
ELSEVIER

Parallel Computing 25 (1999) 2039–2066

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Methods for parallel computation of complex flow problems☆

Tayfun Tezduyar*, Yasuo Osawa

Mechanical Engineering and Materials Science, Army HPC Research Center, Rice University, MS 321, 6100 Main Street, Houston, TX 77005, USA

Received 10 January 1999; received in revised form 15 March 1999

Abstract

This paper is an overview of some of the methods developed by the Team for Advanced Flow Simulation and Modeling (T★AFSM) [http://www.mems.rice.edu/TAFSM/] to support flow simulation and modeling in a number of "Targeted Challenges". The "Targeted Challenges" include unsteady flows with interfaces, fluid–object and fluid–structure interactions, airdrop systems, and air circulation and contaminant dispersion. The methods developed include special numerical stabilization methods for compressible and incompressible flows, methods for moving boundaries and interfaces, advanced mesh management methods, and multi-domain computational methods. We include in this paper a number of numerical examples from the simulation of complex flow problems.

Keywords: Computational fluid dynamics; Flow simulation; Stabilization methods; Compressible flow; Incompressible flow; Multidomain computational methods

1. Introduction

In this paper we provide an overview of some of the methods developed by the Team for Advanced Flow Simulation and Modeling (T★AFSM) [http://

☆ The work reported in this paper was partially sponsored by NSF (grant no. CTS-9896278), NASA JSC (grant no. NAG9-1049), AFOSR (contract no. F49620-98-1-0214) and by the AHPCRC under the auspices of the Department of the Army, ARL cooperative agreement no. DAAH04-95-2-0003/contract no. DAHH04-95-0008. The content does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

* Corresponding author. Tel.: +1-713-527-6051; fax: +1-713-285-5423.
E-mail address: tezduyar@rice.edu (T. Tezduyar).

0167-8191/99/$ - see front matter
PII: S0167-8191(99)00080-0

www.mems.rice.edu/TAFSM/]. These methods have been developed to support flow simulation and modeling in a number of "Targeted Challenges" identified by our team as areas of computational technology and science where we expect to make an impact. The "Targeted Challenges" include unsteady flows with interfaces, fluid–object and fluid–structure interactions, airdrop systems, and air circulation and contaminant dispersion.

Unsteady flows with interfaces can involve two-fluid (such as two different liquids or a liquid and a gas) or free-surface flows. For example, simulation of operational stability of vehicles carrying bulk liquids requires solution of this class of flow problems. The main challenge here is that the location of the interface is also an unknown and must be determined together with the solution of the Navier–Stokes equations. This class of problems is a subset of a larger class flow problems with moving boundaries and interfaces, where the location of these boundaries or interfaces is an unknown that needs to be computed as part of the overall solution.

Fluid–object interactions is another subset of flow problems with moving boundaries and interfaces. This class of problems involves interactions between solid objects and the fluid these objects are moving in. Also involved are the interactions between the objects themselves, such as collisions and groupings. In the modeling of this class of problems, the flow is governed by the Navier–Stokes equations. The three-dimensional dynamics of the objects is governed by Newton's Laws. The fluid forces acting on these particles are calculated from the computed flow field. The motion of the particles is influenced by fluid forces, and in turn influences the fluid behavior. Therefore the governing equations need to be solved in a coupled fashion. Most of our attention in this class of problems has been focused on fluid–particle interactions, where the particles are of spherical shape. Fluid–structure interactions involve fluids with moving boundaries and unsteady interfaces between the fluid and the structure. This is somewhat similar to fluid–object interactions. However now the "objects" are deformable, and these deformations need to be determined coupled with the solution of the Navier–Stokes equations. In this category, we have been focusing on parachute fluid–structure interactions, and fluid–structure interactions in interior flows with moving mechanical components.

Examples of airdrop systems are: aerodynamic behavior of round and ram-air parachutes, aerodynamic interaction between an aircraft and a paratrooper, and a parachute crossing the wake flow of an aircraft. These simulations involve Fluid–object and fluid–structure interactions. They also involve aerodynamics of complex shapes, and, in some cases, unsteady long-wake flows generated by such complex objects.

Examples of air circulation and contaminant dispersion are: how a contaminant introduced near a vehicle or a cluster of buildings spreads around that vehicle or those buildings; and how a contaminant spreads inside a building. These simulations involve solution of the Navier–Stokes equations interior or exterior to some complex geometries. After the flow field is determined, using the velocity field computed, a time-dependent advection–diffusion equation is solved to compute the time-evolution of the passive contaminant.

The methods developed to support simulation and modeling of the classes of problems described above include: special numerical stabilization methods for both compressible and incompressible flows, methods for moving boundaries and interfaces, advanced mesh management methods, iterative solution techniques for large nonlinear equation systems that need to be solved at every time step of a computation, parallel implementations, multi-domain computational methods, and space–time contact techniques. All methods developed are for flow problems involving complex geometries, and all software was developed and implemented on parallel platforms by the T★AFSM. All simulations, except those for testing a new method, are carried out in three-dimensional. Furthermore, all computations are performed on parallel computing platforms.

This overview article is largely based on earlier publications by the T★AFSM, particularly a recent overview article on flow simulation methods for complex flow problems [1]. In Section 2 we review the governing equations used in the computations. Stabilized finite element formulations and methods for computation of moving boundaries and interfaces are reviewed in Section 3. The deforming-spatial-domain/stabilized space–time (DSD/SST) formulation is reviewed in Section 4. The mesh update methods for the DSD/SST formulation are described in Section 5. In Section 6 we describe the enhanced-discretization interface-capturing technique (EDICT). Construction of the function spaces for the EDICT is described in Section 7. Section 8 provides an overview of the iterative solution methods and parallel computing platforms used. In Section 9 we briefly review the multi-domain method (MDM), designed to handle problems with long-wake flows. A new space–time contact technique (STCT) is introduced in Section 10. In Section 11 we report several examples of flow simulations.

2. Governing equations

In both compressible and incompressible flow cases, the governing equations used in numerical modeling are the time-dependent Navier–Stokes equations. The space and time domains will be denoted by Ω_t and $(0,T)$, where Γ_t is the boundary of Ω_t. In some cases the spatial domain may change with respect to time, and the subscript t indicates such time-dependence. This will be the case if in the formulation addressing flows with moving boundaries and interfaces the spatial domain is defined to be the part of the space occupied by the fluid(s). 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., the gravity) are represented by $\mathbf{f}(\mathbf{x},t)$.

2.1. Compressible flows

The Navier–Stokes equations of compressible flows can be written as

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

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 δ_{ij} are the components of the identity tensor, q_i the components of the heat flux vector, and $[\mathbf{T}]_{ij}$ 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. The equation of state corresponds to the ideal gas assumption.

Eq. (1) can further be written in the following form:

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

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 as

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

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

where ρ is assumed to be constant, and

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

where **I** is the identity tensor. This equation set is completed with an appropriate set of boundary conditions and an initial condition consisting of a divergence-free velocity field specified over the entire domain

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

If the problem does not involve any moving boundaries or interfaces, the spatial domain does not need to change with respect to time, and the subscript t can be dropped from Ω_t and Γ_t. This might be the case even for flows with moving boundaries and interfaces if in the formulation used the spatial domain is not defined to be the part of the space occupied by the fluid(s). For example, we can select a fixed spatial domain, and model the fluid–fluid interfaces by assuming that the domain is occupied by two immiscible fluids, A and B, with densities ρ_A and ρ_B and viscosities μ_A and μ_B.

Remark 1. When we model a liquid–gas interaction, we let Fluid A be the liquid and Fluid B the gas. If we model a free-surface problem where Fluid B is irrelevant, we assign a sufficiently low density to Fluid B.

An interface function ϕ serves as a marker identifying Fluids A and B with the definition $\phi=\{1$ for Fluid A and 0 for Fluid B$\}$. The interface between the two fluids is approximated to be at $\phi=0.5$. In this context, ρ and μ are defined as

$$\rho=\phi\rho_A+(1-\phi)\rho_B, \tag{12}$$

$$\mu=\phi\mu_A+(1-\phi)\mu_B. \tag{13}$$

The evolution of the interface function ϕ, and therefore the motion of the interface, is governed by a time-dependent advection equation:

$$\frac{\partial\phi}{\partial t}+\mathbf{u}\cdot\nabla\phi=0 \quad \text{on } \Omega \quad \forall t\in(0,T), \tag{14}$$

Remark 2. One can also see Eqs. (12)–(14) as those representing the constitutive law of the fluid system. How accurately this law will be modeled will depend on how accurately the front between $\phi=1$ and $\phi=0$ will be represented and advected.

Remark 3. We will not address here the surface tension effects at the interfaces.

3. Interface-tracking and interface-capturing methods, stabilized finite element formulations

In computation of flow problems with moving boundaries and interfaces, depending on the nature of the problem, we can use an interface-tracking or interface-capturing method. An interface-tracking method requires meshes which "track" the interfaces. The mesh needs to be updated as the flow evolves. In an interface-cap-

turing method, the computations are based on fixed spatial domains, where an interface function, such as the one described in Section 2, needs to be computed to "capture" the interface. The interface is captured within the resolution of the finite element mesh covering the area where the interface is.

The Deformable-Spatial Domain/Stabilized Space-Time (DSD/SST) formulation is an interface-tracking method, and was first introduced in [2,3] in the context incompressible flows, followed by the version for compressible flows [4]. In the DSD/SST method the finite element formulation of the problem is written over its associated space–time domain. This automatically takes into account the motion of the boundaries and interfaces. At each time step of a computation, the locations of the boundaries and interfaces are calculated as part of the overall solution.

