A finite-element study of two- and three-dimensional incompressible viscoelastic flows in a planar lid-driven cavity and concentric rotating cylinders is presented. The hardware platforms consist of both homogeneous and heterogeneous clusters of workstations. A semi-implicit time-stepping Taylor–Galerkin scheme is employed using the message passing mechanism provided by the Parallel Virtual Machine libraries. DEC-alpha, Intel Solaris and AMD-K7(Athlon) Linux clusters are utilized. Parallel results are compared against single processor (sequentially) solutions, using the parallelism paradigm of domain decomposition. Communication is effectively masked and practically ideal, linear speed-up with the number of processors is realized. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: distributed parallel computation; finite elements; viscoelastic fluids; cavity flow; rotating flow

1. INTRODUCTION

In recent years, parallel computation has become increasingly more important for solving large-scale computational fluid dynamics problems, which arise in many areas of science and engineering involving both compressible and incompressible flow regimes. We are particularly interested in complex viscoelastic flows, of immediate relevance to the processing industries, associated with polymer, foods and oil products. The mathematical modelling of such flows, typically generates complex three-dimensional (3D) systems of partial differential equations of mixed type. Common discretization approaches adopted, such as finite element, finite volume or spectral element formulations, transform these systems from differential to algebraic form, generating large numbers of degrees of freedom. Over the preceding decade, with the advance of computer hardware and developments in sophisticated numerical algorithms, it has become easier to solve complex flows, albeit of limited size. To make satisfactory progress in this

* Correspondence to: M. F. Webster, Institute of Non-Newtonian Fluid Mechanics, Department of Computer Science, University of Wales Swansea, Swansea SA2 8PP, U.K.
† E-mail: m.f.webster@swansea.ac.uk

Contract/grant sponsor: UK EPSRC; contract/grant number: (GR/L14916)

Received October 2000
Revised January 2002

Copyright © 2002 John Wiley & Sons, Ltd.

area often fine resolution meshes are required, that may also involve adaptive meshing. This places practical limitations upon the range and scope of the problems that may be tackled in terms of size, necessitating a shift from a sequential to a parallel mode of computation.

Parallel computation may be viewed in a distributed manner, where memory and processors are distinct, or in a combined form where memory is shared. The distributed model involves sending and receiving messages and configuring a processor network (e.g. master/slave). Over the last few years there has been an increase in the availability of software to perform such message passing. Recent developments in parallel computing using message passing protocols, such as Theoretical Chemistry Message Passing Tool Kit (TCMSG), Parasoft Express (EXPRESS), Network Linda (LINDA), Message Passing Interface (MPI) and Parallel Virtual Machine (PVM), have given impetus to the design of parallel algorithms to solve very large time consuming three-dimensional problems. In this paper, following the study of Reference [1], PVM version 3.4.3 is adopted using a Master/Slave configuration.

In the solution of viscoelastic flows, inverting and resolving linear systems of equations constitutes a large proportion of CPU time overhead. The nature of the problem to be solved may be coupled, leading to large system matrices or alternatively, decoupled which may be handled iteratively.

Examples of using PVM for parallelizing numerical codes are described in References [2, 3]. In Reference [2], two-dimensional parallel computation of viscoelastic flows, steady-state solutions for entry flow and stick-slip flow problems are obtained employing the POLYFLOW code with a finite-element algorithm. A non-linear system of partial differential equations of mixed-type KBKZ integro-differential equations are solved over unstructured triangular meshes. Both coupled (MIXl and SU4 $\times$ 4) and decoupled (iterative) system approaches are contrasted. For the solution of the linear systems involved, a semi-iterative approach is adopted in the style of a Dual Schur Complement technique.

A time-stepping finite-volume method was used in Reference [3] with the SIMPLER algorithm. Results are presented for the solution of flow past a cylinder between two parallel plates, employing an exponential PTT model, unstructured triangular meshes, and a decoupled solution procedure. The algebraic linear system of equations are solved sequentially through a SIMPLER iteration procedure with an explicit Gauss–Seidel solver. For domain decomposition, various methods were tested, with RSB being recommended for mesh partitioning.

For parallel computation, both [2, 3] have used PVM as the message passing mechanism on MIMD parallel computers in master/slave style. Intel IPSC/860 hypercube and Convex Meta Series shared-memory were used by Keunings [2] and DEC-alpha clusters were used by Dou and Phan-Thien [3].

For any fixed mesh, the performance of these parallel implementations are presented demonstrating monotonically increasing speed-up and monotonically decreasing efficiency with increasing number of domains (processors). Running with a fixed number of processors illustrates the increase in speed-up and efficiency with an increasing number of mesh elements.

The present study adopts a semi-implicit Taylor–Galerkin/pressure-correction finite-element time-marching scheme, that has been developed and refined over the last decade. This scheme, initially implemented sequentially in FORTRAN 77, is appropriate for the simulation of incompressible isothermal Newtonian, fibre suspension, generalized inelastic and viscoelastic flows [4–8]. Parallel implementations of this algorithm for Newtonian and Generalized inelastic fluids for flow past a rigid sphere in a tube employing unstructured meshes is

Copyright © 2002 John Wiley & Sons, Ltd.

described in References [1, 9]. Parallelization was achieved using PVM on a cluster of diskless Sun Spare-1 workstations with domain decomposition and a degree-of-freedom approach. For the solution of the linear algebraic systems, a hybrid method was adopted, i.e. direct matrix inversion (Choleski factorization) for the pressure stiffness matrix and iteration for the momentum equations. Parallel efficiency close to the ideal (linear) was achieved.

In this paper we extend our previous work, describing how the algorithm has been implemented in parallel for viscoelastic fluids using differential constitutive models of single-mode type.

