COMPUTATION OF INCOMPRESSIBLE FLOWS WITH IMPLICIT FINITE ELEMENT IMPLEMENTATIONS ON THE CONNECTION MACHINE[†]

M. Behr, A. Johnson
Department of Aerospace Engineering and Mechanics,
Army High Performance Computing Research Center, and Supercomputer Institute
University of Minnesota, 1200 Washington Avenue South, Minneapolis, MN 55415, USA

J. Kennedy
Thinking Machines Corporation, Cambridge, MA 02142, USA

S. Mittal and T. Tezduyar
Department of Aerospace Engineering and Mechanics,
Army High Performance Computing Research Center, and Supercomputer Institute
University of Minnesota, 1200 Washington Avenue South, Minneapolis, MN 55415, USA

Received 27 April 1992
Revised manuscript received 13 January 1993

Abstract

Two implicit finite element formulations for incompressible flows have been implemented on the Connection Machine supercomputers and successfully applied to a set of time-dependent problems. The stabilized space-time formulation for moving boundaries and interfaces, and a new stabilized velocity-pressure-stress formulation are both described, and significant aspects of the implementation of these methods on massively parallel architectures are discussed. Several numerical results for flow problems involving moving as well as fixed cylinders and airfoils are reported. The parallel implementation, taking full advantage of the computational speed of the new generation of supercomputers, is found to be a significant asset in fluid dynamics research. Its current capability to solve large-scale problems, especially when coupled with the potential for growth enjoyed by massively parallel computers, make the implementation a worthwhile enterprise.

1. Introduction

In recent years, it has become apparent that the improvements in computation speed delivered by single processing units are beginning to stagnate. Fundamental physical limitations begin to constrain hardware designers, and the order-of-magnitude improvements we have come to expect every few years have been replaced by more modest incremental gains. These dissatisfactions have been compensated by the promise of highly scalable performance offered by massively parallel (MP) computing. However to take advantage of these new capabilities, programming techniques and algorithm design have to be extensively modified. Because of this, another unexpected challenge has been added to the task of numerical analysts aiming to exploit the latest computer technology.

[†]This research was sponsored by NASA-JSC under grant NAG 9-449, by NSF under grant MSM-8796352, and by ALCOA foundation. Partial support for this work has also come from the Army Research Office contract number DAAL03-89-C-0038 with the Army High Performance Computing Research Center at the University of Minnesota.

The difficulty is especially acute in the case of finite element methods. The inherent flexibility of such methods, which allows one to use unstructured meshes or mix various types of interpolations in the same problem, becomes a drawback when parallelization is considered. Finite element codes often have complex patterns of communications between variables, in stark contrast to the regular communication which is preferred in MP architectures. Nevertheless the efforts to make parallel machines an integral part of finite element universe have been intensifying. A replacement of standard programming techniques with those more suitable for Single-Instruction-Multiple-Data (SIMD) architectures has been proposed by Belytschko et al. [1]. The *exchange* explicit procedure described in [1] bypasses the traditional formation of stiffness matrices and their assembly. Substantial work in the application of explicit and domain-decomposition methods on parallel machines has been done by Farhat et al. — see [2] and references therein. In the context of implicit finite element implementations, the pioneering work has been done by Johnsson and Mathur [3–6]. In their work, storage schemes optimal for the SIMD machines, along with a conjugate gradient solver, were developed and applied to three-dimensional stress analysis. Another alternative to the traditional construction of the stiffness matrix has been proposed by Johan et al. [7], and applied to fluid dynamics problems. The mesh-to-processor mapping techniques discussed there, as well as those developed by Farhat [8], are relevant to most parallel finite element implementations.

Here we describe our experiences in adapting implicit finite element formulations for use with massively parallel computers, restricted so far to SIMD type machines. We concern ourselves with two distinct formulations, but the implementation issues are common to both. By far the most challenging part of the design is the method of solution of the large linear systems of equations arising from the finite element discretization. For this purpose, we adapted an iterative solver based on the Generalized Minimum Residual (GMRES) technique. To improve the convergence of the method, a diagonal scaling is used. On the other hand, the formation of elemental stiffness matrices and force vectors is naturally parallelizeable, due to local nature of all operations involved.

The two formulations used to illustrate the implementation techniques are novel and interesting in their own right. The stabilized space-time formulation for moving boundaries and interfaces, introduced in [9] and [10], allows for simulation of flows in deforming domains with unprecedented ease, and has already been used to add to the physical knowledge base in the field of fluid-structure interaction [11]. The stabilized velocity-pressure-stress formulation introduced in [12] uses the deviatoric stress tensor as an additional unknown, and allows for arbitrary combinations of interpolations for all variables. It is also applicable to viscoelastic fluid flows with complex time-dependent constitutive equations.

In Sections 2 and 3 we briefly outline the finite element formulations adapted for use on the Connection Machine computers. A formal description of the discretization process is given in Section 4, while in Section 5 we discuss in detail the parallel implementation issues, common to both formulations. Selected results are presented in Sections 6 and 7, and main points of the study are reiterated in Section 8.

It is our hope that the material covered here will smooth the transition path for other finite element researchers who wish to take advantage of the massively parallel performance, but find their current code design less than optimal for the task. The parallel implementation section in particular contains a fair amount of detailed information and should be applicable to a wide range of finite element formulations.

2. Space-Time Velocity-Pressure Formulation

We consider a viscous, incompressible fluid occupying at an instant $t \in (0, T)$ a bounded region $\Omega_t \subset \mathbb{R}^{n_{\rm sd}}$, with boundary Γ_t, where $n_{\rm sd}$ is the number of space dimensions. The velocity and

pressure, $\mathbf{u}(\mathbf{x},t)$ and $p(\mathbf{x},t)$, are governed by the Navier-Stokes equations:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u}\right)-\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} = \mathbf{0} \qquad \text{on } \Omega_t \quad \forall t\in(0,T), \tag{1}$$

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0 \qquad \text{on } \Omega_t \quad \forall t\in(0,T), \tag{2}$$

where ρ is the fluid density, and $\boldsymbol{\sigma}$ is the stress tensor. For a fluid with viscosity μ, this tensor can be decomposed into the isotropic and deviatoric parts:

$$\boldsymbol{\sigma}=-p\mathbf{I}+\mathbf{T}, \quad \mathbf{T}=2\mu\boldsymbol{\varepsilon}(\mathbf{u}), \quad \boldsymbol{\varepsilon}(\mathbf{u})=\frac{1}{2}\left(\boldsymbol{\nabla}\mathbf{u}+(\boldsymbol{\nabla}\mathbf{u})^T\right). \tag{3}$$

Both the Dirichlet and Neumann-type boundary conditions are taken into account, represented as:

$$\mathbf{u} = \boldsymbol{g} \qquad \text{on } (\Gamma_t)_{\boldsymbol{g}}, \tag{4}$$

$$\mathbf{n}\cdot\boldsymbol{\sigma} = \boldsymbol{h} \qquad \text{on } (\Gamma_t)_{\boldsymbol{h}}, \tag{5}$$

where $(\Gamma_t)_{\boldsymbol{g}}$ and $(\Gamma_t)_{\boldsymbol{h}}$ are complementary subsets of the boundary Γ_t. The initial condition consists of a divergence-free velocity field specified over the entire domain:

$$\mathbf{u}(\mathbf{x},0)=\mathbf{u}_0 \quad \text{on } \Omega_0. \tag{6}$$

In order to construct the finite element function spaces for the space-time method, we partition the time interval $(0,T)$ into subintervals $I_n=(t_n,t_{n+1})$, where t_n and t_{n+1} belong to an ordered series of time levels $0=t_0<t_1<\cdots<t_N=T$. Let $\Omega_n=\Omega_{t_n}$ and $\Gamma_n=\Gamma_{t_n}$. We shall define the space-time slab Q_n as the domain enclosed by the surfaces Ω_n, Ω_{n+1}, and P_n, where P_n is the surface described by the boundary Γ_t as t traverses I_n. As it is the case with Γ_t, surface P_n can be decomposed into $(P_n)_{\boldsymbol{g}}$ and $(P_n)_{\boldsymbol{h}}$ with respect to the type of boundary condition (Dirichlet or Neumann) being applied. For each space-time slab we define the following finite element interpolation function spaces for the velocity and pressure:

$$(\mathcal{S}^h_{\mathbf{u}})_n = \left\{\mathbf{u}^h \mid \mathbf{u}^h\in\left[H^{1h}(Q_n)\right]^{n_{\mathrm{sd}}}, \mathbf{u}^h\doteq\boldsymbol{g}^h \quad \text{on } (P_n)_{\boldsymbol{g}}\right\}, \tag{7}$$

$$(\mathcal{V}^h_{\mathbf{u}})_n = \left\{\mathbf{u}^h \mid \mathbf{u}^h\in\left[H^{1h}(Q_n)\right]^{n_{\mathrm{sd}}}, \mathbf{u}^h\doteq\mathbf{0} \quad \text{on } (P_n)_{\boldsymbol{g}}\right\}, \tag{8}$$

$$(\mathcal{S}^h_p)_n = (\mathcal{V}^h_p)_n = \left\{p^h \mid p^h\in H^{1h}(Q_n)\right\}. \tag{9}$$

Here $H^{1h}(Q_n)$ represents the finite dimensional function space over the space-time slab Q_n. Over the element domain, this space is formed by using first-order polynomials in space and, depending on our choice, zeroth- or first-order polynomials in time. Globally, the interpolation functions are continuous in space, but discontinuous in time.