The interface-tracking and interface-capturing methods described in this paper are based on stabilization techniques. These stabilization techniques are the streamline-upwind/Petrov–Galerkin (SUPG) [5–9] pressure-stabilizing/Petrov–Galerkin (PSPG) [10,11], and Galerkin/least-squares (GLS) [2,3,12,13] formulations. The SUPG method is one of the earliest and most widely used stabilized methods. The SUPG formulation for incompressible flows was first introduced in [5]. The SUPG formulation for compressible flows, on the other hand, was first introduced, in the context of conservation variables, in 1983 in [6]. Following that, several researches designed and studied SUPG-like methods for compressible flows. For example, the Taylor–Galerkin method, which appeared in the literature in 1984, is very similar, and under some conditions identical, to one of the SUPG methods introduced in [6]. The PSPG formulation was introduced in [10] and assures numerical stability while allowing us to use equal-order interpolation functions for velocity and pressure and other unknowns. These stabilization techniques prevent numerical oscillations and instabilities when the flow involves high Reynolds and/or Mach numbers and strong shocks and boundary layers. In SUPG, PSPG and GLS formulations, the stabilization is accomplished without introducing excessive numerical dissipation (i.e. without "overstabilizing"). Overstabilizing is not always easy to be fully aware of, as the symptoms are not necessarily qualitative. The SUPG, PSPG and GLS formulations were developed with this concern in mind, and perform quite well when the implementation is based on a sound understanding of these methods.

4. DSD/SST formulation

In the DSD/SST method, the finite element formulation of the governing equations is written over a sequence of N 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}. At each time step, the integrations involved in finite element formulation are performed over Q_n. The finite element interpolation functions are discontinuous across the space–time slabs. In the computations reported here, we use first-order polynomials as interpolation functions. 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 Q_n is decomposed into space–

time elements Q_n^e, where $e = 1, 2, \ldots, (n_{\text{el}})_n$. 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 another.

The DSD/SST formulations for compressible and incompressible flows are based on the same concepts and look very similar (see [2–4,14]. We review them here for completeness.

4.1. DSD/SST formulation for compressible flows

For each slab Q_n, we define appropriate finite-dimensional space–time function spaces $\mathscr{S}_n^h$ and $\mathscr{V}_n^h$ corresponding to the trial solutions and weighting functions, respectively. While the superscript h implies that these are finite-dimensional function spaces, the subscript n implies that corresponding to different space–time slabs we might have different spatial discretizations. The DSD/SST 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 $\forall \mathbf{W}^h \in \mathscr{V}_n^h$:

$$\begin{aligned}
&\int_{Q_n} \mathbf{W}^h \cdot \left(\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \right) \mathrm{d}Q + \int_{Q_n} \left(\frac{\partial \mathbf{W}^h}{\partial x_i} \right) \cdot \left(\mathbf{K}_{ij}^h \frac{\partial \mathbf{U}^h}{\partial x_j} \right) \mathrm{d}Q \\
&\quad + \int_{\Omega_n} (\mathbf{W}^h)_n^+ \cdot \left((\mathbf{U}^h)_n^+ - (\mathbf{U}^h)_n^- \right) \mathrm{d}\Omega + \sum_{e=1}^{(n_{\text{el}})_n} \int_{Q_n^e} \boldsymbol{\tau} (\mathbf{A}_k^h)^{\mathrm{T}} \left(\frac{\partial \mathbf{W}^h}{\partial x_k} \right) \\
&\quad \cdot \left[\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} - \frac{\partial}{\partial x_i} \left(\mathbf{K}_{ij}^h \frac{\partial \mathbf{U}^h}{\partial x_j} \right) \right] \mathrm{d}Q + \sum_{e=1}^{(n_{\text{el}})_n} \int_{Q_n^e} \delta \left(\frac{\partial \mathbf{W}^h}{\partial x_i} \right) \\
&\quad \cdot \left(\frac{\partial \mathbf{U}^h}{\partial x_i} \right) \mathrm{d}Q = \int_{P_n} \mathbf{W}^h \cdot H^h \, \mathrm{d}P.
\end{aligned} \tag{15}$$

Here H^h represents the Neumann-type boundary condition, P_n is the lateral boundary of the space–time slab, and $\boldsymbol{\tau}$ and δ are the stabilization parameters. The solution to (15) is obtained sequentially for all space–time slabs $Q_0, Q_1, Q_2, \ldots, Q_{N-1}$, and the computations start with

$$(\mathbf{U}^h)_0^- = \mathbf{U}_0^h, \tag{16}$$

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

The first three integrals, together with the right-hand side, represent the time-discontinuous Galerkin formulation of (5), where 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. The details regarding the stabilization and the space–time formulation can be found in [4]. For problems not involving moving boundaries and interfaces, Eq. (15) can be reduced to a semi-discrete formulation by dropping the third integral and converting all space–time integrations to spatial integrations.

4.2. DSD/SST formulation for incompressible flows

The trial function spaces for velocity and pressure will be denoted by $(\hat{\mathcal{S}}^h_{\mathbf{u}})_n$ and $(\hat{\mathcal{S}}^h_p)_n$. The weighting function spaces corresponding to momentum equation and incompressibility constraint will be denoted by $(\hat{\mathcal{V}}^h_{\mathbf{u}})_n$ and $(\hat{\mathcal{V}}^h_p)_n$ $(=(\hat{\mathcal{S}}^h_p)_n)$. The DSD/SST formulation of Eqs. (8) and (9) can be written as follows: given $(\mathbf{u}^h)^-_n$, find $\mathbf{u}^h \in (\hat{\mathcal{S}}^h_{\mathbf{u}})_n$ and $p^h \in (\hat{\mathcal{S}}^h_p)_n$ such that $\forall \mathbf{w}^h \in (\hat{\mathcal{V}}^h_{\mathbf{u}})_n$ and $\forall q^h \in (\hat{\mathcal{V}}^h_p)_n$:

$$\begin{aligned}
&\int_{Q_n} \mathbf{w}^h \cdot \rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}\right) \mathrm{d}Q + \int_{Q_n} \boldsymbol{\varepsilon}(\mathbf{w}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\, \mathrm{d}Q \\
&\quad + \int_{Q_n} q^h \nabla \cdot \mathbf{u}^h \, \mathrm{d}Q + \int_{\Omega_n} (\mathbf{w}^h)^+_n \cdot \rho\left((\mathbf{u}^h)^+_n - (\mathbf{u}^h)^-_n\right) \mathrm{d}\Omega \\
&\quad + \sum_{e=1}^{(n_{\mathrm{el}})_n} \int_{Q^e_n} \tau_{\mathrm{MOM}} \frac{1}{\rho}\left[\rho\left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{w}^h\right) - \nabla \cdot \boldsymbol{\sigma}(q^h, \mathbf{w}^h)\right] \\
&\quad \cdot \left[\rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}\right) - \nabla \cdot \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\right] \mathrm{d}Q \\
&\quad + \sum_{e=1}^{(n_{\mathrm{el}})_n} \int_{Q^e_n} \tau_{\mathrm{CONT}} \nabla \cdot \mathbf{w}^h \rho \nabla \cdot \mathbf{u}^h \, \mathrm{d}Q = \int_{P_n} \mathbf{w}^h \cdot \mathbf{h}^h \, \mathrm{d}P.
\end{aligned} \tag{17}$$

Here $\mathbf{h}^h$ represents the Neumann-type boundary condition associated with the momentum equation, and τ_{MOM} and τ_{CONT} are the stabilization parameters.

The solution to (17) is obtained sequentially for all space–time slabs $Q_0, Q_1, Q_2, \ldots, Q_{N-1}$, and the computations start with

$$(\mathbf{u}^h)^-_0 = \mathbf{u}^h_0. \tag{18}$$

The first four integrals, together with the right-hand side, represent the time-discontinuous Galerkin formulation of (8) and (9), where 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 corresponding to momentum equation and incompressibility constraint.

The reader can refer to Tezduyar et al. [2,3] and Behr and Tezduyar [14] 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. (17) 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.

5. Mesh update for interface-tracking methods

In interface-tracking methods, as the computations proceed, the mesh needs to be updated to accommodate the changes in the spatial domain. It is essential that this is accomplished as effectively as possible. How the mesh can best be updated depends

on several factors, such as the complexity of the interface and overall geometry, how unsteady the interface is, and how the starting mesh was generated. In general, the mesh update could have two components: moving the mesh as much as it is possible, and remeshing (i.e. generating fully or partially a new set of nodes and elements) when the element distortion becomes too high.

Most real-world problems require simulations with complex geometries. A complex geometry typically requires an automatic mesh generator to start with. We developed our own automatic mesh generator to have a number of special features, such as structured layers of elements around solid surfaces and high-speed mesh generation. The automatic, three-dimensional mesh generator we have developed was described in [15]. It has been used very effectively in a number of simulations (for example see [16,17]). The latest version of this mesh generator can generate approximately 0.67 million elements/min on a modern PC (400 MHz-PentiumII). It also has the capability to generate meshes for spatially periodic domains. This automatic mesh generator has the capability to provide structured layers of elements around solid objects with reasonable geometric complexity. With this capability, we can fully control the mesh resolution near solid objects. This feature can be used for more accurate representation of the boundary layers.

