Simulation of compressible viscous flows on a variety pf MPPs: computational algorithms for unstructured dynamic meshes and performance results

Charbel Farhat, Stéphane Lanteri

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Simulation of Compressible Viscous Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performance Results

Charbel Farhat , Stéphane Lanteri

N° 2154

Janvier 1994

PROGRAMME 6

Calcul scientifique,

modélisation

et logiciel numérique

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Simulation of Compressible Viscous Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performance Results

Charbel Farhat *, Stéphane Lanteri **

Programme 6 — Calcul scientifique, modélisation et logiciel numérique
Projet Sinus

Rapport de recherche n ° 2154 — Janvier 1994 — 43 pages

Abstract: Here we report on our effort in simulating unsteady and steady viscous flows on the iPSC-860, the CM-5 and the KSR-1 MPPs (Massively Parallel Processors), using a Monotonic Upwind Scheme for Conservation Laws finite volume/finite element method on fully unstructured fixed and moving meshes. We advocate mesh partitioning with message passing as a portable paradigm for parallel processing. We present and discuss several performance results obtained on all three MPP systems in terms of interprocessor communication costs, scalability, and sheer performance.

Key-words: Computational Fluid Dynamics, Bidimensional flows, Euler equations, Navier-Stokes equations, Non-structured meshes, Moving meshes, Aeroelastic computations, Parallel processing.

(Résumé : tsvp)

*Department of Aerospace Engineering and Center for Aerospace Structures, University of Colorado, Boulder, CO 80309-0429, (USA), e-mail : charbel@boulder.colorado.edu
**INRIA Sophia-Antipolis, Projet Sinus, 2004, Route des Lucioles, B.P. 93, 06902 Sophia-Antipolis, (FRANCE), e-mail : lanteri@sophia.inria.fr

Simulation d'Ecoulements de Fluides Compressibles Visqueux sur Calculateurs Massivement Parallèles: Algorithmes de Calcul sur des Maillages Dynamiques Non-Structurés et Résultats de Performance

Résumé : Nous rapportons ici les résultats de nos travaux sur la simulation numérique d'écoulements de fluides compressibles visqueux en régimes stationnaire et instationnaire, sur l'iPSC-860, la CM-5 et la KSR-1. Nous utilisons un algorithme de résolution en maillages non-structurés statiques ou dynamiques reposant sur une formulation mixte volumes/éléments finis de type M.U.S.C.L. (*Monotonic Upwind Scheme for Conservation Laws*). La stratégie de parallélisation adoptée combine l'utilisation de techniques de partitionnement de maillage et une programmation dans un modèle par transfert de message. Nous présentons et comparons plusieurs résultats de performance obtenus sur chacun des calculateurs testés, en termes de coûts de communication inter processeurs, scalabilité de l'algorithme de résolution et performance absolue.

Mots-clé : Mécanique des Fluides Numérique, Ecoulements bidimensionnels, Equations d'Euler, Equations de Navier-Stokes, Maillages non-structurés, Maillages mobiles, Calculs aéroélastiques, Calcul parallèle.

Contents

1 Introduction

In this paper, we detail our approach to the simulation of large scale, steady and unsteady, compressible viscous flows on massively parallel processors. We consider the numerical solution of the two-dimensional Navier-Stokes equations using a mixed finite element/finite volume formulation based on unstructured triangular meshes. The spatial approximation method combines a Galerkin centered approximation for the viscous terms, and a Roe upwind scheme for the computation of the convective fluxes. Higher order accuracy is achieved through the use of a piecewise linear interpolation method that follows the principle of the MUSCL (Monotonic Upwind Scheme for Conservative Laws) procedure. The temporal solution is carried out via a 3-step variant of the explicit Runge-Kutta method which lends itself to parallel processing. An ALE (Arbitrary Lagrangian Eulerian) formulation is incorporated in the fluid solver to allow the grid points to displace in Lagrangian fashion, or be held fixed in Eulerian manner, or be moved in some specified way to give a continuous and automatic re-zoning capability, depending on the needs of the physical problem to be solved.

Explicit solvers are naturally amenable to parallel processing because they essentially involve local computations on vertices, and/or edges, and/or triangles of a mesh. However, unstructured meshes induce indirect addressing memory operations that are costly on many hardware architectures. In particular, the present mixed finite element/finite volume solver incurs multiple gather/scatter operations between vertex and triangle based arrays. Therefore, in this paper we highlight the impact of irregular data access patterns on the computational scalability of a parallel unstructured solver, and emphasize the importance of data locality in achieving high level performances. These concerns and our thrive for developing a portable code have lead us to adopt mesh partitioning with message-passing as a paradigm for parallel processing.

The remainder of this paper is organized as follows. Section 2 describes the mathematical model of the problem and the approximation methods involved in the numerical solution algorithm. Section 3 identifies the main computational kernels, and motivates the selected parallelization strategy. Overlapping and non-overlapping mesh partitions are presented, discussed, and contrasted. Finally, Sections 4 and 5 report and analyze the performance results obtained on the iPSC-860, the KSR-1, and the CM-5 parallel processors for various external and internal viscous flow simulations, with fixed and moving meshes.

2 Simulation of compressible viscous flows

We are interested in the numerical simulation of two-dimensional compressible viscous flows around or within, fixed, or moving and deforming bodies. Here, we overview the spatial and temporal discretization methods that have been previously detailed in Farhat, Fezoui and Lanteri [3] for fixed meshes, and outline the mesh updating procedure adopted for aeroelastic computations.

2.1 Governing equations

Let $\Omega \subset \mathbb{R}^2$ be the flow domain of interest and Γ be its boundary. The conservative law form of the equations describing two-dimensional Navier-Stokes flows is given by:

$$\frac{\partial}{\partial t} W + \vec{\nabla}.\vec{\mathcal{F}}(W) = \frac{1}{Re}\vec{\nabla}.\vec{\mathcal{R}}(W) \tag{1}$$

where $\vec{x}$ and t denote the spatial and temporal variables, and

$$W = (\rho \ , \ \rho u \ , \ \rho v \ , \ E)^T \quad , \quad \vec{\nabla} = \left(\frac{\partial}{\partial x} \ , \ \frac{\partial}{\partial y} \right)^T$$

and

$$\vec{\mathcal{F}}(W) = \begin{pmatrix} F(W) \\ G(W) \end{pmatrix} \quad , \quad \vec{\mathcal{R}}(W) = \begin{pmatrix} R(W) \\ S(W) \end{pmatrix}$$

$F(W)$ and $G(W)$ denote the convective fluxes and are given by:

$$F(W) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{pmatrix} \quad , \quad G(W) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{pmatrix}$$

while $R(W)$ and $S(W)$ denote the diffusive fluxes and are given by:

$$R(W) = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + \frac{\gamma k}{Pr}\frac{\partial \varepsilon}{\partial x} \end{pmatrix} \quad , \quad S(W) = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} + \frac{\gamma k}{Pr}\frac{\partial \varepsilon}{\partial y} \end{pmatrix}$$

In the above expressions, ρ is the density, $\vec{U} = (u, v)$ is the velocity vector, E is the total energy per unit of volume, p is the pressure, ε is the specific internal energy, τ_{xx}, τ_{xy}, and τ_{yy} are the components of the two-dimensional Cauchy stress tensor, k is the normalized thermal conductivity, $Re = \dfrac{\rho_0 U_0 L_0}{\mu_0}$ where ρ_0, U_0, L_0 and μ_0 denote the characteristic density, velocity, length, and diffusivity is the Reynolds number, and $Pr = \dfrac{\mu_0 C_p}{k_0}$ is the Prandtl number.

The velocity, energy, and pressure are related by the equation of state for a perfect gas:

$$p = (\gamma - 1)(E - \frac{1}{2}\rho \parallel \vec{U} \parallel^2)$$

where γ is the ratio of specific heats ($\gamma = 1.4$ for air), and the specific internal energy is related to the temperature via:

$$\varepsilon = C_v T = \frac{E}{\rho} - \frac{1}{2} \parallel \vec{U} \parallel^2$$

The components of the Cauchy stress tensor are related to the velocities via:

$$\tau_{xx} = \frac{2}{3}\mu \left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right) \quad , \quad \tau_{yy} = \frac{2}{3}\mu \left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} \right) \quad , \quad \tau_{xy} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)$$

where μ denotes the normalized viscosity.

In order to account for a potential motion or deformation of the computational grid (in that case we have $\Omega = \Omega(t)$) , the following coordinate transformation is introduced:

$$\left\{ \begin{array}{ll} \vec{\xi} & = \vec{\xi}(\vec{x}, t) \\ \tau & = t \end{array} \right. \tag{2}$$

Throughout this paper, it is assumed that the Jacobian of the above transformation defined as:

$$J = \det \left(\frac{\partial \vec{x}}{\partial \vec{\xi}} \mid_t \right)$$

does not vanish at any point and any time. Introducing the grid velocity:

$$\vec{w} = \frac{\partial \vec{x}}{\partial t} \mid_{\vec{\xi}}$$

and using Eqs. (2), Eq. (1) can be transformed into the following ALE (Arbitrary Lagrangian Eulerian) formulation (see, for example, Donea [2]):

$$\frac{\partial (JW)}{\partial t} \mid_{\vec{\xi}} + J \vec{\nabla} . \vec{\mathcal{F}}_c(W) = \frac{1}{Re} \vec{\nabla} . \vec{\mathcal{R}}(W) \tag{3}$$

where:

$$\vec{\mathcal{F}}_c(W) = \left(\begin{array}{c} F_c(W) \\ G_c(W) \end{array} \right)$$

and $F_c(W)$ and $G_c(W)$ are the convected convective fluxes given by:

$$F_c(W) = \left(\begin{array}{c} \rho \bar{u} \\ \rho u \bar{u} + p \\ \rho \bar{u} v \\ E \bar{u} + p u \end{array} \right) \quad , \quad G_c(W) = \left(\begin{array}{c} \rho \bar{v} \\ \rho u \bar{v} \\ \rho v \bar{v} + p \\ E \bar{v} + p v \end{array} \right)$$

and:

$$\left\{ \begin{array}{ll} \bar{u} & = u - w_x \\ \bar{v} & = v - w_y \end{array} \right. \tag{4}$$

2.2 Boundary conditions

The boundary $\Gamma(t)$ of the flow domain is partitioned into a wall boundary $\Gamma_w(t)$ and an infinity boundary $\Gamma_\infty(t)$: $\Gamma(t) = \Gamma_w(t) \cup \Gamma_\infty(t)$. Let $\vec{n}(t)$ denote the outward unit normal at any point of $\Gamma(t)$, and $\vec{U}_w$ and T_w denote the wall velocity and temperature.