The stabilized space-time formulation for deforming domains can be then written as follows: given $(\mathbf{u}^h)^-_n$, find $\mathbf{u}^h\in(\mathcal{S}^h_{\mathbf{u}})_n$ and $p^h\in(\mathcal{S}^h_p)_n$ such that

$$\begin{aligned}
&\int_{Q_n}\mathbf{w}^h\cdot\rho\left(\frac{\partial\mathbf{u}^h}{\partial t}+\mathbf{u}^h\cdot\boldsymbol{\nabla}\mathbf{u}^h\right)dQ+\int_{Q_n}\boldsymbol{\varepsilon}(\mathbf{w}^h):\boldsymbol{\sigma}(p^h,\mathbf{u}^h)dQ-\int_{(P_n)_{\boldsymbol{h}}}\mathbf{w}^h\cdot\boldsymbol{h}^h dP\\
&+\int_{Q_n}q^h\rho\boldsymbol{\nabla}\cdot\mathbf{u}^h dQ+\int_{\Omega_n}(\mathbf{w}^h)^+_n\cdot\rho\left((\mathbf{u}^h)^+_n-(\mathbf{u}^h)^-_n\right)d\Omega\\
&+\sum_{e=1}^{(n_{\mathrm{el}})_n}\int_{Q^e_n}\tau\left[\rho\left(\frac{\partial\mathbf{w}^h}{\partial t}+\mathbf{u}^h\cdot\boldsymbol{\nabla}\mathbf{w}^h\right)-\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}(q^h,\mathbf{w}^h)\right]\cdot\left[\rho\left(\frac{\partial\mathbf{u}^h}{\partial t}+\mathbf{u}^h\cdot\boldsymbol{\nabla}\mathbf{u}^h\right)-\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}(p^h,\mathbf{u}^h)\right]dQ\\
&+\sum_{e=1}^{(n_{\mathrm{el}})_n}\int_{Q^e_n}\delta\boldsymbol{\nabla}\cdot\mathbf{w}^h\,\rho\boldsymbol{\nabla}\cdot\mathbf{u}^h dQ=0, \quad \forall\mathbf{w}^h\in(\mathcal{V}^h_{\mathbf{u}})_n, \quad \forall q^h\in(\mathcal{V}^h_p)_n.
\end{aligned} \tag{10}$$

This process is applied sequentially to all the space-time slabs $Q_1, Q_2, \ldots, Q_{N-1}$. In the variational formulation given by (10), the following notation is being used:

$$(\mathbf{u}^h)_n^{\pm} = \lim_{\varepsilon \to 0} \mathbf{u}(t_n \pm \varepsilon), \tag{11}$$

$$\int_{Q_n} (\ldots) dQ = \int_{I_n} \int_{\Omega_t^h} (\ldots) d\Omega dt, \tag{12}$$

$$\int_{P_n} (\ldots) dP = \int_{I_n} \int_{\Gamma_t^h} (\ldots) d\Gamma dt. \tag{13}$$

The computations start with

$$(\mathbf{u}^h)_0^+ = \mathbf{u}_0. \tag{14}$$

Remarks

1. In the variational formulation given by (10), the first four terms constitute the standard Galerkin formulation of the problem. The fifth term provides continuity, in a weak sense, of the velocity field in time.

2. The sixth term in the formulation (10) is a least-squares form of the momentum equation, which provides necessary stability for advection dominated flows in the presence of sharp streamwise boundary layers. The same term stabilizes the method against spurious pressure modes which arise from certain combinations of interpolations for velocity and pressure, including the equal-order bilinear interpolation used in current computations. See [9] for definition of the stabilization coefficient τ.

3. Stability at high Reynolds numbers is enhanced by the addition of the final term in the formulation (10), which is a least-squares form of the continuity equation. The coefficient δ is defined in [11].

4. Both stabilization terms are of the weighted residual variety, and do not compromise the consistency of the Galerkin formulation.

3. Velocity-Pressure-Stress Formulation

The physical problem under consideration remains the same as the one defined by equations (1)–(6), i.e., Navier-Stokes equations for flows of incompressible Newtonian fluid. However, the $n_{\mathrm{sd}}(n_{\mathrm{sd}}+1)/2$ independent components of the deviatoric stress tensor $\mathbf{T}$ are treated as additional unknowns, and equation (3) enters the variational formulation directly. The case of deforming domains is not covered here, so the subscripts denoting domain time level are dropped. The interpolation function spaces for the velocity, pressure and deviatoric stress tensor are given as:

$$\mathcal{S}_{\mathbf{u}}^h = \left\{ \mathbf{u}^h \mid \mathbf{u}^h \in \left[H^{1h}(\Omega) \right]^{n_{\mathrm{sd}}}, \mathbf{u}^h \doteq \boldsymbol{g}^h \quad \text{on } \Gamma_{\boldsymbol{g}} \right\}, \tag{15}$$

$$\mathcal{V}_{\mathbf{u}}^h = \left\{ \mathbf{u}^h \mid \mathbf{u}^h \in \left[H^{1h}(\Omega) \right]^{n_{\mathrm{sd}}}, \mathbf{u}^h \doteq \mathbf{0} \quad \text{on } \Gamma_{\boldsymbol{g}} \right\}, \tag{16}$$

$$\mathcal{S}_p^h = \mathcal{V}_p^h = \left\{ p^h \mid p^h \in H^{1h}(\Omega) \right\}, \tag{17}$$

$$\mathcal{S}_{\mathbf{T}}^h = \mathcal{V}_{\mathbf{T}}^h = \left\{ \mathbf{T}^h \mid \mathbf{T}^h \in \left[H^{1h}(\Omega) \right]^{n_{\mathrm{sd}}(n_{\mathrm{sd}}+1)/2} \right\}. \tag{18}$$

The velocity-pressure-stress formulation is an extension of Method II described in [12] to time-dependent problems, and can be written as follows: find $\mathbf{u}^h \in \mathcal{S}^h_{\mathbf{u}}$, $p^h \in \mathcal{S}^h_p$ and $\mathbf{T}^h \in \mathcal{S}^h_{\mathbf{T}}$ such that

$$\begin{aligned}
&\int_\Omega \mathbf{w}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{u}^h \right) d\Omega - \int_\Omega \boldsymbol{\nabla} \cdot \mathbf{w}^h p^h d\Omega + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w}^h) : \mathbf{T}^h d\Omega - \int_{\Gamma_h} \mathbf{w}^h \cdot \boldsymbol{h}^h d\Gamma \\
&+ \frac{1}{2\nu} \int_\Omega \mathbf{S}^h : \mathbf{T}^h d\Omega - \int_\Omega \mathbf{S}^h : \boldsymbol{\varepsilon}(\mathbf{u}^h) d\Omega + \int_\Omega q^h \rho \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega \\
&+ \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \tau \left[\rho \left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{w}^h \right) + \boldsymbol{\nabla} q^h - \boldsymbol{\nabla} \cdot \mathbf{S}^h \right] \cdot \left[\rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{u}^h \right) + \boldsymbol{\nabla} p^h - \boldsymbol{\nabla} \cdot \mathbf{T}^h \right] d\Omega \\
&+ \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \alpha 2\nu \left[\frac{1}{2\nu} \mathbf{S}^h - \boldsymbol{\varepsilon}(\mathbf{w}^h) \right] : \left[\frac{1}{2\nu} \mathbf{T}^h - \boldsymbol{\varepsilon}(\mathbf{u}^h) \right] d\Omega \\
&+ \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \delta \boldsymbol{\nabla} \cdot \mathbf{w}^h \rho \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega = 0, \quad \forall \mathbf{w}^h \in \mathcal{V}^h_{\mathbf{u}}, \quad \forall q^h \in \mathcal{V}^h_p, \quad \forall \mathbf{S}^h \in \mathcal{V}^h_{\mathbf{T}}.
\end{aligned} \tag{19}$$

Remarks

5. Aside from stabilization terms described in Section 2, the formulation given by (19) includes a least-squares form of the constitutive equation (3). Consequently, this formulation can be applied in conjunction with arbitrary combinations of interpolation functions for all variables, including presently employed equal-order bilinear combination.

6. Definitions of the coefficients τ, δ and α, as well as stability proof and error analysis for the steady-state case are given in [12].

7. In the computations that follow, formulation (19) has been time-discretized with the Crank-Nicholson scheme. The use of discontinuous Galerkin discretization (space-time method) is planned for future.

8. The time derivative of the velocity weighting function represents the variation of the time derivative of velocity itself. For example, in the case of the space-time method, this term is the true time derivative of the weighting function. On the other hand, in the case of Euler-type time discretization with time step Δt, the term $\partial \mathbf{u}^h / \partial t$ is replaced by $(\mathbf{u}^h_{n+1} - \mathbf{u}^h_n)/\Delta t$, with $\mathbf{u}^h_n$ known, and thus the variation term becomes $\mathbf{w}^h / \Delta t$.

4. Discrete Finite Element Equations

In order to discuss the implementation aspects, we have to describe in more detail the process of forming and solving the linear equation system from the abstract variational forms (10) and (19). We will use as an example the first of the two formulations, i.e., the space-time formulation (10). The description of the parallelization techniques is equally applicable to the velocity-pressure-stress formulation (19) with only minor modifications.

To simplify the notation, we introduce a composite trial solution

$$\bar{\mathbf{u}}^h = \left\{ \bar{u}^h_i \right\}_{i=1,n_{\mathrm{dof}}} = \bar{u}^h_i \bar{\mathbf{e}}_i, \quad \bar{u}^h_i = \begin{cases} u^h_i & 1 \le i \le n_{\mathrm{sd}} \\ p^h & i = n_{\mathrm{sd}} + 1 \end{cases}, \tag{20}$$

where $n_{\rm dof} = n_{\rm sd} + 1$ is the number of degrees of freedom per node. In the presence of Dirichlet boundary conditions, the composite solution is decomposed into its known and unknown parts

$$\bar{\mathbf{u}}^h = \bar{\mathbf{v}}^h + \bar{\boldsymbol{g}}^h, \quad \bar{\mathbf{v}}^h = \bar{v}_i^h \bar{\mathbf{e}}_i, \quad \bar{\boldsymbol{g}}^h = \bar{g}_i^h \bar{\mathbf{e}}_i, \tag{21}$$

with

$$\bar{v}_i^h = \begin{cases} v_i^h & 1 \le i \le n_{\rm sd} \\ p^h & i = n_{\rm sd}+1 \end{cases}, \quad \bar{g}_i^h = \begin{cases} g_i^h & 1 \le i \le n_{\rm sd} \\ 0 & i = n_{\rm sd}+1 \end{cases}. \tag{22}$$

Then the composite solution is represented in terms of basis (shape) functions:

$$\bar{v}_i^h = \sum_{A \in \eta - \eta_{g_i}} N_A d_{iA}, \quad \bar{g}_i^h = \sum_{A \in \eta_{g_i}} N_A g_{iA}, \tag{23}$$

where η and η_{g_i} represent the set of all nodes, and the set of Dirichlet nodes for the degree of freedom i, respectively. Similar representation applies to the composite weighting function:

$$\bar{\mathbf{w}}^h = \bar{w}_i^h \bar{\mathbf{e}}_i, \quad \bar{w}_i^h = \sum_{A \in \eta - \eta_{g_i}} N_A c_{iA}. \tag{24}$$