Automatic mesh generation might become an overwhelming cost especially when the number of elements become very large or when frequency of remeshing has to be high. Sometimes special-purpose mesh generators designed for specific problems can be used. Depending on the complexity of the problem, such mesh generators might involve a high initial design cost, but minimal mesh generation cost. We selected this path in a number of our simulations, and were able to overcome the mesh generation issues very effectively (see for example [18]).

In mesh moving strategies, the only rule the mesh motion needs to follow is that at the interface the normal velocity of the mesh has to match the normal velocity of the fluid. Beyond that, the mesh can be moved in any way desired, with the main objective being to reduce the frequency of remeshing. In three-dimensional simulations, if the remeshing requires calling an automatic mesh generator, the cost of automatic mesh generation becomes a major reason for trying to reduce the frequency of remeshing. Furthermore, when we remesh, we need to project the solution from the old mesh to the new one. This introduces projection errors. Also, in three-dimensional, the computing time consumed by this projection step is not a trivial one. All these factors constitute a strong motivation for designing mesh update strategies which minimize the frequency of remeshing.

In some cases where the changes in the shape of the computational domain allow it, a special-purpose mesh moving method can be used in conjunction with a special-purpose mesh generator. In such cases, simulations can be carried out without calling an automatic mesh generator and without solving any additional equations to determine the motion of the mesh. An earlier example, three-dimensional parallel computation of sloshing in a vertically vibrating container, can be found in [14].

In general, however, we use an automatic mesh moving scheme [19,20] to move the nodal points, as governed by the equations of linear elasticity. The motion of the internal nodes is determined by solving these additional equations, with the

boundary conditions for these mesh motion equations specified in such a way that they match the normal velocity of the fluid at the interface. The structured layers of elements generated around solid objects (mentioned above) move "glued" to these solid objects. No equations are solved for the motion of the nodes in these layers, because these nodal motions are not governed by the equations of elasticity. This also results in some cost reduction. But more importantly, the user continues to have full control of the mesh resolution in these layers. For early examples of automatic mesh moving combined with structured layers of elements, see [15,16].

6. Enhanced-discretization interface-capturing technique (EDICT)

With interface-tracking methods, sometimes the interface might be too complex or unsteady to track while keeping the frequency of remeshing at an acceptable level. Not being able to reduce the frequency of remeshing in three-dimensional might introduce overwhelming mesh generation and projection costs, making the computations with the interface-tracking method no longer feasible. In such cases, interface-capturing methods, which do not normally require costly mesh update techniques, could be used with the understanding that the interface will not be represented as accurately as we would have with an interface-tracking method (for related discussions see [21–23]). Not needing a mesh update strategy makes the interface-capturing methods more flexible than the interface-tracking methods. However, for comparable levels of spatial discretization, interface-capturing methods yield less accurate representation of the interface. These methods can be used as practical alternatives to carry out the simulations when compromising the accurate representation of the interfaces becomes less of a concern than facing major difficulties in updating the mesh to track such interfaces. The desire to increase the accuracy of our interface-capturing methods without adding a major computational cost lead us to seeking techniques with a different kind of "tracking".

The Enhanced-Discretization Interface-Capturing Technique (EDICT) was first introduced in [24] with this kind of philosophy. The objective was to enhance the spatial discretization around an interface so that we could have higher accuracy in representing that interface. We start with the basic approach of an interface-capturing technique such as the volume of fluid (VOF) method [25]. The Navier–Stokes equations are solved over a non-moving mesh with an interface function serving as a marker identifying the two fluids (see Section 3).

In writing the stabilized finite element formulation for the EDICT (see [21]), the notation we use for representing the finite-dimensional function spaces is very similar to the one we used in Section 4. The trial function spaces corresponding to velecity, pressure and interface function are denoted, respectively, by $(\mathcal{S}^h_{\mathbf{u}})_n$, $(\mathcal{S}^h_p)_n$, and $(\mathcal{S}^h_\phi)_n$. The weighting function spaces corresponding to the momentum equation, incompressibility constraint and time-dependent advection equation are denoted by $(\mathcal{V}^h_{\mathbf{u}})_n$, $(\mathcal{V}^h_p)_n$ $(=(\mathcal{S}^h_p)_n)$, and $(\mathcal{V}^h_\phi)_n$. The subscript n in this case allows us to use different spatial discretizations corresponding to different time levels. We will give more precise definition of these function spaces in Section 7.

The stabilized formulations of Eqs. (8), (9), and (14) can be written as follows: given $\mathbf{u}_n^h$ and ϕ_n^h, find $\mathbf{u}_{n+1}^h \in (\mathscr{S}_{\mathbf{u}}^h)_{n+1}$, $p_{n+1}^h \in (\mathscr{S}_p^h)_{n+1}$, and $\phi_{n+1}^h \in (\mathscr{S}_\phi^h)_{n+1}$, such that, $\forall \mathbf{w}_{n+1}^h \in (\mathscr{V}_{\mathbf{u}}^h)_{n+1}$, $\forall q_{n+1}^h \in (\mathscr{V}_p^h)_{n+1}$, and $\forall \psi_{n+1}^h \in (\mathscr{V}_\phi^h)_{n+1}$:

$$\begin{aligned}
&\int_\Omega \mathbf{w}_{n+1}^h \cdot \rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}\right) \mathrm{d}\Omega + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w}_{n+1}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\, \mathrm{d}\Omega + \int_\Omega q_{n+1}^h \nabla \cdot \mathbf{u}^h \, \mathrm{d}\Omega \\
&\quad + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \left(\tau_{\mathrm{SUPG}} \mathbf{u}^h \cdot \nabla \mathbf{w}_{n+1}^h + \frac{\tau_{\mathrm{PSPG}}}{\rho} \nabla q_{n+1}^h\right) \cdot \left[\rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}\right)\right. \\
&\quad \left. - \nabla \cdot \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\right] \mathrm{d}\Omega + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \tau_{\mathrm{CONT}} \nabla \cdot \mathbf{w}_{n+1}^h \rho \nabla \cdot \mathbf{u}^h \, \mathrm{d}\Omega = \int_\Gamma \mathbf{w}_{n+1}^h \cdot \mathbf{h}^h \, \mathrm{d}\Gamma,
\end{aligned} \tag{19}$$

$$\begin{aligned}
&\int_\Omega \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \nabla \phi^h\right) \mathrm{d}\Omega \\
&\quad + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \tau_\phi \mathbf{u}^h \cdot \nabla \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \nabla \phi^h\right) \mathrm{d}\Omega = 0,
\end{aligned} \tag{20}$$

where τ_{SUPG}, τ_{PSPG}, τ_{CONT} and τ_ϕ are the stabilization parameters:

$$\tau_{\mathrm{SUPG}} = \left(\left(\frac{2\|\mathbf{u}^h\|}{h}\right)^2 + \left(\frac{4\nu}{h^2}\right)^2\right)^{-(1/2)}, \tag{21}$$

$$\tau_{\mathrm{PSPG}} = \tau_{\mathrm{SUPG}}, \tag{22}$$

$$\tau_{\mathrm{CONT}} = \frac{h}{2}\|\mathbf{u}^h\| z, \tag{23}$$

where

$$z = \begin{cases} \left(\frac{Re_u}{3}\right) & Re_u \leqslant 3, \\ 1 & Re_u > 3, \end{cases}$$

$$\tau_\phi = \frac{h}{2\|\mathbf{u}^h\|}, \tag{24}$$

where Re_u is the cell Reynolds number.

Remark 4. In Eq. (19), the first three integrals, together with the right-hand side, represent the Galerkin formulation of (8) and (9). The first series of element-level integrals in the formulation are the SUPG and PSPG stabilization terms. The second series of element-level integrals are the least-squares stabilization terms based on the incompressibility constraint. In Eq. (20), the first integral represents the Galerkin formulation of (14), while the series of element-level integrals are the SUPG stabilization terms.

Remark 5. In time discretization, the time derivatives, $\partial\mathbf{u}/\partial t$ and $\partial\phi/\partial t$ are represented as follows:

$$\frac{\partial \mathbf{u}}{\partial t} = \frac{\mathbf{u}^h_{n+1} - \mathbf{u}^h_n}{\Delta t}, \tag{25}$$

$$\frac{\partial \phi}{\partial t} = \frac{\phi^h_{n+1} - \phi^h_n}{\Delta t}, \tag{26}$$

where Δt is the time step size between time levels n and $n+1$. In this time discretization, the functions $\mathbf{u}^h$, p^h and ϕ^h are represented as follows:

$$\mathbf{u}^h \leftarrow (1-\alpha)\mathbf{u}^h_n + \alpha\mathbf{u}^h_{n+1}, \tag{27}$$

$$p^h \leftarrow p^h_{n+1}, \tag{28}$$

$$\phi^h \leftarrow (1-\alpha)\phi^h_n + \alpha\phi^h_{n+1}, \tag{29}$$

where α is a time-integration parameter controlling the stability and accuracy of the integration. Normally we set $\alpha = 0.5$.

To have a sharper representation of the interface function ϕ, we incorporate to the formulation a two-step interface-sharpening algorithm [23]. In the first step, unacceptable values of ϕ are filtered out. Since we want $0 \leqslant \phi \leqslant 1$, any value of ϕ less than 0 or greater than 1 are clipped and set to 0 and 1, respectively. In the second step, $\phi = 0.5$ is assumed to be the interface and ϕ is sharpened around 0.5 as follows:

$$\text{for } |\phi_{n+1} - \phi_n| \leqslant b:$$

$$\begin{aligned} &\phi \leftarrow 2^{a-1}\phi^a \quad \text{for } 0.0 \leqslant \phi \leqslant 0.5, \\ &\phi \leftarrow 1.0 - 2^{a-1}(1.0-\phi)^a \quad \text{for } 0.5 < \phi \leqslant 1.0, \end{aligned} \tag{30}$$

where b and a are user-defined parameters. In computations reported in this paper, $b = 0.1$ and $a = 2.0$.

