Originals Computational Mechanics 18 (1996) 397–412 © Springer-Verlag 1996

Flow simulation and high performance computing

T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, M. Litke

Abstract Flow simulation is a computational tool for exploring science and technology involving flow applications. It can provide cost-effective alternatives or complements to laboratory experiments, field tests and prototyping. Flow simulation relies heavily on high performance computing (HPC). We view HPC as having two major components. One is advanced algorithms capable of accurately simulating complex, real-world problems. The other is advanced computer hardware and networking with sufficient power, memory and bandwidth to execute those simulations. While HPC enables flow simulation, flow simulation motivates development of novel HPC techniques. This paper focuses on demonstrating that flow simulation has come a long way and is being applied to many complex, real-world problems in different fields of engineering and applied sciences, particularly in aerospace engineering and applied fluid mechanics. Flow simulation has come a long way because HPC has come a long way. This paper also provides a brief review of some of the recently-developed HPC methods and tools that has played a major role in bringing flow simulation where it is today. A number of 3D flow simulations are presented in this paper as examples of the level of computational capability reached with recent HPC methods and hardware. These examples are, flow around a fighter aircraft, flow around two trains passing in a tunnel, large ram-air parachutes, flow over hydraulic structures, contaminant dispersion in a model subway station, airflow past an automobile, multiple spheres falling in a liquid-filled tube, and dynamics of a paratrooper jumping from a cargo aircraft.

Communicated by S. N. Atluri, 13 May 1996

T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, M. Litke, Aerospace Engineering and Mechanics, Army HPC Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415, USA

Correspondence to: T. Tezduyar

Sponsored by ARO, ARPA, NASA-JSC, and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/ contract number DAAH04-95-C-0008. The content does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. CRAY C90 time and support for the second author was provided in part by the Minnesota Supercomputer Institute.

Dedicated to the 10th anniversary of Computational Mechanics

1 Introduction

In recent years, the power and diversity of flow simulation as a tool for exploring science and technology involving flow applications have reached new levels. This offers a promising future for providing cost-effective alternatives or complements to laboratory experiments, field tests and prototyping. This new power of flow simulation is based mainly on recent developments in high performance computing (HPC). Both of what we see as the two major components of HPC have contributed to this. These components are: advanced algorithms capable of accurately simulating complex, real-world problems; and advanced computer hardware and networking with sufficient power, memory and bandwidth to execute those simulations. It is also true that there is a two-way exchange between flow simulation and HPC; while HPC enables flow simulation, flow simulation motivates development of novel HPC techniques. In this paper, we show that flow simulation has come a long way and to a point of being a practical tool for complex, real-world problems in different fields of engineering and applied sciences, in our case, aerospace engineering and applied fluid mechanics. In this paper we also briefly review some of the recently-developed HPC methods and tools that have played a major role in making flow simulation a powerful tool that it is today. In this review and in providing examples, we include some material extracted from other articles written by these authors and their group, particularly a recent article by Tezduyar et al. (1996).

We consider applications involving compressible as well as incompressible flows, mostly with time-dependent behavior, and mostly with moving boundaries and interfaces. The governing equations for this class of problems are given in Section 2. The flow simulation capabilities developed recently to address this type of applications cover a large class of flow regimes and patterns: flows involving complex geometries (Tezduyar et al. (1994), Johnson and Tezduyar (1996a)), flows with free surfaces (Soulaimani et al. (1991), Behr and Tezduyar (1994)), tidal flows (Kashiyama et al. (1995)), two-liquid interfaces (Johnson and Tezduyar (1994), Ray et al. (1996)), fluid-particle interactions (Johnson and Tezduyar (1995), Johnson and Tezduyar (1996b), fluid-structure interactions (Mittal and Tezduyar (1995), Ray et al. (1996)), 2D aeroelasticity (Farhat et al. (1995)) and moving mechanical components (Ray et al. (1996)). We also see a significant amount of activity in HPC-based flow simulation of applications in chemical engineering and materials science (see for example Salinger et al. (1994), Xiao et al. (1995)).

Because of the geometric complexities involved in a typical, real-world problem, especially if combined with the presence

of moving boundaries and interfaces, the finite element method with unstructured meshes becomes one of the best strategies for flow simulation, and in fact sometimes the only practical choice. The advances of the last two decades in stabilized finite element formulations have made this method an even more credible and powerful approach in flow computations. The most notable stabilization techniques are, streamline-upwind/Petrov-Galerkin (SUPG) formulation for incompressible flows (Brooks and Hughes (1982)), SUPG formulation for compressible flows (Tezduyar and Hughes (1983), Le Beau and Tezduyar (1991), Aliabadi et al. (1993)), Galerkin/least-squares (GLS) formulation (Hughes et al. (1989), Hansbo and Szepessy (1990)), and pressure-stabilizing/Petrov-Galerkin (PSPG) formulation for incompressible flows (Tezduyar (1991)). The stabilization prevents the numerical oscillations and instabilities that might be encountered when the flow involves high Reynolds and/or Mach numbers and strong shocks and sharp boundary layers. The stabilization also permits the use of equal-order interpolation functions for velocity and pressure and other unknowns, which is a significant advantage in parallel implementations. In all cases, the stabilization is accomplished still in the context of a weighted residual formulation and without introducing excessive numerical dissipation, and that is the main attractive point about this class of methods. A brief review of these stabilization techniques is given in Section 3, together with the specific stabilized finite element formulations employed in carrying out the flow simulations reported in this paper.

When we need to solve problems with changing spatial domains, such as those encountered in flows with moving boundaries and interfaces, depending on the circumstances, one can chose between fixed grid and moving grid methods. As the most notable moving grid methods, we can list the moving finite elements, arbitrary Lagrangian-Eulerian, and the space-time formulations. The application of the stabilized space-time finite element formulation to flows with moving boundaries and interfaces was introduced first in the context of incompressible flows in Tezduyar et al. (1992c), and soon after that in the context of compressible flows in Aliabadi and Tezduyar (1993). The stabilized space-time formulation was in fact used earlier by other researchers to solve problems with fixed spatial domains (Hughes and Hulbert (1988), Hansbo and Szepessy (1990), Shakib (1988)), however we do not see any major advantage in using this relatively costly method for problems with just fixed domains. When the flow problem does not involve any moving boundaries and interfaces, we find it equally satisfactory, and a lot less costly, to use the PSPG formulation which is basically a reduced version of the GLS method, but which is still a weighted residual method. Section 3 also includes the description of the stabilized space-time formulations used in carrying out the flow simulations reported in this paper.

Mesh generation continues to be an important issue and sometimes a serious bottleneck in large-scale computations. The 3D finite element meshes needed can be generated, depending on the complexity of the problem geometry, by using either special mesh generators designed for specific problems, or an automatic mesh generator. We believe that still a significant amount of research and development effort needs to go into 3D automatic mesh generation, and there are some research groups who are doing that rather successfully. One of the most notable of these groups is the one led by Shephard (see Shephard and Georges (1991) and Özturan et al. (1994)). This group is also particularly interested in research on parallel mesh generation as well as mesh adaptation. For the class of flow simulations we are interested in, we needed a 3D automatic mesh generation capability which allows structured layers of elements near solid surfaces and unstructured meshes elsewhere in the domain, and we developed a mesh generation tool (Johnson and Tezduyar (1996a)) which satisfies this and some other requirements, and yet continues to evolve to meet the special needs of our simulations as such needs arise. The mesh generation issues are covered in more detail in Section 4.

In our parallel computations of flows problems with moving boundaries and interfaces, the changes in the shape of the spatial domain are handled by updating the mesh by using the combination of moving the mesh and remeshing (i.e., generating a new set of elements and grid points) when the mesh distortion becomes too high (Tezduyar et al. (1992d), Johnson and Tezduyar (1995)). Moving the mesh is accomplished by special mesh moving techniques designed for specific problems and by an automatic mesh moving algorithm for more general cases. This automatic mesh moving is based on the motion of the grid points being governed by the equations of elasticity, with the boundary conditions imposed by the motion of the fluid boundaries and interfaces. Similar mesh moving techniques were used earlier by other researchers, e.g. Lynch (1982). Mesh moving issues were also addressed by Farhat and Lanteri (1994) by using a pseudo-structural model with fictitious springs. The methods for 3D mesh update are covered in Section 5.

The large coupled, nonlinear equation systems that need to be solved at every time step in a time-accurate computation (or every pseudo-time step in a steady-state computation) are solved using iterative methods (Behr et al. (1993), Kalro and Tezduyar (1995)). The main components of these iterative methods, are: a) formation of the residual vector of the linear equation system (that needs to be solved at every iteration of a Newton-Raphson sequence used in solving the coupled, nonlinear equations); b) designing a preconditioning matrix which cost-effectively and reasonably approximates the Jacobian matrix in each Newton-Raphson iteration; and c) updating the solution vector in an optimal way based on the residual and solution increment vectors of the linear equation system. In all our parallel computations, we form the residual vector of the linear equation systems by using, depending on the size of the problem, the element-level matrices or a method which is free from even element-level matrices. Such matrix-free methods were successfully used also by other researchers in the context of parallel computations (Johan et al. (1991), Johan et al. (1995)). For preconditioning matrices, we are currently using simple ones, such as diagonal or nodal-block-diagonal preconditioners (Shakib et al. (1989)), in our production computations. We are also experimenting with more sophisticated ones, such as the clustered element-by-element (CEBE) and mixed CEBE and cluster companion (CC) preconditioners (Liou and Tezduyar (1992), Tezduyar et al. (1992b)). To update the solution vector, we typically use the GMRES technique (Saad and Schultz (1986)). This is a very common update technique for iterative computations involving non-symmetric matrices.

The parallel computations can be carried out on shared or distributed memory systems, or sometimes a combination of the two in the context multi-platform computing or by connecting several identical shared memory systems together. Parallel implementations on shared memory systems (e.g., a 20-processor SGI ONYX or a 12-processor SGI Power Challenge) are based on the fact that the processors use the same central memory, and therefore this parallel computing paradigm is relatively simple to adapt to. For distributed memory systems (e.g., a 512-processor Thinking Machines CM-5 or a 512-processor CRAY T3D), the implementations could be based on data-parallel or message-passing paradigms, and this requires more elaborate work. The user-friendly Connection Machine software libraries made available by a group of researchers at Thinking Machines (Johnsson and Mathur (1989), Mathur and Johnsson (1992)), encouraged a number of researchers to implement their finite element flow simulation formulations on the Connection Machine systems (Johan et al. (1992), Tezduyar et al. (1992d)). Our parallel computations on the Connection Machine systems for flow problems were reported first in Tezduyar et al. (1992d) for 2D and axisymmetric simulations, and soon after that in Tezduyar et al. (1992a) and Tezduyar et al. (1993) for 3D simulations. More 3D simulations were reported in Behr and Tezduyar (1994), Mittal and Tezduyar (1994), Tezduyar et al. (1994), Aliabadi and Tezduyar (1995), Mittal and Tezduyar (1995), Johnson and Tezduyar (1995) and Kalro et al. (1996). The 3D simulations we carried out on the T3D were reported first in Tezduyar et al. (1995). Some of the other notable parallel computing activities for flow simulations were reported by Farhat and his coworkers, for example in Farhat et al. (1993) and Farhat and Lanteri (1994) for 2D viscous flows, and in Farhat et al. (1995) for 2D aeroelasticity. The parallel computing studies by Shephard and his coworkers (Özturan et al. (1994)) emphasize automatic mesh generation and mesh adaptivity.