The variational formulation (10) may be written in abstract form

$$a(\bar{\mathbf{w}}^h, \bar{\mathbf{v}}^h + \bar{\boldsymbol{g}}^h) = (\bar{\mathbf{w}}^h, \bar{\boldsymbol{h}}^h)_{P_n}, \tag{25}$$

where $\bar{\boldsymbol{h}}^h$ is $\boldsymbol{h}^h$ extended to the composite solution space in a manner analogous to $(22)_2$. Because of nonlinearity of the functional $a(\cdot\,,\cdot)$ in some of the components of the second vector argument, we apply an iterative Newton-Raphson technique to obtain corrections to the current value $\bar{\mathbf{v}}^{h*}$:

$$\bar{\mathbf{v}}^h = \bar{\mathbf{v}}^{h*} + \Delta\bar{\mathbf{v}}^h, \quad a(\bar{\mathbf{w}}^h, \bar{\mathbf{v}}^h + \bar{\boldsymbol{g}}^h) = a(\bar{\mathbf{w}}^h, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h) + a_I(\bar{\mathbf{w}}^h, \Delta\bar{\mathbf{v}}^h), \tag{26}$$

where $a_I(\cdot\,,\cdot)$ is bilinear. Note that for a linear problem (e.g., Stokes equation), $a_I(\cdot\,,\cdot) = a(\cdot\,,\cdot)$. From (26) follows a decomposition of the nodal unknown vector introduced in (23):

$$d_{jB} = d_{jB}^* + \Delta d_{jB}, \tag{27}$$

and at each nonlinear iteration step we have to solve the following equation:

$$\begin{aligned} \sum_{j=1}^{n_{\rm dof}} \left(\sum_{B \in \eta - \eta_{g_i}} a_I(N_A \mathbf{e}_i, N_B \mathbf{e}_j) \Delta d_{jB} \right) &= (N_A \mathbf{e}_i, \bar{\boldsymbol{h}}^h)_{P_n} \\ &- a(N_A \mathbf{e}_i, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h), \ A \in \eta - \eta_{g_i}, \ 1 \le i \le n_{\rm dof}. \end{aligned} \tag{28}$$

Equation (28) may be written in a matrix form as follows:

$$\begin{gathered} \mathbf{K}\Delta\mathbf{d} = \mathbf{F}, \quad \mathbf{K} = [K_{PQ}], \quad \Delta\mathbf{d} = \{\Delta d_Q\}, \quad \mathbf{F} = \{F_P\}, \\ K_{PQ} = a_I(N_A \mathbf{e}_i, N_B \mathbf{e}_j), \quad F_P = (N_A \mathbf{e}_i, \bar{\boldsymbol{h}}^h)_{P_n} - a(N_A \mathbf{e}_i, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h), \\ P = \mathtt{ID}(i, A), \quad Q = \mathtt{ID}(j, B), \quad A, B \in \eta - \eta_{g_i}, \quad 1 \le i, j \le n_{\rm dof}, \end{gathered} \tag{29}$$

where ID is the mapping assigning equation numbers to the nodal degrees of freedom. In the standard finite element implementation, the global matrix $\mathbf{K}$ and vector $\mathbf{F}$ are assembled, or can be thought of as being assembled, from the element-level contributions:

$$\begin{gathered}\mathbf{k}^e = \left[k^e_{pq}\right], \quad \mathbf{f}^e = \left\{f^e_p\right\}, \quad 1 \le p, q \le n_{\mathrm{ee}},\\ k^e_{pq} = a_I(N_a\mathbf{e}_i, N_b\mathbf{e}_j)^e, \quad f_p = (N_a\mathbf{e}_i, \bar{\boldsymbol{h}}^h)_{P^e_n} - a(N_a\mathbf{e}_i, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h)^e,\\ p = n_{\mathrm{dof}}(a-1)+i, \quad q = n_{\mathrm{dof}}(b-1)+j, \quad 1 \le a, b \le n_{\mathrm{en}}, \quad 1 \le i, j \le n_{\mathrm{dof}}, \end{gathered} \tag{30}$$

where n_{en} is the number of nodes in a space-time element, $n_{\mathrm{ee}} = n_{\mathrm{dof}} n_{\mathrm{en}}$ is the number of element equations (or degrees of freedom), while N_a and N_b denote the restrictions of the shape functions to the local elemental space-time domains. Similarly the superscript e denotes restriction of the integral forms to the single element domain.

5. Parallel Implementation

Implementation of the finite element equations (29)–(30) on a distributed memory, massively parallel computer entails architecture-specific considerations which are not relevant in the context of conventional shared-memory scalar or vector machines. The following discussion is intended to rely only on abstract features of the hardware and software environment and should be applicable, in principle, to a variety of distributed memory, massively parallel systems. The actual implementation of the numerical examples is performed on Connection Machine models CM-2, CM-200 and CM-5.

Remark

9. The primary software features which this discussion relies upon are the existence of a *data parallel* style language implemented with *virtual processor* constructs along with :SERIAL and :NEWS *layout* directives for arrays (see Appendix A), such as Connection Machine Fortran (CMF, see [13]). Equivalent data layout controls will be available in the High Performance FORTRAN (HPF) standard, now under development. It should be noted that this same algorithm may be reconstructed from other data parallel languages (such as C*, a data parallel extension to ANSI C offered by the Thinking Machines Corporation) or from a *message passing* programming model (message passing requiring much more detailed low-level management on behalf of the programmer).

Before a detailed discussion of the parallel implementation of the finite element matrix formulation (29)–(30), it is useful to outline the organization of the global numerical algorithm as shown in Box 1. Of the ingredients highlighted in Box 1, the discussion here is restricted primarily to the formulation of the array elements in the linearized system (Step 5 in Box 1) and the solution of the corresponding linear system of equations (Step 6 in Box 1). Preprocessing, postprocessing and data mapping strategies (Steps 1, 9 and 2, respectively, in Box 1) are not considered here. Discussions of parallel preprocessing, such as parallel mesh generation techniques, can be found in [14] and [15]. Data mapping strategies for distributing data to processing nodes in distributed memory, massively parallel computers is important from the perspective of optimizing performance. Typically, the objective of this mapping problem is to determine the mapping which distributes data to the processors in a way which minimizes the communication time associated with sending data between processors. An optimum strategy would balance the objectives of maximizing locality of the data and of minimizing data contention, both within the communication network channels and at the interface of the processing node and the network. This optimal mapping problem is

1. Preprocessing (data input, mesh generation, etc.)
2. Mapping of data onto processors
3. Time step loop
4. Nonlinear iteration loop
5. Formation of array elements in the linearized system ("FORM")
6. Solution of the linear system ("SOLVE")
7. End nonlinear iteration loop
8. End time loop
9. Postprocessing (data visualization, etc.)

Box 1: Structure of a Parallel Nonlinear Finite Element Program

considered to be *NP complete* (nondeterministic polynomial time, see [16], Chapter 36). That is, obtaining a true minimum would be intractable. In fact, maximizing locality is itself NP complete. Consequently, current mapping strategies are simply heuristic methods to reduce communication costs. The mapping strategy used does not influence the quality of the solution, but rather the speed at which the solution is obtained.

Remark

10. It should be noted that in some situations, the mathematical properties of the solution scheme, such as the accuracy and stability associated with a given time increment, or the number of nonlinear iteration taken per time step, may be slightly affected by mapping due to the usual interaction between machine precision and the order in which algebraic operations are performed in finite precision arithmetic. This effect however, should be insignificant in all but extremely ill-conditioned problems.

Although the mapping problem is not addressed here, the data structure constructed is such that a preprocessed map of the data onto processors may be included without difficulty. Discussions concerning this mapping problem may be found in [17–20]. One should note that most of the mapping algorithms previously proposed address only one of the above objectives. For example, randomization (see [6]) addresses reducing data contention within network and at the network interface. Spectral bisection (see [17,19,20]) addresses increasing data locality. Johan [20] combines randomization and spectral bisection.

The dominant computational portions of most finite element formulations are the formation of array elements in the linearized system ("FORM" phase) and the solution of the corresponding linear system of equations ("SOLVE" phase). For implicit time integration methods, the solution of the linear system of equations is typically the dominant computational portion, while for explicit time integration methods the formation of array elements in linearized system constitutes the major computational burden. Only implicit time integration is considered here.

A key step in implementing the finite element method (29)–(30) on a distributed memory, massively parallel machine is to construct a data structure which will circumvent unneeded communication of data between processing nodes. Since, for this class of machines, remote memory access (accessing data on a remote processor) is approximately an order of magnitude slower than

is local memory access (accessing data on the local processor), remote memory access should be avoided when possible. A natural and convenient data structure which addresses this issue is one in which two data sets are used, a "FORM" (element) and a "SOLVE" (node) data set. The construction of these data sets along with parallel algorithms for the "FORM" and "SOLVE" phases are discussed below.

"FORM" Phase

First, considering the "FORM" phase, we note that prior to the assembly of the element objects (30) into global objects (29), this phase can be viewed as a collection of computations which are strictly local to each element. That is, to construct the element matrices (30), only data associated with a given element is used to construct that element's local or element matrices. Consequently, a data structure which collects all the element information for element e into a single processor p will enable the formation of the element matrices (30) for element e in processor p with no communication between neighboring processors. Such a data structure may be constructed naturally within data parallel constructs for :SERIAL and :NEWS array axis layouts (see Appendix for a brief discussion of these layout constructs). In particular, consider arrays $f_{\rm el}(:,:)$ and $k_{\rm el}(:,:,:)$ in a pseudocode language with the declaration attributes

REAL $f_{\rm el}(n_{\rm ee\_max}, n_{\rm el})$,

REAL $k_{\rm el}(n_{\rm ee\_max}, n_{\rm ee\_max}, n_{\rm el})$,

where $n_{\rm ee\_max}$ is the maximum number of element degrees of freedom (number of nodes per element times the number of degrees of freedom per node for the element type having the maximum such product, $n_{\rm ee} = n_{\rm dof}\, n_{\rm en}$) and $n_{\rm el}$ is the total number of elements. The REAL statement is to be interpreted as the REAL attribute in FORTRAN. The declaration is completed with the layout directive

CMF$ LAYOUT $f_{\rm el}$(:SERIAL,:NEWS), $k_{\rm el}$(:SERIAL,:SERIAL,:NEWS).