7. Construction of function spaces for EDICT

The basic concept behind the construction of the function spaces used for EDICT is not complicated. However, the description requires certain formalism and involves a number of guidelines. For completeness, we repeat them here from [21]. In constructing the function spaces corresponding to time level n, we start with a base mesh (Mesh-1), with the set of elements and nodal points denoted by ϵ^1_n and η^1_n. The subscript n implies that Mesh-1 itself might change from one-time level to other.

A second-level and more refined mesh (Mesh-2) is constructed over a subset $(\epsilon^1_n)^2_n$ of these elements. Mesh-2 is generated by patching together the second-level meshes generated over each of the elements in $(\epsilon^1_n)^2_n$ (see Fig. 1).

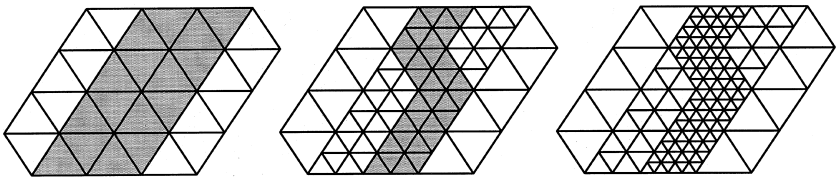

Fig. 1. EDICT multi-level meshes.

Remark 6. The second subscript n implies that for a given Mesh-1, the elements of this mesh which are declared to be in $(\epsilon_n^1)_n^2$ might change from one-time level to other. An element which might be declared to be in $(\epsilon_n^1)_n^2$ at some time level, might fall out of it at some other time, and yet come back in again some time later.

Remark 7. For each element in ϵ_n^1, there will be a unique second-level mesh. Therefore, if an element is declared to be in $(\epsilon_n^1)_n^2$ for a second time, the refined mesh generated over that element at the earlier declaration can be reused. If an automatic mesh generator is being used to generate these second-level meshes, the cost for that mesh generation will be a one-time cost.

The set of elements and nodal points for Mesh-2 are denoted by ϵ_n^2 and η_n^2.

A third-level and even more refined mesh (Mesh-3) is constructed over a subset $(\epsilon_n^2)_n^3$ of the elements in Mesh-2. This will be generated by patching together the third-level meshes generated over each of the elements in $(\epsilon_n^2)_n^3$ (see Fig. 1).

Remark 8. The statements in Remarks 6 and 7 apply, with the mesh-level numbers referred to in Remarks 6 and 7 shifted up by one.

The set of elements and nodal points for Mesh-3 are denoted by ϵ_n^3 and η_n^3.

Remark 9. At this time we limit ourselves to linear elements in two-dimensional and three-dimensional, and bilinear and trilinear elements, respectively, in two-dimensional and three-dimensional.

We construct $\mathbf{u}_n^h$ as follows:

$$\mathbf{u}_n^h = \mathbf{u}_n^1 + \mathbf{u}_n^2. \tag{31}$$

The function $\mathbf{u}_n^1$ comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^1, excluding those "surrounded" by the elements in $(\epsilon_n^1)_n^2$. The function $\mathbf{u}_n^1$ also needs to satisfy the Dirichlet-type boundary conditions, except at those nodes that have been surrounded at the boundary of Ω. The function $\mathbf{u}_n^2$ comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^2, excluding those at the boundaries of the zones covered by the elements in ϵ_n^2. However we do include the

nodes at the boundary of Ω unless they coincide with the nodes in η_n^1 that have not been surrounded.

We construct p_n^h in exactly the same way, except for recognizing the fact that the references to Dirichlet-type boundary conditions do not apply:

$$p_n^h = p_n^1 + p_n^2. \tag{32}$$

We construct ϕ_n^h with more enhancement:

$$\phi_n^h = \phi_n^1 + \phi_n^2 + \phi_n^3. \tag{33}$$

The function ϕ_n^1 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^1, excluding those "surrounded" by the elements in $(\epsilon_n^1)_n^2$. The function ϕ_n^1 also needs to satisfy the Dirichlet-type boundary conditions, except at those nodes that have been surrounded at the boundary of Ω. The function ϕ_n^2 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^2, excluding those at the boundaries of the zones covered by the elements in ϵ_n^2. We also exclude the nodes surrounded by the elements in $(\epsilon_n^2)_n^3$. However we do include the nodes at the boundary of Ω unless they coincide with the nodes in η_n^1 that have not been surrounded. The function ϕ_n^3 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^3, excluding those at the boundaries of the zones covered by the elements in ϵ_n^3. However we do include the nodes at the boundary of Ω unless they coincide with the nodes in η_n^2 that have not been surrounded.

The weighting functions are constructed in a similar fashion:

$$\mathbf{w}_n^h = \mathbf{w}_n^1 + \mathbf{w}_n^2, \tag{34}$$

$$q_n^h = q_n^1 + q_n^2, \tag{35}$$

$$\psi_n^h = \psi_n^1 + \psi_n^2 + \psi_n^3. \tag{36}$$

The components of each weighting function are defined in the same way as we did for the trial functions, except that the weighting functions need to satisfy the homogeneous form of the Dirichlet-type boundary conditions.

Our objective with this enhanced discretization is to capture the interface as accurately as possible by using more refined meshes for the velocity, pressure, and interface function, and possibly even more refined meshes for the interface function. This is done in a dynamic fashion by defining $(\epsilon_n^1)_n^2$ and $(\epsilon_n^2)_n^3$ depending on which elements in ϵ_n^1 and ϵ_n^2 the interface is passing through, and re-define these subsets occasionally to track the interface.

Remark 10. We update $(\epsilon_n^1)_n^2$ and $(\epsilon_n^2)_n^3$ not every time step but with sufficient frequency to keep the interface within the zones covered by these subsets of elements. How many time steps one can carry out the simulation without re-defining these subsets of elements will depend on, among other things, how "wide" we decide to keep these zones around the interface.

Remark 11. Whenever we redefine these subsets the mesh generation cost will not be a significant one. If we are using an automatic mesh generator for the second- and third-level meshes, we will be able to use and re-use the meshes which were generated (and stored) the first time these meshes were needed.

Remark 12. It is possible to eliminate ϕ_n^3 by not choosing to go to a third-level of refinement. It is also possible to design the second- and third-level meshes in such a way that they overlap. One of the advantages in keeping them as non-overlapping meshes is that, by keeping Mesh-2 "wider" than Mesh-3, one can chose to limit the existence (as an unknown) of ϕ to Mesh-2, and therefore solving for it only over the part of the computational domain covered by Mesh-2. With this, we have to make sure that the interface remains in Mesh-2 zone. Since our objective will be to keep the interface in Mesh-3 zone, this would also keep it in Mesh-2 zone, even if the interface occasionally falls slightly out of the Mesh-3 zone.

8. Iterative solution methods and parallel computing platforms

The finite element formulations reviewed in the earlier sections fall into two categories: a space–time formulation with moving meshes or a semi-discrete formulation with non-moving meshes. Full discretizations of these formulations lead to coupled, nonlinear equation systems which need to be solved at every time step of the simulation. Whether we are using a space–time formulation or a semi-discrete formulation we can represent the equation system that needs to be solved as follows:

$$\mathbf{N}(\mathbf{d}_{n+1}) = \mathbf{F}. \tag{37}$$

Here $\mathbf{d}_{n+1}$ is the vector of nodal unknowns. In a semi-discrete formulation, this vector contains the unknowns associated with marching from time level n to $n+1$. In a space–time formulation, it contains the unknowns associated with the finite element formulation written for the space–time slab Q_n. The time-marching formulations described earlier can also be used for solving a steady-state flow problem. In such cases we call the time steps the "pseudo time steps", and solve Eq. (37) at every pseudo time step. In most of our simulations the number of unknowns in Eq. (37) is around 1–10 million. In some of our computations we exceed 100 million equations.

We solve Eq. (37) with the Newton–Raphson method:

$$\left.\frac{\partial \mathbf{N}}{\partial \mathbf{d}}\right|_{\mathbf{d}_{n+1}^k} \left(\Delta \mathbf{d}_{n+1}^k\right) = \mathbf{F} - \mathbf{N}\left(\mathbf{d}_{n+1}^k\right), \tag{38}$$

where k is the step counter for the Newton–Raphson sequence, and $\Delta \mathbf{d}_{n+1}^k$ is the increment computed for $\mathbf{d}_{n+1}^k$. The linear equation system represented by (38) needs to be solved at every step of the Newton–Raphson sequence. We can represent Eq. (38) as as linear equation system of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{39}$$

In the class of computations we typically carry out, this equation system would be too large to solve with a direct method. Therefore we solve it iteratively. At each iteration, we need to compute the residual of this system,

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}. \tag{40}$$

This can be acheived in several different ways. The computation can be based on a sparse-matrix storage of **A**. It can also be based on storing just element-level matrices, or even just element-level vectors. This last strategy is also called a matrix-free technique. After the residual computation, we compute a candidate correction to **x** as given by the expression

$$\Delta\mathbf{y} = \mathbf{P}^{-1}\mathbf{r}, \tag{41}$$

where **P**, the preconditioning matrix, is an approximation to **A**. **P** has to be simple enough to form and factorize efficiently. However, it also has to be sophisticated enough to yield a desirable convergence rate. How to update the solution vector **x** by using $\Delta\mathbf{y}$ is also a major subject in iterative solution techniques. Several update methods are available, and we use the GMRES [26] method. We have been focusing our research related to iterative methods mainly on computing the residual **r** efficiently and selecting a good preconditioner **P**. While moving in this direction, we have always been keeping in mind that the iterative solution methods we develop need to be implemented on parallel computing platforms. For example, the parallel methods we designed for the residual computations include those which are element-matrix-based [27], element-vector-based [27], and sparse-matrix-based [28].