Currently, the majority of our parallel computations are carried out on the AHPCRC's 896-node CM-5 and the Minnesota Supercomputer Center's 512-node T3D. Some of our smaller scale parallel computations are performed on the AHPCRC's 20-processor SGI Onyx. The applications that will be presented in this paper are: supersonic flow around fighter airplane; flow around two trains passing each other in a tunnel; steady-state decent of a large ram air parachute; longitudinal dynamics and flare maneuver of a large ram air parachute; flow past the spillway of the Olmsted Dam; contaminant dispersion in a model subway station; airflow past an automobile; multiple spheres falling in a liquid-filled tube; and dynamics of a paratrooper jumping from a cargo plane.

2
Governing equations

The flow simulations are based on the solution of time-dependent Navier-Stokes equations of compressible and incompressible flows. In stating those equations here, Ω_t and $(0, T)$ will denote the space and time domains, where Γ_t is the boundary of Ω_t. In general, the spatial domain may change with respect to time, and the subscript t indicates such time-dependence. The symbols $\rho(\mathbf{x}, t)$, $\mathbf{u}(\mathbf{x}, t)$, $p(\mathbf{x}, t)$ and $e(\mathbf{x}, t)$ represent the density, velocity, pressure and the total energy, respectively. The external forces (e.g. gravity) are represented by $\mathbf{f}(\mathbf{x}, t)$.

2.1
Compressible flows

The Navier-Stokes equations of compressible flows can be written in the following vector form:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} - \frac{\partial \mathbf{E}_i}{\partial x_i} = \mathbf{0} \quad \text{on } \Omega_t \quad \forall t \in (0, T), \tag{1}$$

where $\mathbf{U} = (\rho, \rho u_1, \rho u_2, \rho u_3, \rho e)$ is the vector of conservation variables, and $\mathbf{F}_i$ and $\mathbf{E}_i$ are, respectively, the Euler and viscous flux vectors defined as

$$\mathbf{F}_i = \begin{pmatrix} u_i \rho \\ u_i \rho u_1 + \delta_{i1} p \\ u_i \rho u_2 + \delta_{i2} p \\ u_i \rho u_3 + \delta_{i3} p \\ u_i (\rho e + p) \end{pmatrix}, \tag{2}$$

$$\mathbf{E}_i = \begin{pmatrix} 0 \\ [\mathbf{T}]_{i1} \\ [\mathbf{T}]_{i2} \\ [\mathbf{T}]_{i3} \\ -q_i + [\mathbf{T}]_{ik} u_k \end{pmatrix}. \tag{3}$$

Here $[\mathbf{T}]_{ij}$ are the components of the Newtonian viscous stress tensor:

$$\mathbf{T} = 2\mu \boldsymbol{\varepsilon}(\mathbf{u}), \tag{4}$$

where μ is the dynamic viscosity and $\boldsymbol{\varepsilon}$ is the strain rate tensor, and q_i are the components of the heat flux vector. The equation of state is modeled with the ideal gas assumption.

Equation (1) can also be written in the following form:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}_i \frac{\partial \mathbf{U}}{\partial x_i} - \frac{\partial}{\partial x_i}\left(\mathbf{K}_{ij} \frac{\partial \mathbf{U}}{\partial x_j}\right) = \mathbf{0} \text{ on } \Omega_t \quad \forall t \in (0, T), \tag{5}$$

where

$$\mathbf{A}_i = \frac{\partial \mathbf{F}_i}{\partial \mathbf{U}}, \tag{6}$$

$$\mathbf{K}_{ij} \frac{\partial \mathbf{U}}{\partial x_j} = \mathbf{E}_i. \tag{7}$$

Appropriate sets of boundary and initial conditions are assumed to accompany Eq. (5).

2.2
Incompressible flows

The Navier-Stokes equations of incompressible flows can be written in the following vector form:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \mathbf{f}\right) - \nabla \cdot \boldsymbol{\sigma} = 0 \quad \text{on } \Omega_t \quad \forall t \in (0, T), \tag{8}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega_t \quad \forall t \in (0, T), \tag{9}$$

where ρ is assumed to be constant, and

$$\boldsymbol{\sigma} = -p\mathbf{I} + \mathbf{T}. \tag{10}$$

This equation set is completed by an appropriate set of boundary conditions and an initial condition consisting of a divergence-free velocity field specified over the entire domain:

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0, \quad \nabla \cdot \mathbf{u}_0 = 0 \quad \text{on } \Omega_0. \tag{11}$$

2.3
Turbulence model

Most computational grids which are in practical use today are not able to resolve the flow features well enough to fully capture turbulence effects. Acknowledging this restriction, significant effort is spent in the computational community to devise and improve turbulence modeling. Several reviews of the work in this area have been published, see e.g. Speziale (1991). A notable attempt has been made recently to integrate the concepts of turbulence modeling and stabilization techniques (Hughes (1995), Hughes and Stewart (1996)).

In our high-Reynolds number flow computations, we employ a simple Smagorinsky turbulence model (Smagorinsky (1963), Kato and Ikegawa (1991)). In this approach, the kinematic viscosity ν is augmented by an eddy viscosity:

$$\nu \leftarrow \nu + (Ch)^2 (2\boldsymbol{\varepsilon}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{u}))^{1/2}, \tag{12}$$

where C is the model constant and h is the element length. Unless stated otherwise, the constant C is set to 0.15.

3
Stabilized finite element formulations

The SUPG formulation for incompressible flows was introduced in Hughes and Brooks (1979), with detail description of the formulation and numerical examples given in Brooks and Hughes (1982). The SUPG formulation for compressible flows on the other hand was first introduced, in the context of conservation variables, in Tezduyar and Hughes (1983). Following that, several researchers designed and studied SUPG-like methods for compressible flows. For example, the Taylor-Galerkin stabilization method, which appeared in the literature in 1984 in an article by Donea (1984), is very similar, and under some conditions identical to one of the stabilization methods introduced in Tezduyar and Hughes (1983). Among other researchers who used the SUPG-like methods for compressible flows in the context of conservation variables formulation in 1984 and the following years are Johnson and his coworkers (Johnson et al. (1984), Johnson and Saranen (1986)). Also following Tezduyar and Hughes (1983), the SUPG formulation for compressible flows was recast in entropy variables and supplemented with a shock-capturing term (Hughes et al. (1987), Shakib (1988)). It was shown in Le Beau and Tezduyar (1991) and Aliabadi et al. (1993) that, the SUPG formulation introduced in Tezduyar and Hughes (1983), when supplemented with a similar shock-capturing term, is very comparable in accuracy to the one employing entropy variables. In fact, it was shown in Le Beau et al. (1993) for inviscid flows and in Aliabadi et al. (1993) for viscous flows that for the 2D comparative test problems computed, the SUPG formulations in conservation and entropy variables yield rather indistinguishable results.

For the compressible and incompressible flow applications later discussed in Section 8, the space-time formulation begins with the weak form of the governing equations being written over the associated space-time domain of the problem, by dividing this domain into a sequence of space-time slabs Q_n, where Q_n is the slice of the space-time domain between the time levels t_n and t_{n+1}. The integrations involved in the weak form are then performed over Q_n. The finite element interpolation functions used are continuous in space but discontinuous across time levels. To reflect this situation, we use the notation $(\cdot)_n^-$ and $(\cdot)_n^+$ to denote the function values at t_n as approached from below and above respectively. Each space-time slab Q_n is decomposed into space-time elements Q_n^e, where $e = 1, 2, \ldots, (n_{el})_n$. The subscript n used with n_{el} is to account for the general case in which the number of space-time elements may change from one space-time slab to other. In our computations, we use first-order polynomials as interpolation functions.

3.1
Compressible flows

In the finite element formulation of compressible flows, for each slab Q_n, we first define appropriate finite-dimensional function spaces $\mathscr{S}_n^h$ and $\mathscr{V}_n^h$ corresponding to the trial solutions and weighting functions, respectively. The stabilized space-time formulation of (5) can then be written as follows: given $(\mathbf{U}^h)_n^-$, find $\mathbf{U}^h \in \mathscr{S}_n^h$ such that $\forall \mathbf{W}^h \in \mathscr{V}_n^h$:

$$\int_{Q_n} \mathbf{W}^h \cdot \left(\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \right) dQ + \int_{Q_n} \left(\frac{\partial \mathbf{W}^h}{\partial x_i} \right) \cdot \left(\mathbf{K}_{ij}^h \frac{\partial \mathbf{U}^h}{\partial x_j} \right) dQ$$

$$+ \int_{\Omega_n} (\mathbf{W}^h)_n^+ \cdot ((\mathbf{U}^h)_n^+ - (\mathbf{U}^h)_n^-)\, d\Omega$$

$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau (\mathbf{A}_k^h)^T \left(\frac{\partial \mathbf{W}^h}{\partial x_k} \right) \cdot \left[\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} - \frac{\partial}{\partial x_i} \left(\mathbf{K}_{ij}^h \frac{\partial \mathbf{U}^h}{\partial x_j} \right) \right] dQ$$

$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \delta \left(\frac{\partial \mathbf{W}^h}{\partial x_i} \right) \cdot \left(\frac{\partial \mathbf{U}^h}{\partial x_i} \right) dQ = \int_{(P_n)_H} \mathbf{W}^h \cdot \boldsymbol{H}^h \, dP. \tag{13}$$

Here $\boldsymbol{H}^h$ represents the Neumann-type boundary condition, $(P_n)_H$ is the part of the slab boundary with such conditions, and τ and δ are the stabilization parameters.

The solution to (13) is obtained sequentially for $Q_1, Q_2, \ldots, Q_{N-1}$, starting with

$$(\mathbf{U}^h)_0^- = \mathbf{U}_0^h, \tag{14}$$

where $\mathbf{U}_0$ is the specified initial value of the vector $\mathbf{U}$.

In the formulation given by the Eq. (13), the first three integrals, together with the right-hand-side, represent the time-discontinuous Galerkin formulation of (5). The third integral enforces, weakly, the continuity of the conservation variables in time. The first series of element-level integrals are the SUPG stabilization terms, and the second series are the shock-capturing terms added to the formulation. The details regarding the stabilization and the space-time formulation can be found in Aliabadi and Tezduyar (1993). For problems not involving moving boundaries and interfaces, Eq. (13) can be reduced to

a semi-discrete formulation by dropping the third integral, and by converting all space-time integrations to spatial integrations.

3.2
Incompressible flows

In the finite element formulation of incompressible flows, for each slab Q_n, we first define appropriate finite-dimensional trial solution ($(\mathcal{S}^h_{\mathbf{u}})_n$ and $(\mathcal{S}^h_p)_n$) and weighting function ($(\mathcal{V}^h_{\mathbf{u}})_n$ and $(\mathcal{V}^h_p)_n = (\mathcal{S}^h_p)_n$) spaces for the velocity and pressure.

The stabilized space-time formulation of Eqs. (8) and (9) can then be 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 $\forall \mathbf{w}^h \in (\mathcal{V}^h_{\mathbf{u}})_n$ and $\forall q^h \in (\mathcal{V}^h_p)_n$:

$$\int_{Q_n} \mathbf{w}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f} \right) dQ + \int_{Q_n} \boldsymbol{\varepsilon}(\mathbf{w}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\, dQ$$

$$+ \int_{Q_n} q^h \nabla \cdot \mathbf{u}^h\, dQ + \int_{\Omega_n} (\mathbf{w}^h)_n^+ \cdot \rho ((\mathbf{u}^h)_n^+ - (\mathbf{u}^h)_n^-)\, d\Omega$$

$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau_{\mathrm{MOM}} \frac{1}{\rho} \left[\rho \left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{w}^h \right) - \nabla \cdot \boldsymbol{\sigma}(q^h, \mathbf{w}^h) \right]$$

$$\cdot \left[\rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f} \right) - \nabla \cdot \boldsymbol{\sigma}(p^h, \mathbf{u}^h) \right] dQ$$

$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau_{\mathrm{CONT}} \nabla \cdot \mathbf{w}^h \rho \nabla \cdot \mathbf{u}^h\, dQ = \int_{(P_n)_{\boldsymbol{h}}} \mathbf{w}^h \cdot \boldsymbol{h}^h\, dP. \qquad (15)$$

Here $\boldsymbol{h}^h$ represents the Neumann-type boundary condition imposed, $(P_n)_{\boldsymbol{h}}$ is the part of the slab boundary with such conditions, and τ_{MOM} and τ_{CONT} are the stabilization parameters.