On the wall boundary $\Gamma_w(t)$, a no-slip condition and a Dirichlet condition on the temperature are imposed:

$$\vec{U} = \vec{U}_w \quad , \quad T = T_w \tag{5}$$

No boundary condition is specified for the density. Hence, the total energy per unit of volume and the pressure on the wall are given by:

$$p = (\gamma - 1)\rho C_v T_w \quad , \quad E = \rho C_v T_w + \frac{1}{2}\rho \parallel \vec{U}_w \parallel^2 \tag{6}$$

For external flows around airfoils, the viscous effects are assumed to be negligible at infinity, so that a uniform free-stream state vector W_∞ is imposed on $\Gamma_\infty(t)$:

$$\rho_\infty = 1 \quad , \quad \vec{U}_\infty = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} \quad , \quad p_\infty = \frac{1}{\gamma M_\infty^2} \tag{7}$$

where α is the angle of attack, and M_∞ is the free-stream Mach number.

For internal flows, $\Gamma_\infty(t)$ is partitioned into upstream and downstream boundaries which are in general in contact with the wall boundary: $\Gamma_\infty(t) = \Gamma_\infty^{in}(t) \cup \Gamma_\infty^{out}(t)$. In that case, the previous definition of W_∞ is improved using a parabolic profile for the free-stream velocity. For example, for the horizontal flow between two plates shown in Figure 1, one can specify:

$$\vec{U}_\infty = (0 \ , \ v_\infty(y))^T \tag{8}$$

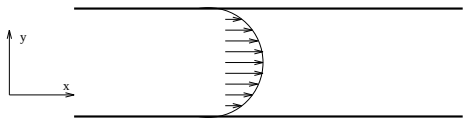

Figure 1: Horizontal velocity profile for internal flows

2.3 Spatial discretization

The flow domain $\Omega(t)$ is assumed to be a polygonal bounded region of $\mathbb{R}^2$. Let $\mathcal{T}_h$ be a standard triangulation of $\Omega(t)$, and h the maximal length of the edges of $\mathcal{T}_h$. A vertex of a triangle T is denoted by S_i, and the set of its neighboring vertices by $K(i)$. At each vertex S_i, a cell $C_i(t)$ is constructed as the union of the subtriangles resulting from the subdivision by means of the medians of each triangle of $\mathcal{T}_h$ that is connected to S_i (see Figure 2). The

boundary of $C_i(t)$ is denoted by $\partial C_i(t)$, and the unit vector of the outward normal to $\partial C_i(t)$ by $\vec{\nu}_i(t) = (\nu_{ix}(t), \nu_{iy}(t))$. The union of all these control volumes constitutes a discretization of domain $\Omega(t)$:

$$\Omega_h = \bigcup_{i=1}^{N_V} C_i \quad , \quad N_V \text{ : number of vertices of } \mathcal{T}_h$$

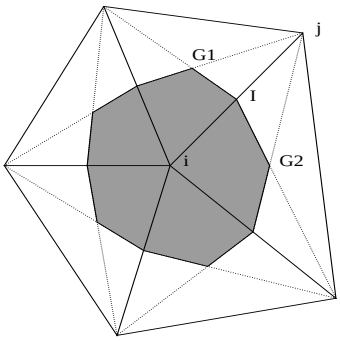

Figure 2: Control volume in an unstructured grid

The spatial discretization method adopted here combines the following features (see [3] for details):

- a finite volume upwind approximation method for the convective fluxes. Second order spatial accuracy is achieved using an extension of Van Leer's MUSCL technique [14] to unstructured meshes;
- a classical Galerkin finite element centered approximation for the diffusive fluxes.

Let $C_i^{\vec{x}}$ and $C_i^{\vec{\xi}}$ denote the two representations of $C_i(t)$ in the coordinate systems defined by $\vec{x}$ and $\vec{\xi}$, respectively. By definition, $C_i^{\vec{\xi}}$ is a fixed reference representation of $C_i(t)$. Integrating Eq. (3) over $C_i^{\vec{\xi}}$ yields:

$$\int\int_{C_i^{\vec{\xi}}} \frac{\partial(JW)}{\partial t}|_{\vec{\xi}} d\vec{\xi} + \int\int_{C_i^{\vec{\xi}}} J\vec{\nabla}.\vec{\mathcal{F}}_c(W) d\vec{\xi} = \int\int_{C_i^{\vec{\xi}}} \frac{1}{Re}\vec{\nabla}.\vec{\mathcal{R}}(W) d\vec{\xi} \tag{9}$$

Given that the time derivative is computed for a constant $\vec{\xi}$ and that $C_i^{\vec{\xi}}$ does not depend on the time t, the mapping $\vec{\xi} = \vec{\xi}(\vec{x}, t)$ and the identity $d\vec{x} = Jd\vec{\xi}$ can be used to transform Eq. (9) into:

$$\frac{d}{dt}\int\int_{C_i^{\vec{x}}(t)} W d\vec{x} + \int\int_{C_i^{\vec{x}}(t)} \vec{\nabla}.\vec{\mathcal{F}}_c(W) d\vec{x} = \int\int_{C_i^{\vec{x}}(t)} \frac{1}{Re}\vec{\nabla}.\vec{\mathcal{R}}(W) d\vec{x} \tag{10}$$

Finally, integrating Eq. (10) by parts leads to:

$$\begin{aligned} \frac{d}{dt}\int\int_{C_i^{\vec{x}}(t)} W d\vec{x} \;&+\; \sum_{j\in K(i)} \int_{\partial C_{ij}^{\vec{x}}(t)} \vec{\mathcal{F}}_c(W).\vec{\nu}_i(t) d\sigma \qquad &< 1 > \\ &+ \int_{\partial C_i^{\vec{x}}(t)\cap\Gamma_w(t)} \vec{\mathcal{F}}_c(W).\vec{n}_i(t) d\sigma \qquad &< 2 > \\ &+ \int_{\partial C_i^{\vec{x}}(t)\cap\Gamma_\infty(t)} \vec{\mathcal{F}}_c(W).\vec{n}_i(t) d\sigma \qquad &< 3 > \\ &= -\frac{1}{Re}\sum_{T, S_i\in T} \int\int_T \vec{\mathcal{R}}(W).\vec{\nabla} N_i^T d\vec{x} \quad &< 4 > \end{aligned} \tag{11}$$

where $\partial C_{ij}^{\vec{x}}(t) = \partial C_i^{\vec{x}}(t) \cap \partial C_j^{\vec{x}}(t)$, and $N_i^T = N_i^T(x,y)$ is the P1 shape function defined at the vertex S_i and associated with the triangle T.

A first order finite volume discretization of $< 1 >$ goes as follows:

$$< 1 > = \; A_i^{n+1} W_i^{n+1} - A_i^n W_i^n + \Delta t \sum_{j\in K(i)} \Phi_{\mathcal{F}_c}(W_i^n, W_j^n, \tilde{\vec{\nu}}_{ij}) \tag{12}$$

where A_i^n denotes the area of the control volume $C_i^{\vec{x}}$ measured at time t^n, $\vec{\nu}_{ij}(t)$ denotes a spatial mean value of the normal to $\partial C_{ij}^{\vec{x}}(t)$, the tilde superscript designates a temporal mean value between t^n and t^{n+1}, and $\Phi_{\mathcal{F}_c}$ denotes a numerical flux function that approximates the following quantity:

$$\Delta t \Phi_{\mathcal{F}_c}(W_i, W_j, \tilde{\vec{\nu}}_{ij}) \approx \int_{t^n}^{t^{n+1}} \int_{\partial C_{ij}^{\vec{x}}(t)} \vec{\mathcal{F}}_c(W).\vec{\nu}_i(t) d\sigma \tag{13}$$

Upwinding is introduced by extending Roe's approximate Riemann solver [12] to dynamic meshes and computing $\Phi_{\mathcal{F}_c}$ as follows:

$$\begin{aligned} \Phi_{\mathcal{F}_c}(W_i, W_j, \tilde{\vec{\nu}}_{ij}) \;&= \frac{\vec{\mathcal{F}}_c(W_i) + \vec{\mathcal{F}}_c(W_j)}{2}.\tilde{\vec{\nu}}_{ij} \\ &- \mid \mathcal{A}_R(W_i, W_j, \tilde{\vec{\nu}}_{ij}) - (\tilde{\vec{w}}.\tilde{\vec{\nu}}_{ij}) I \mid \frac{(W_j - W_i)}{2} \end{aligned} \tag{14}$$

where $\mathcal{A}_R$ is Roe's mean value of the flux Jacobian matrix $\dfrac{\partial \vec{F}(W)}{\partial W}$, I is the identity matrix, and the dot product $\tilde{\vec{w}}.\tilde{\vec{\nu}}_{ij}$ is computed as suggested recently by N'Konga and Guillard [11]:

$$\tilde{\vec{w}}.\tilde{\vec{\nu}}_{ij} = \frac{1}{2}\left(\vec{w}(P_{1ij}) + \vec{w}(P_{2ij})\right).\tilde{\vec{\nu}}_{ij} \tag{15}$$

where P_{1ij} and P_{2ij} are the end points of the bi-segment $\partial C_{ij}^{\vec{x}}(\frac{t^n + t^{n+1}}{2})$ (see Figure 3).

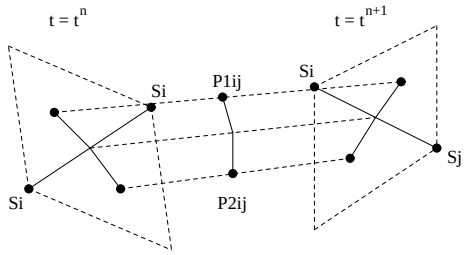

Figure 3: Computation of $\vec{w}.\vec{\nu}_{ij}$

The control volume area A_i^{n+1} is updated using the following finite volume scheme:

$$A_i^{n+1} = A_i^n + \Delta t \sum_{j \in K(i)} \tilde{\vec{w}}.\tilde{\vec{\nu}}_{ij} \tag{16}$$

Following the MUSCL technique, second order accuracy is achieved in expression (13) via a piecewise linear interpolation of the states W_{ij} and W_{ji} at the interface between cells C_i and C_j. This requires the evaluation of the gradient of the solution at each vertex as follows:

$$\begin{cases} W_{ij}^{\star} &= W_i^{\star} + \frac{1}{2}(\vec{\nabla}W)_i^{\beta}.\vec{S_i S_j} \\ W_{ji}^{\star} &= W_j^{\star} - \frac{1}{2}(\vec{\nabla}W)_j^{\beta}.\vec{S_i S_j} \end{cases} \tag{17}$$

where $W^{\star} = (\rho\ ,\ u\ ,\ v\ ,\ p)^T$ — in other words, the interpolation is performed on the physical variables instead of the conservative variables. The approximate nodal gradients $(\vec{\nabla}W)_{i,j}^{\beta}$ are obtained using a β-combination of centered and fully upwind gradients:

$$(\nabla\vec{W})_i^{\beta} = (1-\beta)(\nabla\vec{W})_i^{Cent} + \beta(\nabla\vec{W})_i^{Upw} \tag{18}$$

The half-upwind scheme ($\beta = \frac{1}{2}$) is simply obtained by means of a linear interpolation of the Galerkin gradients computed on each triangle of C_i:

$$\left(\nabla\vec{W}\right)_i^{\beta=\frac{1}{2}} = \frac{\iint_{C_i^{\vec{x}}} \nabla\vec{W}|_T d\vec{x}}{\iint_{C_i^{\vec{x}}} d\vec{x}} = \frac{1}{area(C_i^{\vec{x}})} \sum_{T \epsilon C_i^{\vec{x}}} \frac{area(T)}{3} \sum_{k=1,k\epsilon T}^{3} W_k^{\star} \vec{\nabla} N_k^T \tag{19}$$

and the centered gradient $(\nabla\vec{W})_i^{Cent}$ $(\beta = 0)$ is given by any vector that verifies:

$$(\nabla\vec{W})_i^{Cent}.S_i\vec{S}_j = W_j - W_i \tag{20}$$

The second term $< 2 >$ and the third term $< 3 >$ of Eq. (11) include the contributions of the boundary conditions and are evaluated as follows:

Wall boundary : the no-slip condition is enforced with a strong formulation and therefore the corresponding boundary integral in $< 2 >$ is not explicitly computed.