In preconditioning design, we developed some advanced preconditioners such as the clustered-element-by-element (CEBE) preconditioner [29] and the mixed CEBE and cluster companion (CC) preconditioner [30]. We have implemented, with quite satisfactory results, the CEBE preconditioner in conjunction with an ILU approximation [28]. However, our typical computations are based on diagonal and nodal-block-diagonal preconditioners. These are very simple preconditioners, but are also very simple to implement on parallel platforms.

Some of our parallel computations are based on shared-memory platforms, such as a 12-processor SGI POWER CHALLENGE, a 20-processor SGI ONYX, and, more recently, a 2-processor SGI ONYX2. Most of our simulations are carried out on distributed-memory platforms. In this category, our earlier computations were based on data-parallel paradigm on a 512-node Thinking Machines CM-5. The majority of our more recent computations is based on message-passing paradigm on computing platforms such as a 256-node CRAY T3E. More on our parallel implementations can be found in [27].

9. Multi-domain method (MDM) for computation of long-wake flows

Methods designed for a general class of challenging flow problems sometimes need to be re-designed and/or optimized for more specific classes of problems which pose their own specific challenges. One of these specific classes of problems is three-

dimensional simulation of unsteady wake flow in the far downstream of an object. The analysis of the far wake behavior is important to understand its effect on secondary objects.

For example, as paratroopers are deployed from aircraft flying in formation, the paratrooper jumping from a trailing aircraft crosses the unsteady wake flow generated by the leading aircraft. This could subject a paratrooper/parachute system to destabilizing aerodynamical forces. Similarly, a small, trailing aircraft might face the destabilizing wake of the larger, leading aircraft. In both cases, simulations would involve computation of unsteady wake flow in the long region between the two objects, and the influence of this wake flow on the secondary object. This creates a major computational challenge because the two objects are separated by a large distance compared to the length scales of the objects, and therefore unsteady flows need to be computed accurately over long-wake regions.

We have developed the Multi-Domain Method (MDM) [31] to address this challenge. The simulation domain is divided into an ordered sequence of overlapping subdomains. Subdomain-1 (SD-1) and Subdomain-3 (SD-3) would be used for computation of flow around the primary and secondary objects, respectively (see Fig. 2). Because these objects would have complex geometries, such as an aircraft or a parachute, these domains are discretized with unstructured meshes, and the flow solver is based on a general-purpose finite element implementation. Subdomain-2 (SD-2), which connects SD-1 and SD-3, would be used for computation of the wake flow generated by the primary object. SD-2 is discretized with highly refined structured meshes. A special-purpose finite element implementation for structured meshes has been optimized to yield much higher computational speeds compared to a general-purpose implementation. Computation over SD-2 can also be accomplished by other methods which might be more desirable for structured grids.

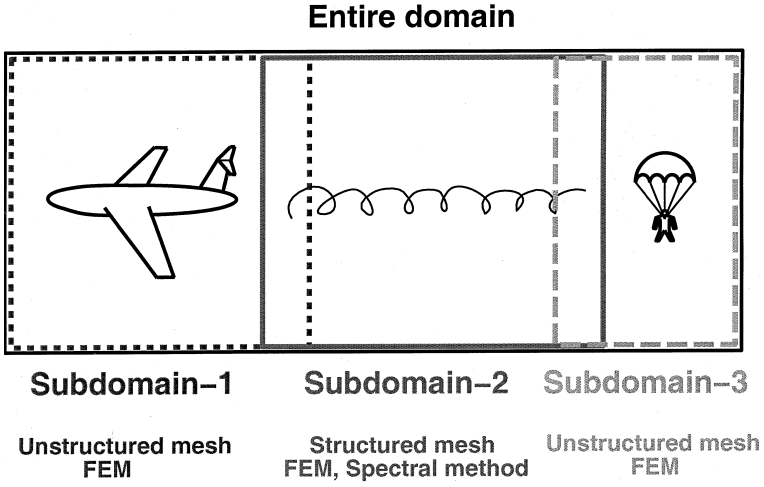

Fig. 2. MDM: The basic concept.

10. Space-time contact technique (STCT)

In this section we introduce the Space-Time Contact Technique (STCT) for computation of time-dependent problems with contacting and de-contacting surfaces. The DSD/SST formulation, combined with the STCT, provides a natural mechanism to handle contact and de-contact problems in an implicit fashion. Formulating this class of problems in the space–time domain can help us develop more effective methods.

Let us imagine a one-dimensional problem, shown in Fig. 3, where there is a possibility that between the time levels t_n and t_{n+1} the liquid free surface on the right might be contacting a wall. Let us first perform some intermediate calculations for this space–time slab in which we update the positions of the free-surface nodes by assuming that the motion of these free-surface nodes are not constrained by the wall. Let us say that these calculations show that the new position of Node-1 at t_{n+1} is at location $(x_1)_2$, which is beyond the wall. Next, on the wall, we predict the temporal position of Node-3. Node-3 represents the contact point in the space–time domain. We calculate this predicted value of $t_3 - t_1$ from $(x_1)_3 - (x_1)_1$ and $(u_1)_1^+$. We can now redo the calculations for this modified space–time slab. Although we will be using a predicted value for $t_3 - t_1$, we can see the calculation for this modified space–time slab as one in which $(x_1)_4$ becomes a known, and $t_3 - t_1$ becomes an unknown. This is as opposed to the intermediate calculations, in which $(x_1)_2$ was an unknown, and $t_2 - t_1$ was known. We complete the calculations for this space–time slab by performing a sufficient number of iterations, in which we update $t_3 - t_1$ and $(u_1)_1^+$, as well as p_1^+, p_3, and p_4^-.

In extending this to multi-dimensional cases, we can see the picture as follows. In the intermediate calculations, we would have $t_2 - t_1$ as known and $(x_1)_2$, $(x_2)_2$, and $(x_3)_2$ as unknowns. In the calculations for the modified space–time slab, $(x_1)_4$ would become a known, and we would have $t_3 - t_1$ as unknown, together with $(x_2)_4$ and $(x_3)_4$. Node-4 would symbolically represent more than one node. We will later provide a picture for a simple two-dimensional case.

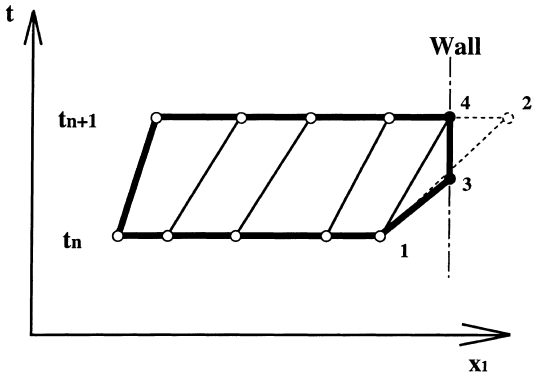

Fig. 3. STCT: Fluid contacting wall in one dimension.

Let us now imagine another one-dimensional case, shown in Fig. 4, where there is a possibility that between the time levels t_n and t_{n+1} the liquid on the right might be de-contacting the wall. Again, we first perform some intermediate calculations for this space–time slab. In these calculations we assume that Node-1 stays on the wall and maps to Node-2 at t_{n+1}. During these intermediate calculations, we also predict the liquid pressure on the wall, namely p_1^+ and p_2^-. Next, on the wall, we predict the temporal position of Node-3 by calculating t_3 when the liquid pressure becomes 0 (we assume that the viscous stress is neglible). Node-3 represents the de-contact point in the space–time domain. We calculate the predicted value of $t_3 - t_1$ from p_1^+ and p_2^- by seeking the 0 of the linear function $p(t)$. We can now redo the calculations for the modified space–time slab. At each iteration of the calculations for this modified space–time slab, we update $(u_1)_4^-$, $(x_1)_4$, p_1^+, p_3, and p_4^-, as well as $t_3 - t_1$ by seeking the 0 of the linear function $p(t)$ based on the updated values of p_1^+ and p_3.

Fig. 5 shows a simple two-dimensional case where we expect that between the time levels t_n and t_{n+1} Node-1 might be contacting the wall. In principle the calculation process is very similar to the one-dimensional contact problem. In the intermediate calculations, we have $t_2 - t_1$ as known and $(x_1)_2$, $(x_2)_2$, and $(x_3)_2$ as unknowns. In the calculations for the modified space–time slab, $(x_1)_{4a}$ and $(x_1)_{4b}$ become knowns, and we have $t_3 - t_1$ as unknown, together with $(x_2)_{4a}$ and $(x_2)_{4b}$. We complete the calculations for this space–time slab by performing a sufficient number of iterations, in which we update $t_3 - t_1$, $(x_2)_3$, $(x_2)_{4a}$, $(x_2)_{4b}$, $(u_1)_1^+$, $(u_2)_1^+$, $(u_2)_3$, $(u_2)_{4a}^-$, and $(u_2)_{4b}^-$, as well as p_1^+, p_3, p_{4a}^-, and p_{4b}^-.