By definition of the *CMF$ LAYOUT* constructs, for a fixed element $i_{\rm el}$, the element level vector $f_{\rm el}(1 : n_{\rm ee\_max}, i_{\rm el})$, of $n_{\rm ee\_max}$ components, resides in the memory of processor $p(i_{\rm el})$. Furthermore, the element level matrix $k_{\rm el}(1 : n_{\rm ee\_max}, 1 : n_{\rm ee\_max}, i_{\rm el})$ of $n^2_{\rm ee\_max}$ components resides in the memory of the same processor $p(i_{\rm el})$. Consequently, since these comments are true for all $i_{\rm el}$, $i_{\rm el} \in [1, n_{\rm el}]$, the element data associated with $\mathbf{f}^e$ in (29) may be stored in $f_{\rm el}(1 : n_{\rm ee\_max}, e)$, while the element data associated with $\mathbf{k}^e$ in (29) may be stored in $k_{\rm el}(1 : n_{\rm ee\_max}, 1 : n_{\rm ee\_max}, e)$, so that $\mathbf{f}^e$ and $\mathbf{k}^e$ reside in the same (virtual) processor for each e, $e \in [1, n_{\rm el}]$.

Remarks

11. The physical processor identified by $p(i_{\rm el})$ is a function of the mapping strategy used in mapping the data set to the machine. For our purposes, we will assume that this mapping is provided to us either by the default mapping issued by the compiler in initializing an array whose :NEWS dimension has extent $n_{\rm el}$, or by a mapping strategy such as mentioned in Step 2 of Box 1.

12. Note that described data structure for the "FORM" phase has redundant storage of nodal values corresponding to nodes which are shared by more than one element. This trade-off between memory and performance is common in a wide variety of parallel algorithms in computational physics. Fortunately, the total memory within contemporary distributed memory, massively parallel systems is substantially larger than that in traditional vector supercomputers so that this redundant use of memory does not become an unacceptable burden.

With this data structure, the "FORM" phase may be carried out in parallel, with no interprocessor communication. To demonstrate this explicitly, consider the first term in the weak form (10), namely

$$\int_{Q_n} \mathbf{w}^h \cdot \rho \frac{\partial \mathbf{u}^h}{\partial t} dQ. \tag{31}$$

The corresponding contribution to the Galerkin form (25) is

$$a_t(\bar{\mathbf{w}}^h, \bar{\mathbf{v}}^h + \bar{\boldsymbol{g}}^h) = \int_{Q_n} \rho \bar{\mathbf{w}}^h \, \mathbf{I}^{\mathbf{uu}} \, \frac{\partial(\bar{\mathbf{v}}^h + \bar{\boldsymbol{g}}^h)}{\partial t} dQ, \tag{32}$$

where

$$\mathbf{I}^{\mathbf{uu}} = \begin{bmatrix} \mathbf{I}_{n_{\mathrm{sd}} \times n_{\mathrm{sd}}} & \mathbf{0}_{n_{\mathrm{sd}} \times 1} \\ \mathbf{0}_{1 \times n_{\mathrm{sd}}} & 0 \end{bmatrix} \tag{33}$$

is used to *filter* out the pressure degree of freedom. The contributions to $a_I(N_A \mathbf{e}_i, N_B \mathbf{e}_j)$ and $a(N_A \mathbf{e}_i, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h)$ in (28) become

$$\begin{aligned} a_{I,t}(N_A \mathbf{e}_i, N_B \mathbf{e}_j) &= \int_{Q_n^e} \rho N_A \frac{\partial N_B}{\partial t} \delta_{ij}^{\mathbf{uu}} dQ^e \,, \quad A, B \in \eta - \eta_{g_i}, \\ a_t(N_A \mathbf{e}_i, \bar{\mathbf{v}}^{h*} + \bar{\boldsymbol{g}}^h) &= \int_{Q_n^e} \rho N_A \frac{\partial N_B}{\partial t} \delta_{ij}^{\mathbf{uu}} (d_{jB}^* + g_{jB}) dQ^e \,, \quad A \in \eta - \eta_{g_i}, \quad B \in \eta, \end{aligned} \tag{34}$$

where $1 \le i, j \le n_{\mathrm{dof}}$ and $\delta_{ij}^{\mathbf{uu}} = \mathbf{I}_{ij}^{\mathbf{uu}}$. Hence, $a_{I,t}(N_A \mathbf{e}_i, N_B \mathbf{e}_j) = 0$ when $i = n_{\mathrm{dof}}$ or $j = n_{\mathrm{dof}}$, consistent with (32). The contribution to $\mathbf{k}^e$ then takes the form

$$k_{pq}^{e,t} = a_{I,t}(N_a \mathbf{e}_i, N_b \mathbf{e}_j)^e \,. \tag{35}$$

That is,

$$\mathbf{k}^{e,t} = \begin{bmatrix} \hat{\mathbf{k}}_{11}^{e,t} & \hat{\mathbf{k}}_{12}^{e,t} & \dots & \hat{\mathbf{k}}_{1 n_{\mathrm{en}}}^{e,t} \\ \hat{\mathbf{k}}_{21}^{e,t} & \hat{\mathbf{k}}_{22}^{e,t} & \dots & \hat{\mathbf{k}}_{2 n_{\mathrm{en}}}^{e,t} \\ \vdots & \vdots & & \vdots \\ \hat{\mathbf{k}}_{n_{\mathrm{en}} 1}^{e,t} & \hat{\mathbf{k}}_{n_{\mathrm{en}} 2}^{e,t} & \dots & \hat{\mathbf{k}}_{n_{\mathrm{en}} n_{\mathrm{en}}}^{e,t} \end{bmatrix}, \tag{36}$$

where

$$\hat{\mathbf{k}}_{ab}^{e,t} = \begin{bmatrix} \int_{Q_n^e} \rho N_a \frac{\partial N_b}{\partial t} dQ^e & 0 & 0 & 0 \\ 0 & \int_{Q_n^e} \rho N_a \frac{\partial N_b}{\partial t} dQ^e & 0 & 0 \\ 0 & 0 & \int_{Q_n^e} \rho N_a \frac{\partial N_b}{\partial t} dQ^e & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad 1 \le a, b \le n_{\mathrm{en}}. \tag{37}$$

In order to express this in pseudocode, numerical integration of a function $\chi(\mathbf{x})$ over the element domain Q_n^e is represented in the form

$$\int_{Q_n^e} \chi(\mathbf{x}) dQ^e = \sum_{l=1}^{n_{\mathrm{int}}} \chi(\boldsymbol{\xi}_l) J_l W_l \,, \tag{38}$$

REAL $k^{e,t}(n_{\text{ee\_max}}, n_{\text{ee\_max}}, n_{\text{el}})$, $f^{e,t}(n_{\text{ee\_max}}, n_{\text{el}})$, $shape(0:4, max(n_{\text{en}}), n_{\text{el}})$
REAL $d(n_{\text{ee\_max}}, n_{\text{el}})$, $\rho(n_{\text{el}})$, $J(n_{\text{el}})$, $W(n_{\text{el}})$
CMF$ LAYOUT $k^{e,t}$(:SERIAL,:SERIAL,:NEWS)
CMF$ LAYOUT $f^{e,t}$(:SERIAL,:NEWS)
CMF$ LAYOUT $shape$(:SERIAL,:SERIAL,:NEWS),
CMF$ LAYOUT d(:SERIAL,:NEWS), ρ(:NEWS)
CMF$ LAYOUT J(:NEWS), W(:NEWS)
for $l = 1 : n_{\text{int}}$
call **shape** {compute $shape(:,:,:)$, $J(:)$, $W(:)$}
call **rho** {compute $\rho(:)$}
for $a, b = 1 : n_{\text{en}}$
for $i, j = 1 : n_{\text{sd}}$
  $p = n_{\text{dof}} * (a-1) + i$
  $q = n_{\text{dof}} * (b-1) + j$
  $k^{e,t}(p,q,:) = k^{e,t}(p,q,:) + shape(0,a,:) * shape(4,b,:) * \rho(:) * J(:) * W(:)$
  $f^{e,t}(p,:) = f^{e,t}(p,:) - shape(0,a,:) * shape(4,b,:) * \rho(:) * d(q,:) * J(:) * W(:)$
end for
end for
end for

Box 2: Pseudocode for $\mathbf{k}^{e,t}$ and $\mathbf{f}^{e,t}$

where n_{int} is the number of quadrature points in Q_n^e, J_l is the determinant of the Jacobian of the isoparametric mapping between the parent domain of the element (i.e., the unit cube with local coordinate system $\boldsymbol{\xi}$) and the physical domain Q_n^e (with coordinate system $\mathbf{x}$), evaluated at the lth quadrature point $\boldsymbol{\xi}_l$, and W_l is the quadrature weight for that point. Hence, defining

$$\begin{aligned} shape(0, a, i_{\text{el}}) &= N_a \quad \text{defined on } Q_n^{i_{\text{el}}}, \\ shape(4, a, i_{\text{el}}) &= \frac{\partial N_a}{\partial t} \quad \text{defined on } Q_n^{i_{\text{el}}}, \end{aligned} \tag{39}$$

the pseudocode for the "FORM" phase contribution of (31) to $\mathbf{k}^e$ and $\mathbf{f}^e$ takes the form shown in Box 2.

In conceptual terms, the "FORM" phase is constructed in such a way, that all the physical information required to form the element arrays within a single element is stored in :SERIAL dimensions for each element, with a :NEWS dimension along the number of elements. The actual format in which this information is stored along :SERIAL dimensions may be arranged according to individual preferences. Note that a serial computer or a shared-memory multiprocessor would interpret these layout constructs simply as comments (they begin with "c") and they would not affect the results of the computation.

"SOLVE" Phase

In the "SOLVE" phase, a parallel implementation of the GMRES(m) method proposed by

Saad and Schultz in [21] is considered. The parallel implementation GMRES(m) is discussed only briefly here. A comprehensive discussion of the implementation of the classical GMRES(m) method within an unstructured finite element framework is provided in [22]. It should be noted that an alternative, matrix-free version of the GMRES(m) method has been implemented on the Connection Machine [7]. For simplicity, the numerical examples in Sections 6 and 7 report results only for diagonal preconditioning, but block diagonal and element-by-element preconditioners have been implemented, and work on a cluster-element-by-element preconditioner is in progress.