Inflow and outflow boundaries : at these boundaries, a precise set of compatible exterior data that depend on the flow regime and the velocity direction must be specified. Here, a *plus-minus* flux splitting is applied between exterior data and interior values. More specifically, the boundary integral $< 3 >$ is evaluated using a non-reflective version of the flux-splitting of Steger and Warming [13]:

$$\int_{\partial C_i^{\vec{x}}(t) \cap \Gamma_\infty(t)} \vec{\mathcal{F}}_c(W).\vec{n}_i(t) d\sigma = \mathcal{A}_c^+(W_i, \vec{n}_{i\infty}(t)).W_i + \mathcal{A}_c^-(W_i, \vec{n}_{i\infty}(t)).W_\infty \tag{21}$$

Finally, the viscous integral $< 4 >$ is evaluated using a classical Galerkin finite element P1 method. The components of the stress tensor and those of ∇N_i^T are constant in each triangle. The velocity vector in a triangle is computed as follows:

$$U_T = \frac{1}{3} \sum_{k=1,k\epsilon T}^{3} U^k$$

and the viscous fluxes are approximated as follows:

$$\vec{\mathcal{R}}_i(T) = \iint_T \vec{\mathcal{R}}(W).\nabla N_i^T d\vec{x} = area(T) \left(R_T \frac{\partial N_i^T}{\partial x} + S_T \frac{\partial N_i^T}{\partial y} \right)$$

where R_T and S_T are the constant values of $R(W)$ and $S(W)$ in the triangle T.

2.4 Time integration

The resulting semi-discrete fluid flow equations can be written as:

$$\frac{dW}{dt} + \psi(W) = 0$$

Because it lends itself to massive parallelism, the following 3-step variant of the explicit Runge-Kutta algorithm is selected for time integrating the above equations:

$$\begin{cases} W^{(0)} &= W^n = W(t = n\Delta t) \\ \\ W^{(k)} &= \dfrac{A_i^n}{A_i^{n+1}} W^{(0)} + \dfrac{1}{A_i^{n+1}} \alpha_k \Delta t \Psi(W^{(k-1)}) \qquad k = 1, 2, 3 \\ \\ W^{(3)} &= W^{n+1} \end{cases} \tag{22}$$

This scheme is often referred to as a low-storage Runge-Kutta algorithm because only the solution at substep $k-1$ is needed to compute the one at substep k. The coefficients α_k are the standard Runge-Kutta coefficients and are given by:

$$\alpha_k = \frac{1}{4-k}$$

The above time integration algorithm is third order accurate in the linear case, and second order accurate in the general non-linear case.

2.5 Dynamic meshes

In this work, an unstructured dynamic fluid mesh is represented by a pseudo structural model (see, for example, Batina [1]) where a fictitious linear spring is associated with each edge connecting two fluid grid points S_i and S_j and is attributed the following stiffness:

$$k_{ij} = \frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} \tag{23}$$

The grid points located on the downstream and upstream boundaries are held fixed. The motion of those points located on the wall boundary is determined from the wall motion and/or deformation. At each time step t^{n+1}, the new position of the interior grid points is determined from the solution of a displacement driven pseudo structural problem via a two-step iterative procedure. First, the displacements of the interior grid points are predicted by extrapolating the previous displacements at time steps t^n and t^{n-1} in the following manner:

$$\begin{cases} \tilde{\delta} x_i = 2\delta^n x_i - \delta^{n-1} x_i \\ \\ \tilde{\delta} y_i = 2\delta^n y_i - \delta^{n-1} y_i \end{cases} \tag{24}$$

with $\delta^n x = x^{n+1} - x^n$. Next, the above predictions are corrected with a few explicit Jacobi relaxations on the static equilibrium equations as follows:

$$\left\{ \begin{array}{ll} \delta^{n+1}x_i & = \dfrac{\sum\limits_{j\in K(i)} k_{ij}\tilde{\delta}x_j}{\sum\limits_{j\in K(i)} k_{ij}} \\ \delta^{n+1}y_i & = \dfrac{\sum\limits_{j\in K(i)} k_{ij}\tilde{\delta}y_j}{\sum\limits_{j\in K(i)} k_{ij}} \end{array} \right. \tag{25}$$

Finally, the new positions are computed as:

$$\left\{ \begin{array}{ll} x_i^{n+1} & = x_i^n + \delta^{n+1}x_i \\ y_i^{n+1} & = y_i^n + \delta^{n+1}y_i \end{array} \right. \tag{26}$$

3 Computational and parallel implementation issues

3.1 Identification of the computational kernels

From Eqs. (11-22), it follows that our fluid solver contains essentially two kernels of elementary computations, one for the convective fluxes, and the other for the diffusive ones. Both type of computations can be described as three-step sequences of the form `Gather/Compute/Scatter`.

3.1.1 The convective flux kernel

The evaluation of the second term of $< 1 >$ in Eq. (11) using the numerical flux function $\Phi_{\mathcal{F}_c}$ (14) with the second order approximation outlined in Eq. (17) can be summarized as follows:

$$\left\{ \begin{array}{ll} H_{ij} & = \Phi_{\mathcal{F}_c}(W_i, W_j, \vec{\nu}_{ij}(t)) \approx \int\limits_{\partial C_{ij}(t)} \vec{\mathcal{F}}_c(W).\vec{\nu}_i(t)d\sigma \\ H_{ji} & = -H_{ij} \end{array} \right. \tag{27}$$

where:

$$\vec{\nu}_{ij}(t) = \int\limits_{\partial C_{ij}(t)} \vec{\nu}_i d\sigma = \vec{\nu}_1(t) + \vec{\nu}_2(t) \tag{28}$$

Essentially, one-dimensional elementary convective fluxes are computed at the intersection between the control volumes $C_i(t)$ and $C_j(t)$ (see Figure 4). Each elementary flux contributes to a flux balance at the boundary of the control volume $C_i(t)$. This balance involves the accumulation over the set of neighboring vertices $K(i)$ of all computed fluxes.

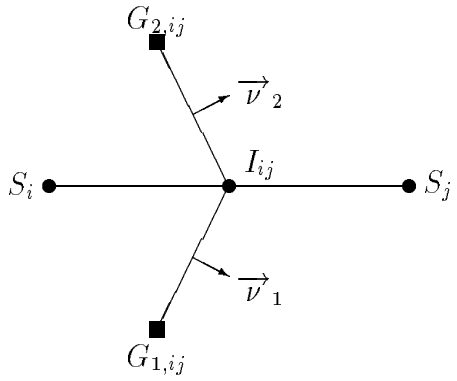

Figure 4: Evaluation of a convective flux along an edge $\{S_i, S_j\}$

From the second of Eqs. (27), it follows that only H_{ij} needs to be computed in order to update the flux balances at the two end-point nodal values of edge $E_{ij} = \{S_i, S_j\}$. Therefore, the most efficient way for evaluating the convective fluxes is to loop over the list of the mesh edges and compute as follows:

For `each edge` $E_{ij} = \{S_i, S_j\}$ `of` $\mathcal{T}_h$ **Do**

`Gather` $W_i = W(S_i)$, $W_i = W(S_i)$
`Gather` $\vec{\nabla} W_i = \vec{\nabla} W(S_i)$, $\vec{\nabla} W_i = \vec{\nabla} W(S_i)$
`Compute` H_{ij}
`Scatter` $\Phi_i = \Phi_i + H_{ij}$
`Scatter` $\Phi_j = \Phi_j - H_{ij}$

End Do

3.1.2 The diffusive flux and nodal gradient kernel

In the last term $< 4 >$ of Eq. (11), the elementary diffusive flux $\vec{\mathcal{R}}_i(T)$ is constant in each triangle T. Its evaluation requires accessing the values of the physical state W at the three vertices S_i , S_j and S_k :

$$\left\{ \begin{array}{rl} \vec{\mathcal{R}}_i(T) & = \displaystyle\iint_T \vec{\mathcal{R}}(W).\nabla \vec{N}_i^T d\vec{x} \\ & = area(T) \left(R_T \dfrac{\partial N_i^T}{\partial x} + S_T \dfrac{\partial N_i^T}{\partial y} \right) \end{array} \right. \tag{29}$$

The values of R_T and S_T contribute to the diffusive fluxes at all three vertices of triangle T. The sum symbol in $< 4 >$ is a clear indication of a gather operation. Clearly, the most

efficient way for evaluating the convective fluxes is to loop over the list of the mesh triangles and compute as follows:

For `each element` $T_{ijk} = \{S_i, S_j, S_k\}$ `of` $\mathcal{T}_h$ **Do**

`Gather` $W_i = W(S_i)$, $W_j = W(S_j)$, $W_k = W(S_k)$
`Compute` R_T, S_T
`Scatter` $\mathcal{V}_i = \mathcal{V}_i + \vec{\mathcal{R}}_i(T)$
`Scatter` $\mathcal{V}_j = \mathcal{V}_j + \vec{\mathcal{R}}_j(T)$
`Scatter` $\mathcal{V}_k = \mathcal{V}_k + \vec{\mathcal{R}}_k(T)$

End Do

The evaluation of the half-upwind nodal gradient (19) follows the same computational pattern described above.

3.2 The mesh partitioning with message-passing parallel paradigm

In addition to efficiency and parallel scalability, portability should be a major concern. With the proliferation of computer architectures, it is essential to adopt a programming model that does not require rewriting thousands of lines of code — or even worse, altering the architectural foundations of a code — every time a new parallel processor emerges. Here, we are neither referring to differences between programming languages, nor to differences between the multitude of parallel extensions to a specific programming language. We are more concerned about the impact of a given parallel hardware architecture on the software design, and sometimes, on the solution algorithm itself. For example, a data parallel code written for the CM-2 or CM-5 machines could require major rehauling before it can be adapted to an iPSC computer. A parallel-do-loop based code can be easily ported across different true shared memory multiprocessors, but may require substantial modifications before it can run successfully on some distributed memory systems.

