Parallel Solutions of Three-Dimensional Compressible Flows

Stephane Lanteri

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallel Solutions of Three-Dimensional Compressible Flows

Stéphane Lanteri

N° 2594

Juin 1995

PROGRAMME 6

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Parallel Solutions of Three-Dimensional Compressible Flows

Stéphane Lanteri *

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

Rapport de recherche n° 2594 — Juin 1995 — 50 pages

Abstract: In this report, we present parallel solutions of realistic three-dimensional flows obtained on the `Intel Paragon`, the `Cray T3D` and the `Ibm SP2` MPPs (Massively Parallel Processors). The solver under consideration is a representative subset of an existing industrial code, `N3S-MUSCL` (a three-dimensional compressible Navier-Stokes solver, see Chargy[3]). It implements a mixed finite element/finite volume formulation on unstructured tetrahedral meshes. Defining a good strategy for the parallelisation of an unstructured mesh based solver is a challenge, particularly when one aims at reaching a high level of performance while maintaining portability of the source code between scalar, vector and parallel machines. The parallelisation strategy adopted in this study combines mesh partitioning techniques and a message-passing programming model. The mesh partitioning algorithms and the generation of the corresponding communication data-structures are gathered in a preprocessor in order to introduce a minimum change in the original serial code. The portability from one message passing parallel system to another may be enhanced with the use of a communication library such as `PVM`.

Key-words: Computational Fluid Dynamics, Three-dimensional flows, Euler equations, Navier-Stokes equations, Unstructured meshes, Parallel computing.

(Résumé : tsvp)

*INRIA Sophia-Antipolis, Projet Sinus, 2004 Route des Lucioles, B.P. 93, 06902 Sophia-Antipolis, (FRANCE), e-mail : lanteri@sophia.inria.fr

Résolutions Parallèles d'Ecoulements Compressibles Tridimensionnels

Résumé : Dans ce rapport nous présentons des résultats de simulations numériques d'écoulements tridimensionnels sur les calculateurs massivement parallèles `Intel Paragon`, `Cray T3D` et `Ibm SP2`. Le solveur considéré est un sous-ensemble représentatif du code industriel `N3S-MUSCL` (un solveur des équations de Navier-Stokes tridimensionnelles pour des fluides compressibles, voir Chargy[3]). Ce solveur est basé sur une formulation mixte éléments finis/volumes finis en maillages tétraédriques non-structurés. La parallélisation de solveurs en maillages non-structurés constitue une tâche non-triviale, en particulier lorsque la stratégie adoptée doit conduire à l'obtention de bonnes performances tout en maximisant la portabilité du code source résultant entre architectures séquentielles, vectorielles et parallèles. La stratégie de parallélisation adoptée ici combine des techniques de partitionnement de maillages et une programmation dans un modèle par transfert de messages. Par ailleurs, la portabilité du code source résultant peut être maximisée avec l'utilisation d'une librairie de communication standard telle que `PVM`.

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

Contents

1 Introduction

In the past ten years, computer technology has grown rapidly especially with regard to parallel architectures. Several configurations of massively parallel processors are today installed worldwide [5], most of them being part of academic or research institute computing means. In order for those parallel architectures to be widely adopted by the industrial community, significant advances in software environments that facilitate the parallelisation step while maintaining the portability, yet need to be achieved; on the other part, it is often necessary to demonstrate the capabilities of such computing architectures on realistic large-scale applications. In the field of computational fluid dynamics, codes that solve the Euler or even the full Navier-Stokes equations around or inside complex geometries are currently used in production mode by aircraft or engine manufacturers. These codes are often based on finite element discretisations of the computational domain. In the present paper, we consider the parallelisation of a representative subset of an existing industrial code, `N3S-MUSCL` (a three-dimensional compressible Navier-Stokes solver, see Chargy[3] and also [23] for some results on hypersonic flows) which runs on scalar and vector machines. The targeted parallel architectures are of MIMD type. This work is complementary to what was previously presented in Loriot and Fezoui[17] and Fezoui *et al.*[13]. Other works in this direction can be found in Johan *et al.*[14] and Morano and Mavriplis[19] for experiences on the Connection Machine CM-5, Mavriplis *et al.*[18] for the parallelisation of a multigrid Euler solver on the Intel Touchstone Delta.