Note that the "FORM" data structure is not well suited for the "SOLVE" phase due to the redundancy in the storage of nodal values. As a result, an additional "SOLVE" data structure is introduced which is identical to the traditional assembled data structure in the finite element literature (see, e.g., [23]). A key feature of the GMRES implementation is that, for the system $\mathbf{K}\Delta\mathbf{d} = \mathbf{F}$ in (29), the stiffness matrix $\mathbf{K}$ is stored only on the element level (i.e., in the "FORM" data structure). Only the vectors $\mathbf{d}$, $\mathbf{F}$, and other intermediate vectors of the same dimension, introduced by the GMRES(m) algorithm, are assembled into the "SOLVE" data structure. The dimension and layout of vectors involved in the "SOLVE" phase become

REAL v($n_{\text{dof\_global}}$)

CMF$ LAYOUT v(:NEWS)

where n_{dof_global} is the total number of global degrees of freedom in the problem. Hence, such vectors get distributed across all the processors in the machine (using either the compiler's default mapping or a mapping specified otherwise). The only communication encountered between the "FORM" and "SOLVE" data structures in the GMRES algorithm occurs when $\mathbf{K}$ is referenced; i.e., in constructing the preconditioner and in the *matrix-vector product*. In particular, a matrix-vector product of the form $\mathbf{Kv}$, where $\dim(\mathbf{v}) = \dim(\Delta\mathbf{d})$, is implemented as (1) a *gather* of $\mathbf{v}$ from the "SOLVE" data structure to the "FORM" data structure in the form of element level vectors $v^e(:,:)$ of the same dimension and layout as $f^e(:,:)$ above, (2) an element level, *in-processor* matrix-vector multiplication of $k^e(:,:,:)$ with $v^e(:,:)$ and (3) a *scatter* of $v^e(:,:)$ from the "FORM" data structure to $\mathbf{v}$ in the "SOLVE" data structure. Note that the in-processor matrix-vector multiplication occurs with no communication between processors, and that the scatter operation involves discarding elemental matrix-vector product results corresponding to Dirichlet nodes.

A noteworthy feature of the current GMRES(m) implementation is that the solution of the Hessenberg system is conducted in a serial fashion on a single processor. Alternative schemes may also be constructed. The required dimension of the Krylov space is related to the quality of the preconditioner and the matrix properties. For well-behaved compressible problems, a Krylov space of dimension 5 to 10 is typically adequate. For incompressible problems, a Krylov space of dimension 20 or more may be needed due to the large condition number of the matrix operator associated with the incompressibility constraint. It is interesting to note that, as observed on the Connection Machine model CM-5, for a problem with 130,000 degrees of freedom, for a Krylov space of dimension 20, the amount of CPU time spent idle by the system in waiting for solution of the Hessenberg system is below 3% of the entire GMRES elapsed time. That is, even with a serial solution of the Hessenberg system, the Connection Machine is still utilized 97% efficiently in the GMRES solver. This number is only slightly larger for other sizes of Krylov space (up to 60). Krylov spaces dramatically larger than this will likely require more sophisticated methods of solution of the Hessenberg system, which would take advantage of the parallelization.

Gather and Scatter

The gather and scatter communication steps in the matrix-vector product are the dominant contributors to computation time in the "SOLVE" phase. These communication steps use a mesh

specific connectivity array $iconn(1{:}n_{\text{ee\_max}},1{:}n_{\text{el}})$ derived from `ID` used in Section 4. For each elemental degree of freedom, this matrix returns either a global equation number, or zero if the particular elemental degree of freedom is prescribed by the boundary conditions. A gather operation can be expressed using the FORALL statement in the form:

$$\text{FORALL } (i_{\text{ee}}{=}1{:}n_{\text{ee\_max}},\ i_{\text{el}}{=}1{:}n_{\text{el}},\ iconn(i_{\text{ee}},i_{\text{el}}).\text{NE}.0)$$
$$v^e(i_{\text{ee}},i_{\text{el}}) = v(iconn(i_{\text{ee}},i_{\text{el}}))$$

and a scatter in the form:

$$\text{FORALL } (i_{\text{dof\_global}}{=}1{:}n_{\text{dof\_global}})$$
$$v(i_{\text{dof\_global}}) = v(i_{\text{dof\_global}}) + \text{SUM}(v^e, \text{MASK} = iconn.\text{EQ}.i_{\text{dof\_global}}).$$

The FORALL statement, present in Connection Machine Fortran and High Performance Fortran, is a parallel counterpart of a serial loop. Although the above expressions provide the desired functionality, the computational expense associated with these communication steps demands the use of high performance gather and scatter functions. On the Connection Machine, the `sparse_util_scatter` and `sparse_util_gather` functions from the Connection Machine Scientific Software Library (CMSSL) [24] are used. Randomization of the global equation numbers, i.e., the *iconn* array, is used to reduce path contention.

6. Numerical Examples for the Space-Time Velocity-Pressure Formulation

Flow past an oscillating airfoil

The stabilized space-time velocity-pressure implementation was used to simulate flow past a pitching NACA 0012 airfoil, at Reynolds number 1,000 based on the unit free-stream velocity and a unit chord length. In this simulation, a sinusoidal oscillation between the angles of attack of 10 and 30 degrees is prescribed for the airfoil, with a non-dimensional frequency of 1.0. The time step size is 0.02. The space-time formulation allows for the deformation of the mesh according to the airfoil inclination. Therefore the deformation is periodic and bounded, and the desired deformation could be achieved without remeshing. The mesh generator in this case provides a flexible zone surrounding the rigid mesh core near the airfoil, and is capable of generating meshes at wide ranges of angle of attack, with no significant element distortion. The mesh used in this problem consists of 6,460 spatial elements, and lead to approximately 39,000 unknowns in a single space-time slab. The resulting equation system is solved iteratively, with a Krylov subspace size of 60, with 7 GMRES iterations and an average of 3 nonlinear iterations per time step. In Figure 1 the vorticity field is shown at several different instants in the simulation. The same figure shows also two typical mesh deformations. There is a good qualitative agreement between the results obtained here and those reported for a similar airfoil motion in [25] and [26].

Vortex-induced vibrations of a cylinder in cross-flow direction

This problem represents a simple model of fluid-structure interaction. A cylinder mounted on lightly damped springs is allowed to move in the vertical direction in response to the fluid forces acting on it. The motion of the cylinder alters the vortex shedding mechanism of the cylinder significantly and leads to several interesting physical phenomena. A detailed numerical investigation of such an oscillator can be found in [11]. The motion of the cylinder is governed by the following equation:

$$\frac{\partial^2 Y}{\partial t^2} + 2\pi F_n \zeta \frac{\partial Y}{\partial t} + (\pi F_n)^2 Y = \frac{C_L}{M}. \tag{40}$$

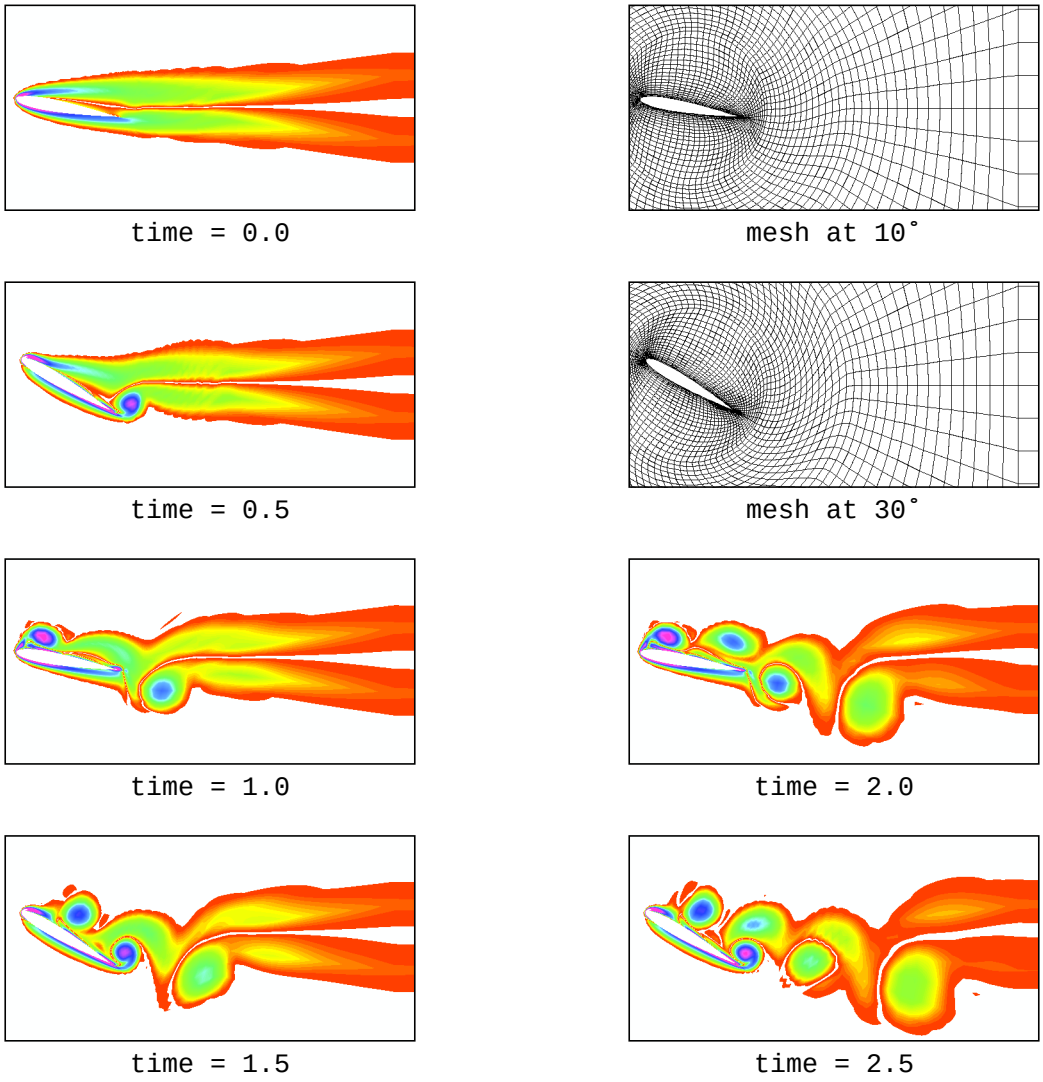

Figure 1. Flow past an oscillating airfoil at Reynolds number 1,000: vorticity fields and typical mesh deformations.