The stress equations are solved using a decoupled approach as for the momentum equations. For viscoelastic calculations, a single-mode Oldroyd-B model is employed. Such a model may reflect memory and constant shear viscosity fluid properties. Triangular elements in two dimensions and tetrahedral elements in three dimensions are employed with quadratic interpolation for velocity and stress, and linear interpolation for pressure. A domain decomposition method is adopted on both shared and distributed memory clusters for homogeneous and heterogeneous environments.

Two types of problem are investigated. The first concerns the two- and three-dimensional flow of Newtonian and viscoelastic fluids within a lid-driven cavity—a confined-flow moving boundary problem. This problem is industrially relevant for processing applications such as short-dwell and coating (flexible blade, roller, packing tape, etc.). It is also of importance for testing numerical methods and understanding viscoelastic effects. The simplicity and regularity of the geometry, presence of recirculation regions in the centre and stagnation in the corners, and singularity at the edges of the lid with strong extension, present interesting flow phenomena. The discontinuous nature of the velocity boundary conditions at the two edges/points where the moving lid is in contact with two adjacent solid walls, generates flow singularities. The combination of these features, has resulted in the lid-driven cavity flow becoming a popular benchmark problem in the field of computational fluid dynamics, upon the basis of which convergence and stability may be established.

The second problem investigated in this study is the two- and three-dimensional flow of Newtonian and viscoelastic fluids between concentric rotating cylinders. This problem arises in the food industry, for example, within dough mixing. The availability of an analytical solution and the simplicity of the domain, permits algorithm validation within a cylindrical polar coordinate system.

Results are presented on the performance of the parallel algorithm for these two problems, providing speed-up and parallel efficiency measures for a variety of different sized meshes and network configurations.

2. PROBLEM SPECIFICATION

The particular problems simulated in this study are two- and three-dimensional flows of planar steady lid-driven cavity of unit dimension and concentric rotating cylinders for Oldroyd-B fluid. Initially, a solution is obtained for viscous Newtonian fluid.

Schematic diagrams for the three-dimensional domains of interest covering the cubic lid-driven cavity and concentric rotating cylinder flows are displayed in Figure 1. Symmetric reflected structured meshes are chosen. Two-dimensional finite-element meshes are shown in Figure 2. In two dimensions, two triangular elements within a rectangle are used, while in

Copyright © 2002 John Wiley & Sons, Ltd.

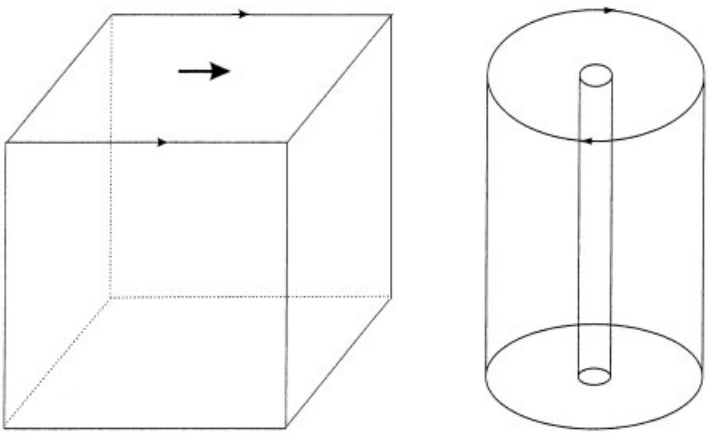

Figure 1. Schematic diagrams for planar lid-driven cavity and concentric rotating cylinder domains.

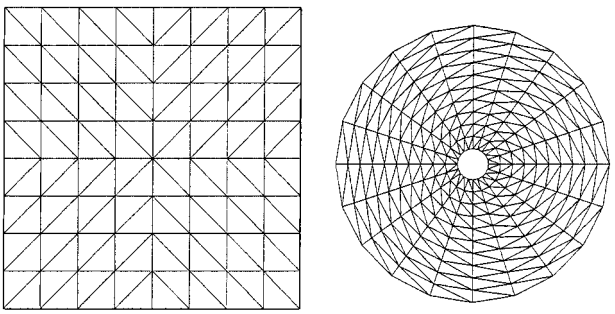

Figure 2. Two-dimensional finite-element meshes for planar lid-driven cavity and concentric rotating cylinder flows.

Table I. Two-dimensional mesh data.

Meshes	Elts	Nodes	DOF(N)	DOF(V)
lid cav				
10^2	200	441	1003	2326
20^2	800	1681	3803	8846
40^2	3200	6561	14803	34486
60^2	7200	14641	33003	76926
rot cyl	400	840	1900	4420

three-dimension six tetrahedra form a cube. The total number of elements, nodes and degrees of freedom are presented in Tables I and II according to the relevant dimensions and fluids, where N and V denote Newtonian and viscoelastic cases, respectively. The degrees of

Copyright © 2002 John Wiley & Sons, Ltd.

Table II. Three-dimensional mesh data.

Meshes	Elts	Nodes	DOF(N)	DOF(V)
lid cav				
4^3	384	729	2312	6686
8^3	3072	4913	15468	44946
10^3	6000	9261	29114	84680
rot cyl	6000	9240	29040	84480

freedom increase by a factor of approximately three with the introduction of stress variables in the viscoelastic case.

The statement of the flow problem is completed by prescribing appropriate initial and boundary conditions. Simulations start from a quiescent state for both two- and three-dimensional lid-driven cavity and concentric rotating cylinder flows. No-slip velocity boundary conditions are imposed on the solid surfaces/walls of both problems.

For lid-driven cavity flow, a constant velocity on the top moving-lid and a fixed pressure ($p = 0$) at the departing flow edge/point on the lid in three/two dimensions are applied. For viscoelastic flows, a fixed stress tensor is assumed at the entering edge/point of the lid in three/two dimensions, respectively.