Based on our "hands on" experience with a dozen of different parallel processors, we believe that the mesh partitioning and message-passing lead to portable software designs for parallel computational mechanics. Essentially, the underlying mesh is assumed to be partitioned into several submeshes, each defining a subdomain. The same "old" serial code can be executed within every subdomain. The assembly of the subdomain results can be implemented in a separate software module and optimized for a given machine. This approach enforces data locality, and therefore is suitable for all parallel hardware architectures. For example, we have shown in [9] that for unstructured meshes, this approach produces substantially better performance results on the KSR-1 than the acclaimed virtual shared memory programming model. Note that in this context, message-passing refers to the assembly phase of the subdomain results. However, it does not imply that messages have to be explicitly

exchanged between the subdomains. For example, message-passing can be implemented on a shared memory multiprocessor as a simple access to a shared buffer, or as a duplication of one buffer into another one.

In this work, we use essentially the same code on the iPSC-860, the KSR-1, and the CM-5 parallel processors. This code also runs on a workstation. We consider mesh partitions with and without overlapping for reasons that are discussed next.

3.2.1 Overlapping mesh partitions

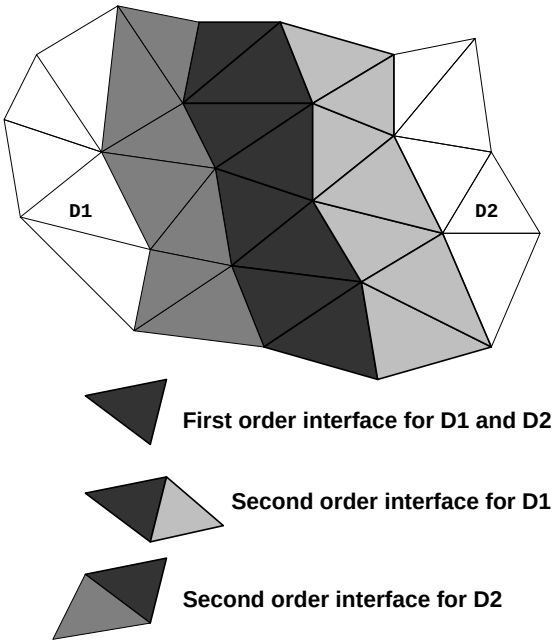

Figure 5: Overlapping mesh partition

The reader can verify that for the computations described herein, mesh partitions with overlapping simplify the programming of the subdomain interfacing module. Only one communication step is required, after the local physical states have been updated. Depending on the order of the spatial approximation, the overlapping region can be one or three triangles wide (see Figure 5 below). For a first order spatial approximation, we have by definition:

$$\begin{cases} W_{ij}^{\star} &= W_{i}^{\star} \\ \\ W_{ji}^{\star} &= W_{j}^{\star} \end{cases} \tag{30}$$

which shows that the overlapping region needs in that case to be only one triangle wide. However, mesh partitions with overlapping also have a drawback: they incur redundant floating-point operations.

For fixed meshes and overlapping partitions, the main loop of the parallel fluid solver described herein goes as follows:

Repeat $step = step + 1$

```
Compute the local time steps
```
For $srk = 1$ `to` $nsrk$ **Do**
```
    Compute the nodal gradients
    Compute the diffusive fluxes
    Compute the convective fluxes
    Update the physical states
    Exchange the conservatives variables
```
End Do

Until $step = step_{max}$

In the above pseudo code, $step_{max}$ denotes the maximum number of time steps, and $nsrk$ denotes the number of steps in the Runge-Kutta integration algorithm. All overlapping mesh partitions used in this investigation were generated by the decomposer described in [10].

3.2.2 Non-overlapping mesh partitions

Non-overlapping mesh partitions (see Figure 6 below) incur little redundant floating-point operations but induce one additional communication step. While physical state variables are exchanged between the subdomains in overlapping mesh partitions, partially gathered nodal gradients and partially gathered fluxes are exchanged between subdomains in non-overlapping ones.

For fixed meshes and non-overlapping partitions, the main loop of the parallel fluid solver described herein goes as follows:

Repeat $step = step + 1$

```
Compute the local time steps
```
For $srk = 1$ `to` $nsrk$ **Do**
```
    Compute the nodal gradients
    Compute the diffusive fluxes
    Exchange the nodal gradients
    Compute the convective fluxes
    Exchange the convective fluxes
    Update the physical states
```
End Do

Until $step = step_{max}$

All non-overlapping mesh partitions discussed in this investigation were generated by the TOP/DOMDEC software described in [4].

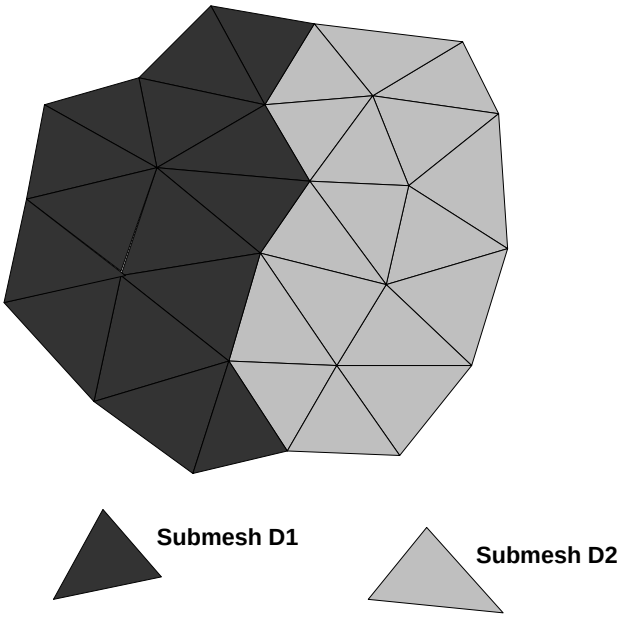

Figure 6: Non-overlapping mesh partition

3.2.3 To overlap or not to overlap?

To answer this question, we analyze the communication requirements of both families of mesh partitions, and the amount of redundant computations they incur for two-dimensional fixed problems. Because we are interested in a comparative study, it suffices to consider the case of a single interface between two subdomains with uniform triangulations.

Let n_I^{nov} denote the number of interface vertices in an non-verlapping mesh partition (see Figure 7). In the first communication step, $8 \times n_I^{nov}$ words related to the nodal gradients are exchanged between the two subdomains. In the second communication step, $4 \times n_I^{nov}$ fluxes are exchanged. Hence, the total communication cost per subdomain is given by:

$$T_{com}^{nov} = 2 \times T_s + 12 \times n_I^{nov} \times T_r \tag{31}$$

where T_s denotes the startup time of a message, and T_r denotes the transmit time for a 64-bit word.

In a non-overlapping mesh partition, the only redundant computations are those associated with the evaluation of the convective fluxes along the interface edges. Since an elementary convective flux requires about 200 floating-point operations, the total time per subdomain associated with redundant computations can be estimated as:

$$T_{red}^{nov} = 200 \times (n_I^{nov} - 1) \times T_a \tag{32}$$

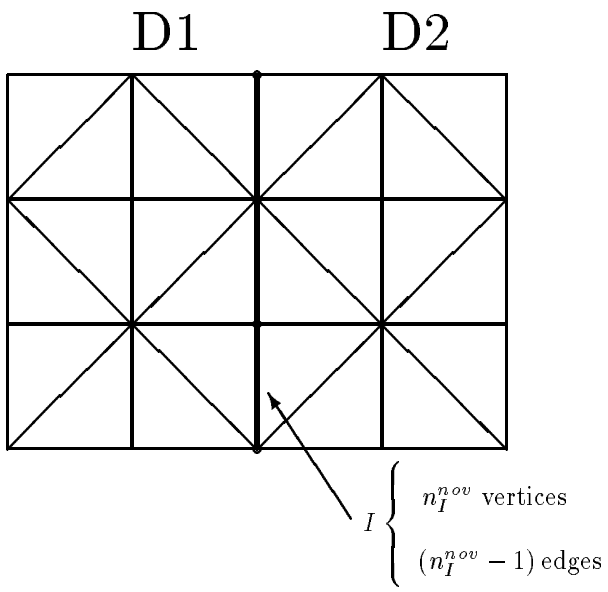

Figure 7: Analysis of a non-overlapping mesh partition

where T_a denotes the times it takes to perform a single floating-point operation.

From Figure 8, it follows that the total number of overlapping vertices in an overlapping mesh partition is $4 \times n_I^{nov}$. In this case, only one communication step is required to exchange 4 components of the physical state at half of the overlapping vertices. Hence, the total amount of communication per subdomain is given by:

$$T_{com}^{ov} = T_s + 8 \times n_I^{nov} \times T_r \tag{33}$$

In an overlapping mesh partition, redundant computations are performed during the evaluation of both the convective and diffusive fluxes. Given that an elementary diffusive flux requires about 100 floating-point operations, the total time per subdomain associated with redudant computations is given in that case by:

$$T_{red}^{ov} = (200 \times (4 \times n_I^{nov}) + 100 \times (4 \times (n_I^{nov} - 1)) \times T_a \tag{34}$$

where $4 \times n_I^{nov}$ is the number of overlapping edges that generate redundant convective flux computations, and $4 \times (n_I^{nov} - 1)$ is the number of overlapping triangles that generate redundant diffusive flux computations.

From Eqs. (31-33), it follows that for sufficiently large messages we have:

$$\frac{T_{com}^{ov}}{T_{com}^{nov}} = \frac{T_s + 8 \times n_I^{nov} \times T_r}{2 \times T_s + 12 \times n_I^{nov} \times T_r} \approx \frac{2}{3} \tag{35}$$

which shows that overlapping the mesh partitions reduces the communication costs by 33%.