Here Y represents the normalized vertical displacement of the cylinder. The displacement and the velocity of the cylinder are normalized by its radius and the free-stream velocity respectively. M is the non-dimensional mass/unit length of the cylinder, ζ is the structural damping coefficient associated with the system, and C_L denotes the lift coefficient for the cylinder. F_n is the reduced natural frequency of the spring-mass system and is defined as:

$$F_n = \frac{2f_n a}{U}, \tag{41}$$

where a is the radius of the cylinder, U is the free stream velocity and f_n is the actual natural frequency of the system.

For our problem, $F_n = 0.22$, $M = 472.74$ and $\zeta = 3.3 \times 10^{-4}$. The Reynolds number for the simulation (based on the free-stream velocity and the cylinder diameter) is 300. For this value of the Reynolds number the reduced natural frequency of the spring-mass system is larger than the Strouhal number for flow past a fixed cylinder. Unsteady flow past a fixed cylinder at the

same Reynolds number is used as the initial condition for this simulation. The time step for the computations is 1.0. The finite element mesh consists of 4209 nodes and 4060 elements. At each time step approximately 25,000 equations are solved using GMRES in conjunction with a diagonal preconditioner. The dimension of the Krylov space used is 30.

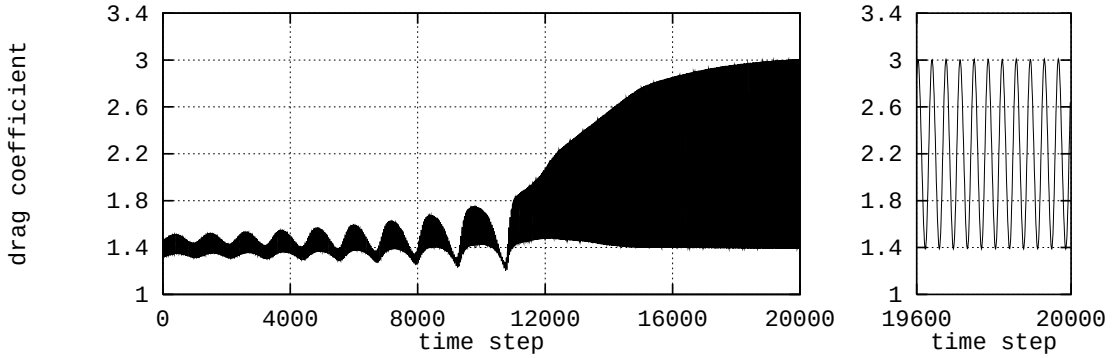

Figure 2. Vertically oscillating cylinder at Reynolds number 300: time history of the drag coefficient.

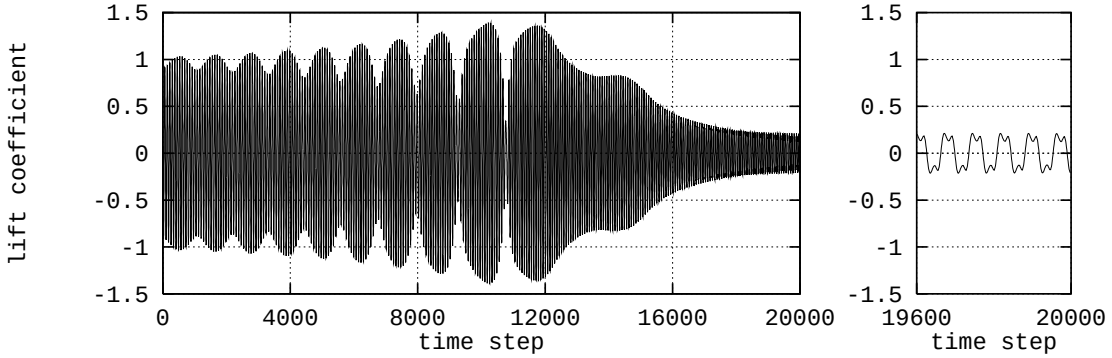

Figure 3. Vertically oscillating cylinder at Reynolds number 300: time history of the lift coefficient.

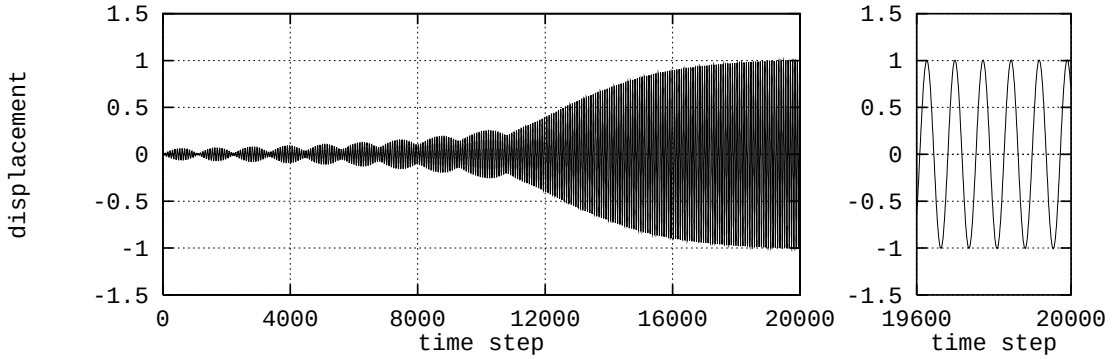

Figure 4. Vertically oscillating cylinder at Reynolds number 300: time history of the normalized vertical displacement.

Figures 2–4 show, respectively, the global and terminal time histories of the drag and lift coefficients and the normalized vertical displacement of the cylinder. It can be observed that, initially the oscillator exhibits the phenomenon of beats. At a later time, the vortex shedding

frequency of the cylinder locks on to the natural frequency of the spring-mass system. Finally the cylinder reaches a periodic oscillation amplitude of approximately one radius. As a result of these oscillations, the drag and torque acting on the cylinder increase substantially, while there is a decrease in the amplitude of the lift acting on the cylinder. Figure 5 shows a sequence of frames

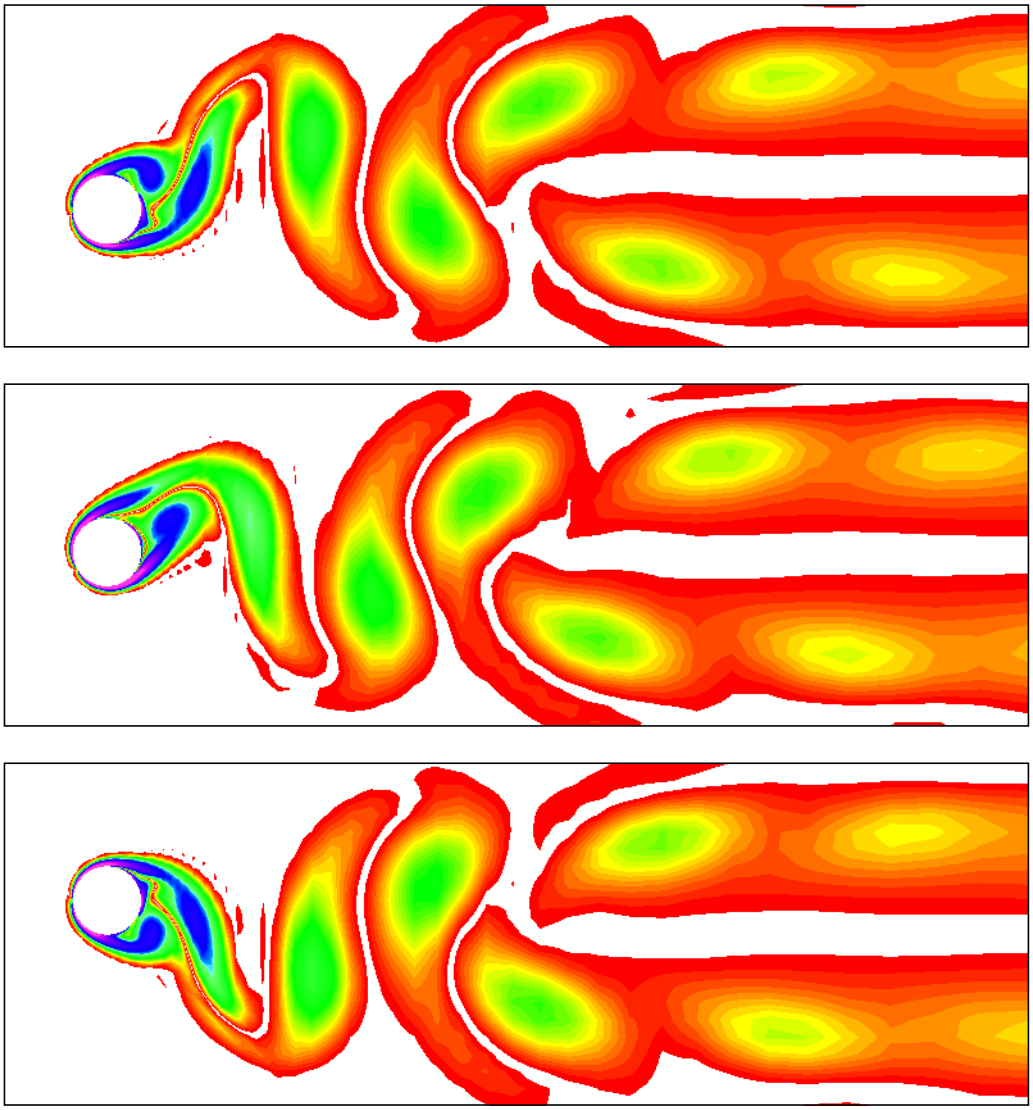

Figure 5. Vertically oscillating cylinder at Reynolds number 300: vorticity field at the lowest, mean and highest positions of the cylinder.

displaying the vorticity field during one period of the cylinder motion. The first and last frames correspond, respectively, to the lowest and highest positions of the cylinder while the middle frame corresponds to the mean location of the cylinder.