For concentric rotating cylinder flow, the lid/stirrer is held stationary and the outer cylinder (vessel) and base rotate at a constant rate. A fixed pressure ($p = 0$) is assumed at the outer rotating cylinder in three/two dimensions.

For these time-stepping computations, an appropriate time-step value of $\Delta t = 0.01$ is chosen and a relative termination tolerance of 10^{-5} is enforced. To confirm correctness of three-dimensional computations, two-dimensional cross-sectional flows are computed at a central sliced mid-plane of the three-dimensional geometry.

3. GOVERNING SYSTEM OF EQUATIONS

The two- and three-dimensional isothermal flow of incompressible viscoelastic fluid can be modelled through a system comprising of the generalized momentum transport, conservation of mass and viscoelastic stress constitutive equations. In the absence of body forces, Equations (1)–(5) represent the governing system. Incompressibility is expressed, via the conservation of mass, as

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

The conservation of momentum transport equation is

$$\rho \frac{\partial \mathbf{u}}{\partial t} = \nabla \cdot \sigma - \rho \mathbf{u} \cdot \nabla \mathbf{u} \tag{2}$$

where, $\mathbf{u}$ is the fluid velocity vector field, σ is the Cauchy stress tensor, ρ is the fluid density, t represents time and $\nabla\cdot$ and ∇ are divergence and gradient operators, respectively. The Cauchy

Copyright © 2002 John Wiley & Sons, Ltd.

stress tensor can be expressed in the form

$$\sigma = -p\delta + \mathbf{T}_e \tag{3}$$

where p is the isotropic fluid pressure (per unit density), δ is the Kronecker delta tensor, whilst $\mathbf{T}_e$ is the extra-stress tensor.

For the upper-convected Oldroyd-B differential model the extra-stress tensor $\mathbf{T}_e$ is given by

$$\mathbf{T}_e = 2\mu_2\mathbf{d} + \boldsymbol{\tau} \tag{4}$$

$$\lambda_1 \overset{\nabla}{\boldsymbol{\tau}} + \boldsymbol{\tau} = 2\mu_1\mathbf{d} \tag{5}$$

where $\mathbf{d}$ is the rate-of-strain tensor, $\boldsymbol{\tau}$ the polymeric stress tensor and $\overset{\nabla}{\boldsymbol{\tau}}$ the upper-convected material derivative. μ_1 is the polymeric and μ_2 the solvent fluid viscosities, and λ_1 is a relaxation time (single/averaged). Then, the total viscosity $\mu = \mu_1 + \mu_2$, and the ratio μ_1/μ_2, is taken as $\frac{1}{9}$.

4. NUMERICAL METHOD

The numerical algorithm employed is a time-marching semi-implicit TGPC scheme [4–8], based on a fractional-step formulation. This involves discretization, first in the temporal domain, adopting a Taylor series expansion in time and a pressure-correction operator-split, to build a second-order time-stepping scheme. This scheme has been developed extensively elsewhere. The pressure-correction split follows a Crank–Nicolson style (see Reference [10]), requiring the solution of the temporal difference of pressure. The predictor–corrector doublet at step 1, is a Lax–Wendroff scheme, here of Taylor–Galerkin form, explicit on convection terms. Semi-implicit aspects arise, via treatment of diffusion terms, that removes the excessive stability limitation due to diffusion. Only the Courant conditional must be respected, and for the present structured meshes, a fixed time step is found quite adequate [4, 5].

Spatial discretization is achieved via Galerkin approximation for the momentum equation and Streamline-Upwind-Petrov–Gelerkin for the constitutive equation. The finite-element basis functions are quadratic for velocity and stress, and linear for pressure, taken over triangles for two dimensions and tetrahedra in three. Galerkin-weighted integrals are evaluated by exact integration, while for Streamline-Upwind-Petrov–Galerkin integrals, seven and 15 point Gauss quadrature rules are invoked for two- and three-dimensional problems, respectively.

Utilizing conventional inner-product notation for square-integrable functions, a weighted-residual form of the equations is obtained by projecting the momentum equation onto the space of test functions $\mathscr{U}_0$, the continuity equation onto the test function space $\mathscr{P}$, the constitutive equation onto the test functions $\mathscr{T}_0$ and integrating over the spatial domain Ω. After integration by parts via the divergence theorem, a semi-implicit non-dimensional weak-form statement of the problem is

Stage 1a: Given $(\mathbf{u}^n, \boldsymbol{\tau}^n, p^n) \in \mathscr{U} \times \mathscr{T} \times \mathscr{P}$, find $\mathbf{u}^{n+1/2} \in \mathscr{U}$ and $\boldsymbol{\tau}^{n+1/2} \in \mathscr{T}$ such that

$$\left[\left(\frac{2}{\Delta t}(\mathbf{u}^{n+1/2} - \mathbf{u}^n), v\right) + \frac{1}{2Re}(\nabla(\mathbf{u}^{n+1/2} - \mathbf{u}^n), \nabla v)\right]$$
$$= \left(\left(p^n - \frac{1}{Re}\nabla\mathbf{u}^n - \boldsymbol{\tau}\right), \nabla v\right) - (((\mathbf{u}\cdot\nabla)\mathbf{u})^n, v) \quad \forall v \in \mathscr{U}_0 \tag{6}$$

Copyright © 2002 John Wiley & Sons, Ltd.

$$\left(\frac{2We}{\Delta t}(\boldsymbol{\tau}^{n+1/2}-\boldsymbol{\tau}^{n}),s\right)=((2\mu_1\mathbf{d}-\boldsymbol{\tau}-We\mathbf{u}\cdot\nabla\boldsymbol{\tau}),s)^{n}$$
$$+((We\{\boldsymbol{\tau}\cdot\nabla\mathbf{u}+(\boldsymbol{\tau}\cdot\nabla\mathbf{u})^{T}\}),s)^{n}\quad\forall s\in\mathscr{T}_0 \tag{7}$$