We realize that the two-dimensional computations will require a three-dimensional mesh generation in the space–time domain, and the three-dimensional computations will require a four-dimensional mesh generation. However, we also realize that these will be only partial mesh generations, limited to the contact zones.

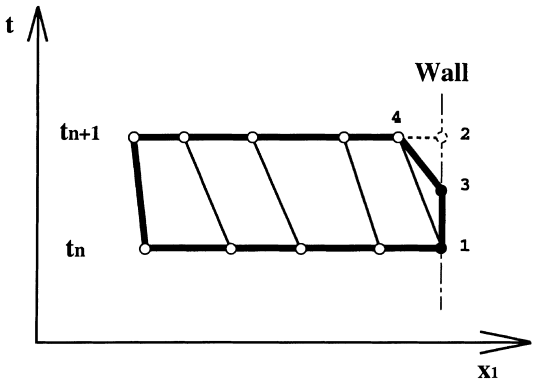

Fig. 4. STCT: Fluid de-contacting wall in one dimension.

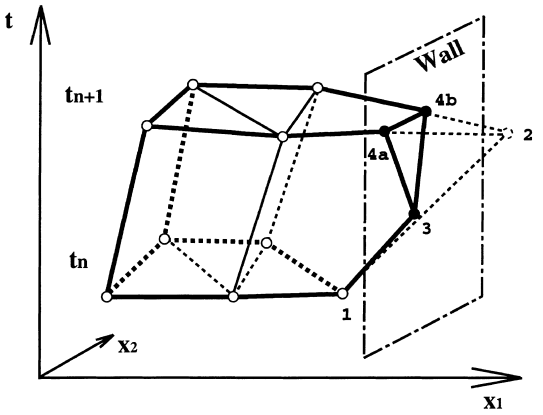

Fig. 5. STCT: Fluid contacting wall in two dimensions.

11. Examples of flow simulations

In this section we present examples of flow simulations carried out by using the methods and computing platforms described in the earlier sections. Each case is described briefly and references are given to our earlier publications for more information.

Contaminant dispersion in a model subway station. The subway station has two entrances on each side and four vents located on the upper surface. This three-dimensional simulation is carried out on the CRAY T3D in two stages. First, the Navier–Stokes equations are solved to obtain the flow velocity. This velocity field is used in the second stage in the time-dependent contaminant advection–diffusion equation to obtain the concentration of the contaminant. The contaminant is released from a point source with constant strength. The unstructured mesh used in this simulation consists of 187612 nodes and 1116992 tetrahedral elements. The steady-state solution of the flow equations is obtained by solving over 0.65 million coupled, nonlinear equations at every pseudo-time step. For the contaminant dispersion, at every time step, we solve a linear system with more than 0.15 million equations. Fig. 6 shows the mesh and the contaminant concentration at an instant. For more on this simulation see [32].

Fluid–object interactions with 1000 spheres falling in a liquid-filled tube. The core method for this simulation is the DSD/SST formulation. The methods layered around this include: an efficient distributed-memory implementation of the formulation; fast automatic mesh generation; a mesh update method based on automatic mesh moving with remeshing only as needed; an efficient method for projecting the solution after each remesh; and multi-platform (heterogeneous) computing. Here, while mesh partitioning, flow computations, and mesh movements were performed on the AHPCRC's 512-node Thinking Machines CM-5, automatic mesh generation and projection of the solution were accomplished on a 2-processor SGI ONYX2. The two systems communicated over a high-speed network as often as the compu-

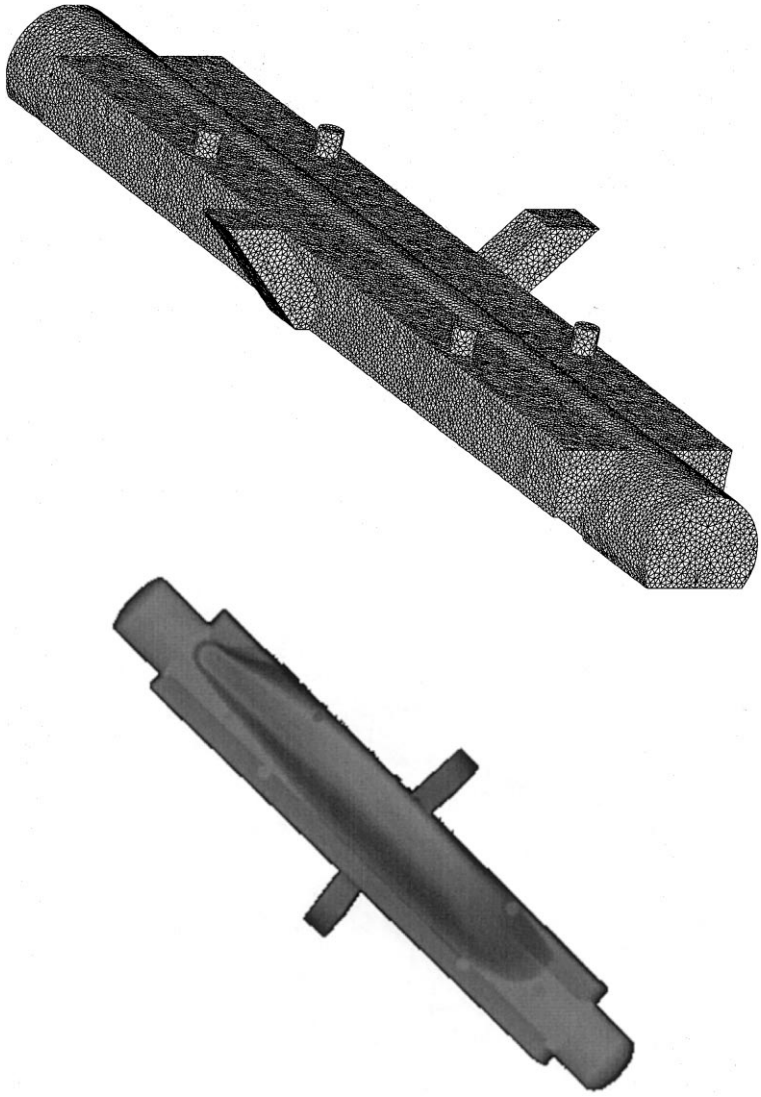

Fig. 6. Contaminant dispersion in a model subway station. Picture shows the mesh and the contaminant concentration at an instant.

tation required remeshing. In current simulations of this class of problems, the CM-5 has been replaced by the AHPCRC's newly acquired CRAY T3E-1200. The spheres, in addition to interacting with the fluid, interact and collide with each other and with the tube wall. The average Reynolds number is around 8. The mesh size is approximately 2.5 million tetrahedral elements, resulting in about 5.5 million coupled, nonlinear equations to be solved every time step. The number of time steps is around 1100 in the simulation. Fig. 7 shows the spheres at four different instants during the

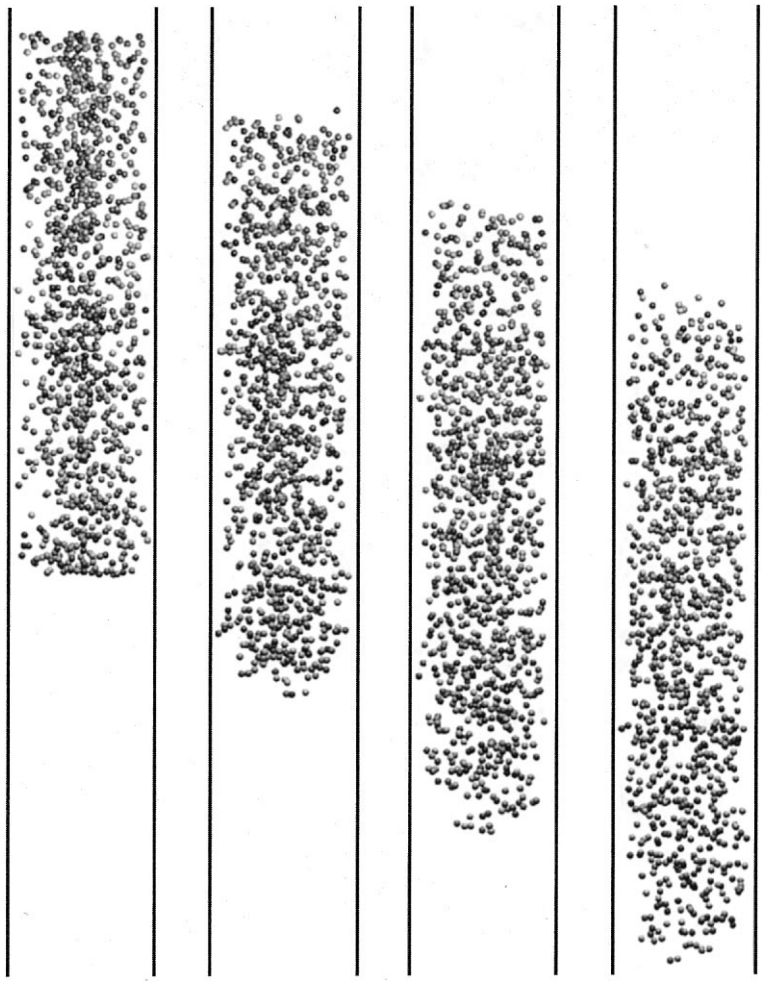

Fig. 7. Fluid–object interactions with 1000 spheres falling in a liquid-filled tube. The figure shows the spheres at four different instants during the simulation. The first picture shows the initial distribution.

simulation. The first picture shows the initial distribution. The colors are for identification purpose only. For more on this simulation see [33].