7. Numerical Examples for the Velocity-Pressure-Stress Formulation

The velocity-pressure-stress implementation is used to compute two-dimensional flow past a fixed cylinder, at Reynolds number 1000. The upper and lower boundaries are flow symmetry lines, and the downstream boundary is traction-free. The mesh consists of 21,408 quadrilateral elements, with continuous bilinear interpolation functions for all variables. The element size near the cylinder surface is of the order 0.01. A time step size of 0.05 was selected to provide sufficient resolution of the vortex shedding periods. A diagonal scaling with no preconditioning was applied to the system. The Krylov subspace size is 20, with 5 outer GMRES iterations, and at most 4 nonlinear iterations

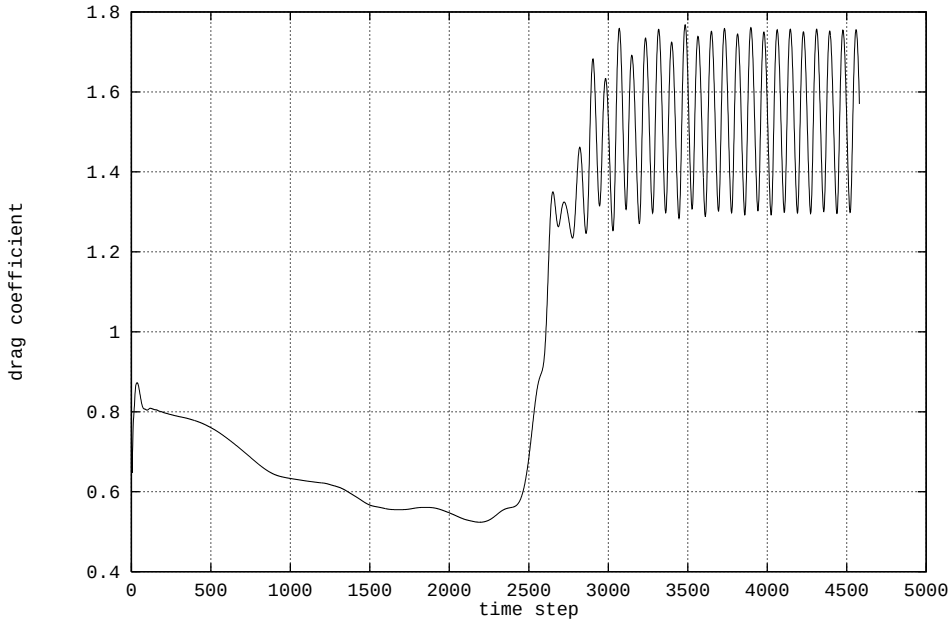

Figure 6. Flow past a cylinder at Reynolds number 1000: time history of the drag coefficient.

per time step. The simulation was continued for 4580 time steps after restart from a steady-state solution at Reynolds number 100, and reached a periodic state with Strouhal number 0.241, as seen in the lift and drag coefficient plots in Figures 6 and 7. Three different vorticity fields are shown in Figure 8. The three frames represent the fully developed periodic flow field, at an instant when the lift coefficient attains the maximum, zero and minimum values, respectively.

Benchmark measurements performed for this problem, revealed the total computation speed of 561 MFLOPS on CM-200 computer with 32,768 processors, using a 4-CPU Sun 4/690 front end. This figure includes parallel output of data to the DataVault mass storage system. In the "SOLVE" phase, the speed of communication-bound GMRES solver routine applied to the system with 129,610 degrees of freedom was 518 MFLOPS. On the other hand, the highly parallel "FORM" phase achieved 1607 MFLOPS on the same machine. All computations were performed in double (64-bit) precision. The time needed to compute one nonlinear iteration for this problem was 6.5 seconds, compared to 150.6 seconds consumed for the same purpose on a Cray Y-MP M-92 CPU. The Cray implementation, although vectorized, likely was not optimal with respect to memory access in the gather and scatter stages of the GMRES algorithm.

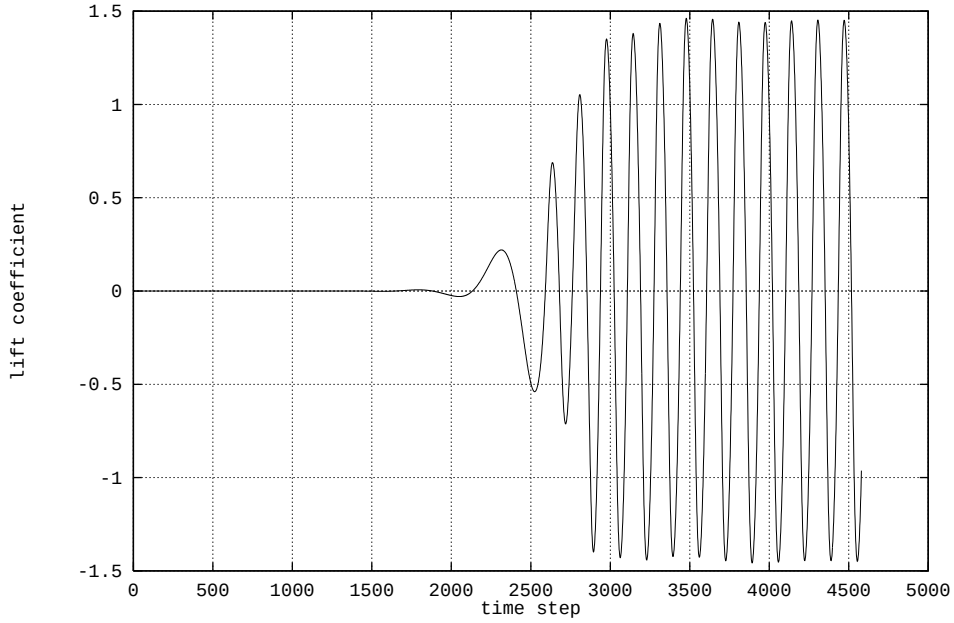

Figure 7. Flow past a cylinder at Reynolds number 1000: time history of the lift coefficient.

8. Summary and Concluding Remarks

We have briefly described two finite element formulations for incompressible flows, and given detailed account of the issues involved in adapting them for use on massively parallel computers. The stabilized space-time method which is applicable to problems involving moving boundaries and interfaces was used to simulate incompressible flows past oscillating bodies such as cylinders and airfoils. The velocity-pressure-stress formulation was applied to a fixed cylinder problem. The two implementations share a parallel design, which is a major focus of this work.

The parallel program design proposed here uses traditional notions of element level matrices common to implicit finite element implementations. Consequently, it provides a straightforward extension to traditional implicit implementations. A useful alternative to using element level matrices is the *matrix-free* GMRES implementation suggested by Johan [20]. The key advantage of the matrix-free GMRES lies in the reduced memory consumption, as the element matrices need not be stored. A disadvantage of the matrix-free GMRES is the need to frequently recompute the residual of the problem in the inner iteration loop of the GMRES algorithm. Such residual formations are used in place of the matrix-vector product in the matrix-based GMRES solver. For classical continuum models, such as the Navier-Stokes equations considered here, 1-point spatial quadrature typically leads to residual formation cost lower than that of the matrix-vector product. On the other hand, higher order integration, e.g., $2 \times 2 \times 2$ or $3 \times 3 \times 3$ quadrature for three-dimensional brick elements, leads to high residual formation costs, which exceed the expense associated with the matrix-vector product. For more elaborate material models such as those entailing micromechanical models for inelasticity in solids (discrete defect models, polycrystalline models, etc.), the residual formation rapidly becomes more expensive than the matrix-vector product, even for 1-point quadrature. Another advantage of implementations employing classical element-level matrices is the fact that the information contained in these matrices can be put to other uses, such as construction of certain preconditioners.

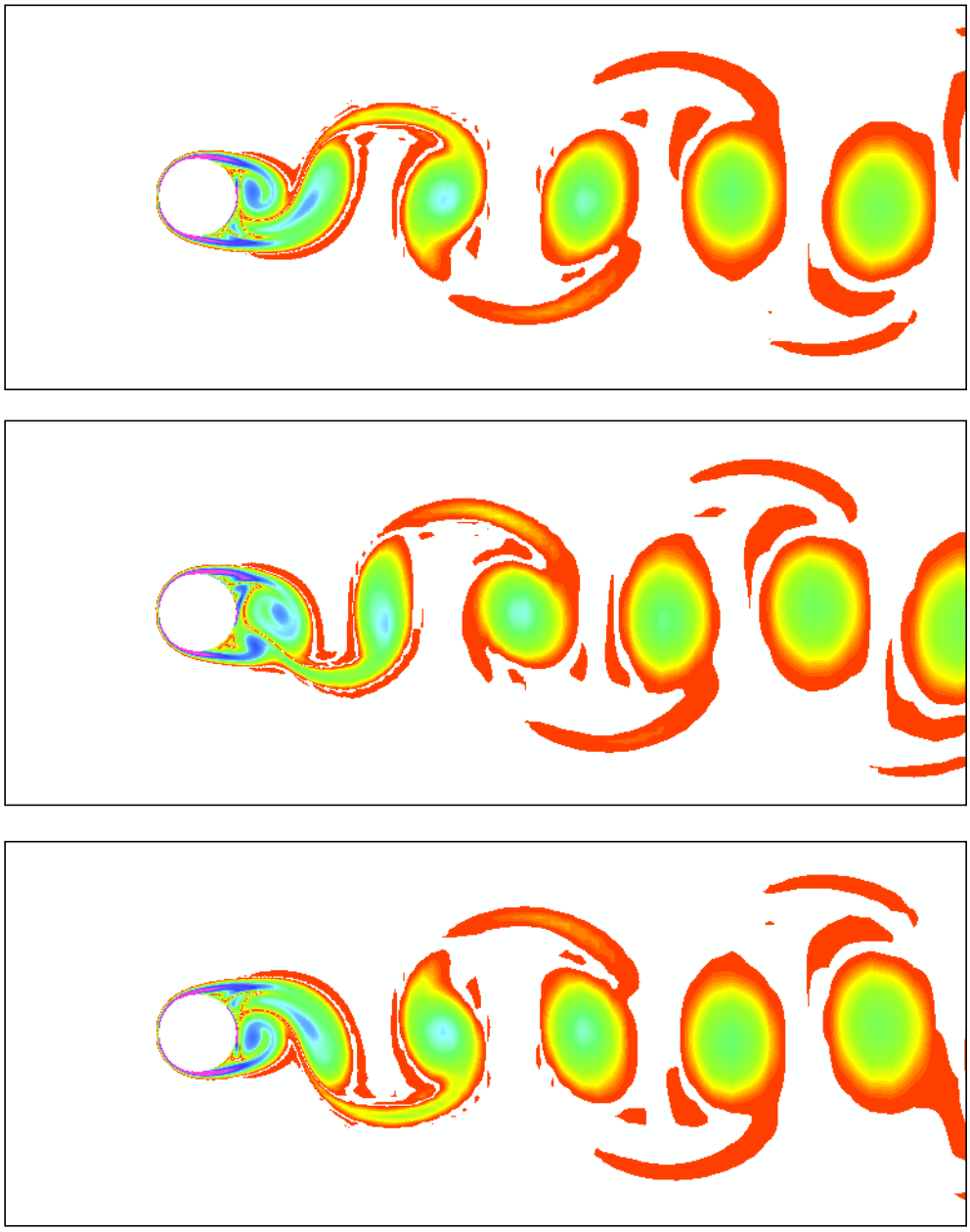

Figure 8. Flow past a cylinder at Reynolds number 1000: vorticity field at time steps 4472, 4518 and 4554