The solution to (15) is obtained for all of the space-time slabs $Q_1, Q_2, \ldots, Q_{N-1}$ sequentially, and the computations start with

$$(\mathbf{u}^h)_0^- = \mathbf{u}_0^h. \qquad (16)$$

In the formulation given by Eq. (15), the first four integrals, together with the right-hand-side, represent the time-discontinuous Galerkin formulation of (8)–(9). The fourth integral enforces, weakly, the continuity of the velocity field in time. The two series of element-level integrals in the formulation are the least-squares stabilization terms. The reader can refer to Tezduyar et al. (1992c) and Behr and Tezduyar (1994) for further details regarding the space-time formulation for incompressible flows, including definitions of the stabilization parameters. For problems not involving moving boundaries and interfaces, Eq. (15) can be reduced to a semi-discrete formulation by dropping the fourth integral and the term $\partial \mathbf{w}^h / \partial t$, and by converting all space-time integrations to spatial integrations.

4
3D mesh generation for complex geometries

As numerical simulations become larger and are applied to problems with complex geometries, the mesh generation and management phases play more and more of an important role in the simulation process. To discretize a domain representing a given geometry, there are two different approaches we can take. We could create a special purpose mesh generator which is designed for a specific application, or we could use an automatic mesh generator which can be applied to many different cases. The two approaches are very different and both have advantages and disadvantages, but usually the type of problem under consideration will determine which approach is taken.

4.1
Special-purpose mesh generation

A special-purpose mesh generator (see e.g. Kalro et al. (1996)) is one that is developed for a specific geometry and can only create meshes for the class of geometries it was originally designed for. An example of such a mesh generator might be one developed for flow past a flat plate where the geometry would be the discretization of a box. Another example might be a mesh generator developed for flow past a wing where only certain parameters about the wing, such as the airfoil type, pitch angle, and wing aspect ratio can be changed. This approach has several advantages:

- Once the special-purpose mesh generator has been written, the cost of generating a finite element mesh is minimal in terms of computing time.
- The user has direct control over the structure of the mesh and its refinement.
- In problems with moving boundaries and interfaces, such as free surfaces, moving mechanical components, and fluid-structure and fluid-particle interactions, the nodal coordinates need to be updated every time step, and usually a special-purpose mesh moving scheme can be incorporated within the mesh generator. By using such a mesh moving scheme, the deformation of the mesh is user controlled and requires very little computer time to generate a new set of nodes every time the mesh moves.

The disadvantages of this approach to mesh generation is the often time-consuming task of creating the special-purpose mesh generator for a specific application. If the application involves a geometry of any significant complexity, this method is very difficult to apply.

4.2
Automatic mesh generation

Automatic mesh generators, on the other hand, provide much more flexibility in discretizing complex domains. They involve little or no assumptions on the shape of the domain, and because of that, a mesh can be generated for almost any geometry. Also, the mesh generator is general and can be used in many applications. The disadvantage of this approach is that the user cannot specify exactly the type of mesh structure and refinement they may wish to have in all areas of the domain. Our automatic mesh generation program is based on Delaunay methods (Barth (1992), Joe (1991), Borouchaki et al. (1995)) which are probably the most general type and yield high quality meshes in 3D. Some other popular automatic mesh generation methods are advancing front (Baker (1989)) and finite octree (Shephard and Georges (1991)). More comprehensive overviews of several automatic mesh generation methods can be found in Baker (1989), Bern and Eppstein (1992) and George (1991).

Our automatic mesh generation program has three components (Johnson and Tezduyar (1996a)). The first component is an interactive 3D geometric modeler based on Bézier surfaces (Farin (1993)). By using this modeling program, geometries of almost any complex shape can be created. The second compo-

nent is an automatic surface mesh generator which takes the geometric model as input and discretizes it into a surface mesh composed of triangular elements. The refinement level within this surface mesh is controlled by user-specified parameters within the geometric model. The third component is the actual 3D volumetric mesh generator which takes the surface mesh as input and generates the 3D mesh composed of tetrahedral elements. The level of refinement within the 3D mesh is determined by the refinement of the surface mesh. We also have the capability of creating thin, structured layers of elements around solid surfaces while using a totally unstructured mesh in the rest of the domain. By creating these thin layers, the boundary layer features of the flow can be modeled more accurately which is important when trying to represent viscous flows using unstructured meshes.

5
3D mesh update techniques for moving boundaries and interfaces

A large class of fluid dynamics applications involve moving boundaries and interfaces, and in such numerical simulations methods are needed to handle the changes in the shape of the domain. We employ two strategies, which may be used in combination, to handle these changes in the domain shape. These include techniques based on moving the nodal coordinates of the mesh, and those based on remeshing (i.e. generating a new set of nodes and elements).

5.1
Special-purpose mesh moving scheme

The space-time finite element method gives us the ability to move the mesh during a simulation. The motion of the boundary is known (prescribed or determined at every nonlinear iteration) and we have two methods to specify the displacement of the internal nodes. One approach is to use a special-purpose mesh moving scheme (usually associated with a special-purpose mesh generator) to specify the movement of the mesh based on some pre-defined motion (Kalro et al. (1996)). In problems where the motion is pre-determined or constrained in some fashion, this motion can be programmed within the mesh generator, and every time the nodes are required to move a new mesh is generated in the new configuration. The connectivity of the new mesh remains the same as the old one so we are, in effect, just moving the nodes. This method is desirable due to the fact that the deformation of the mesh is user controlled and bounded, but its applicability is limited to known or a constrained motion for simple geometries. An example of such an application is flow past a wing that is allowed to translate and change its pitch angle. The translation can be handled by translating the entire mesh, and the change in pitch angle can be incorporated within the mesh generator (i.e. the pitch angle becomes an input parameter to the mesh generator).

5.2
Automatic mesh moving scheme

For more arbitrary or unknown motions, we employ an automatic mesh moving scheme (Johnson and Tezduyar (1994), Johnson and Tezduyar (1996b)). In this method, we solve the modified equations of linear elasticity to determine the internal nodal displacements with the specified movement of the boundary as a boundary condition. This system is solved using the finite element method every time the mesh is required to move. The modification we introduce to the equations of linear elasticity relies on eliminating from the formulation the Jacobian of the transformation between the element and physical domains. By doing this, smaller elements have a higher effective stiffness coefficient and retain their shape better during mesh motion. This method can be applied to any type of mesh with arbitrary motion, but this comes at the cost of solving a finite element formulation, possibly at every nonlinear iteration. Also, this method is usually the only option for moving the mesh if an unstructured mesh, typically coming from an automatic mesh generator, is being used.

5.3
Remeshing

If, while using the automatic mesh moving scheme, the quality of the mesh is degraded by the mesh motion, we can remesh by generating a new set of nodes and elements and by projecting the solution from the old mesh on to the new one (Johnson and Tezduyar (1996b)). This remeshing is usually carried out in conjunction with an automatic mesh generator. We can, of course, use remeshing to handle all mesh motion, but this is undesirable due to the cost of using the automatic mesh generator, projection of the solution, and the projection errors introduced. By combining remeshing with the automatic mesh moving scheme, we can reduce the remeshing cost as much as possible. We can also reduce the projection errors by using advanced projection schemes (Johnson (1995), de Sampaio et al. (1993)) such as incorporating this projection within the jump term of the space-time finite element formulation.

5.4
Hybrid mesh moving scheme

During a typical numerical simulation involving mesh motion, we use the automatic mesh moving scheme and remeshing in combination with each other. The automatic mesh moving scheme is our main mechanism of mesh motion, but when we find that this mesh has become too distorted, we remesh. During a simulation, we typically track some measures of mesh distortion based on element volume and aspect ratio, and after each computer run, we check these measures to determine if remeshing should take place. If so, we generate a new mesh using an automatic mesh generator (often on a different computer than the one used to carry out the simulation) and project the solution onto the new mesh. This remeshing procedure generates new restart information which is then used in the next computer run. By using these techniques in combination, we can handle an almost unlimited range of motion while still minimizing the cost and errors associated with remeshing.

6
Iterative solution strategies for large, coupled, nonlinear equation systems

As outlined in Section 3, the finite element formulations we developed are either semi-discrete formulations, which are used to solve problems with fixed spatial domains, or space-time formulations, used to solve problems with changing spatial domains, such as those encountered in flows with moving boundaries and interfaces. Regardless of this distinction, the formulations like the ones outlined in Section 3 will lead to a

coupled, nonlinear equation system:

$$\mathbf{N}(\mathbf{d}_n) = \mathbf{F}, \tag{17}$$

where $\mathbf{d}_n$ is the vector of unknowns associated with marching from time step $n-1$ to n in a semi-discrete formulation, or associated with space-time slab n in a space-time formulation. This system is typically large, sometimes involving tens of millions of equations in our computations. It needs to be solved at every time step, in the case of transient simulations, or at every pseudo-time step, in the case of steady-state simulations.

For the nonlinear system of Eq. (17), the Newton-Raphson iterations

$$\left.\frac{\partial \mathbf{N}}{\partial \mathbf{d}}\right|_{\mathbf{d}_n^k} (\Delta \mathbf{d}_n^k) = \mathbf{F} - \mathbf{N}(\mathbf{d}_n^k), \tag{18}$$

each require the solution of a linear equation system

$$\mathbf{A}_n^k \mathbf{x}_n^k = \mathbf{R}_n^k, \tag{19}$$

where k is the nonlinear iteration counter, $\mathbf{A}_n^k = (\partial \mathbf{N}/\partial \mathbf{d})|_{\mathbf{d}_n^k}$ is the nonsymmetric Jacobian matrix, $\mathbf{x}_n^k = \Delta \mathbf{d}_n^k$ is the vector of increments for unknown solution values, and $\mathbf{R}_n^k = \mathbf{F} - \mathbf{N}(\mathbf{d}_n^k)$ is the vector of residuals. When discussing the process of the solution of the linear equation system (19), the sub- and superscripts identifying the time step and nonlinear iteration will be dropped, as only one such system is solved at a given time.

The size of the linear equation systems under consideration precludes the use of direct solutions techniques. We focus instead on iterative strategies. Such strategies are typically based on selection of a preconditioning matrix (which approximates the original matrix) and an update technique which governs how the solution vector is updated as corrections are computed based on the preconditioning matrix selected.

6.1
Preconditioning

The choice of the preconditioning matrix for the iterative solvers on parallel machines is becoming more varied in recent years. For production runs, the diagonal and nodal-block-diagonal preconditioning (Shakib (1988)) are still extensively used, and work well for compressible flow problems, and adequately for incompressible flow problems. We have also developed more complex preconditioners, among them are the clustered-element-by-element (CEBE) preconditioner (Liou and Tezduyar (1992)), and the mixed CEBE and cluster companion (CC) preconditioner (Tezduyar et al. (1992b)). We later implemented some of these complex preconditioners on parallel platforms.