Defining a way to implement a given serial algorithm on a parallel machine depends of course on the level of parallelism of the algorithm as well as on some of the characteristics of the selected machine, but also on some previously set objectives. As an example, in some of our previous works (see Fezoui and Lanteri[10], Farhat *et al.*[6]), our main goal was to achieve a maximum level of performance, thus accepting to introduce significant modifications in the original serial algorithm. This approach is considered feasible for "simple" code which generally deal with the solution of two-dimensional problems. Distributed solutions of three-dimensional problems using finite-element solvers based on fully unstructured meshes clearly represents a more challenging task. As a first condition, we require to minimize the programming effort on the original serial algorithm in order to be able to maintain and upgrade the resulting parallel version of the code. Moreover, we expect the latter to be as little processor architecture dependent as possible. That is, it must run on both serial and parallel machines (at least of the same kind) with no further modification. From this viewpoint, the highest efficiency to be achieved on any particular machine is left to an optimisation phase.

The main characteristic of `N3S-MUSCL` is that it is based on finite volume schemes using finite element type grids (tetrahedra), which results in complex data structures. The conservative form of the Navier-Stokes equations is discretised using a mixed finite element/finite volume method on fully unstructured meshes. The convective fluxes are computed by means of an upwind scheme which is chosen to be Roe's scheme. Second-order spatial accuracy is achieved by using an extension to unstructured meshes of the "Monotonic Upwind Scheme

for Conservative Laws" (MUSCL) method introduced by van Leer. A standard Galerkin approximation is used to evaluate the viscous fluxes. Two strategies are considered for advancing the solution in time. An explicit formulation (a predictor/corrector scheme) can be used for steady as well as for unsteady flow simulations. The second approach is best suited to steady flow simulations; it makes use of a linearised implicit formulation.

The parallelisation strategy adopted in this study has been already successfully applied in the two-dimensional case (see Fezoui *et al.*[11], Farhat and Lanteri[8]). It combines mesh partitioning techniques and a message-passing programming model. The mesh partitioning algorithms and the generation of the corresponding communication data-structures have been gathered in a a special purpose software package in order to introduce a minimum change in the original serial code. The portability from one message passing parallel system to another has been enhanced through the use of `PVM` for the implementation of the communication steps. However, depending on the targeted parallel system, we have also been interested in testing the native communication library proposed by the constructor. In that case, a small percentage of the code has required modifications in order to switch from `PVM` to the corresponding communication library. Two partitioning approaches have been investigated here. The first one makes use of overlapping mesh partitions; it contributes to minimize the programming effort on the original serial algorithm but is characterized by redundant arithmetic operations. The second strategy uses non-overlapping mesh partitions and demands additional programming effort. As it has been already noticed in the two-dimensional case (see Farhat and Lanteri[8] for more details), the latter strategy is the one that yields the best scalability properties of the resulting parallel solver.

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 parallelisation strategy. Finally, Section 4 reports and analyzes the performance results obtained on several parallel systems.

2 The FEM/FVM Navier-Stokes Solver

Here, we overview the spatial and temporal discretisation methods that are detailed in Chargy[3] for the numerical solution of the full three-dimensional Navier-Stokes equations.

2.1 Governing Equations

Let $\Omega \subset I\!R^3$ be the flow domain of interest and Γ be its boundary. The conservative law form of the equations describing three-dimensional Navier-Stokes flows is given by:

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

where $W = W(\vec{x}, t)$; $\vec{x}$ and t denote the spatial and temporal variables, and:

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

and:

$$\vec{\mathcal{F}}(W) = \begin{pmatrix} F_x(W) \\ F_y(W) \\ F_z(W) \end{pmatrix} \quad , \quad \vec{\mathcal{R}}(W) = \begin{pmatrix} R_x(W) \\ R_y(W) \\ R_z(W) \end{pmatrix}$$