However from Eqs. (31-34), it follows that for a sufficiently large n_I^{nov} we have:

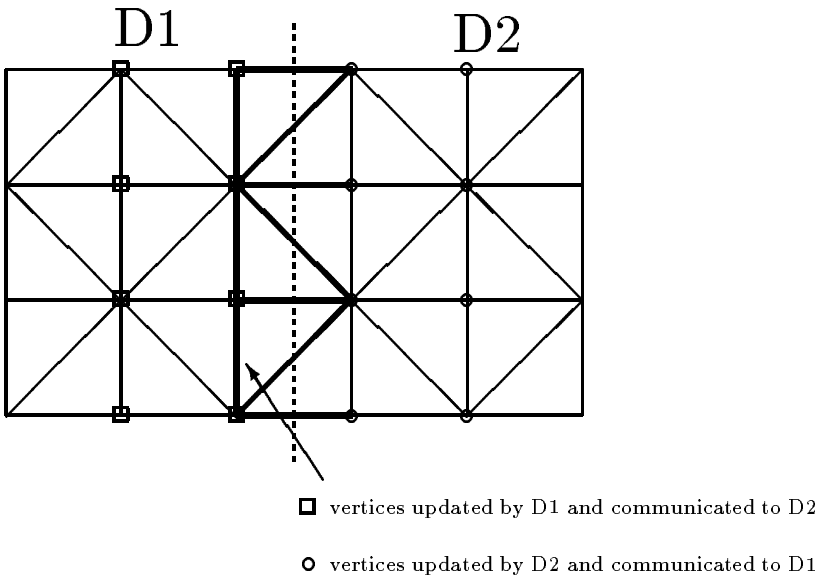

Figure 8: Analysis of an overlapping mesh partition

$$(T_{com}^{ov} + T_{red}^{ov}) - (T_{com}^{nov} + T_{red}^{nov}) \approx 1000 \times n_I^{nov} \times T_a(1 - \frac{1}{250} \times \frac{T_r}{T_a}) \tag{36}$$

which suggests that overlapping the mesh partitions will incur a greater total parallel overhead than not overlapping them, because of the resulting redundant computations. For example on the iPSC-860, $T_a = 0.2 \times 10^{-6}$ seconds (sustained 5 Mflops per processor), $T_r = 3.184 \times 10^{-6}$ seconds (sustained 2.5 Mbytes/second), and $(T_{com}^{ov} + T_{red}^{ov}) - (T_{com}^{nov} + T_{red}^{nov}) \approx 0.18 \times 10^{-3} \times n_I^{ov} > 0$. Nevertheless, this specific result also suggest that for two-dimensional problems, similar performance results will be obtained for mesh partitions with or without overlapping.

Finally, we caution the reader than different conclusions may be drawn for three-dimensional problems where overlapping can also require significantly more storage.

4 Performance results on a variety of MPPs

In this section, we discuss the parallel performance results obtained on various configurations of the iPSC-860, the KSR-1, and the CM-5 parallel processors.

4.1 Focus problem

We consider the numerical simulation of the unsteady viscous flow around a fixed NACA0012 airfoil, starting impulsively from a uniform flow. The angle of attack is set to 30°, and the free stream Mach number to 0.1. Several physical solutions of this problem were previously reported in [3] for different Reynolds numbers. All performance results reported herein are for 100 iterations and 64-bit arithmetic. More importantly, **the redundant floating-point operations are not counted** when evaluating the Mflop rate, which is a strict approach to benchmarking.

A partial view of an unstructured triangulation of the computational domain is given in Figure 9. Seven meshes with increasing sizes have been generated. Their characteristics are summarized in Table 1 below where N_V denotes the number of vertices, N_T the number of triangles, and N_E the number of edges.

MESH	N_V	N_T	N_E
M1	8119	15998	24117
M2	16116	30936	47052
M3	32236	63992	96228
M4	63974	127276	191250
M5	131035	261126	392161
M6	262717	523914	786631
M7	523196	1044504	1567700

Table 1 : seven meshes and their characteristics

Throughout the remainder of this paper, the following nomenclature is used for the investigated mesh partitioning algorithms:

- **SCT** : Sector [10]
- **RIB** : Recursive inertial bisection [6]
- **GRD** : Greedy [6]
- **RGB** : Recursive graph bisection [16]
- **RSB** : Recursive spectral bisection [16]

4.2 Parallel scalability for increasing size problems

Parallel scalability is evaluated here for problems where the subdomain size is fixed, and the total size is increased with the number of processors. Note that because we are dealing with

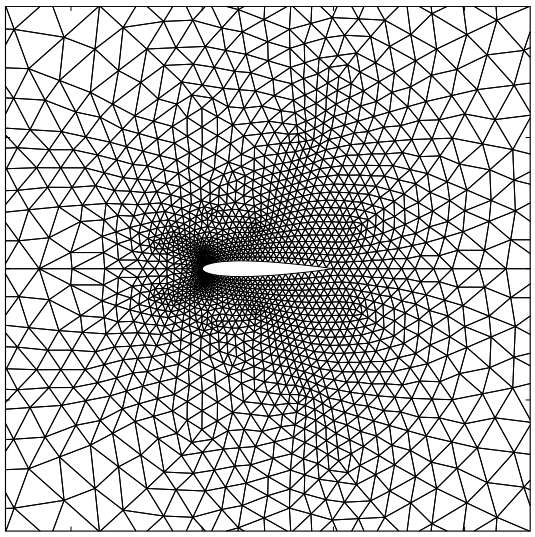

Figure 9: Partial view of a NACA0012 mesh

unstructured meshes, some slight deviations are inevitable. Overlapping mesh partitions are generated using the **RIB** heuristic.

Tables 2-3 summarize the performance results obtained on the iPSC-860 and KSR-1 parallel systems. The number of processors is denoted as N_p. The parallel CPU time and the Mflop rate are shown to remain almost constant when the problem size is increased with the number of processors, which demonstrates the scalability of the parallel solver. The slight degradations in efficiency are mainly attributed to overlapping since redundant operations are not accounted for in the evaluation of the Mflop rate. The KSR-1 processor cell is a RISC-style superscalar 64-bit unit operating at a peak of 40 Mflops. Clearly, despite a rather large number of gather/scatter operations, 25% of this peak performance is attained.

4.3 Influence of the mesh partitioning algorithm

Next, we focus on mesh *M6* with 32 processors and non-overlapping partitions, and investigate the influence of the partitioning algorithm on parallel performance. Tables 4-5 report the measured CPU time and Mflop rates. "Conv" and "Diff" designate respectively the convective and diffusive fluxes. In all cases, the **RSB** algorithm yields the fastest solution time, even when it does not produce the smallest communication time. The reason is that, for mesh *M6*, the **RSB** algorithm does a better job than the others at generating subdomains that are well balanced vertex-wise, element-wise, and edge-wise, simultaneously.

N_V	N_p	CPU Time	Mflop/s	Comm Time	% Comm
8119	1	491.2 s	6	0 s	0
16116	2	491.7 s	11	28.6 s	5.81
32236	4	514.7 s	22	31.3 s	6.09
63974	8	519.5 s	43	40.0 s	7.71
131035	16	536.6 s	85	40.3 s	7.52
262717	32	548.6 s	159	44.8 s	8.17

Table 2 : Parallel scalability for increasing size problems
Computations with *overlapping* mesh partitions on the iPSC-860

N_V	N_p	CPU Time	Mflop/s	Comm Time	% Comm
8119	1	272.3 s	10	0 s	0
16116	2	272.3 s	20	1.8 s	0.67
32236	4	286.0 s	39	3.2 s	1.18
63974	8	289.3 s	77	4.3 s	1.52
131035	16	306.7 s	149	5.9 s	1.95
262717	32	316.1 s	276	8.4 s	2.66

Table 3 : Parallel scalability for increasing size problems
Computations with *overlapped* mesh partitions on the KSR-1

Decomp	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
GRD	550.0 s	330.4 s	163.4 s	47.4 s	158
RGB	556.3 s	335.1 s	157.9 s	62.7 s	158
RSB	538.0 s	326.9 s	153.7 s	53.8 s	162

Table 4 : Influence of the mesh partitioning algorithm
Computations with *non-overlapping* mesh partitions on an iPSC-860/32

Decomp	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
GRD	343.7 s	165.9 s	112.8 s	18.9 s	254
RGB	340.1 s	168.3 s	105.1 s	18.5 s	256
RSB	322.5 s	160.4 s	101.4 s	14.2 s	270

Table 5 : Influence of the mesh partitioning algorithm
Computations with *non-overlapping* mesh partitions on a KSR-1/32

The reader can verify that the numbers reported in columns 3 to 5 of Tables 4-5 do not add up to the total CPU time reported in column 2. The difference corresponds to various parallel and sequential overheads. On the iPSC-860, these overheads represent less than 2% of the total solution time. However on the KSR-1, they represent about 14% of the total CPU time. Indeed, the computing mode on the iPSC-860 is parallel by default, while on the KSR-1 it is sequential by default. Hence, many fork-join type of procedures are necessary on the KSR-1, which explains the relatively large amount of overhead.

Also, note that using the same number of processors, the KSR-1 is twice as fast as the iPSC-860 at computing the convective fluxes, but only 1.5 times faster at computing the diffusive ones. This is because the evaluation of the convective fluxes requires less indirect addressing than the evaluation of the diffusive ones.

4.4 Parallel scalability for fixed size problems

Tables 6-8 summarize the performance results obtained on the iPSC-860 and KSR-1 parallel processors for overlapping mesh partitions generated by the **RIB** algorithm. For the largest mesh *M7*, a Gigaflop performance level is attained using 128 processors of the KSR-1 system. Good scalability is observed on both machines.

Finally, Tables 9-10 compare the performances of the iPSC-860 and KSR-1 parallel systems for the case of mesh *M6* and non-overlapping mesh partitions generated by the **GRD** algorithm. For the same number of processors, the KSR-1 machine is reported to be 1.67 times faster than the iPSC-860, even though its basic processor is supposed to be 1.5 times slower than that of the iPSC-860.

4.5 Performance results on the CM-5

Recently, we have implemented our fluid solver on a 32 processor CM-5 system using Fortran 77 on a node and the CMMD message passing library (this corresponds to the Sparc model of computation on the CM-5). For mesh *M6* and a 32 subdomain decomposition with overlapping using the **RIB** algorithm, Table 11 reports the measured performance results and compares them to those obtained on an iPSC-860/32 and a KSR-1/32 computers.

Clearly, the results reported in Table 11 for the CM-5 are not as impressive as those reported, for example, in [15]. This can be attributed to several factors including the use of the message-passing model for portability reasons, the variety of gather/scatter operations required by our specific fluid solver, and perhaps our strict approach to performance benchmarking.