6.2
Iterative update techniques

As an update technique, we employ the Generalized Minimum Residual (GMRES) (Saad and Schultz (1986)), which is a popular method for nonsymmetric systems stemming from Navier-Stokes or Euler equations. In the GMRES procedure, the full system is projected onto a much smaller Krylov subspace, and the residual is minimized on that subspace. The size of the Krylov space required for convergence is usually smaller than 15 for compressible flow problems (Shakib et al. (1989)). This requirement increases for incompressible flow problems and finer meshes.

6.3
Element matrix-based schemes

A related issue is the interaction of the linear equation system matrix with residual-type vectors, which occurs at various stages of the solution update process. For moderately sized problems, the matrix-vector products are computed directly from the element-level matrices, without the global sparse matrix ever being assembled. Employing an element-level data structure, the matrix-vector product $\mathbf{Ax}$ is actually computed as $\mathscr{A}(\mathbf{a}^e\mathbf{x}^e)$, using the fact that $\mathbf{A} = \mathscr{A}\,\mathbf{a}^e$, where $\mathbf{a}^e$ are the element-level matrices, $\mathbf{x}^e$ is the vector $\mathbf{x}$ distributed (gathered) to the element level, and $\mathscr{A}$ represents the finite element assembly operator.

6.4
Matrix-free schemes

For larger problems, a matrix-free scheme is typically used, which results in significantly lower memory demands, but can involve more computations than its element matrix-based counterpart described earlier. In this scheme, the matrix-vector product is approximated by two residual evaluations, eliminating the need to store the element-level matrices (Johan et al. (1991)). The matrix-vector product $\mathbf{A}(\mathbf{d})\mathbf{x}$ can be written as $(\mathbf{R}(\mathbf{d}+\varepsilon\mathbf{x}) - \mathbf{R}(\mathbf{d}))/\varepsilon$, where ε is a suitably small number. In matrix-free computations, the complex analytical task of finding the Jacobian matrix $\mathbf{A}(\mathbf{d})$ of the nonlinear system is also avoided.

7
Parallel computing

For a number of years, the computation speed delivered by a single processing unit has been growing only moderately, and so did the performance of the traditional scalar and vector supercomputers. In contrast, parallel computers compensate for the limited speed of a single processing node by linking tens, hundreds or thousands of nodes together with fast communication channels. Using this approach, computational speeds on the order of trillion floating point operations per second (TeraFLOPS) are within grasp, with sustained rates of 10–50 GigaFLOPS achievable as of now. More often than not, however, the parallel algorithm design and programming techniques have to take specific hardware features into consideration, and thus diverge from the methods which were used on scalar machines. For the time being, the task of using a parallel machine efficiently is largely the burden of the numerical analyst, as the parallel compilers and operating systems are yet unable to hide various characteristic features of the parallel machine, especially the memory hierarchy. This burden is smaller on moderately parallel machines which use a shared memory architecture, but quite acute on highly parallel engines employing distributed memory.

All the numerical examples presented in Section 8 were obtained on either CM-5, T3D, or 20-processor Silicon Graphics ONYX. Our implementations on the distributed memory CM-5 use a data-parallel computing paradigm, exemplified by the Connection Machine Fortran (CMF) programming language, which hides somewhat the distinction between on-processor

and off-processor memory from the user. However, for efficient computation, every effort must still be taken to minimize the amount of the compiler-generated interprocessor communication. The implementations on the distributed memory T3D use a message-passing computing paradigm, making the interprocessor communication explicit. The said communication is accomplished using the Parallel Virtual Machine (PVM) library. SGI multi-processor (MP) implementations, applicable to ONYX and Power Challenge machines, take advantage of the true shared memory architecture of this machine. Consequently, the issue of the interprocessor communication gives way to scalar optimization as the main approach for increasing the efficiency of the implementation.

7.1
Distributed memory implementations

The first two implementations, based on the CM-5 and the T3D, share a number of features. The chief among them is the explicit partitioning of the finite element mesh into contiguous subdomains, to be assigned later to individual processors. The corresponding nodes which are completely interior to the subdomain are also assigned to that processor. The nodes which belong to the subdomain interfaces are randomly assigned to one of the subdomains sharing that node, and to that subdomain's processor. An example of the partitioning of an unstructured finite element mesh of a spillway of a dam is shown in Fig. 1. The main form of the communication encountered in the "parallel-ready" solution techniques, such as those discussed in Section 6, takes the form of exchange between element-level (stiffness matrices, local residuals) and the node-level (global residuals and increments) data structures (Kennedy et al. (1994)). The transfer of data from the element-level to the node-level takes the form of a scatter, while the movement in the opposite direction takes the form of a gather. In the presence of partitioning, both these operations can be performed in two stages. In one stage, constituting the bulk of data motion, only local gather or scatter are performed, with no interprocessor communication. In the second stage, the sub-domain surface data is communicated to appropriate remote processors. On the CM-5, this technique is built into the Connection Machine Scientific Software Library (CMSSL) partitioned gather and scatter subroutines. On the T3D, PVM-based counterparts of these routines were written by the authors.

7.2
Shared memory implementations

Shared memory machines, such as the SGI MP products, are multiprocessor architectures with a central memory, and as a result, interprocessor communication does not figure significantly in the overall cost. This model of parallelism focuses on distributing segments of `DO` loops across processors. This is achieved by embedding compiler directives, as is illustrated below in a code fragment for a dot product:

```
    dot = 0.0
C$DOACROSS REDUCTION(dot)
    do  i = 1, n
      dot = dot + a(i)*b(i)
    end do
```

The `DOACROSS` directive tells the compiler to distribute `n/npes` iterations of the loop to each of `npes` processors. Here `dot` is treated as a *reduction* type variable, i.e., each processor computes its share of the dot product in its private copy of `dot`, these contributions are at the end summed together. In our implementation, the `DOACROSS` directive is applied to the loop over the elements, thus the formation of element level quantities in parallel is easily achieved. The gather and scatter operations are embedded within the element loop. All quantities are stored at the nodal level, and are gathered to element level, when needed, using the element connectivity data. Implementation of the scatter is not straightforward. As mentioned earlier, some nodes are shared by elements, which may be allocated to different processors. Direct assembly of the global data components associated with such nodes would result in data collisions, leading to erroneous results. Thus, as a preprocessing step, such nodes are isolated using the element connectivity. Each processor assembles its share for these nodes, these are then summed outside the element loop, similar to the two-step scatter algorithm used on the CM-5 and T3D. The use of mesh partitioning reduces the number of nodes requiring this special treatment. Matrix-free solvers are used to minimize memory requirements. With the current implementation, it was possible to simulate, for example, the flow around a ram-air parachute with more than 1.1 million unknowns with less than 280 Mbytes of memory.

7.3
Performance measurements

There are many criteria which may be used in evaluating the performance of a numerical method. The one which is perhaps the most relevant to a computational scientist who attempts to solve a given problem, is the time-to-solution. This benchmark takes into account all the diverse factors, such as the efficiency of the chosen algorithm, utilization of the particular hardware platform as a percentage of its peak speed, and even the effort required to include additional capability into the numerical code. However, the multitude of the quantities rolled into this benchmark makes it quite difficult to compare between different researchers. The more universal, and more widespread, performance benchmark is the raw computational speed, typically expressed in FLoating-point Operations Per Second (FLOPS). While acknowledging the limited value of such isolated performance figures, we still believe that they give at least an approximate measure of the capability of a given algorithm-architecture combination.

We provide therefore, for each of the three architectures mentioned, the approximate speed rating of our finite element codes. For the distributed-memory architectures, the ratio of communication to computation is also a telling figure. The CM-5 performance of the class of codes used by the authors has been discussed extensively in Kennedy et al. (1994). We reiterate that the communication-free portions of the code, such as the formation of element-level residuals, are being executed at approximately 24×10^6 FLOPS (24 MFLOPS) per processing node. The ratio of communication to computation depends greatly on the size of the Krylov space in the GMRES solver; typically it ranges from 18% for a matrix-free implementation, to 31% for an element matrix-based implementation. As a matrix-free computation is dominated by the formation of the element-level residuals, the overall speed is close to

that of the communication-free portion of the code. In an element matrix-based implementation, as the other parts of the computation assume more important role, they reduce the overall speed to as low as 10 MFLOPS per node.

On the T3D, the speed of the communication-free portions of the implementation is measured at around 24 MFLOPS per node. The low-latency communication network of that machine results in an improved communication-to-computation ratio in a 9% range, in the case of a matrix-free code. The overall speed for such code is observed to be 18 MFLOPS per node. For both CM-5 and T3D, these figures are consistent for all partition sizes, and they are taken on a minimum partition which is able to accommodate given problem.

In the SGI MP shared memory implementation, the element-level residual formation proceeds at 22 MFLOPS per node on a 150MHz R4400-based ONYX. In the absence of performance penalties associated with a distributed memory architecture, the overall speed is quite similar, at 19 MFLOPS per node. The maximum number of processors which was found to sustain this performance is 16. On an R8000-based Power Challenge, the overall speed is observed to be approximately 45 MFLOPS per node.

8
Examples

All computations are carried out in 3D and on parallel platforms.

8.1
Supersonic flow past a fighter aircraft

In this problem, supersonic flow past a fighter aircraft at Mach 2.0 is simulated. The flow is assumed to be inviscid and governed by the Euler equations of compressible flows.

The model was generated using our interactive modeling program. The model is very detailed and most of the important components of the aircraft are included. Since the geometry of the aircraft is assumed to be symmetric, only half of the aircraft is being modeled. The surface mesh of this model contains 33,868 nodes and 67,732 triangular elements. A 3D mesh with 185,483 nodes and 1,071,580 tetrahedral elements was produced using our automatic mesh generator.

The computation was carried out on the T3D. The steady-state solution is obtained using a matrix-free iterative strategy and local-time-stepping approach (see Aliabadi and Tezduyar (1995)).

We also used a full unstructured mesh so as to simulate the entire aircraft. This mesh contained 367,867 nodes and 2,143, 160 tetrahedral elements. In this case, we carried out the computations for the same flow conditions by using the CM-5 with 512 nodes. To reach the steady-state solution, at every pseudo-time step, more than 1.7 million coupled nonlinear equations were solved.

The top and bottom images in Fig. 2 show the temperature and Mach number distributions on the aircraft surface respectively. The steady-state solution contains a conical shock wave attached to the nose of the aircraft, and other shock and expansion waves in the other parts of the flow-field. The readers can refer to Aliabadi and Tezduyar (1995), Johan et al. (1994), Morgan et al. (1992) for other inviscid compressible flow simulations around different types of aircraft, using unstructured meshes.

8.2
Flow around high-speed trains in a tunnel

Aerodynamics plays a crucial role in the design and development of high-speed trains from the point of view of cost-effectiveness, safety, comfort, and minimal impact on the environment, amongst many other factors. Trains passing each other, especially in tunnels, create pressure transients which may threaten the structural integrity of the trains besides causing some discomfort to the passengers.

We use the stabilized space-time finite element formulation of compressible flows for this simulation. The mesh used is rather coarse for a 3D problem with intricate geometry and therefore we attach only a qualitative significance to the results. The mesh consists of 101,888 hexahedral elements and 230,982 space-time nodes, and requires the solution of 900,274 equations. The Reynolds number based on steady speed and train length is ~ 67 million. The Smagorinsky LES turbulence model is used to account for the effect of the subgrid structures. To accommodate the motion of the trains we have developed a special algebraic mesh moving and remeshing scheme where elements are transplanted from regions forward of the traints to regions aft of the trains when they become overtly distorted. The total number of nodes and elements remains fixed during the simulation.

At the start of the simulation the trains are at rest near the tunnel entrances and accelerate to 100 m/s, which is the speed at which they pass each other. When the trains achieve a constant speed the pressure at the nose is ~ 0.12 MPa and the corresponding pressure coefficient is ~ 2.46. The trailing tip is surrounded by a low pressure region due to rarefaction of the air. The simulations were carried out on the CM-5.