$F_x(W)$, $F_y(W)$ and $F_z(W)$ denote the convective fluxes and are given by:

$$F_x(W) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{pmatrix} \quad , \quad F_y(W) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{pmatrix} \quad , \quad F_z(W) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{pmatrix}$$

while $R_x(W)$, $R_y(W)$ and $R_z(W)$ denote the diffusive fluxes and are given by:

$$R_x(W) = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + \dfrac{\gamma k}{Pr}\dfrac{\partial \varepsilon}{\partial x} \end{pmatrix}$$

$$R_y(W) = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + \dfrac{\gamma k}{Pr}\dfrac{\partial \varepsilon}{\partial y} \end{pmatrix}$$

$$R_z(W) = \begin{pmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + \dfrac{\gamma k}{Pr}\dfrac{\partial \varepsilon}{\partial z} \end{pmatrix}$$

In the above expressions, ρ is the density, $\vec{U} = (u, v, w)^T$ is the velocity vector, E is the total energy per unit of volume, p is the pressure, ε is the specific internal energy, k is the

normalised 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; τ_{xx}, τ_{xy}, τ_{xz}, τ_{yz}, τ_{yy} and τ_{zz} are the components of the three-dimensional Cauchy stress tensor. 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:

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

2.2 Boundary Conditions

The boundary Γ of the flow domain is partitioned into a wall boundary Γ_w and an infinity boundary Γ_∞: $\Gamma = \Gamma_w \cup \Gamma_\infty$. Let $\vec{n}$ denote the outward unit normal at any point of Γ, and T_w denote the wall temperature. On the wall boundary Γ_w, a no-slip condition and a Dirichlet condition on the temperature are imposed:

$$\vec{U} = \vec{0} \quad , \quad T = T_w \tag{2}$$

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:

$$p = (\gamma - 1)\rho C_v T_w \quad , \quad E = \rho C_v T_w \tag{3}$$

The viscous effects are assumed to be negligible at infinity, so that a uniform free-stream state vector W_∞ is adopted as a representation of the solution on Γ_∞:

$$\rho_\infty = 1 \quad , \quad \vec{U}_\infty = (u_\infty \ , \ v_\infty \ , \ w_\infty) \quad \text{with} \quad \parallel \vec{U}_\infty \parallel = 1 \quad , \quad p_\infty = \frac{1}{\gamma M_\infty^2} \tag{4}$$

where α is the angle of attack, and M_∞ is the free-stream Mach number. This approach is generally well suited to external flows around bodies, however for internal flows the interaction between the body surface and the freestream boundaries can yield to the apparition of non-physical oscillations. In that case an adapted definition of the above free-stream state vector (for instance by taking into account a velocity profile) or a more sophisticated treatment of the corresponding boundary conditions may be necessary.

2.3 Spatial Discretisation

The flow domain Ω is assumed to be a polyhedral bounded region of IR^3. Let $\mathcal{T}_h$ be a standard tetrahedrisation of Ω, and h the maximal length of the edges of $\mathcal{T}_h$. A vertex of a tetrahedron T is denoted by S_i, and the set of its neighboring vertices by $K(i)$. At each vertex S_i, a control volume C_i is constructed as the union of the subtetrahedra resulting from the subdivision by means of the medians of each tetrahedron of $\mathcal{T}_h$ that is connected to S_i (see Fig. 1). The boundary of C_i is denoted by ∂C_i, and the unit vector of the outward normal to ∂C_i by $\vec{\nu}_i = (\nu_{ix}, \nu_{iy}, \nu_{iz})$. The union of all these control volumes constitutes a discretisation of domain Ω:

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

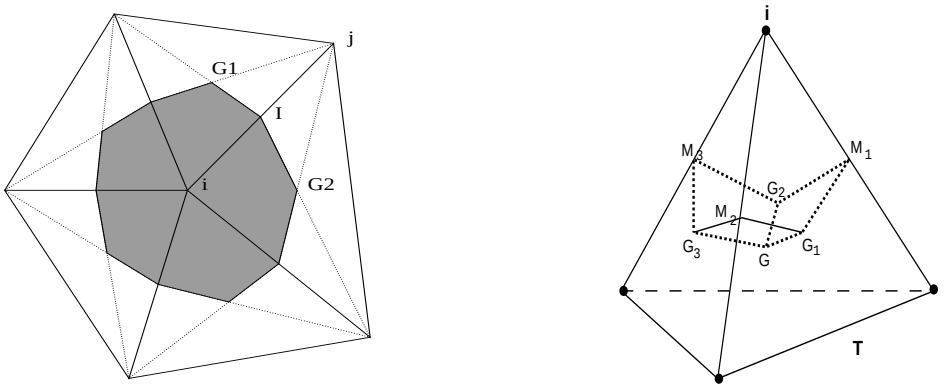

Figure 1: 2D control surface (left) and contribution to a 3D control volume (right)