Sloshing in a tanker driving over a bump. This three-dimensional simulation was carried out on the CRAY T3E. We use the interface-capturing technique with no enhanced discretization. The tanker, moving at 10 m/s, drives over a bump 30 cm-high. The suspension system of the tanker absorbs the initial displacements due to the bump and transfers the generated forces to the structure of the tanker. The three-dimensional rigid-body dynamics equations are coupled to the finite element formulation of the flow problem and solved simultaneously for the motion of the tanker as function of time. The fluid dynamics equations are written in a non-inertial frame.

The mesh has 343 560 hexahedral elements and 357 911 nodes. At each time step, a coupled system of nonlinear equations with 1 704 661 unknowns is solved. Fig. 8 shows, at different instants, the motion of the tanker, sloshing and pressure distribution. For more on this simulation see [23].

Two-dimensional sloshing in a container. A container is filled 2/3 with water and 1/3 with air. It is suddenly subjected to the gravitational acceleration ($g = 9.8$ m/s^2) and a horizontal acceleration of magnitude 0.2 g. We compute this problem first with the DSD/SST formulation, where the computational domain is discretized using 6000 quadrilateral elements and 6161 nodes. We will refer to this solution as Solution-IT. Next, we compute the problem with the EDICT. The base mesh, Mesh-1, consists of 30 000 triangular elements and 15 251 nodes. Solution-1 is obtained by using the base discretization, where all trial and weighting functions come only from Mesh-1. Solution-2 is obtained by using the EDICT, where all trial and weighting functions come from Mesh-1 $\oplus$ Mesh-2. Solution-3 is obtained by using a more enhanced discretization, where the trial and weighting functions for velocity and pressure come from Mesh-1 $\oplus$ Mesh-2, and for the interface function from Mesh-1 $\oplus$ Mesh-2 $\oplus$ Mesh-3. Fig. 9 shows, on the left, at $t = 0.2$ s, Mesh-1 together with Mesh-2 and Mesh-3 (both shown on top of Mesh-1); and on the right, the time histories of the horizontal forces exerted on the container for all four solutions. We assume that the target solution is Solution-IT. Solution-1 has significant frequency and amplitude errors. Solution-2 is superior to Solution-1, and Solution-3 is the one in best agreement with Solution-IT. For more on these simulations see [34].

Axisymmetric filling/impact. We have a container in the shape of a circular cylinder. The lower half of the container is filled with a liquid. The upper half is filled

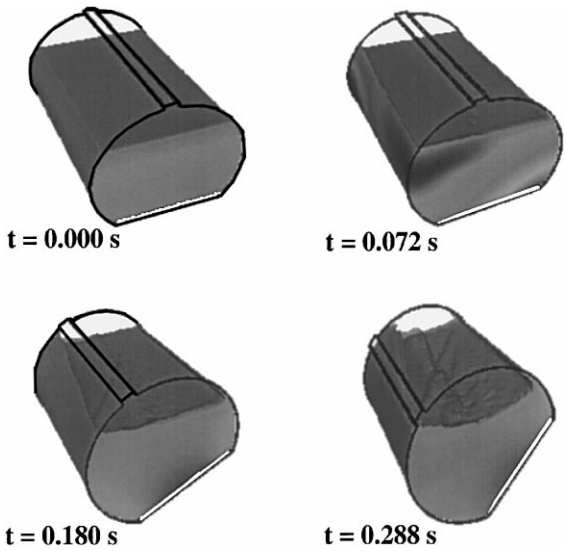

Fig. 8. Sloshing in a tanker driving over a bump. Picture shows, at different instants, the motion of the tanker, sloshing and pressure distribution.

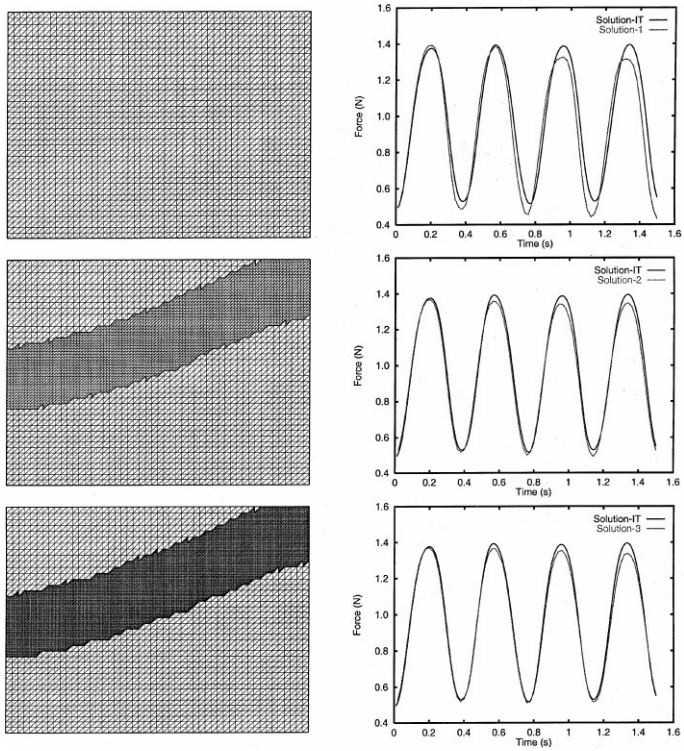

Fig. 9. Two-dimensional sloshing in a container. Picture shows, on the left, at $t = 0.2$ s, Mesh-1 together with Mesh-2 and Mesh-3 (both shown on top of Mesh-1); and on the right, the time histories of the horizontal forces exerted on the container for all four solutions.

with air. At $t = 0.0$ s we start injecting the same liquid through a circular section positioned concentrically at the top of the cylinder. The injection stream has a uniform flow speed. The base mesh, Mesh-1, consists of 30 000 triangular elements and 15 251 nodes. Mesh-2 and Mesh-3 are generated in the same way as they were generated in the previous test problem. The trial and weighting functions for velocity and pressure come from Mesh-1 $\oplus$ Mesh-2, and for the interface function from Mesh-1 $\oplus$ Mesh-2 $\oplus$ Mesh-3. Fig. 10 shows a sequence of air–liquid interactions seen at different instants during the simulation of this problem. The pictures show the injection stream impacting the still liquid, formation of surface waves, and entrapment of air in the liquid. For more on this simulation see [21].

A parachute crossing the wake of an aircraft. In a joint project with US Army Natick Research, Development, and Engineering Center, the MDM has been demonstrated in computation of a parachute/payload system crossing the wake of an aircraft. Simulations were carried out on the AHPCRC's CRAY T3E-1200. Fig. 11 shows the air–pressure distribution on the surface of the aircraft and the streamlines, as well as the streamlines for the parachute.

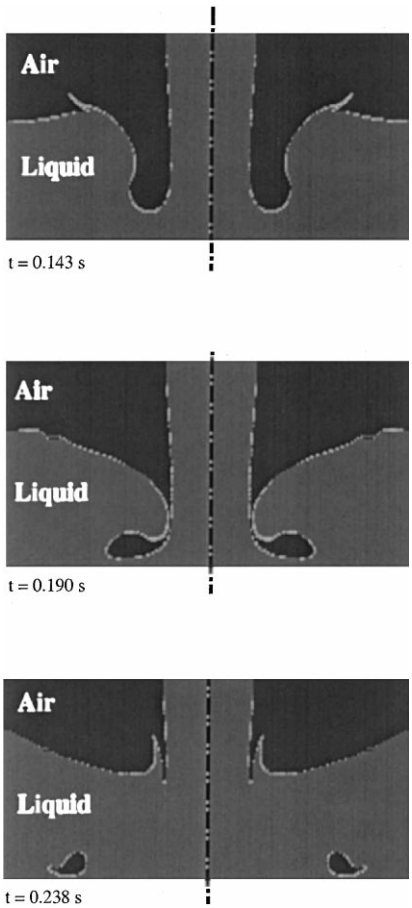

Fig. 10. Axisymmetric filling/impact. A sequence of air–liquid interactions seen at different instants.

Unsteady long-wake flow behind a circular cylinder. The MDM, developed to address a challenging army application, has also benefited three-dimensional simulation of a classical problem: unsteady wake flow behind a circular cylinder. At Reynolds number 140, using three subdomains, we computed the wake flow as far as 300 diameters downstream (see [35,36]. SD-2 and SD-3 contain no objects and are covered with structured meshes. Computations were carried out on the AHPCRCs CRAY T3E-1200, and involved, at every time step of the simulation, solution of coupled, nonlinear equation systems with more than 0.8 million unknowns for SD-1, and more than 17 million unknowns for each of SD-2 and SD-3. We were able to extend our computations sufficiently downstream, and with sufficient accuracy, to successfully capture (in SD-3) the second phase of the Karman vortex street, which has been observed in laboratory experiments, and which has double the spacing between the vortices compared to the first phase (see Fig. 12).