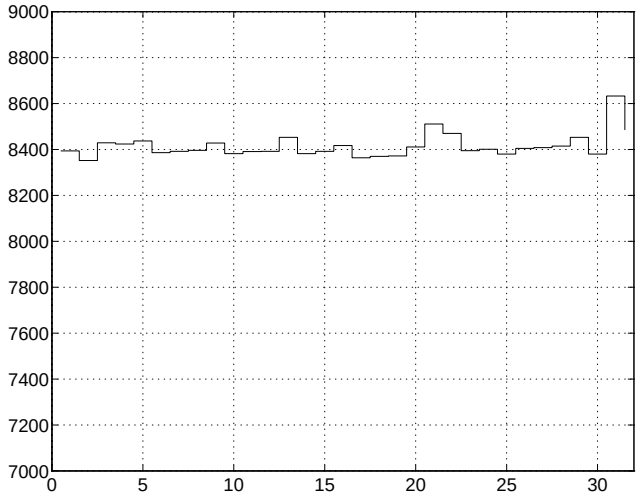

Figure 10: Vertex-wise load bal. - 32 *non-overlapping* sub.
GRD algorithm

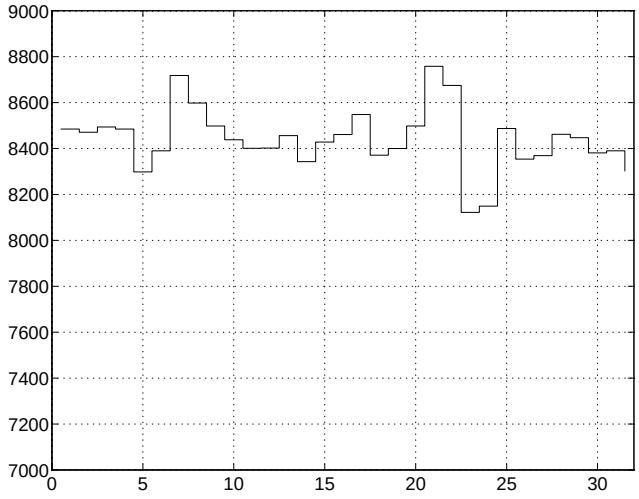

Figure 11: Vertex-wise load bal. - 32 *non-overlapping* sub.
RGB algorithm

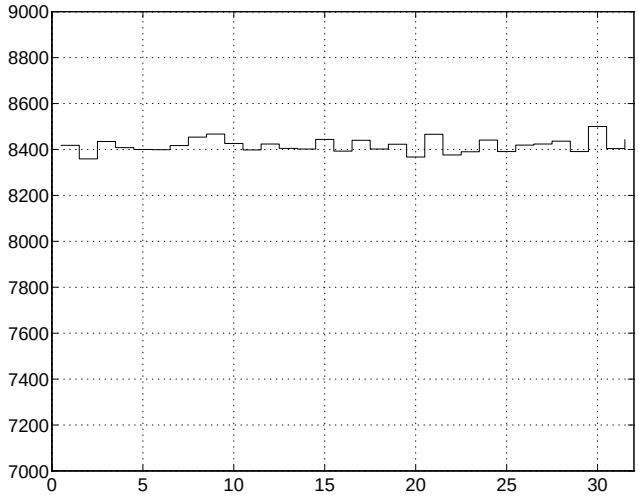

Figure 12: Vertex-wise load bal. - 32 *non-overlapping* sub.
RSB algorithm

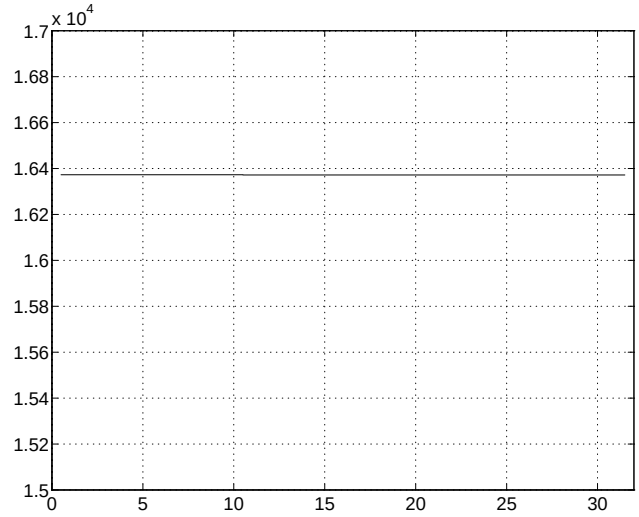

Figure 13: Element-wise load bal. - 32 *non-overlapping* sub.
GRD algorithm

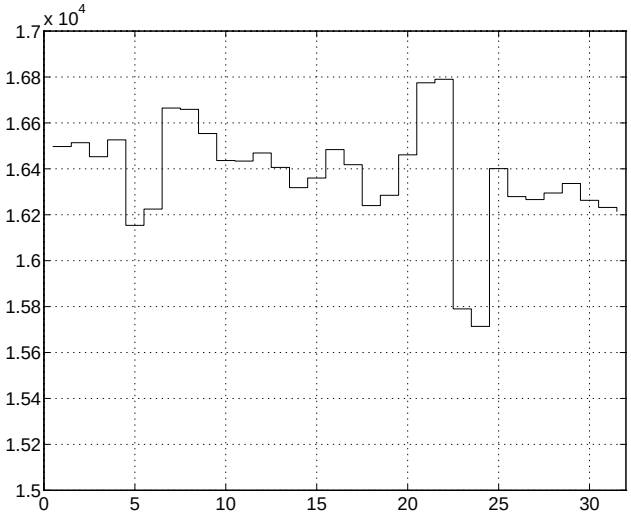

Figure 14: Element-wise load bal. - 32 *non-overlapping* sub.
RGB algorithm

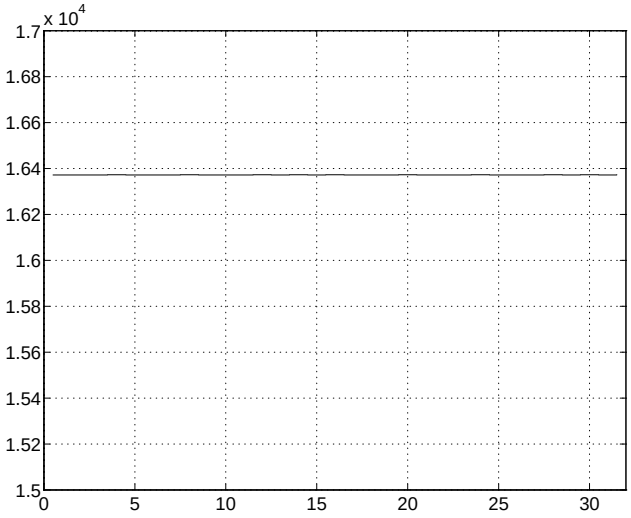

Figure 15: Element-wise load bal. - 32 *non-overlapping* sub.
RSB algorithm

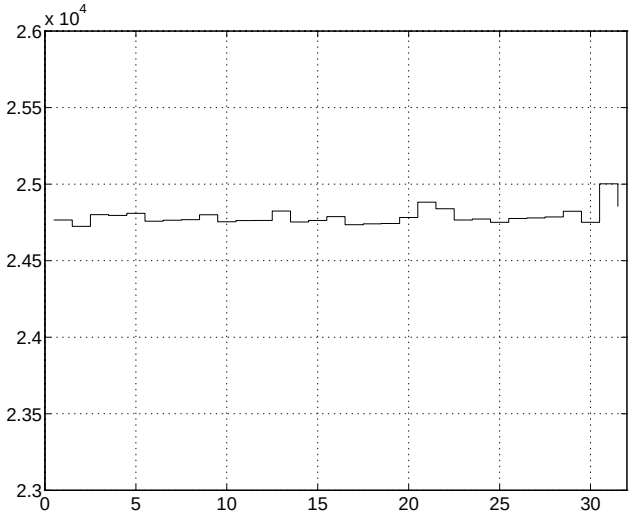

Figure 16: Edge-wise load bal. - 32 *non-overlapping* sub.
GRD algorithm

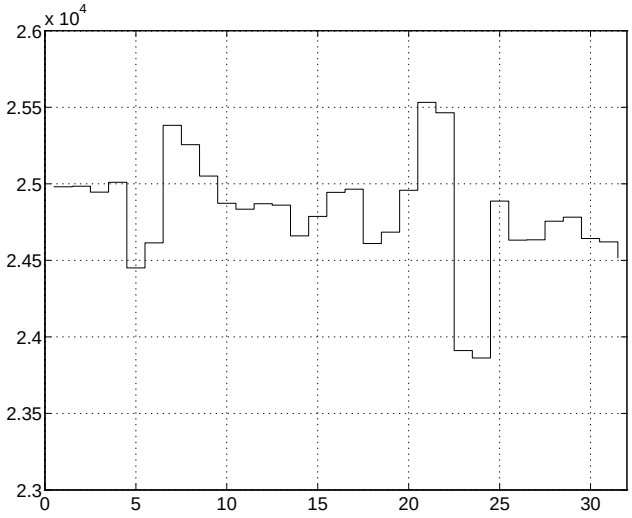

Figure 17: Edge-wise load bal. - 32 *non-overlapping* sub.
RGB algorithm

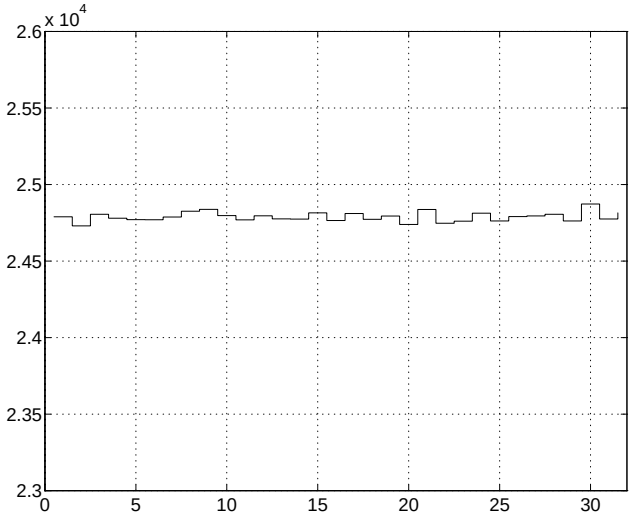

Figure 18: Edge-wise load bal. - 32 *non-overlapping* sub.
RSB algorithm

N_p	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
32	548.7 s	323.1 s	164.6 s	44.9 s	159
64	282.7 s	163.6 s	82.4 s	24.4 s	309
128	144.5 s	82.2 s	42.7 s	18.9 s	617

Table 6 : Performance results for mesh *M6*
Computations with *overlapping* mesh partitions on the iPSC-860

N_p	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
32	316.1 s	155.4 s	111.3 s	8.4 s	276
64	169.7 s	79.0 s	58.1 s	9.8 s	514
128	93.0 s	39.7 s	31.2 s	6.4 s	938

Table 7 : Performance results for mesh *M6*
Computations with *overlapping* mesh partitions on the KSR-1

N_p	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
128	174.0 s	77.7 s	60.2 s	14.0 s	1024

Table 8 : Performance results for mesh *M7*
Computations with *overlapping* mesh partitions on the KSR-1

N_p	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
64	287.7 s	164.0 s	79.8 s	42.3 s	303

Table 9 : Performance results for mesh *M6*
Computations with *non-overlapping* mesh partitions on the iPSC-860

N_p	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
64	171.8 s	83.1 s	51.1 s	14.1 s	508
128	94.8 s	40.1 s	24.3 s	15.2 s	921

Table 10 : Performance results for mesh *M6*
Computations with *non-overlapping* mesh partitions on the KSR-1

MPP	CPU Time	Conv Time	Diff Time	Comm Time	Mflop/s
CM-5	855.4 s	541.3 s	241.7 s	44.0 s	102
iPSC-860	548.6 s	323.1 s	164.6 s	44.8 s	159
KSR-1	316.1 s	155.4 s	111.3 s	8.4 s	276

Table 11 : Performance results for mesh *M6*
Computations with *overlapping* mesh partitions

Next, we report performance results on the CM-5 using a global CM Fortran approach (this corresponds to the Vector Units model of computation on the CM-5), and the parallel version of our solver that was previously developed for the CM-2/200 and described in [3]. In this so called data parallel approach, all local computations are carried out on a control volume. Therefore, all data structures are vertex based. Note that this approach generates a substantial amount of redundant computations. The reported communication timing corresponds to the inter and intra vector units gather/scatter operations. In Table 12, MFU is a Mflop rate that does not account for redundant arithmetic operations, while MFR is a Mflop rate that does.