The spatial discretisation method adopted here combines the following features :

- a finite volume upwind approximation method for the convective fluxes. Second order spatial accuracy is achieved using an extension of van Leer's[24] MUSCL technique to unstructured meshes;
- a classical Galerkin finite element centered approximation for the diffusive fluxes.

Integrating Eq. (1) over C_i yields:

$$\iiint_{C_i} \frac{\partial W}{\partial t} d\vec{x} + \iiint_{C_i} \vec{\nabla}.\vec{\mathcal{F}}(W) d\vec{x} = \iiint_{C_i} \frac{1}{Re} \vec{\nabla}.\vec{\mathcal{R}}(W) d\vec{x} \tag{5}$$

Finally, integrating Eq. (5) by parts leads to:

$$
\begin{aligned}
\iiint_{C_i} \frac{\partial W}{\partial t} d\vec{x} \;&+\; \sum_{j \in K(i)} \int_{\partial C_{ij}} \vec{\mathcal{F}}(W).\vec{\nu}_i d\sigma && < 1 > \\
&+ \int_{\partial C_i \cap \Gamma_w} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma && < 2 > \\
&+ \int_{\partial C_i \cap \Gamma_\infty} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma && < 3 > \\
&= -\frac{1}{Re} \sum_{T, S_i \in T} \iiint_T \vec{\mathcal{R}}(W).\vec{\nabla} N_i^T d\vec{x} && < 4 >
\end{aligned}
\tag{6}
$$

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

2.3.1 Convective Fluxes Computation

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

$$
< 1 > = \; W_i^{n+1} - W_i^n + \Delta t \sum_{j \in K(i)} \Phi_{\mathcal{F}}(W_i^n, W_j^n, \vec{\nu}_{ij}) \tag{7}
$$

where $\Phi_{\mathcal{F}}$ denotes a numerical flux function such that:

$$
\Phi_{\mathcal{F}}(W_i, W_j, \vec{\nu}_{ij}) \approx \int_{\partial C_{ij}} \vec{\mathcal{F}}(W).\vec{\nu}_i d\sigma \tag{8}
$$

Upwinding can be introduced in the computation of Eq. (8) by using Roe's[20] approximate Riemann solver thus computing $\Phi_{\mathcal{F}}$ as follows:

$$
\Phi_{\mathcal{F}}(W_i, W_j, \vec{\nu}_{ij}) = \frac{\vec{\mathcal{F}}(W_i) + \vec{\mathcal{F}}(W_j)}{2}.\vec{\nu}_{ij} - \mid \mathcal{A}_R(W_i, W_j, \vec{\nu}_{ij}) \mid \frac{(W_j - W_i)}{2} \tag{9}
$$

where $\mathcal{A}_R$ is Roe's mean value of the flux Jacobian matrix $\dfrac{\partial \vec{\mathcal{F}}(W)}{\partial W}.\vec{\nu}$.

Following the MUSCL technique, second order accuracy is achieved in Eq. (8) via a piecewise linear interpolation of the states W_{ij} and W_{ji} at the interface between control volumes C_i and C_j. This requires the evaluation of the gradient of the solution at each vertex as follows:

$$
\tilde{W}_{ij} = \tilde{W}_i + \frac{1}{2}(\vec{\nabla}\tilde{W})_i.\vec{S_iS_j} \quad , \quad \tilde{W}_{ji} = \tilde{W}_j - \frac{1}{2}(\vec{\nabla}\tilde{W})_j.\vec{S_iS_j} \tag{10}
$$