Stage 1b: Given $\mathbf{u}^{n},\mathbf{u}^{n+1/2}\in\mathscr{U}$, $\boldsymbol{\tau}^{n+1/2}\in\mathscr{T}$ and $p^{n}\in\mathscr{P}$, find $\mathbf{u}^{*}\in\mathscr{U}$ and $\boldsymbol{\tau}^{n+1}\in\mathscr{T}$ such that

$$\left[\left(\frac{1}{\Delta t}(\mathbf{u}^{*}-\mathbf{u}^{n}),v\right)+\frac{1}{2Re}(\nabla(\mathbf{u}^{*}-\mathbf{u}^{n}),\Delta v)\right]=\left(\left(p^{n}-\frac{1}{Re}\nabla\mathbf{u}^{n}-\boldsymbol{\tau}\right),\nabla v\right)$$
$$-(((\mathbf{u}\cdot\nabla)\mathbf{u})^{n+1/2},v)\quad\forall v\in\mathscr{U}_0 \tag{8}$$

$$\left(\frac{We}{\Delta t}(\boldsymbol{\tau}^{n+1}-\boldsymbol{\tau}^{n}),s\right)=((2\mu_1\mathbf{d}-\boldsymbol{\tau}-We\mathbf{u}\cdot\nabla\boldsymbol{\tau}),s)^{n+1/2}$$
$$+((We\{\boldsymbol{\tau}\cdot\nabla\mathbf{u}+(\boldsymbol{\tau}\cdot\nabla\mathbf{u})^{T}\}),s)^{n+1/2}\quad\forall s\in\mathscr{T}_0 \tag{9}$$

Stage 2: Given $(\mathbf{u}^{*},p^{n})\in\mathscr{U}\times\mathscr{P}$, find $p^{n+1}-p^{n}\in\mathscr{P}$ such that

$$(\theta\nabla(p^{n+1}-p^{n}),\nabla q)=-\frac{1}{\Delta t}(\nabla\mathbf{u}^{*},q)\quad\forall q\in\mathscr{P} \tag{10}$$

Stage 3: Given $(\mathbf{u}^{*},p^{n+1}-p^{n})\in\mathscr{U}\times\mathscr{P}$, find $\mathbf{u}^{n+1}\in\mathscr{U}$ such that

$$\left(\frac{1}{\Delta t}(\mathbf{u}^{n+1}-\mathbf{u}^{*}),v\right)=(\theta(p^{n+1}-p^{n}),\nabla v)\quad\forall v\in\mathscr{U}_0 \tag{11}$$

where Δt is the time lapse.

Here, n denotes the time-step index. Velocity and stress at the half step $n+\frac{1}{2}$ are computed in step 1a from data gathered at level n and in step 1b an intermediate non-solenoidal velocity field $\mathbf{u}^{*}$ and stress field $\boldsymbol{\tau}$ are computed at the full time-step, using the solutions at the level n and $n+\frac{1}{2}$. At this stage, calculations for stress over a full time-step cycle are complete. For pressure this leads naturally to a second step, where a Poisson equation is solved for the pressure difference subject to the non-solenoidal velocity field $\mathbf{u}^{*}$. For second-order accuracy in time the Crank–Nicolson choice of $\theta=0.5$ is adopted. In a third and final step, a solenoidal velocity field at the end of the time-step cycle is captured from the pressure-difference field of step 2.

In the present implementation of our algorithm, spatial discretization is affected by selecting finite-dimensional subspaces within $\mathscr{U}_0$, $\mathscr{U}$, $\mathscr{T}_0$, $\mathscr{T}$ and $\mathscr{P}$ (as stated above). The systems of equations that arise at stage one and three are solved by an iterative Jacobi technique, where only a handful of iterations (five) are required. This has always been our experience in scale-up for steady problems, where preference goes to larger numbers of time-step loops. Hence our expectation for large industrial problems. One may revise this choice to double the iterations, per linearized stage, for transient problems. At stage two, a Poisson equation, for the temporal difference of pressure is solved by a direct Choleski method. The procedures of assembly of right-hand side vectors and Jacobi iteration are highly parallelizable. As this

Copyright © 2002 John Wiley & Sons, Ltd.

algorithm is dominated by these element and iterative compute phases, the time complexity is theoretically expected to be linear in the degrees of freedoms [1]. Speed-up, via parallelism invoked over the number of processors, should also reflect this property, provided communication overhead is minimal and insignificant compared with process calculation time. In addition, the solution-independent structure of the 'Poisson-like' operator implies a one-off Choleski decomposition phase at the outset, prior to the time-step loop. For efficiency, the direct Choleski solution process, necessitates optimized node numbering and bandwidth reduction. For three-dimensional flows, the amount of memory required for the Choleski solver may impose a severe limitation on the size of possible problem to be solved, due to the large number of total nodal unknowns and the associated large bandwidth. This limitation is significantly reduced by appealing to distributed storage for each subdomain problem, and through reordering of node numbers and near-optimal bandwidth reduction [1, 11]. As proposed in Reference [1], for suitable problem and platform configurations, it is practical to recast such large three-dimensional sized problems into sub-problems, for each processor-node of the network.

The semi-implicit Taylor–Galerkin/pressure-correction algorithm was implemented originally in FORTRAN-77. Recently, this algorithm has been restructured and streamlined for efficiency, modifying looping and IF constructs, and incorporating new features of FORTRAN-90, such as direct initialization and assignments, and modular common blocks. By way of example, such changes in the sequential program have led to a speed-up factor of 3.4 in execution time for three-dimensional concentric rotating cylinder flows.