The bottom right image in Fig. 3 shows the pressure history of points on the nose, inner side and trailing tip of one of the trains. As the trains approach each other the high pressure regions interact and give rise to positive peaks. This peak subsides and then dips due to the interaction of low pressure trailing tip of one train with the high pressure nose of the other. Similar behavior is observed at the other points. The pressure changes are maximum at the nose and minimum at the trailing tip. After the trains have passed each other the pressure recovers to the level prior to crossing. The top image in Fig. 3 shows the pressure on the train surfaces at one instant during passage.

From this simulation we estimate a pressure variation of 8 kPa at the nose. It is however important to note that the pressure changes depend very strongly on the aerodynamic shape of the trains (for which we have a very simple model). The bottom left image in Fig. 3 shows the time history of the drag force experienced by one of the trains during passage.

8.3
Steady-state descent of a large ram-air parachute

Ram-air parachutes are finding increasing use in military operations for accurate delivery of large payloads such as supply crates and vehicles. HPC tools are currently being developed (Garrard et al. (1995), Kalro et al. (1996)) to predict the performance of these parachutes, from inflation to full deployment. The aerodynamics of such parachutes involves very complex phenomena, thus very fine meshes are required for an accurate representation of the flow features. The top image in Fig. 4 shows the steady-state pressure distribution on a ram-air

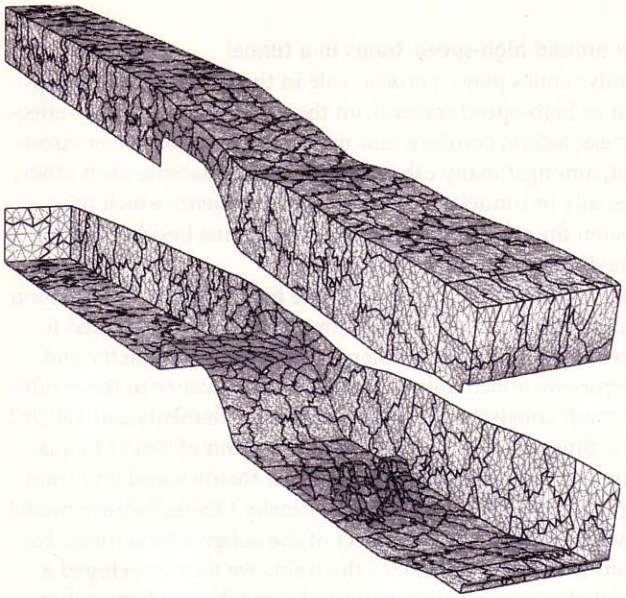

Fig. 1. Partitioning example: Mesh for a spillway of a dam divided into subdomains to be distributed among 256 processors

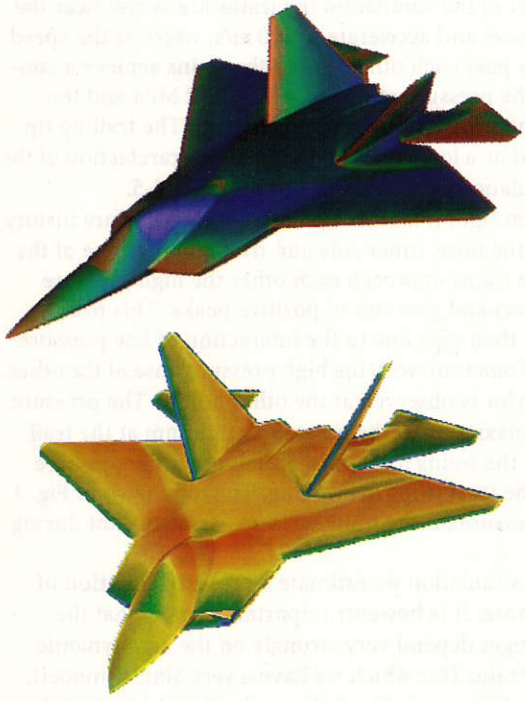

Fig. 2. Supersonic flow past a fighter aircraft: The top and bottom images show the temperature and Mach number distributions on the aircraft surface respectively

parachute with no flaps. The angle of attack is 2 degrees and Reynolds number 10 million. This computation, carried out on a T3D, required the solution of over 38 million coupled, nonlinear equations at every pseudo-time step. This is the largest problem we have solved with our current implementations. The bottom image in Fig. 4 shows the flow past a parachute with flaps. The pressure distribution is indicated on the parachute

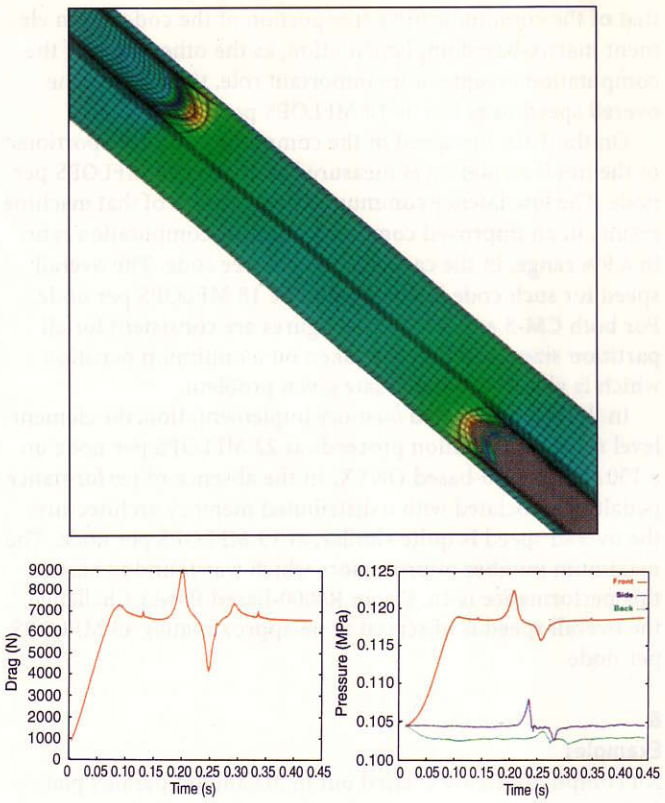

Fig. 3. Flow around high-speed trains in a tunnel: The top image shows the pressure distribution on the train surfaces and a 2D section at one instant. The bottom left image shows the drag experienced by one of the trains during passage. The bottom right image shows the history of pressure at three points on one of the trains

surface, together with stream tubes colored with pressure. This simulation was carried out on the CM-5 and required the solution of 2,363,887 equations at every pseudotime step. This simulation, and the next two, are part of a collaborative project with W. Garrard (University of Minnesota), and K. Stein and E. Steeves (Natick Research, Development, and Engineering Center).

8.4
Longitudinal dynamics of a large ram-air parachute

It is essential to evaluate the capability of a parachute to settle into steady glide, and indicate conditions where dynamic transients are lightly damped (Lingard (1995)). The desired steady glide configuration for the parachute is achieved by rigging the canopy, i.e., adjusting the center of gravity (c.g.) of the system such that equilibrium is obtained in the desired configuration. Mittal and Tezduyar (1994) has investigated the effect of location of the c.g. on the dynamic behavior of 2D airfoils. In this preliminary simulation, the longitudinal characteristics of ram-air parachutes are assessed. A hexahedral mesh with 291,437 nodes and 279,888 elements is used in this space-time computation. The parachute has a Clark Y airfoil cross section and a rigging angle of 9 degrees. The aspect ratio is 3.0 and the ratio of line length to span is 0.6. At every time step the solution of 2,258,496 coupled, nonlinear equations is required. A mesh-moving scheme similar to the one used in Mittal and Tezduyar

(1994) accommodates the pitching motion of the parachute. Further, the motion of the parachute is confined to a vertical plane. The effect of line drag is modeled as a single force acting on the midpoint of the line joining the quarter chord of the canopy to the payload (Lingard (1995)). The parachute is released at 45 degrees with 0 pitch angle. It exhibits a tendency to rapidly pitch forward, accelerate and settle into steady glide. This steady glide configuration is used as an initial condition for flare maneuvers. Figure 5 shows the parachute at two instants. The pressure field and mesh in corresponding section are also shown. This computation was carried out on the T3D.

8.5
Flare maneuver of a large ram-air parachute

One of the favorable characteristics of ram-air parachutes is the capability to deliver loads with reduced landing impact. This maneuver is achieved by pulling on the flaps at either end, and is termed as flaring. The increase in the effective camber creates large aerodynamic forces, this in turn causes the parachute system to decelerate. A specific mesh generator was developed to represent the parachute geometry together with flaps. This mesh also allows for the motion of the flaps during the flare without overt distortion of elements. As a result, the entire flare maneuver is simulated without the requirement to remesh; thus reducing the overheads in the parallel computation. The mesh used results in 3,666,432 coupled, nonlinear equations which are solved at every time step. The space-time finite element formulation is used in this problem. Here, the mesh moves together with the parachute. The initial condition consists of the steady glide configuration of an unconstrained parachute with no flap deflection. The time for the flare maneuver and total flap deflection is obtained from test data. The parachute is treated as a solid body with changing shape. The shape of the parachute during the maneuver is interpolated from the initial and final flap configurations. At the end of the maneuver there is a significant decrease in the horizontal component of the velocity, and this is consistent with flight data. The Reynolds number for this simulation is approximately 10 million. An algebraic turbulence model is used in the computation. This simulation was carried out on the CM-5. Figure 6 shows the pressure distribution on the parachute surface at three instants during the flare maneuver.

8.6
Flow past the spillway of the Olmsted Dam

In this problem we investigate the water flow in the spillway of the Olmsted Dam on the Ohio River. The Dam was under study by the U.S. Army Corps of Engineers for possible modifications of the spillway bed. Several experimental models were constructed with the aim of analyzing the candidate designs for the scour protection of the spillway bed. The geometrical model represents a 48 feet wide section of the navigation pass crest and stilling basin, and includes a long upstream channel, the spillway crest, and a set of underwater obstacles designed to dissipate the energy of the flow. This geometry and its surface discretization have been already shown in Fig. 1. The mesh consists of 139,352 space-time nodes and 396,682 tetrahedral space-time elements. The top surface of the mesh is free and allowed to move in the vertical direction. A stabilized mesh surface movement mechanism is employed (Soulaimani et al. (1991)). In the interior of the domain, the position of the nodes is being updated based on the linear elasticity formulation described in Section 5. The 533,172 flow equations and the 168,846 mesh displacement equations are solved using the GMRES update technique. Shown in Fig. 7 are the pressure field and streamlines, and the steady shape of the free surface achieved in the final stages of the time-dependent simulation. This problem was computed on the CM-5. This simulation is part of a collaborative project with R. Stockstill and C. Berger (Waterways Experiment Station).

8.7
Contaminant dispersion in a model subway station

In this problem, the dispersion of a contaminant in a model subway station is simulated. The subway station has two entrances on each side and four vents at the ceiling. The air is assumed to enter the station from the left (inflow boundary) at a rate of 0.75 m/s. The air is also ventilated through each vent at a rate of 0.15 m/s.

This simulation is carried out on the T3D in two stages. First, the full Navier-Stokes equations are solved to obtain the velocity field throughout the subway station. This velocity field is used in the second stage in the time-dependent contaminant dispersion equation to determine the concentration of the contaminant. Here, the contaminant is released from a point source, with constant strength, located close to the inflow boundary. The Prandtl and Lewis numbers in this computation are 0.72 and 1.0 respectively.