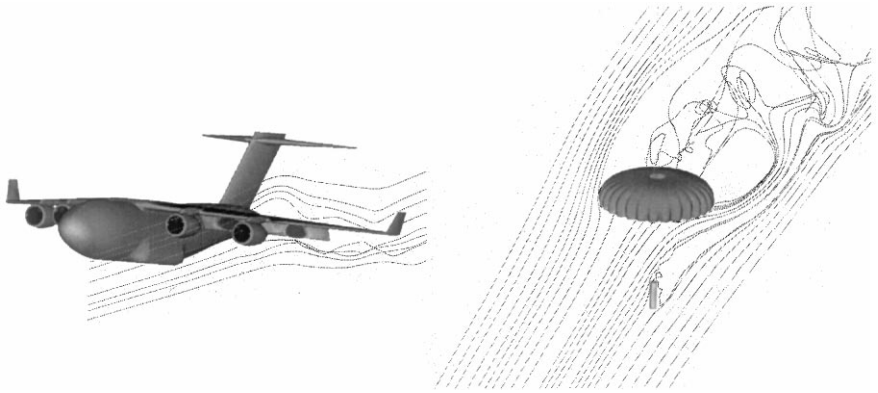

Fig. 11. A parachute crossing the wake of an aircraft. Picture shows air–pressure distribution on the surface of the aircraft and the streamlines, as well as the streamlines for the parachute.

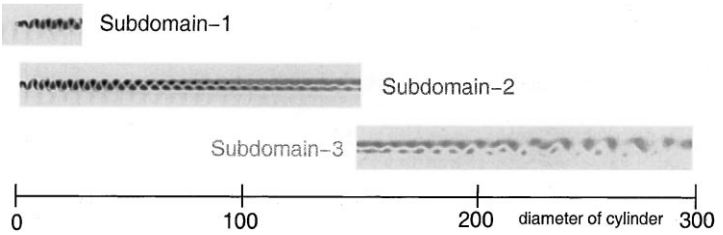

Fig. 12. Unsteady long-wake flow behind a circular cylinder. Picture shows the first and second phases of the Karman vortex street.

References

[1] T.E. Tezduyar, CFD methods for three-dimensional computation of complex flow problems, Journal of Wind Engineering and Industrial Aerodynamics 81 (1999) 97–116.
[2] T.E. Tezduyar, M. Behr, J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces the deforming-spatial-domain/space–time procedure: I. The concept and the preliminary tests, Computer Methods in Applied Mechanics and Engineering 94 (1992) 339–351.
[3] T.E. Tezduyar, M. Behr, S. Mittal, J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain/space–time procedure: II. Computation of free-surface flows two-liquid flows and flows with drifting cylinders, Computer Methods in Applied Mechanics and Engineering 94 (1992) 353–371.
[4] S.K. Aliabadi, T.E. Tezduyar, Space–time finite element computation of compressible flows involving moving boundaries and interfaces, Computer Methods in Applied Mechanics and Engineering 107 (1993) 209–223.
[5] A.N. Brooks, T.J.R. Hughes, 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 (1982) 199–259.
[6] T.E. Tezduyar, T.J.R. Hughes, Finite element formulations for convection dominated flows with particular emphasis on the compressible Euler equations, in: Proceedings of the AIAA 21st Aerospace Sciences Meeting, AIAA Paper 83-0125, Reno, Nevada, 1983.

[7] G.J. LeBeau, T.E. Tezduyar, Finite element computation of compressible flows with the SUPG formulation, in: M.N. Dhaubhadel, M.S. Engelman, J.N. Reddy (Eds.), Advances in Finite Element Analysis in Fluid Dynamics, FED-Vol.123, ASME, New York, 1991, pp. 21–27.
[8] G.J. LeBeau, S.E. Ray, S.K. Aliabadi, T.E. Tezduyar, SUPG finite element computation of compressible flows with the entropy and conservation variables formulations, Computer Methods in Applied Mechanics and Engineering 104 (1993) 397–422.
[9] S.K. Aliabadi, S.E. Ray, T.E. Tezduyar, SUPG finite element computation of viscous compressible flows based on the conservation and entropy variables formulations, Computational Mechanics 11 (1993) 300–312.
[10] T.E. Tezduyar, Stabilized finite element formulations for incompressible flow computations, Advances in Applied Mechanics 28 (1991) 1–44. ← (1992)
[11] T.E. Tezduyar, S. Mittal, S.E. Ray, R. Shih, Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity–pressure elements, Computer Methods in Applied Mechanics and Engineering 95 (1992) 221–242.
[12] T.J.R. Hughes, L.P. Franca, G.M. Hulbert, 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 (1989) 173–189.
[13] P. Hansbo, A. Szepessy, A velocity–pressure streamline diffusion finite element method for the incompressible Navier–Stokes equations, Computer Methods in Applied Mechanics and Engineering 84 (1990) 175–192.
[14] M. Behr, T.E. Tezduyar, Finite element solution strategies for large-scale flow simulations, Computer Methods in Applied Mechanics and Engineering 112 (1994) 3–24.
[15] A.A. Johnson, T.E. Tezduyar, Parallel computation of incompressible flows with complex geometries, International Journal for Numerical Methods in Fluids 24 (1997) 1321–1340.
[16] A.A. Johnson, T.E. Tezduyar, Simulation of multiple spheres falling in a liquid-filled tube, Computer Methods in Applied Mechanics and Engineering 134 (1996) 351–373.
[17] A.A. Johnson, T.E. Tezduyar, Three-dimensional simulation of fuid–particle interactions with the number of particles reaching 100, Computer Methods in Applied Mechanics and Engineering 145 (1997) 301–321.
[18] V. Kalro, S. Aliabadi, W. Garrard, T. Tezduyar, S. Mittal, K. Stein, Parallel finite element simulation of large ram-air parachutes, International Journal for Numerical Methods in Fluids 24 (1997) 1353–1369.
[19] T.E. Tezduyar, M. Behr, S. Mittal, A.A. Johnson, Computation of unsteady incompressible flows with the finite element methods – space–time formulations, iterative strategies and massively parallel implementations, in: P. Smolinski, W.K. Liu, G. Hulbert, K. Tamma (Eds.), New Methods in Transient Analysis, AMD-Vol.143, ASME, New York, 1992, pp. 7–24.
[20] A.A. Johnson, T.E. Tezduyar, Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces, Computer Methods in Applied Mechanics and Engineering 119 (1994) 73–94.
[21] T.E. Tezduyar, S. Aliabadi, M. Behr, Enhanced-discretization interface-capturing technique (EDICT) for computation of unsteady flows with interfaces, Computer Methods in Applied Mechanics and Engineering 155 (1998) 235–248.
[22] I. Güler, M. Behr, T.E. Tezduyar, Parallel finite element computation of free-surface flows, Computational Mechanics 23 (1999) 117–123.
[23] S. Aliabadi, T. Tezduyar, Stabilized-finite-element/interface-capturing technique for parallel computation of unsteady flows with interfaces, Computer Methods in Applied Mechanics and Engineering, 1997, to appear.
[24] T.E. Tezduyar, S. Aliabadi, M. Behr, Enhanced-discretization interface-capturing technique, in: Y. Matsumoto, A. Prosperetti (Eds.), Proceedings of the ISAC'97 High Performance Computing on Multiphase Flows, Japan Society of Mechanical Engineers, 1997, pp. 1–6.
[25] C.W. Hirt, B.D. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, Journal of Computational Physics 39 (1981) 201–225.
[26] Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal of Scientific and Statistical Computing 7 (1986) 856–869.

[27] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, M. Litke, High performance computing techniques for flow simulations, in: M. Papadrakakis (Ed.), Solving Large-Scale Problems in Mechanics: Parallel and Distributed Computer Applications, Wiley, New York, 1996, pp. 363–398.
[28] V. Kalro, T. Tezduyar, Parallel iterative computational methods for three-dimensional finite element flow simulations, Computer Assisted Mechanics and Engineering Sciences 5 (1998) 173–193.
[29] J. Liou, T.E. Tezduyar, Clustered element-by-element computations for fluid flow, in: Horst D. Simon (Ed.), Parallel Computational Fluid Dynamics – Implementation and Results, Scientific and Engineering Computation Series, Ch. 9, MIT Press, Cambridge, MA, 1992, pp. 167–187.
[30] T.E. Tezduyar, M. Behr, S.K. Aliabadi, S. Mittal, S.E. Ray, A new mixed preconditioning method for finite element computations, Computer Methods in Applied Mechanics and Engineering 99 (1992) 27–42.
[31] Y. Osawa, V. Kalro, T.E. Tezduyar, Multi-domain parallel computation of wake flows, Computer Methods in Applied Mechanics and Engineering 174 (1999) 371–391.
[32] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, M. Litke, Flow simulation and high performance computing, Computational Mechanics 18 (1996) 397–412.
[33] A.A. Johnson, T.E. Tezduyar, Advanced mesh generation and update methods for three-dimensional flow simulations, Computational Mechanics 23 (1999) 130–143.
[34] T.E. Tezduyar, S. Aliabadi, EDICT for computation of unsteady flows with interfaces, in: S. Atluri, P. O'Donoghue (Eds.), Modeling and Simulation Based Engineering, Proceedings of International Conference on Computational Engineering Science, Atlanta, Georgia, 1998.
[35] Y. Osawa, V. Kalro, T.E. Tezduyar, A multi-domain method for computation of long-wake flows, in: Proceedings of the Second Ankara International Aerospace Conference, Ankara, Turkey, 1998.
[36] Y. Osawa, V. Kalro, T.E. Tezduyar, A multi-domain computational method for wake flows, in: S. Atluri, P. O'Donoghue (Eds.), Modeling and Simulation Based Engineering, Proceedings of International Conference on Computational Engineering Science, Atlanta, Georgia, 1998.