5. PARALLELIZATION STRATEGY

5.1. Hardware configuration

The target hardware platform is composed of subsystems of both homogeneous and heterogeneous type, involving a number of workstations. Two shared-memory computers have been employed, one with three processors and 256 MB memory and another with four processors and 1 GB memory, together with five single processor DEC-alpha workstations with 64 MB memory, running DEC Unix. In addition, we have also used three single processor Intel workstations with 128 MB memory running Solaris Unix, eight 450 MHz AMD-K6-2 single processor with 64 MB memory running Linux and four 950 MHz AMD-K7 single processor with 256 MB memory running Linux. For the homogeneous network, the system configurations offer shared, as well as distributed memory DEC-alpha combinations. For heterogeneous networks of workstations, DEC-alpha, Intel Solaris and AMD-K Linux systems are used. These workstations communicate through a 100 MBit per seconds Ethernet network.

Version 3.4.3 of the PVM system developed at Oak Ridge National Laboratory has been employed. PVM supports programs written in both C/C++ and FORTRAN by providing a library of low-level communication routines [12].

5.2. Domain decomposition

The parallelization strategies and associated test results are applicable to a wide range of CFD applications. The domain decomposition method provides ample potential opportunity for

Copyright © 2002 John Wiley & Sons, Ltd.

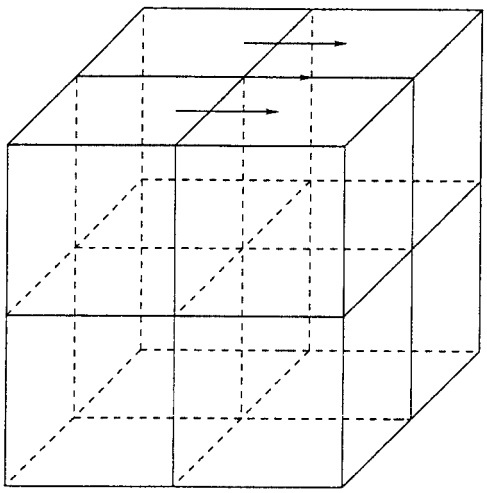

Figure 3. Domain decomposition diagram for planar lid-driven cavity.

parallelization of finite-element methods. In this approach, the domain of interest is partitioned into smaller subdomains of desired size, according to the specification of the available processors. The overall computational load may be equi-partitioned and assigned uniformly among the available processors, resulting in a uniform balancing of computational load (see Figure 3 for the case of eight processors). The interprocessor communication can considerably influence efficiency. In this coarse granularity implementation, each subdomain is assigned to a processor, that computes simultaneously the corresponding subsection of the subdomain. Such a configuration would yield optimal performance when there is no communication amongst the processors.

Load balancing is one of the central issues in parallel computing. At this stage, issues of dynamic load balancing are yet to be investigated. Here, static uniform load distribution is ensured, irrespective of processor speed, using a recursive spectral bisection method [5]. This, with appropriate choice of granularity of parallelism, enables us to handle synchronization of processes, sending and receiving messages, and distributing data efficiently.

Our finite-element algorithm is inherently suitable for parallelization through a variety of paradigms. This is well documented in Reference [9]. Here, we focus on the paradigm of domain decomposition. As our finite-element meshes are structured, adopting a domain decomposition approach, the meshes are partitioned into a number of equal-sized subdomains according to the number of processors available. On each processor, calculations are performed simultaneously for each subdomain over a set of slave processors. On the periphery of each subdomain, shared boundary nodes are computed by a central master (control) processor. The master processor is used to gather the contributions from shared nodes that result from subdomain processes on each processor, and subsequently, redistribute the combined information to each processor. The parallel algorithm (ParTGPC) is illustrated in Table III, across master and slave-processors, accounting for combine-distribute, send-receive, build-solve phases, through the stages per time-step loop.

Copyright © 2002 John Wiley & Sons, Ltd.

Table III. Parallel Taylor–Galerkin algorithm (ParTGPC).

```
Master processor:
     Enter preprocessing information;
     Setup Parallel Virtual Machines;
     Input domain decomposition information free RSB;
     Spawn process on slave processors;
     Input numerical, fluid and algorithm parameters;
     Input mesh information, and initial and boundary conditions;
     Decompose domain and reorder node numbering after band-width reduction,
     and pack all information;
     Distribute information to slave processors after rearranging the
     informations;
     Synchronise the machines and hand-shake with slave processors;
     While not converged
          Stage 1a
              Receive: right-hand-side for stage-1a from each slave processor;
              Redistribute: after combining;
              Loop over Jacobi iteration;
                    Solve stage-1a for interfacing nodes;
          Stage 1b
              Receive: right-hand-side from each slave processor for stage-1b;
              Redistribute: after combining;
              Loop over Jacobi iteration;
                    Solve stage-1b for interfacing nodes;
          Stage 2
              Build: right-hand-side for interfacing nodes on stage-2;
              Solve stage-2 for pressure difference using Choleski on interfacing
              nodes;
          Stage 3
              Receive: right-hand-side from each slave processor for stage-3;
              Redistribute: after combining;
              Loop over Jacobi iteration;
                    Solve stage-3 for interfacing nodes;
     Compute error norm for interfacing nodes;
     Test for convergence;
     Synchronise: the machines and hand-shake with slave processors;
     Receive: results from slave processors;
     Print combine final result;
```

Copyright © 2002 John Wiley & Sons, Ltd.