CPU Time	Conv Time	Diff Time	Comm Time	MFU	MFR
342.0 s	116.9 s	49.0 s	162.0 s	107	238

Table 12 : Performance results for mesh *M5*
Computations with *overlapping* mesh partitions on the CM-5

The results reported in Table 12 for mesh *M5* (that was the largest size we have been able to test with the global data parallel aproach) show that about 50% of the elapsed time is spent in gather/scatter operations, which is consistent with the results obtained by other investigators for two-dimensional finite element fluid problems [7]. This percentage is higher than those observed on the iPSC-860 and KSR-1 systems, but that is because the CM-5 has faster processors.

5 Applications

5.1 Steady viscous flow inside a model jet engine

First, we consider the numerical simulation of a steady viscous flow inside a model jet engine. The free stream Mach number and the Reynolds number are set respectively to 0.2 and 2000. The computational grid is illustrated in Figure 19. Its characteristics are $N_V = 12233$, $N_T = 22936$, and $N_E = 35170$.

This simulation is carried out on the KSR-1 using overlapping mesh partitions generated by the **RIB** algorithm. Here, the pseudo time integration is carried out at CFL=1.9 via a four step Runge-Kutta method with $\alpha_1 = 0.11$, $\alpha_2 = 0.2766$, $\alpha_3 = 0.5$ and $\alpha_4 = 1.0$ (see [8] for more details about these coefficients). A local time step strategy is introduced in order to accelerate convergence. After 1527 iterations, the initial residual is reduced by a factor of 10^4. The resulting steady mach lines are depicted in Figure 20.

Table 13 reports the performance results obtained on the KSR-1, and Table 14 summarizes the characteristics of the generated mesh partitions. $S(p)$ denotes the speed-up using

p processors. N_I denotes the total number of interface vertices, and Max N_I denotes the maximum number of interface vertices per subdomain. The jump of communication costs between the case with $N_p = 2$ and that with $N_p = 4$ can be attributed to the accidental increase in Max N_I, which implies an increase in the maximum message length.

N_p	CPU Time	Mflop/s	Comm Time	% Comm	$S(p)$
1	8946 s	9	0 s	0	1.0
2	4532 s	17	28.7 s	0.63	1.9
4	2136 s	37	49.9 s	2.33	4.1
8	1168 s	68	46.3 s	3.96	7.6

Table 13 : Performance results for an internal flow simulation
Computations with *overlapping* mesh partitions on the KSR-1

N_p	Min N_V	Max N_V	Min N_T	Max N_T	Max N_I
2	6222	6229	11568	11668	113
4	3205	3334	5942	6183	171
8	1593	1774	2964	3204	86

Table 14 : Characteristics of the mesh partitions

5.2 Airfoil flutter and control surface

Next, we consider the flutter simulation of a NACA0012 airfoil in transonic flow. The structural dynamics behavior of the airfoil is represented by a two-spring two-degree of freedom system. The airfoil twist θ is associated with a torsional spring and monitors the angle of attack. The lateral deflection h is associated with a lineal spring and monitors the airfoil bending. The evolution of this system is governed by a set of differential equations that can be written in non-dimensional form as follows (for example, see [5]):

$$\begin{cases} \dfrac{d^2\bar{h}}{d\bar{t}^2} + \dfrac{x_\theta}{2}\dfrac{d^2\theta}{d\bar{t}^2} + \dfrac{4\xi_h\omega_h M_\infty}{V^\star\omega_\theta}\dfrac{d\bar{h}}{d\bar{t}} + \dfrac{4\omega_h^2 M_\infty^2}{V^{\star 2}\omega_\theta^2}\bar{h} = -\dfrac{2M_\infty^2 C_l}{\pi\mu} \\ \dfrac{x_\theta}{2}\dfrac{d^2\bar{h}}{d\bar{t}^2} + \dfrac{\gamma_\theta^2}{4}\dfrac{d^2\theta}{d\bar{t}^2} + \dfrac{\xi_\theta\gamma_\theta^2 M_\infty}{V^\star}\dfrac{d\theta}{d\bar{t}} + \dfrac{\gamma_\theta^2 M_\infty^2}{V^{\star 2}}\theta = \dfrac{2M_\infty^2 C_m}{\pi\mu} \end{cases} \tag{37}$$

In Eqs. (37) above, the bar superscript indicates a non-dimensional variable, and C_l and C_m denote respectively the lift coefficient and the torsional moment. These two quantities are related to the generalized aerodynamics forces Q_h and Q_θ by:

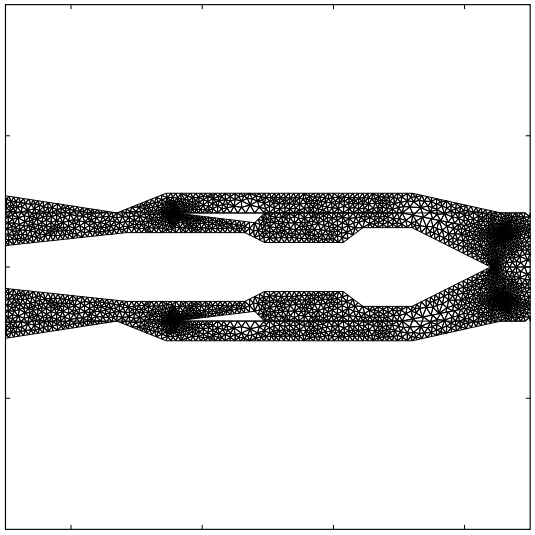

Figure 19: View of the discretization of a model jet engine

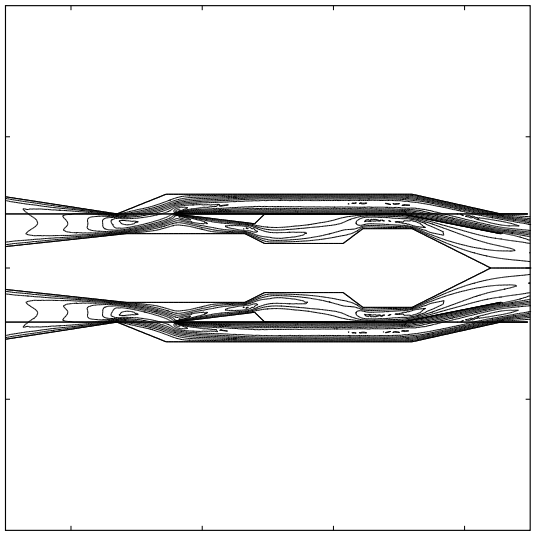

Figure 20: Mach lines : $Min = 0.0$, $Max = 0.6$, $\Delta M = 0.06$

$$\begin{cases} Q_h &= \frac{1}{2}\rho_\infty V_\infty^2 (2b) C_l \\ \\ Q_\theta &= \frac{1}{2}\rho_\infty V_\infty^2 (2b)^2 C_m \end{cases} \tag{38}$$

where $2b$ is the airfoil chord, $V^\star = \dfrac{V_\infty}{b\omega_\theta}$ is the normalized velocity, V_∞ is the free stream velocity, and $\mu = \dfrac{m}{\pi \rho_\infty b^2}$ is the ratio of the airfoil mass per transversal unit to the free stream density.

The effect of an additional control surface such as a flap is simulated with the superposition of a controlled motion of a fraction of the airfoil trailing edge. This secondary motion is defined by the flap angle $\delta(t)$, which obeys the following control law:

$$\delta(t) = G_h h(t) e^{i\varphi_h} + G_\theta \theta(t) e^{i\varphi_\theta} \tag{39}$$

where G_h et G_θ are gain coefficients, and φ_h and φ_θ phase angles (see Figures 21-23).

Two aeroelastic simulations with and without the control surface are performed using mesh *M4* and:

$$2b = 1, \quad m = 1, \quad \xi_\theta = \xi_h = 0$$
$$\omega_\theta = \omega_h = 100 \ rad/s$$
$$\gamma_\theta = 1.865, \quad \mu = 60, \quad a_h = -2, \quad x_\theta = 1.8$$
$$V^\star = 0.6, \quad Q^\star = 0.006, \quad V_\infty = 30.0 m/s$$

The flow initial conditions are identified with the steady solution at $M_\infty = 0.8$ and zero angle of attack. After the steady state is reached, a perturbation in the angle of attack $\Delta\theta = 0.01$ radian is introduced, which causes the airfoil to vibrate and the flow to become unsteady. The governing aeroelastic equations (37) are used to predict the dynamic response of the system. The unsteady fluid flow equations are time integrated with a global time step strategy. At each time step, the dynamic mesh is updated with 8 explicit Jacobi relaxations as described in Section 2.5. All computations are run on both the iPSC-860 and KSR-1 parallel processors.

Figures 24 and 25 report the evolution in time of the angle of attack θ and the lift coefficient C_l. Clearly, when the control surface is enabled with $G_h = G_\theta = 0.75$ and $\varphi_h = \varphi_\theta = \pi$, a stable aeroelastic response is observed. When it is disabled, a flutter instability is reached.