where $\tilde{W} = \left(\rho \;,\; \vec{U} \;,\; p\right)^T$ — in other words, the interpolation is performed on the physical variables instead of the conservative variables. The approximate nodal gradients $(\vec{\nabla}\tilde{W})_i$ are

obtained by means of a linear interpolation of the Galerkin gradients computed on each tetrahedron of C_i:

$$\left(\vec{\nabla}\tilde{W}\right)_i = \frac{\iiint_{C_i} \vec{\nabla}\tilde{W}|_T d\vec{x}}{\iiint_{C_i} d\vec{x}} = \frac{1}{\mathrm{vol}(C_i)} \sum_{T \epsilon C_i} \frac{\mathrm{vol}(T)}{4} \sum_{k=1,k\epsilon T}^{4} \tilde{W}_k \vec{\nabla} N_k^T \tag{11}$$

The construction given by Eq. (10-11) results in a half-upwind (Fromm-like) scheme which is spatially second order accurate but may present spurious oscillations in the solutions, expressing a loss of monotony. One way to circumvent this problem is to make a compromise between the first order scheme and the second order one by applying slope limitation procedures. The one selected here consists in the following :

- in order to measure the upstream and downstream variations of the unknown on the edge $E_{ij} = \{S_i, S_j\}$, we introduce fictitious values $\tilde{W}_{ij}^f$ and $\tilde{W}_{ji}^f$ in addition to the nodal values $\tilde{W}_i$ and $\tilde{W}_j$; they are derived from the nodal gradients (11) as follows :

$$\begin{cases} \tilde{W}_{ij}^f &= \tilde{W}_i - 2(\vec{\nabla}\tilde{W})_i . \vec{S_i S_j} + (\tilde{W}_j - \tilde{W}_i) \\ \tilde{W}_{ji}^f &= \tilde{W}_j - 2(\vec{\nabla}\tilde{W})_j . \vec{S_i S_j} - (\tilde{W}_j - \tilde{W}_i) \end{cases}$$

- approximate values of variations of the unknown W are then obtained by using the van Albada limiter[25] which writes as :

$$\begin{cases} d\tilde{W}_{ij} &= \mathrm{ave}[(\tilde{W}_j - \tilde{W}_i), (\tilde{W}_i - \tilde{W}_{ij}^f)] \\ d\tilde{W}_{ji} &= \mathrm{ave}[(\tilde{W}_i - \tilde{W}_j), (\tilde{W}_j - \tilde{W}_{ji}^f)] \end{cases}$$

where :

$$\mathrm{ave}[a,b] = \begin{cases} \dfrac{a(b^2+\varepsilon^2) + b(a^2+\varepsilon^2)}{a^2+b^2+\varepsilon^2} & \text{if } a.b > 0 \\ 0 & \text{else} \end{cases}$$

Limited arguments for the numerical flux function $\Phi_{\mathcal{F}}$ are then computed by :

$$\tilde{W}_{ij} = \tilde{W}_i + \frac{1}{2} d\tilde{W}_{ij} \quad , \quad \tilde{W}_{ji} = \tilde{W}_j + \frac{1}{2} d\tilde{W}_{ji} \tag{12}$$

When considering flows for which the viscous terms are dominant (small values of the Reynolds number) the above limitation procedure is generally not necessary. Indeed, the second order upwind approximation obtained using Eq. (8-9) and Eq. (10-11) is characterised by the introduction of an amount of numerical diffusion which can prevent a good representation of the viscous effects. One way to minimize the effect of the numerical diffusion consists in combining centered and fully upwind slopes in the computation of the nodal gradients :

$$(\vec{\nabla}\tilde{W})_i^{\beta} = (1-\beta)(\vec{\nabla}\tilde{W})_i^{Cent} + \beta(\vec{\nabla}\tilde{W})_i^{Upw} \tag{13}$$

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

$$(\vec{\nabla}\tilde{W})_i^{Cent}.\vec{S_iS_j} = \tilde{W}_j - \tilde{W}_i \tag{14}$$

Then a simple way to compute the upwind part ($\beta = 1$) of the nodal gradient (13) goes as follows :

$$(\vec{\nabla}\tilde{W})_i^{Upw} = 2(\vec{\nabla}\tilde{W})_i^{\beta=\frac{1}{2}} - (\vec{\nabla}\tilde{W})_i^{Cent}$$

where $(\vec{\nabla}\tilde{W})_i^{\beta=\frac{1}{2}}$ is given by (11).