Table III (*Contd.*)

```
Slave processor:
     Receive: preprocessing information from master processor and unpack all
     information;
     Synchronise: the machines and hand-shake with other processors;
     While not converged

          Stage 1a
            Build: right-hand-side for stage-1a for internal nodes;
            Send: only information of interfacing nodes to master processor;
            Receive: combined information from master processor;
            Loop over Jacobi iteration;
                    Solve stage-1a for internal nodes;

          Stage 1b
            Build: right-hand-side for stage-1b for internal nodes;
            Send: only information of interfacing nodes to master processor;
            Receive: combined information from master processor;
            Loop over Jacobi iteration;
                    Solve stage-1b for internal nodes;

          Stage 2
            Build: right-hand-side for stage-2 for internal nodes;
            Solve stage-2 for pressure difference using Choleski on internal nodes;

          Stage 3
            Build: right-hand-side for stage-3 for internal nodes;
            Send: only information of interfacing nodes to master processor;
            Loop over Jacobi iteration;
                    Solve stage-3 for internal nodes;

     Computer error norm for internal nodes, and information of interfacing nodes
     to master;
     Synchronise: the machines and hand-shake with slave processors;
     Send: result to master processors;
```

5.3. *Speed-up and efficiency*

Numerical computed results are presented for the performance of the parallel TGPC scheme by measuring speed-up, S_n, and efficiency, η_n, defined, respectively, as

$$S_n = \frac{T_s}{T_n}, \qquad \eta_n = \frac{S_n}{n}$$

where T_s is the CPU time in seconds for the sequential algorithm and T_n is the CPU time for the parallel algorithm for n processors. CPU time T_n of parallel computation can be decomposed into computation (T^{comp}) and communication (T^{comm}) time. Timings correspond to total job run-time, inclusive of input–output and communication latency. For a fixed mesh with an increasing number of partitions, the cost of communication increases and this decreases

Copyright © 2002 John Wiley & Sons, Ltd.

Table IV. CPU speed-up and efficiency, 3D lid-driven cavity: 10^3 mesh.

		Newtonian		Viscoelastic	
	Processors	S_n	η_n	S_n	η_n
1	1	1.00	1.00	1.00	1.00
2	2	1.98	0.99	1.99	1.00
3	4	3.99	0.99	3.99	0.99
4	8	7.87	0.98	7.98	0.99

the total efficiency of computation. Whilst, for computation on a fixed number of domains and upon increasing the size of problem (mesh), overall efficiency increases.

6. RESULTS

In this section, for brevity, we present results only for three-dimensional computations, though we point out that these have been cross-checked in symmetry planes against two-dimensional solutions.

6.1. Homogeneous networks

In Table IV, results are displayed for Newtonian and viscoelastic simulations for planar three-dimensional lid-driven, cavity flow on a 10^3 mesh. We have used a single-processor AMD-K6-2 450 MHz Linux workstation for sequential implementation (that lacks the parallel overhead). For parallel implementations, a distributed-memory homogeneous network is established through two, four and eight single-processor AMD-K6-2 450 MHz Linux workstations.

The results in Table IV illustrate linear speed-up with the number of processors and almost a constant efficiency of 1. For three-dimensional Newtonian simulations, a maximum 2% loss of efficiency is observed up to eight processors. This loss of efficiency improves in three-dimensional viscoelastic simulations, where only 1% loss of efficiency is observed for the same network configuration and number of processors.

Figure 4 is included to compare our results against Reference [2] (two flow problems) and Figure 5 against Reference [3] (flow around a cylinder, where the mesh is initially grouped into a coarse submesh of 48 blocks). Parallel strategies and platforms employed in References [2, 3] were identified in Section 1. From these figures it is clear that the observation of others, using PVM on a shared-memory network with eight processors, identifies a loss of efficiency that varies between 10% to above 18%. In contrast, for the present implementation for both three-dimensional Newtonian and viscoelastic flows, masking of communication has been successfully managed (see Reference [1]). Further comparison of these three studies are illustrated in Figures 6 and 7, where speed-up and efficiency of the simulations in References [2, 3] are plotted against the results in Table IV (for viscoelastic flows). The results on efficiency are particularly complimentary to the present study and approach.

For the three-dimensional concentric rotating cylinder flow, we have used a DEC-alpha workstation as a single processor for sequential implementations. For parallel computation, we have constructed a homogeneous network of DEC-alpha processors coupled to a shared-memory resource. As the size of the problem (elements, nodes and degrees of freedom) is

Copyright © 2002 John Wiley & Sons, Ltd.

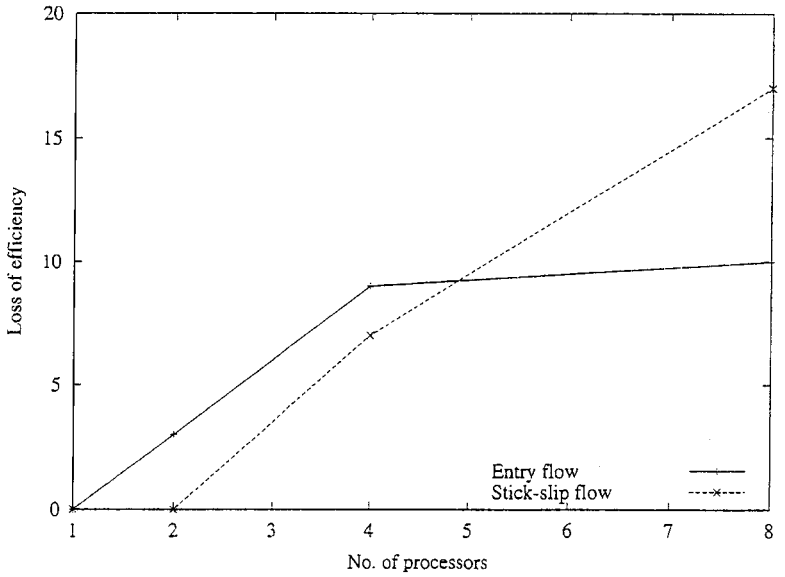

Figure 4. Loss of efficiency [2].