The mesh used in the simulation consists of 187,612 nodes and 1,116,992 tetrahedral elements. The steady-state solution of the incompressible Navier-Stokes equations is obtained by solving over 0.65 million coupled, nonlinear equations at every pseudo-time step. The transient solution of the equations governing the contaminant dispersion involves solution of a linear system with more than 0.15 million equations at every time step.

The images in Fig. 8 show the sections of the subway station together with iso-surfaces of the contaminant concentration at two instants. Another example of a contaminant dispersion calculation, in a flow field around a battle tank, can be found in Tezduyar et al. (1996).

8.8
Airflow past an automobile

In this example, described in more detail in Johnson and Tezduyar (1996a), we simulate airflow past an automobile traveling at a speed of 55 miles per hour. Other numerical simulations of automobile shapes can be found in Kobayashi (1992), Matsunga et al. (1992), Gresho and Chan (1996). We chose to model the automobile after a Saturn SL2. We first generated a mesh for half of the domain (since the geometry is symmetric) using our automatic mesh generation package. This mesh contains 1,407,579 tetrahedral elements. We then reflected the mesh across the symmetry plane to represent the full automobile. This mesh contains 448,695 nodes and 2,815,158 elements. The automatic mesh generator has created three thin structured layers of elements around the automobile surface and wheels.

The Reynolds number for this flow condition is 6.9 million (based on the automobile length), and the flow is assumed to be incompressible and turbulent. For efficient use of the computing resources (512 node CM-5), a matrix-free implementa-

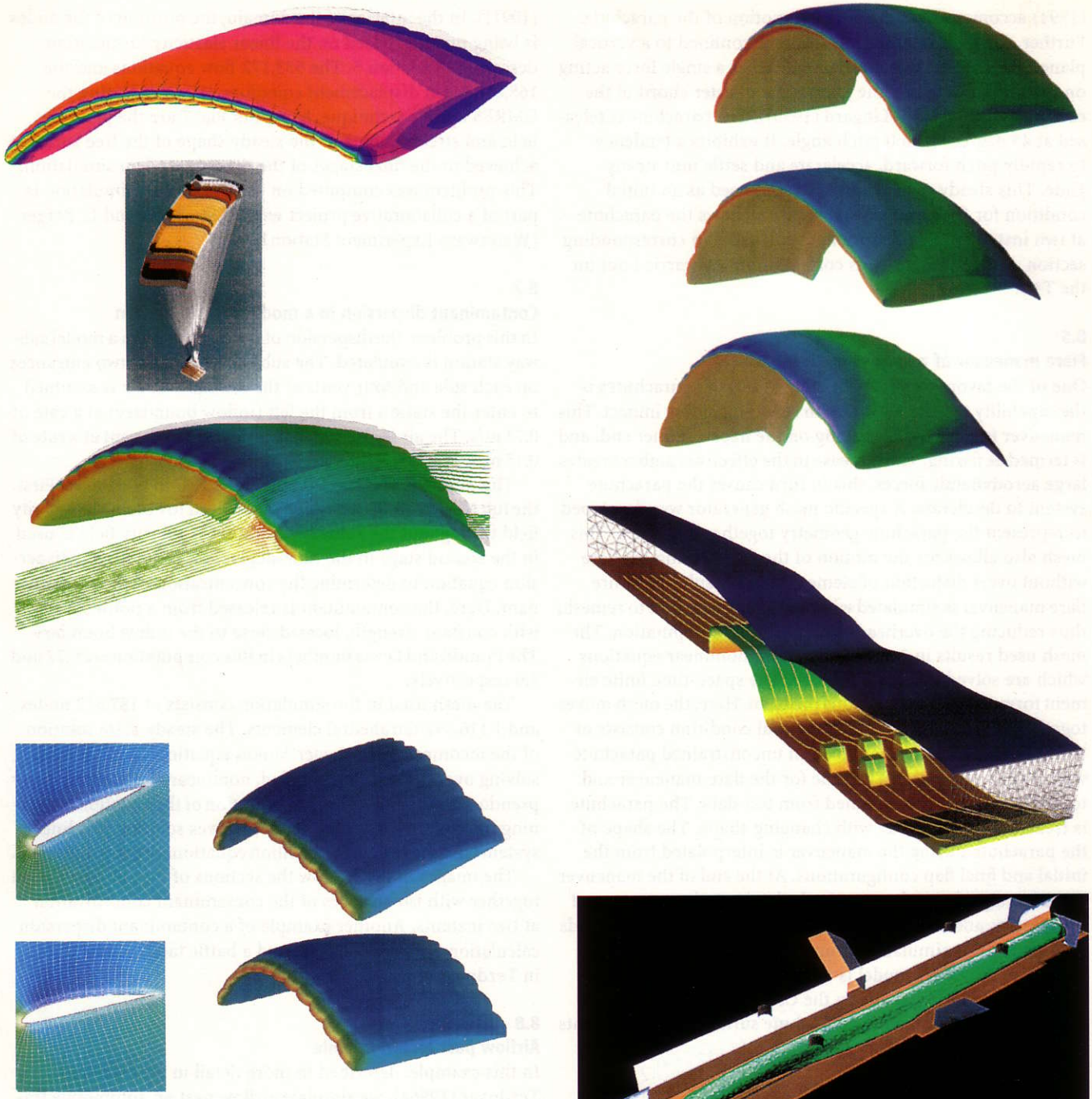

Fig. 4. Steady-state descent of a large ram-air parachute: The top image shows the pressure distribution on the surface of a parachute with no flaps. The bottom image shows the pressure distribution together with stream tubes on a parachute with flaps

Fig. 5. Longitudinal dynamics of a large ram-air parachute: Pressure distribution on the parachute surface and at a cross section at two instants of pitching. The mesh can also be seen in these sections

Fig. 6. Flare maneuver of a large ram-air parachute: Images show the pressure distribution on the parachute surface at three instants of flare

Fig. 7. Flow past the spillway of the Olmsted Dam: The pressure field on the spillway bed and on the free surface, and a set of streamlines

Fig. 8. Contaminant dispersion in a model subway station: The images show the sections of the subway station together with iso-surfaces of the contaminant concentration at two instants

tion of the flow solver is used. The simulation is carried out under road conditions where the wheels are spinning (rotational velocity field is specified on the wheels), and the free-stream velocity is imposed on the road and inflow boundaries. We also carried out simulation under wind–tunnel conditions where both the automobile surface and wheels are stationary, and we apply slip boundary conditions along the road. This second simulation is performed so we can compare the computed drag coefficient with those found experimentally in wind tunnels.

The computed drag coefficient under road conditions is 0.46 and under wind–tunnel conditions is 0.35. For comparision, the stated drag coefficient for a Saturn SL2 is 0.34 (Holt (1990)), but the computational model we used is only a rough approximation to a Saturn SL2. Shown in Fig. 9 is the pressure on the surface of the automobile and a set of streamlines. Velocity at a cutting plane at 1/4 car lengths behind the automobile and at a section cutting the rear wheel are also shown in Fig. 9. All images in the figures are for road conditions. Similar flow behaviour to the ones computed here are observed in Cogotti (1983), Muto (1983), Ahmed et al. (1984).

8.9
Multiple spheres falling in a liquid-filled tube

In this example, described in more detail in Johnson and Tezduyar (1995), Johnson and Tezduyar (1996b), we simulate four cases of multiple spheres falling in a liquid-filled tube. The

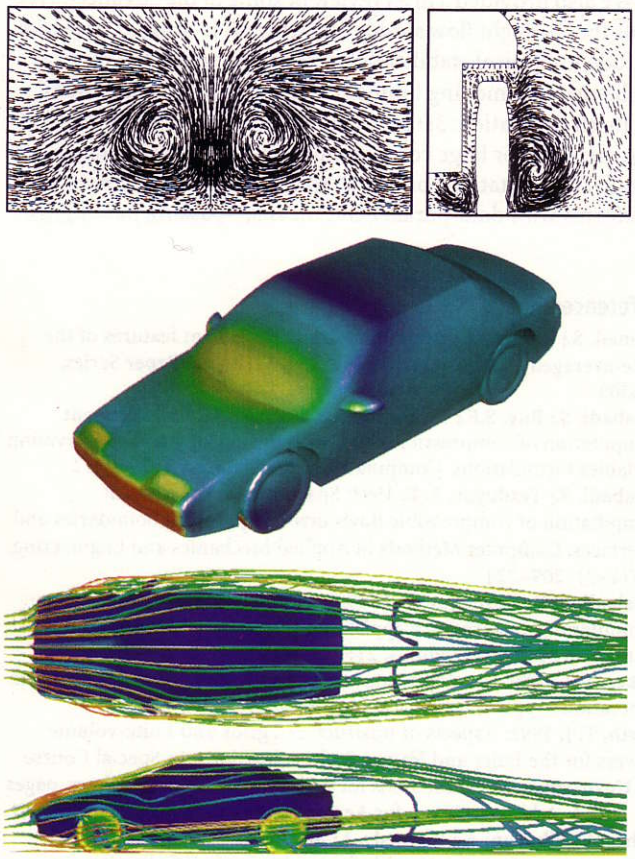

Fig. 9. Airflow past an automobile: On the top are shown velocity vectors at cross-sections 1/4 car lengths behind the automobile and at a section cutting the rear wheel. In the center is shown the pressure distribution on the surface of the automobile, while on the bottom are shown streamlines

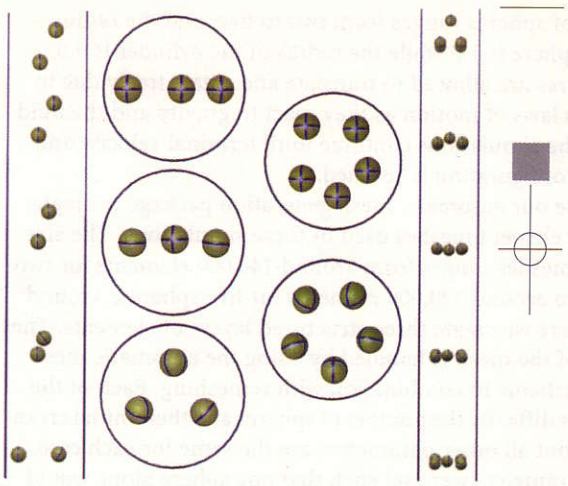

Fig. 10. Multiple spheres falling in a liquid-filled tube: On the left is shown the position of two spheres falling in a liquid-filled tube at five instants during the simulation. At center left is shown the position of three spheres falling in a liquid-filled tube at the initial, neutrally-stable, and stable states. At center right is shown the position of five spheres falling in a liquid-filled tube at the initial and stable states. On the right is shown the position of five spheres falling in a liquid-filled tube at five instants.

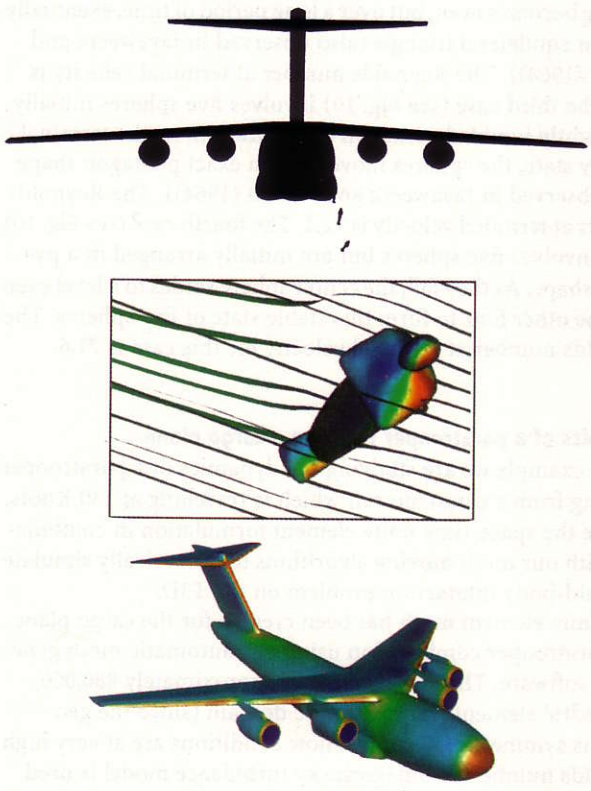

Fig. 11. Dynamics of a paratrooper jumping from a cargo plane: On the top is shown the position and orientation of the paratrooper at different instants during the simulation. On the bottom is shown the steady-state pressure distribution on the surface of the cargo plane, and in the center is the pressure distribution on the paratrooper with streamlines

number of spheres ranges from two to five, and the radius of each sphere is 1.0 while the radius of the cylinder is 5.0. The spheres are allowed to translate and rotate freely due to Newton's laws of motion as they react to gravity and the fluid forces. The simulations continue until terminal velocity and a stable configuration is reached.