For this application, the performance results obtained on the iPSC-860 and KSR-1 parallel processors are summarized in Tables 15-16. Simulation time is reported for 100 steps

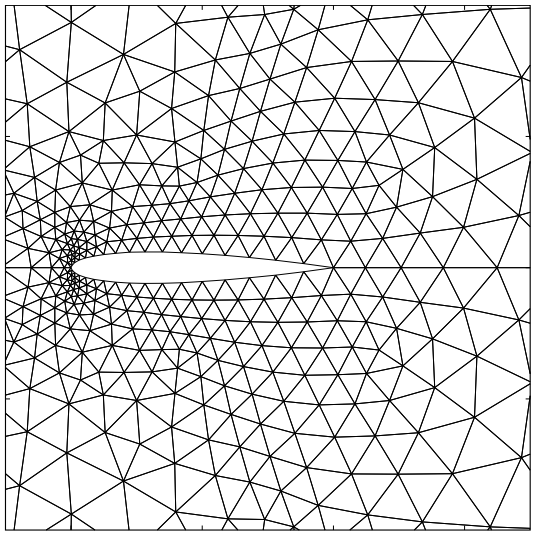

Figure 21: $\alpha = 0^\circ$, $\delta = 0^\circ$, $G_\theta = 0.0$
(mesh is coarsened for clarity)

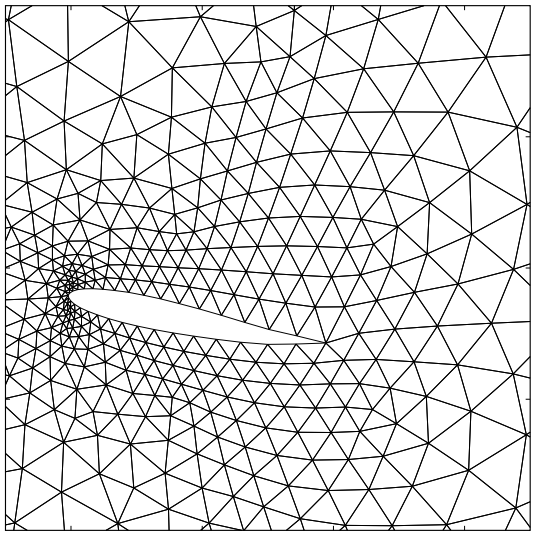

Figure 22: $\alpha = 12^\circ$, $\delta = -6^\circ$, $G_\theta = 0.5$
(mesh is coarsened for clarity)

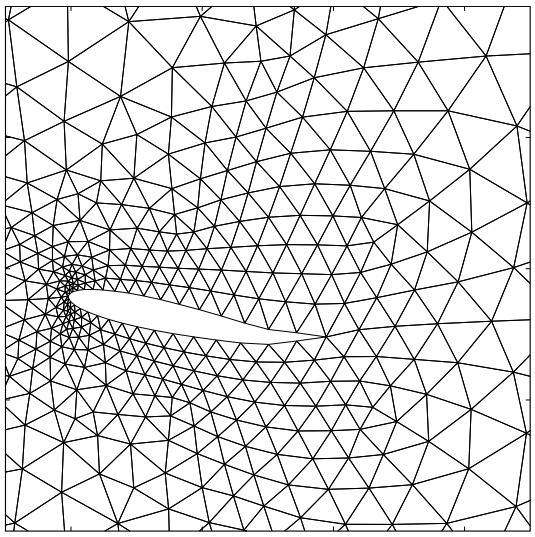

Figure 23: $\alpha = 12^\circ$, $\delta = 12^\circ$, $G_\theta = 1.0$
(mesh is coarsened for clarity)

and includes I/O costs for saving the computed solutions on disk. On the KSR-1 the stuctural model (37) is solved on one processor using shared variables while on the iPSC-860, all processors are solving for the structural displacements.

Both parallel processors are shown to deliver good speed-ups. Interprocessor communication time varies between 9% and 12.7 % on the iPSC-860, and 5.7% and 15.8% on the KSR-1. The cost of updating the dynamic mesh is about 10% of the total cost only. Note that for this simulation, the KSR-1 is not reported to be 1.5 times faster than the iPSC-860 for the same number of processors, unlike in all previous cases. This suggests that I/O on the KSR-1 is more expensive than on the iPSC-860.

N_p	Simulation Time	Flux Comp Time	Mesh Update	Comm Time
16	372.0 s	243.2 s	32.4 s	33.5 s
32	208.0 s	124.0 s	17.7 s	17.1 s
64	101.0 s	64.0 s	10.1 s	12.8 s

Table 15 : 100 steps of an aeroelastic simulation with mesh *M4*
Computations with *overlapping* mesh partitions on the iPSC-860

N_p	Simulation Time	Flux Comp Time	Mesh Update	Comm Time
16	326.0 s	135.0 s	35.8 s	18.8 s
32	176.0 s	69.2 s	19.1 s	18.9 s
64	99.7 s	35.1 s	11.0 s	15.8 s

Table 16 : 100 steps of an aeroelastic simulation with mesh *M4*
Computations with *overlapping* mesh partitions on the KSR-1

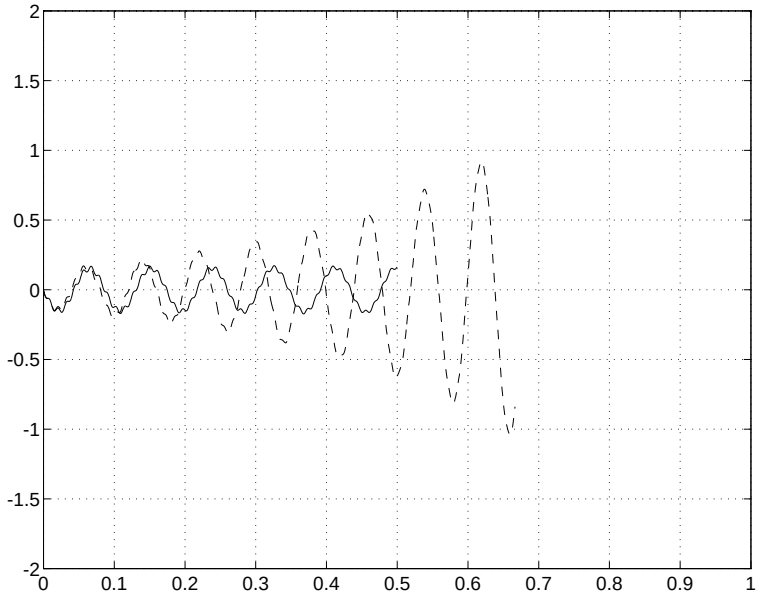

Figure 24: Angle of attack θ (in °) versus physical time t (in seconds)
$G_h = G_\theta = 0.75$, $\varphi_h = \varphi_\theta = \pi$

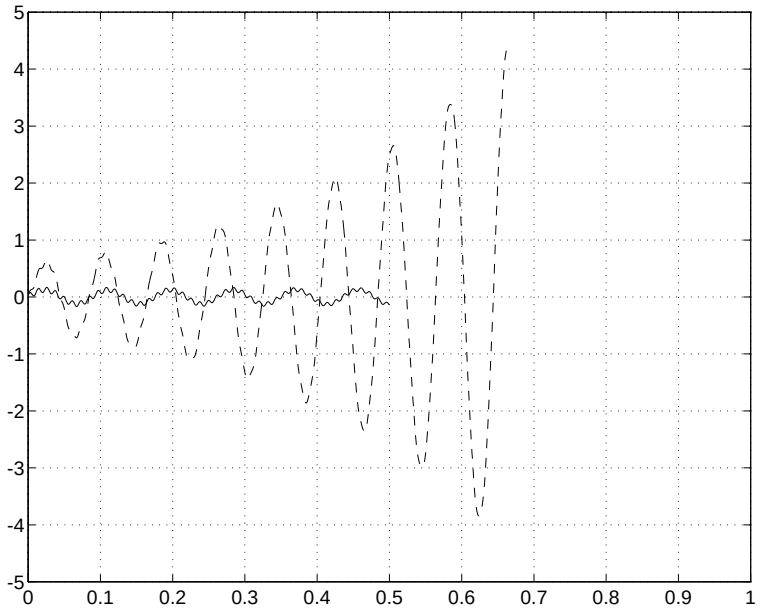

Figure 25: Lift coefficient C_l versus physical time t (in seconds)
$G_h = G_\theta = 0.75$, $\varphi_h = \varphi_\theta = \pi$
- - - - : Without active control surface
——- : With active control surface

Acknowledgments
The authors acknowledge partial support by the NASA Lewis Research Center under Grant NAG-31424, and partial support by the National Science Foundation under Grant ASC-9217394. They also wish to thank M. Loriot for his help in using the decomposer described in [10].

References

[1] BATINA J.T., *Unsteady Euler Airfoil Solutions Using Unstructured Dynamic Meshes*, AIAA Paper No. 89-0115, AIAA 27th Aerospace Sciences Meeting, Reno, Nevada, June 9-12, (1989).

[2] DONEA J., *An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Fluid-Structure Interactions*, Comp. Meth. Appl. Mech. Engng., Vol. 33, pp. 689-723, (1982).

[3] FARHAT C. - FEZOUI L. - LANTERI S., *Two-Dimensional Viscous Flow Computations on the CM-2: Unstructured Meshes, Upwind Schemes and Massively Parallel Computations*, Comp. Meth. in Appl. Mech. and Eng., Vol. 102, pp. 61-88, (1993).

[4] FARHAT C. - LANTERI S. - SIMON H., *TOP/DOMDEC : a Software Tool for Mesh Partitioning and Parallel Processing and Applications to CSM and CFD Computations*, Comput. Sys. Engrg. (To Appear)

[5] FARHAT C. - LIN T.Y., *Transient Aeroelastic Computations Using Multiple Moving Frames of Reference*, AIAA Journal, Vol. 31, No. 3, pp. 597-599, (1993).

[6] FARHAT C. - LESOINNE M., *Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics*, Internat. J. Numer. Meths. Engrg., Vol. 36, pp. 745-764, (1993).

[7] KENNEDY J. et al., *A Strategy for Implementing Implicit Finite Element Methods for Incompressible Fluids on the CM-5*, Symposium on Parallel Finite Element Computations, Minneapolis, Minnesota, October 24-27, (1993).

[8] LALLEMAND M.H., *Dissipative Properties of Runge-Kutta Schemes with Upwind Spatial Approximation for the Euler Equations*, INRIA Report No. 1173, (1990).

[9] LANTERI S. - C. FARHAT, *Viscous Flow Computations on M.P.P. Systems : Implementational Issues and Performance Results for Unstructured Grids*, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, pp. 65-70, (1993).

[10] LORIOT M., *MS3D : Mesh Splitter for 3D Applications, User's Manual*, SIMULOG, (1992).

[11] N'KONGA B. - GUILLARD H., *A Roe Scheme on Non-Structured Meshes for Moving Boundaries Problems*, Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics, Trinity College, Dublin, Ireland, July 22-26, Vol. 1, pp-447-449, (1191).

[12] ROE P. L., *Approximate Riemann Solvers, Parameters Vectors and Difference Schemes*, Journ. of Comp. Phys., Vol. 43, pp. 357-371, (1981).

[13] STEGER J. - WARMING R.F., *Flux vector splitting for the inviscid gas dynamic with applications to finite-difference methods*, Journ. of Comp. Phys., Vol. 40, (2), pp. 263-293, (1981).

[14] VAN LEER B., *Towards the Ultimate Conservative Difference Scheme V : a Second-Order Sequel to Godunov's Method*, Journ. of Comp. Phys., Vol. 32, pp. 361-370, (1979).

[15] ZDENEK J. - MATHUR K. K. - JOHNSSON S. L. - HUGHES T. J. R., *An Efficient Communication Strategy for Finite Element Methods on the Connection Machine CM-5 System*, Thinking Machines Technical Report No. 256, (1993).

[16] SIMON H., *Partitioning of Unstructured Problems for Parallel Processing*, Comput. Sys. Engrg., Vol. 2, pp. 135-148, (1991).

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399