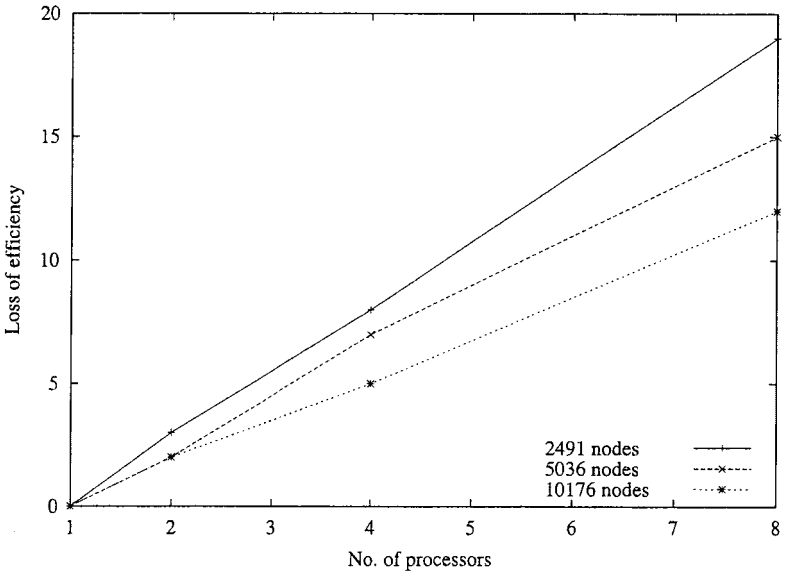

Figure 5. Loss of efficiency [3].

equivalent to the 10^3 three-dimensional lid-driven cavity flow problem, results on speed-up and efficiency are similar. However, due to the use of quadrature points, total computation timing is much higher than that for the planar lid-driven cavity flow problem.

Copyright © 2002 John Wiley & Sons, Ltd.

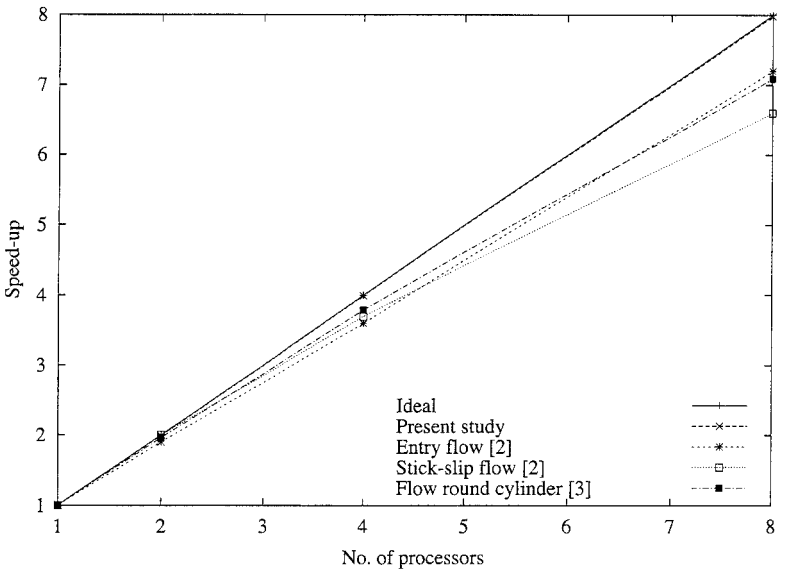

Figure 6. Comparisons of speed-up.

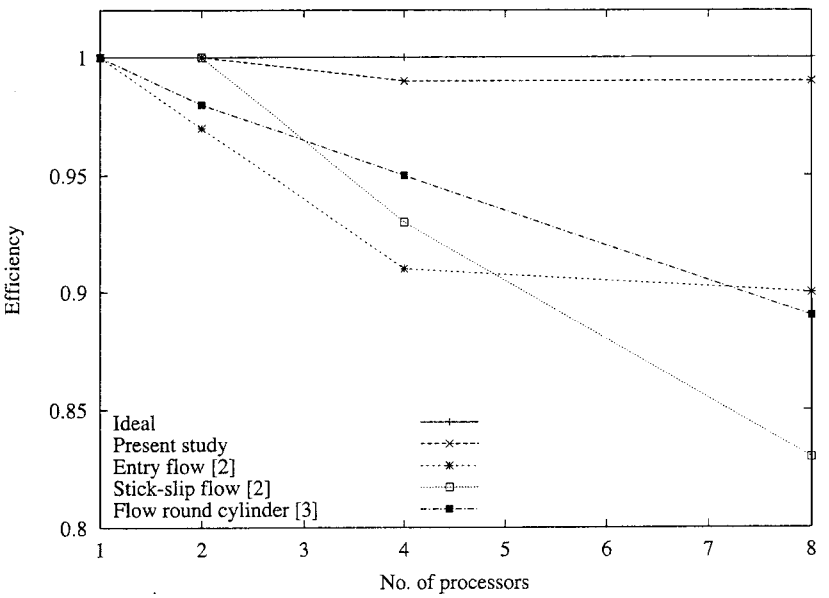

Figure 7. Comparison of efficiency.

6.2. *Heterogeneous networks*

In Table V, results are displayed for Newtonian and viscoelastic simulations for three-dimensional lid-driven cavity flow on a 8^3 mesh. Timings corresponds to total job run-time,

Copyright © 2002 John Wiley & Sons, Ltd.

Table V. CPU time in seconds, three-dimensional lid-driven cavity 8^3 mesh.

	Processors	Newt.	Elastic
1	1 (α)-99% CPU	1434	21 079
2	1 Master (DEC-α) + 2 (α)-50% CPU	1147	20 795
3	2 (α)-90% CPU	713	11 562
4	4 ($2\alpha + 2$Intel)-33% CPU	1089	15 978
5	4 ($2\alpha + 2$Intel)-50% CPU	734	10 681

Table VI. CPU time in seconds.