A key ingredient in the proposed parallel implementation is the interprocessor communication associated with the gather and scatter steps of the "SOLVE" phase. The unstructured nature of finite element meshes leads to general communication patterns across the parallel network. Despite the careful choice of data structures which exploit locality through compiler layout directives, the gather and scatter steps consume the dominant portion of the computation time for the implicit simulations, exceeding the time spent in on-processor computation. The dominance of the communication expenses creates an added dimension to algorithmic design considerations for parallel architectures. Although the FORALL construct (in both Connection Machine Fortran and High Performance Fortran) provides functionality for the gather and scatter operations, it is essential to

take advantage of high performance gather and scatter communication routines such as the Connection Machine Scientific Software Library functions discussed in Section 5. Further improvements to reducing communication expenses can be achieved through effective partitioning algorithms for mapping the data to the processors (Step 2 of Box 1). Spectral partitioning algorithms [17, 20] have shown particularly promising results, particularly when combined with communication functions which provide efficient treatment of both on-processor and off-processor components of the gathered and scattered data. A key observation relative to such partitioning algorithms is that the partitions chosen can be imposed upon the data structures here by a simple permutation of the :NEWS axes of the connectivity array

FORALL (i_{ee}=1:$n_{\mathrm{ee\_max}}$,i_{el}=1,n_{el})

$$iconn(i_{\mathrm{ee}}, iperm(i_{\mathrm{el}})) = iconn(i_{\mathrm{ee}}, i_{\mathrm{el}}),$$

where *iperm* is a permutation array describing element numbers associated with particular processors (see [20]).

By making efficient use of emerging high-performance distributed-memory computers, we are able to routinely solve problems of larger scale, spatial as well as temporal, than the ones we could afford with the state-of-the-art classical architectures. Such increased capabilities more than compensate for the initial investment involved in the transformation of existing serial algorithms.

Acknowledgment

We wish to thank S.L. Johnsson and K. Mathur of Thinking Machines Corporation for their advice and the CMSSL library routines used extensively in our finite element implementation. We would also like to thank J.P. Brunet (TMC), Z. Johan (Stanford University) and A. Raefsky (CENTRIC Engineering Systems) for valuable discussions with the TMC author.

Appendix A. Data Parallel :SERIAL and :NEWS Layout Constructs

The notion of data layout constructs for :SERIAL and :NEWS array dimensions is one of the features of the Connection Machine Fortran. This notion is designed to allow the programmer flexibility in distributing data to processors on massively parallel architectures. In CMF, the declaration of such dimensions appear as compiler directives of the form "CMF$ LAYOUT A(:SERIAL,:NEWS)" which typically are included following the array declaration statements in FORTRAN. Similar compiler directives will appear in High Performance Fortran. The directives are constructed in such a way that they will not alter the results of any computations and they will not have any effect on serial machines since they will appear simply as comment lines to FORTRAN compilers which do not recognize them. The layout directives should be interpreted simply as additional constructs which aid performance on distributed memory, massively parallel platforms.

To demonstrate the effect of these layout directives on the location in memory of data represented by arrays, it is useful to consider the arrays A, B and C defined as follows, assuming a 3 processor machine.

```
INTEGER A(9), B(9), C(3,3)
CMF$ LAYOUT A(:NEWS), B(:SERIAL), C(:SERIAL,:NEWS)
```

with

```
A(1:9) = (1,2,3,4,5,6,7,8,9)
```

B(1:9) = (1,2,3,4,5,6,7,8,9)

C(1,1:3) = (1,2,3)

C(2,1:3) = (4,5,6)

C(3,1:3) = (7,8,9)

The manner in which A, B and C are distributed to the processors in the 3 processor machine is depicted in Table 1.

It should be noted that the notion of *virtual processors* is assumed throughout this discussion. A virtual processor facility allows each physical processor to simulate some number of virtual processors, thus allowing application programs to be relatively independent of the number of physical processors in the machine in which it is executing. Arrays are then allocated in such a way that each :NEWS index is associated with a single virtual processor. Operations on such arrays can be performed on all elements in parallel. For example, the operation A(:) = 3*A(:) + 4*A(:) denotes multiplying all elements of A by 3 (simultaneously, in parallel), then by 4 (in parallel) and summing the result (in parallel). The repeated execution of a physical processor over the virtual processors it represents is called virtual processor, or subgrid, looping. For example, for array A, the *virtual processor ratio*, or subgrid, i.e., the ratio of virtual processors to physical processors, is 3 so that the above addition operation requires about 3 times longer to execute than it would if there were 9 physical processors available. Furthermore, in storing A in memory, each virtual processor has a fraction (1/3) of the memory available to a physical processor. If the processors are *pipelined*, the time required to perform an operation may increase with the virtual processor ratio, so application programs may achieve better performance at higher rather than lower virtual processor ratios.

References

[1] T. Belytschko, E.J. Plaskacz, J.M. Kennedy, and D.L. Greenwell, "Finite element analysis on the Connection Machine", *Computer Methods in Applied Mechanics and Engineering*, **81** (1990) 229–254.

[2] C. Farhat, L. Fezoui, and S. Lanteri, "Mixed finite volume / finite element massively parallel computations: Euler flows, unstructured grids, and upwind computations", in P. Mehrotra, J. Saltz, and R. Voigt, editors, *Unstructured Scientific Computation on Scalable Multiprocessors*. MIT Press, (1992) 253–283.

Array	**Processors**			**Control Processor**
	P=1	P=2	P=3	
A(1:3)	1	2	3	—
A(4:6)	4	5	6	—
A(7:9)	7	8	9	—
B(1:3)	—	—	—	1,2,3
B(4:6)	—	—	—	4,5,6
B(7:9)	—	—	—	7,8,9
C(1,1:3)	1	2	3	—
C(2,1:3)	4	5	6	—
C(3,1:3)	7	8	9	—

Table 1. Data layout for arrays A, B, C for a three-processor machine

[3] S.L. Johnsson and K.K. Mathur, "Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer", *International Journal for Numerical Methods in Engineering*, **27** (1989) 523–546.

[4] S.L. Johnsson and K.K. Mathur, "Data structures and algorithms for the finite element method on a data parallel supercomputer", *International Journal for Numerical Methods in Engineering*, **29** (1990) 881–908.

[5] K.K. Mathur and S.L. Johnsson, "The finite element method on a data parallel computing system", *International Journal of High Speed Computing*, **1** (1989) 29–44.

[6] K.K. Mathur, "On the use of randomized address maps in unstructured three-dimensional finite element simulations", Technical Report TMC-37/CS90-4, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, 1990.

[7] Z. Johan, T.J.R. Hughes, K.K. Mathur, and S.L. Johnsson, "A data parallel finite element method for computational fluid dynamics on the Connection Machine system", *Computer Methods in Applied Mechanics and Engineering*, **99** (1992) 113–134.

[8] C. Farhat, "On the mapping of massively parallel processors onto finite element graphs", *Computers & Structures*, **32** (1989) 347–354.

[9] T.E. Tezduyar, M. Behr, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain/space-time procedure: I. The concept and the preliminary tests", *Computer Methods in Applied Mechanics and Engineering*, **94** (1992) 339–351.

[10] T.E. Tezduyar, M. Behr, S. Mittal, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain/space-time procedure: II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders", *Computer Methods in Applied Mechanics and Engineering*, **94** (1992) 353–371.

[11] S. Mittal and T.E. Tezduyar, "A finite element study of incompressible flows past oscillating cylinders and airfoils", *International Journal for Numerical Methods in Fluids*, **15** (1992) 1073–1118.

[12] M. Behr, L.P. Franca, and T.E. Tezduyar, "Stabilized finite element methods for the velocity-pressure-stress formulation of incompressible flows", *Computer Methods in Applied Mechanics and Engineering*, **104** (1993) 31–48.

[13] Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, *CM Fortran Programming Guide*, 1991.

[14] R. Lohner, J. Camberos, and M. Merriam, "Parallel unstructured grid generation", in *Proceedings of the 10th Computational Fluid Dynamics Conference*, Honolulu, Hawaii, (1991), AIAA Paper 91-1582-CP.

[15] J.F. Dannenhoffer, R. Haimes, and M.B. Giles, "Data compression through the use of grid adaptation techniques", in *Proceedings of the 10th Computational Fluid Dynamics Conference*, Honolulu, Hawaii, (1991) 983–984.

[16] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. McGraw-Hill, New York, 1990.

[17] A. Pothen, H.D. Simon, and K.P. Liou, "Partitioning sparse matrices with eigenvectors of graphs", *SIAM Journal on Matrix Analysis and Applications*, **11** (1990) 430–452.

[18] R. Schreiber and S. Hammond, "Mapping unstructured grid problems to the Connection Machine", RIACS Technical Report TR90.22, NASA Ames Research Center, Moffett Field, CA 94035, October 1990.

[19] A. Pothen, H.D. Simon, and L. Wang, "Spectral nested dissection", Technical Report RNR-92-003, NASA Ames Research Center, Moffett Field, CA 94035, January 1992.

[20] Z. Johan, *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*, Ph.D. thesis, Department of Mechanical Engineering, Stanford University, 1992.

[21] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM Journal of Scientific and Statistical Computing*, **7** (1986) 856–869.

[22] J.G. Kennedy and M. Behr, "A data parallel implementation of GMRES(m)", In preparation, 1992.

[23] T.J.R Hughes, *The Finte Element Method. Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[24] Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, *CMSSL Release Notes - Version 2.2*, June 1991.

[25] I.H. Tuncer, J.C. Wu, and C.M. Wang, "Theoretical and numerical studies of oscillating airfoils", *AIAA Journal*, **28** (1990) 1615–1624.

[26] K. Ohmi, M. Coutanceau, O. Daube, and T.P. Loc, "Further experiments on vortex formation around an oscillating and translating airfoil at large incidences", *Journal of Fluid Mechanics*, **225** (1991) 607–630.