We use our automatic mesh generation package to create the finite element meshes used in these simulations. The size of these meshes ranges from around 140,000 elements for two spheres to around 320,000 elements for five spheres. Around each sphere we create three structured layers of elements. The motion of the mesh is handled by using the automatic mesh moving scheme in combination with remeshing. Each of the four cases differ by the number of spheres and their initial arrangement, but all other parameters are the same for each case. These parameters were set such that one sphere alone would fall at a terminal velocity Reynolds number of 100. In each case, the spheres and the fluid are initially at rest. We used our incompressible flow solver on the CM-5 to perform the calculations.

The first case (see Fig. 10) involves two spheres initially arranged in a staggered configuration. As the spheres fall, the trailling sphere is attracted to the wake of the leading sphere, the spheres eventually collide and then seperate (a behaviour also seen in experiments of Fortes et al. (1987), and in 2D, cylinder simulations of Hu et al. (1992)). They fall side by side throughout the rest of the simulation. The Reynolds number at terminal velocity is 93.1. The second case (see Fig. 10) involves three spheres initially aligned in a row but with uneven spacing. As they fall, they first form a neutrally-stable state where the spacing becomes even, but over a long period of time, eventually form an equilateral triangle (also observed in Jayaweera and Mason (1964)). The Reynolds number at terminal velocity is 86.1. The third case (see Fig. 10) involves five spheres initially in a slightly jumbled pentagon configuration. At the terminal velocity state, the spheres move into an exact pentagon shape (also observed in Jayaweera and Mason (1964)). The Reynolds number at terminal velocity is 72.1. The fourth case (see Fig. 10) again involves five spheres but are initially arranged in a pyramid shape. As they fall, the center sphere settles to a level even with the other four to form this stable state of five spheres. The Reynolds number at terminal velocity for this case is 71.6.

8.10
Dynamics of a paratrooper jumping a cargo plane

In this example we are studying the dynamics of a paratrooper jumping from a cargo aircraft which is travelling at 130 Knots. We use the space-time finite element formulation in combination with our mesh moving algorithms to numerically simulate this fluid-body interaction problem on the T3D.

A finite element mesh has been created for the cargo plane and paratrooper combination using our automatic mesh generation software. This mesh contains approximately 880,000 tetrahedral elements for half of the domain (since the geometry is symmetric). Since the flow conditions are at very high Reynolds numbers, a Smagorinsky turbulence model is used. We use the automatic mesh moving scheme in combination with remeshing to move the mesh in response to the motion of the paratrooper. The paratrooper is allowed to translate and rotate freely governed by Newton's laws of motion.

The initial condition for this time-dependent simulation is the steady-state flow past the cargo plane with the paratrooper located within the open side door of the plane (a portion of the interior of the plane is also modeled). The paratrooper motion is initialized with a small outward velocity. Figure 11 shows the steady-state pressure distribution on the cargo aircraft and also shows a close-up-view of the pressure distribution on the paratrooper with streamlines at one instant during the time-dependent simulation. The last image in Fig. 11 shows the position and orientation of the paratrooper relative to the plane at five instants during the simulation. This simulation was used to test the numerical tools for use in such applications, and is part of a collaborative project with W. Sturek (Army Research Laboratory) and K. Stein (Natick Research, Development, and Engineering Centre).

9
Concluding remarks

Recent developments in high performance computing (HPC) methods and hardware brought flow simulation to a new level. Today, 3D simulation of complex, real-world problems can be carried out with the millions of degrees of freedom needed for reasonable accuracy, and within acceptable amount of computing time. These problems can involve complex geometries and moving boundaries and interfaces such as free surfaces, two-fluid interfaces, fluid-particle and fluid-structure interactions, and moving mechanical components. In this article we provided many examples of such simulations, mostly for applications from aerospace engineering and applied fluid mechanics.

We also provided a brief review of some of the advanced HPC tools that brought flow simulation where it is today. These tools include numerical stabilization techniques; methods for flow problems with moving boundaries and interfaces; automatic 3D mesh generation; 3D mesh update strategies; interative solution methods for large coupled, nonlinear equation systems; and parallel computations on the shared and distributed memory platforms with data-parallel and message-passing paradigms.

References

Ahmed, S.; Ramm, G.; Faltin, G. 1984: Some salient features of the time-averaged ground vehicle wake. SAE Technical Paper Series, 840300

Aliabadi, S.; Ray, S.E.; Tezduyar, T.E. 1993: SUPG finite element computation of compressible flows with the entropy and conservation variables formulations. Computational Mechanics, 11: 300–312

Aliabadi, S.; Tezduyar, T. E. 1993: Space-time finite element computation of compressible flows involving moving boundaries and interfaces. Computer Methods in Applied Mechanics and Engineering, 107(1–2): 209–224

Aliabadi, S. K.; Tezduyar, T. E. 1995: Parallel fluid dynamics computations in aerospace applications. International Journal for Numerical Methods in Fluids, 21: 783–805

Baker, T. J. 1989: Developments and trends in three-dimensional mesh generation. Applied Numerical Mathematics, 5: 275–304

Barth, T. J. 1992: Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. In Special Course on Unstructured Grid Methods for Advection Dominated Flows, pages 6-1 - 6-61. Advisory Group for Aerospace Research and Development

Behr, M.; Johnson, A.; Kennedy, J.; Mittal, S.; Tezduyar, T. E. 1993: Computation of incompressible flows with implicit finite element implementations on the Connection Machine. Computer Methods in Applied Mechanics and Engineering, 108: 99–118

Behr, M.; Tezduyar, T. E. 1994: Finite element solution strategies for large-scale flow simulations. Computer Methods in Applied Mechanics and Engineering, 112: 3–24

Bern, M.; Eppstein, D. 1992: Mesh generation and optimal triangulation. Technical report, Xerox Corporation, Palo Alto Research Center
Borouchaki, H.; Hecht, F.; Saltel, E.; George, P. L. 1995: Reasonably efficient delaunay based mesh generator in 3 dimensions. In Proceedings 4th International Meshing Roundtable, pages 3–14, Sandia National Laboratories
Brooks, A. N.; Hughes, T. J. R. 1982: Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. Computer Methods in Applied Mechanics and Engineering, 32: 199–259
Cogotti, A. 1983: Aerodynamic characteristics of car wheels. In Technological Advances in Vehicle Design Series, SP3; Impact of Aerodynamics on Vehicle Design, pages 173–196, International Journal of Vehicle Design
de Sampaio, P. A. B.; Lyra, P. R. M.; Morgan, K.; Weatherill, N. P. 1993: Petrov–Galerkin solutions of the incompressible Navier–Stokes equations in primitive variables with adaptive remeshing. Computer Methods in Applied Mechanics and Engineering, 106: 143–178
Donea, J. 1984: A Taylor-Galerkin method for convective transport problems. International Journal for Numerical Methods in Engineering, 20: 101–120
Farhat, C.; Fezoui, L.; Lanteri, S. 1993: Two-dimensional viscous flow computations on the CM-2: Unstructured meshes, upwind schemes and massively parallel computations. Computer Methods in Applied Mechanics and Engineering, 102: 61–88
Farhat, C.; Lanateri, S. 1994: Simulation of compressible viscous flows on a variety of MPPs: Computational algorithms for unstructured dynamic meshes and performance results. Computer Methods in Applied Mechanics and Engineering, 119: 35–60
Farhat, C.; Lesoinne, M.; Maman, N. 1995: Mixed explicit/implicit time integration of coupled aeroelastic problems: Three-field formulation, geometric conservation and distributed solution. International Journal for Numerical Methods in Fluids, 21: 807–835
Farin, G. 1993: Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide, Academic Press
Fortes, A. F. Joseph, D. D.; Lundgren, T. S. 1987: Nonlinear mechanics of fluidization of beds of spherical particles, Journal of Fluid Mechanics, 177: 467–483
Garrard, W. L.; Tezduyar, T. E.; Aliabadi, S. K.; Kalro, V.; Luker, J.; Mittal, S. 1995: Inflation analysis of ram air inflated gliding parachutes. In Proceedings of AIAA 13th Aerodynamic Decelerator Systems Technology, AIAA Paper 95–1565, Clearwater Beach, Florida
George, P. L. 1991: Automatic Mesh Generation, Application to Finite Element Methods, John Wiley & Sons
Gresho, P.; Chan, S. 1996: Projection 2 goes turbulent – and fully implicit. to appear in *International Journal for Numerical Methods in Fluids*
Hansbo, P. Szepessy, A. 1990: A velocity-pressure streamline diffusion finite element method for the incompressible Navier-Stokes equations. Computer Methods in Applied Mechanics and Engineering, 84: 175–192.
Holt, D. J. 1990: Saturn: The vehicles. Automotive Engineering, 98(11): 34–43
Hu, H. H.; Joseph, D. D.; Crochet, M. J. 1992: Direct simulation of fluid particle motions. Theoretical and Computational Fluid Mechanics, 3: 285–306
Hughes, T. J. R. 1995: Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles, and the origins of stabilized methods. Computer Methods in Applied Mechanics and Engineering, 127: 387–401
Hughes, T. J. R.; Brooks, A. N. 1979: A multi-dimensional upwind scheme with no crosswind diffusion. In Hughes, T. J. R., editor, Finite Element Methods for Convection Dominated Flows, AMD-Vol. 34, pages 19–35, ASME, New York
Hughes, T. J. R.; Franca, L. P.; Hulbert, G. M. 1989: A new finite element formulation for computational fluid dynamics: VIII. the Galerkin/least-squares method for advective–diffusive equations. Computer Methods in Applied Mechanics and Engineering, 73: 173–189
Hughes, T. J. R.; Franca, L. P.; Mallet, M. 1987: A new finite element formulation for computational fluid dynamics: VI. Convergence analysis of the generalized SUPG formulation for linear time-dependent multi-dimensional advective-diffusive systems. Computer Methods in Applied Mechanics and Engineering, 63: 97–112
Hughes, T. J. R.; Hulbert, G. M. 1988: Space–time finite element methods for elasto-dynamics: formulations and error estimates. Computer Methods in Applied Mechanics and Engineering, 66: 339–363
Hughes, T. J. R.; Stewart, J. R. 1996: A space-time formulation for multiscale phenomena. To appear in *Journal of Computational and Applied Mathematics*
Jayaweera, K. O. L. F.; Mason, B. J. 1964: The behavior of clusters of spheres falling in a viscous fluid. Journal of Fluid Mechanics, 20: 121–128
Joe, B. 1991: Construction of three-dimensional Delaunay triangulations using local transformations. Computer Aided Geometric Design, 8: 123–142
Johan, Z.; Hughes, T. J. R.; Mathur, K. K.; Johnsson, S. L. 1992: A data parallel finite element method for computational fluid dynamics on the Connection Machine system. Computer Methods in Applied Mechanics and Engineering, 99(1): 113–134
Johan, Z.; Hughes, T. J. R.; Shakib, F. 1991: A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids. Computer Methods in Applied Mechanics and Engineering, 87: 281–304
Johan, Z.; Mathur, K. K.; Johnsson, S. L.; Hughes, T. J. R. 1994: An efficient communications strategy for finite element methods on the Connection Machine CM-5 system. Computer Methods in Applied Mechanics and Engineering, 113: 363–387
Johan, Z.; Mathur, K. K.; Johnsson, S. L.; Hughes, T. J. R. 1995: A case study in parallel computation: Viscous flow around an Onera M6 wing. International Journal for Numerical Methods in Fluids, 21: 877–884
Johnson, A. 1995: Mesh Generation and Update Strategies for Parallel Computation of Flow Problems with Moving Boundaries and Interfaces. PhD thesis. University of Minnesota
Johnson, A. A.; Tezduyar, T. E. 1994: Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces. Computer Methods in Applied Mechanics and Engineering, 119: 73–94
Johnson, A. A.; Tezduyar, T. E. 1995: Mesh generation and update strategies for parallel computation of 3D flow problems. In Computational Mechanics, '95, Proceedings of International Conference on Computational Engineering Science, Mauna Lani, Hawaii
Johnson, A. A.; Tezduyar, T. E. 1996a: Parallel computation of incompressible flows with complex geometries. To appear in *International Journal for Numerical Methods in Fluids* See update U1 below.
Johnson, A. A.; Tezduyar, T. E. 1996b: Simulation of multiple spheres falling in a liquid-filled tube. To appear in *Computer Methods in Applied Mechanics and Engineering* See update U2 below.
Johnson, C.; Navert, U.; Pitkäranta, J. 1984: Finite element methods for linear hyperbolic problems. Computer Methods in Applied Mechanics and Engineering, 45: 285–312
Johnson, C.; Saranen, J. 1986: Streamline diffusion methods for the incompressible Euler and Navier-Stokes equations. Mathematics of Computation, 47: 1–18
Johnsson, S. L.; Mathur, K. K. 1989: Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer. International Journal for Numerical Methods in Engineering, 27: 523–546
Kalro, V.; Aliabadi, S.; Garrard, W.; Tezduyar, T.; Mittal, S.; Stein, K. 1996: Parallel finite element simulation of large ram-air parachutes. To appear in *International Journal for Numerical Methods in Fluids* See update U3 below.
Kalro, V.; Tezduyar, T. E. 1995: Parallel finite element computation of 3D incompressible flows on MPPs. In Habashi, W.G., editor, Solution Techniques for Large-Scale CFD Problems, John Wiley & Sons
Kashiyama, K.; Ito, H.; Behr, M.; Tezduyar, T. 1995: Three-step explicit finite element computation of shallow water flows on a massively parallel computer. International Journal for Numerical Methods in Fluids, 21: 885–900