	Processors	Elements	Nodes	DOF	Time (s)
1	2 (1 DEC-$\alpha + 1$ AMD-K Linux)	4^3	729	6686	1150
2	4 (2 DEC-$\alpha + 2$ AMD-K Linux)	8^3	4913	44946	4497
3	8 (4 DEC-$\alpha + 4$ AMD-K Linux)	10^3	9261	84680	4744

inclusive of input–output and communication latency. A DEC-alpha workstation is used as a single processor for the sequential computation. For parallel computation homogeneous and heterogeneous network combinations are constructed. In Table V (rows 4 and 5), we report on timings for a system comprising of two DEC-alpha and two Intel processors. Other combinations have also been employed, demonstrating variation in the number of DEC-alpha and Intel-type processors. The percentage CPU usage indicates results produced on multi-user systems, and therefore, the share of the processor usage received in each case.

For Newtonian flows, the time taken on the single processor with 99% of CPU resource is almost double that on two processors with 90% CPU. This indicates linear dependency on the number of processors and that communication is being dominated by computation. This speed-up is more significant in the viscoelastic case, where we have additional sets of equations and an increase in the number of degrees of freedom. We observe roughly linear speed-up for homogeneous systems, see Table V (2 and 3). This is also true for (4 and 5), heterogeneous systems. That is, assuming that the Intel (450 MHz) and DEC-alpha (433 MHz) processors are almost comparable on base-speed for the current code application. This is borne out by inspection of performance in contrasting (2) to (4) and (5), even when accounting for the percentage of processor usage.

6.3. Scalability

A parallel algorithm is said to be *scalable* if the computation time remains unchanged when the total number of degrees of freedom is increased simultaneously with an increase in the same ratio, as the number of slave processors.

Various heterogeneous network combinations are constructed to demonstrate scalability. To establish two, four and eight slave processor heterogeneous networks, in each case equal numbers of DEC-alpha and AMD-K7 Linux workstations are taken.

In Table VI, we report timings for systems comprising of the above network clusters. These results are cross-checked on other heterogeneous networks, such as those involving combinations of DEC-alpha and Intel Solaris processors.

Copyright © 2002 John Wiley & Sons, Ltd.

From rows 2 and 3 in this table, we observe that doubling the number of processors and degrees of freedom gives the same order of computation time. Furthermore, this is true if we extrapolate the time from row 1, taking the size of problem to be half that in row 2. This demonstrates the scalability of our parallel algorithm. In parallel computation mode, better scalability depends upon ideal speed-up, whilst speed-up depends on problem size.

7. CONCLUSIONS

We have investigated parallel simulations of Newtonian and viscoelastic fluid flows in planar lid-driven cavity and concentric rotating cylinder problems. A variety of homogeneous and heterogeneous clusters of workstations have been tested, with several versions of the Unix operating system. For the range of problems considered and processor clusters adopted, almost linear speed-up with increasing number of processors has been achieved. Hence, the algorithm described appears ideal for parallelization and ripe for further research.

Our next phase of investigation is to study the behaviour of the algorithm on larger clusterings of processors and to solve actual industrial-scale problems. This shall embrace complex three-dimensional flows and multi-mode viscoelastic computations, which will offer further opportunities for parallelization. There, it is anticipated that, with the increase in complexity of the problem, speed-up will eventually degrade from the ideal linear form. This is to be quantified in the future.

ACKNOWLEDGEMENTS

This research is supported by a grant from the UK EPSRC (GR/L14916). We thank the anonymous referees for comments which led to improvements in the presentation.

REFERENCES

1. Grant PW, Webster MF, Zhang X. Coarse grain parallel finite element simulations for incompressible flows. *International Journal for Numerical Methods in Engineering* 1998; **41**:1321–1337.
2. Keunings R. Parallel finite element algorithms applied to computational rheology. *Computers and Chemical Engineering* 1995; **19**(6–7):647–669.
3. Dou H, Phan-Thien N. Parallelization of an unstructured finite volume implementation with PVM: viscoelastic flow around a cylinder. *Journal of Non-Newtonian Fluid Mechanics* 1998; **77**:21–51.
4. Townsend P, Webster MF. An algorithm for the three-dimensional transient simulation of non-newtonian fluid flows. In *Proceedings of International Conference on Numerical Methods in Engineering, NUMETA*, Nijhoff, Dordrecht, 1987.
5. Carew EO, Townsend P, Webster MF. Taylor–Petrov–Galerkin algorithm for viscoelastic flow. *Journal of Non-Newtonian Fluid Mechanics* 1994; **50**:253–287.
6. Baloch A, Webster MF. A computer simulation of complex flows of fibre suspensions. *Computers and Fluids* 1995; **24**(2):135–151.
7. Baloch A, Townsend P, Webster MF. On the highly elastic flows. *Journal of Non-Newtonian Fluid Mechanics* 1995; **59**:111–128.
8. Matallah H, Townsend P, Webster MF. Recover and stress-splitting schemes for viscoelastic flows. *Journal of Non-Newtonian Fluid Mechanics* 1998; **75**:139–166.
9. Grant PW, Webster MF, Zhang X. Solving computational fluid dynamics problems on unstructured grids with distributed parallel processing. In *Parallel Algorithms for Irregularly Structured Problems*, Ferreria A, Rolim J (eds), Lecture Notes in Computer Science, vol. 980, Springer: Berlin, Lyon, France, September 1995; 187–197.

Copyright © 2002 John Wiley & Sons, Ltd.

10. van Kan J. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM Journal on Science and Statistical Computing* 1986; **7**(3):870–891.
11. Hawken DM, Tamaddon-Jahromi HR, Townsend P, Webster MF. A Taylor–Galerkin-based algorithm for viscous incompressible flow. *International Journal for Numerical Methods in Fluids* 1990; **10**:327–351.
12. Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam V. *PVM: Parallel Virtual Machine. A User Guide and Tutorial for Networked Parallel Computing*. MIT Press: Cambridge, 1994.

Copyright © 2002 John Wiley & Sons, Ltd.