Publication Updates
U1. A.A. Johnson and T.E. Tezduyar, "Parallel Computation of Incompressible Flows with Complex Geometries", *International Journal for Numerical Methods in Fluids*, **24** (1997) 1321-1340.
U2. A.A. Johnson and T.E. Tezduyar, "Simulation of Multiple Spheres Falling in a Liquid-Filled Tube", *Computer Methods in Applied Mechanics and Engineering*, **134** (1996) 351-373.
U3. V. Kalro, S. Aliabadi, W. Garrard, T. Tezduyar, S. Mittal, and K. Stein, "Parallel Finite Element Simulation of Large Ram-Air Parachutes", *International Journal for Numerical Methods in Fluids*, **24** (1997) 1353-1369.

Kato, C.; Ikegawa, M. 1991: Large eddy simulation of unsteady turbulent wake of a circular cylinder using the finite element method. In Celik, I.; Kobayashi, T.; Ghia, K. N.; Kurokawa, J., editors, Advances in Numerical Simulation of Turbulent Flows, FED-Vol. 117, pages 49–56, New York, ASME

Kennedy, J. G.; Behr, M.; Kalro, V.; Tezduyar, T. E. 1994: Implementation of implicit finite element methods for incompressible flows on the CM-5. Computer Methods in Applied Mechanics and Engineering, 119: 95–111

Kobayashi, T. 1992: A review of CFD methods and their application to automobile aerodynamics. In SAE Special Publication SP-908, Vehicle Aerodynamics: Wake Flows, CFD, and Aerodynamic Testing, pages 53–64

Le Beau, G. J.; Ray, S. E.; Aliabadi, S. K.; Tezduyar, T. E. 1993: SUPG finite element computation of compressible flows with the entropy and conservation variables formulations. Computer Methods in Applied Mechanics and Engineering, 104: 397–422

Le Beau, G. J.; Tezduyar, T. E. 1991: Finite element computation of compressible flows with the SUPG formulation. In Dhaubhadel, M. N.; Engelman, M. S.; Reddy, J. N., editors, Advances in Finite Element Analysis in Fluid Dynamics, FED-Vol. 123, pages 21–27, New York. ASME

Lingard, J. S. 1995: Ram-air parachute design. In AIAA 13th Aerodynamic Decelerator Conference, 2nd ADS Technology Seminar, Clearwater Beach, Florida

Liou, J.; Tezduyar, T. E. 1992: Clustered element-by-element computations for fluid flow. In Simon, H. D., editor, Parallel Computational Fluid Dynamics–Implementation and Results, Scientific and Engineering Computation Series, Chapter 9, pages 167–187, MIT Press, Cambridge, Massachusetts

Lynch, D. R. 1982: Unified approach to simulation on deforming elements with application to phase change problems. Journal of Computational Physics, 47(3): 387–411

Mathur, K. K.; Johnsson, S. L. 1992: Communication primitives for unstructured finite element simulations on data parallel architectures. Computer Systems in Engineering, 3: 63–71

Matsunaga, K.; Miyata, H.; Aoki, K.; Zhu, M. 1992: Finite-difference simulation of 3D vortical flows past road vehicles. In SAE Special Publication SP-908, Vehicle Aerodynamics: Wake Flows, CFD, and Aerodynamic Testing, pages 65–84

Mittal, S.; Tezduyar, T. E. 1994: Massively parallel finite element computation of incompressible flows involving fluid-body interactions. Computer Methods in Applied Mechanics and Engineering, 112: 253–282

Mittal, S.; Tezduyar, T. E. 1995: Parallel finite element simulation of 3d incompressible flows–Fluid-structure interaction. International Journal for Numerical Methods in Fluids, 21: 933–953

Morgan, K.; Peraire, J.; Peiró J. 1992: Unstructured grid methods for compressible flows. In Special Course on Unstructured Grid Methods for Advection Dominated Flows, pages 5-1–5-39. Advisory Group for Aerospace Research and Development

Muto, S. 1983: The aerodynamic drag coefficient of a passenger car and methods for reducing it. In Technological Advances in Vehicle Design Series, SP3; Impact of Aerodynamics on Vehicle Design, pages 57–69, International Journal of Vehicle Design

Özturan, C.; deCougny, H. L.; Shephard, M. S.; Flaherty, J. E. 1994: Parallel adaptive mesh refinement and redistribution on distributed memory computers. Computer Methods in Applied Mechanics and Engineering, 119: 123–137

Ray, S. E.; Wren, G. P.; Aliabadi, S. K.; Tezduyar, T. E. 1996: High performance computation of fluid-structure interaction and two-phase flow problems. In Proceedings of the 20th Army Science Conference, Norfolk, Virginia, to appear

Saad, Y.; Schultz, M. 1986: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal of Scientific and Statistical Computing, 7: 856–869

Salinger, , A. G.; Xiao, Q.; Zhou, Y.; Derby, J. J. 1994: Massively parallel finite element computations of three-dimensional, time-dependent, incompressible flows in materials processing systems. Computer Methods in Applied Mechanics and Engineering, 119: 139–156

Shakib, F. 1988: Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations. PhD thesis, Department of Mechanical Engineering, Stanford University

Shakib, S.; Hughes, T. J. R.; Johan, Z. 1989: A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis. Computer Methods in Applied Mechanics and Engineering, 75: 415–456

Shephard, M. S.; Georges, M. K. 1991: Automatic three-dimensional mesh generation by the finite octree technique. International Journal for Numerical Methods in Engineering, 32: 709–749

Smagorinsky, J. 1963: General circulation experiments with the primitive equations. Monthly Weather Review, 91(3): 99–165

Soulaimani, A.; Fortin, M.; Dhatt, G.; Ouellet, Y. 1991: Finite element simulation of two- and three-dimensional free surface flows. Computer Methods in Applied Mechanics and Engineering, 86: 265–296

Speziale, C. G. 1991: Analytical methods for the development of Reynolds-stress closures in turbulence. Annual Review of Fluid Mechanics, 23: 107–157

Tezduyar, T. E. 1991: Stabilized finite element formulations for incompressible flow computations. Advances in Applied Mechanics, 28: 1–44

Tezduyar, T. E.; Aliabadi, S.; Behr, M.; Johnson, A.; Kalro, V.; Litke, M. 1996: High performance computing techniques for flow simulations. In Papadrakakis, M., editor, Solving Large-Scale Problems in Mechanics: Parallel and Distributed Computer Applications, John Wiley & Sons, to appear

Tezduyar, T. E.; Aliabadi, S.; Behr, M.; Johnson, A.; Kalro, V.; Waters, C. 1995: 3D simulation of flow problems with parallel finite element computations on the Cray T3D. In *Computational Mechanics '95, Proceedings of International Conference on Computational Engineering Science*, Mauna Lani, Hawaii

Tezduyar, T.; Aliabadi, S.; Behr, M.; Johnson, A.; Mittal, S. 1992a: Massively parallel finite element computation of three-dimensional flow problems. In Proceedings of the 6th Japan Numerical Fluid Dynamics Symposium, pages 15–24, Tokyo, Japan

Tezduyar, T.; Aliabadi, S.; Behr, M.; Johnson, A.; Mittal, S. 1993: Parallel finite-element computation of 3D flows. IEEE Computer, 26-10(10): 27–36

Tezduyar, T. E.; Aliabadi, S. K.; Behr, M.; Mittal, S. 1994: Massively parallel finite element simulation of compressible and incompressible flows. Computer Methods in Applied Mechanics and Engineering, 119: 157–177

Tezduyar, T. E.; Behr, M.; Aliabadi, S. K.; Mittal, S.; Ray, S. E. 1992b: A new mixed preconditioning method for finite element computations. Computer Methods in Applied Mechanics and Engineering, 99: 27–42

Tezduyar, T. E.; Behr, M.; Liou, J. 1992c: 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(3): 339–351

Tezduyar, T. E.; Behr, M.; Mittal, S.; Johnson, A. A. 1992d: Computation of unsteady incompressible flows with the finite element methods–space-time formulations, iterative strategies and massively parallel implementations. In Smolinski, P.; Liu, W. K.; Hulbert, G.; Tamma, K., editors, New Methods in Transient Analysis, AMD-Vol. 143, pages 7–24, New York, ASME

Tezduyar, T. E.; Hughes, T. J. R. 1983: Finite element formulations for convection dominated flows with particular emphasis on the compressible Euler equations. In Proceedings of AIAA 21st Aerospace Sciences Meeting, AIAA Paper 83-0125, Reno, Nevada

Xiao, Q.; Salinger, A. G.; Zhou, Y.; Derby, J. J. 1995: Massively parallel finite element analysis of coupled, incompressible flows: A benchmark computation of baroclinic annulus waves. International Journal for Numerical Methods in Fluids, 21: 1007–1014