The second term $< 2 >$ and the third term $< 3 >$ of Eq. (6) 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[22]:

$$\int_{\partial C_i \cap \Gamma_\infty} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma = \mathcal{A}^+(W_i, \vec{n}_{i\infty}).W_i + \mathcal{A}^-(W_i, \vec{n}_{i\infty}).W_\infty \tag{15}$$

2.3.2 Viscous Fluxes Computation

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 tetrahedron. The velocity vector in a tetrahedron is computed as follows:

$$\vec{U}_T = \frac{1}{4}\sum_{k=1,k\epsilon T}^{4} \vec{U}^k$$

and the viscous fluxes are approximated as follows:

$$\vec{\mathcal{R}}_i(T) = \iiint_T \vec{\mathcal{R}}(W).\vec{\nabla} N_i^T \, d\vec{x} = \mathrm{vol}(T) \left(R_x(T) \frac{\partial N_i^T}{\partial x} + R_y(T) \frac{\partial N_i^T}{\partial y} + R_z(T) \frac{\partial N_i^T}{\partial z} \right)$$

where $R_x(T)$, $R_y(T)$ and $R_z(T)$ are constant values on the tetrahedron T.

2.4 Time Integration

Assuming that $W(\vec{x}, t)$ is constant over the control volume C_i (in other words, a mass lumping technique is applied to the temporal term of Eq. (6)), we obtain the following semi-discrete fluid flow equations:

$$\mathrm{vol}(C_i) \frac{dW_i^n}{dt} + \Psi(W_i^n) = 0 \quad , \quad i = 1, \cdots, N_V \tag{16}$$

where $W_i^n = W(\vec{x}_i, t^n)$, $t^n = n\Delta t^n$ and:

$$\Psi(W_i^n) = \sum_{j \in K(i)} \Phi_{\mathcal{F}}(W_{ij}, W_{ji}, \vec{\nu}_{ij}) + \int_{\partial C_i \cap \Gamma_\infty} \vec{\mathcal{F}}(W).\vec{n}_i d\sigma + \frac{1}{Re} \sum_{T, S_i \in T} \vec{\mathcal{R}}_i(T) \tag{17}$$

2.4.1 Explicit Time Advancing Procedure

A predictor-corrector can be selected for time integrating the semi-discrete equations Eq. (16). This explicit scheme which is of second-order accuracy and cheap in terms of CPU costs (this is often a requirement of industrial codes). First we predict a state $\tilde{W}^{n+\frac{1}{2}}$ using the Euler equations :

$$\tilde{W}_i^{n+\frac{1}{2}} = \tilde{W}_i^n - \frac{\Delta t^n}{2} \tilde{\mathcal{A}}(W_i^n).(\vec{\nabla} \tilde{W})_i \tag{18}$$

In the second phase (correction), the fluxes are evaluated using the predicted state:

$$W_i^{n+1} = W_i^n - \Delta t^n \left[\sum_{j \in K(i)} \Phi(\tilde{W}_{ij}^{n+\frac{1}{2}}, \tilde{W}_{ji}^{n+\frac{1}{2}}, \vec{\nu}_{ij}) + \frac{1}{Re} \sum_{T, S_i \in T} \vec{\mathcal{R}}_i(T) \right] \tag{19}$$

2.4.2 Implicit Time Advancing Procedure

Explicit time integration procedures are subjected to a stability condition expressed in terms of a CFL (Courant-Friedrichs-Lewy) number. When one is interested in looking for steady solutions of the Euler or Navier-Stokes equations, an efficient time advancing strategy can be obtained by means of an implicit linearised formulation. Here, we briefly recall the main ingredients of this approach which is described in details in Fezoui and Stoufflet[12]. A Newton-like implicit linearised version of Eq. (16) writes as :

$$\frac{\mathrm{vol}(C_i)}{\Delta t^n}\delta W_i^{n+1} + \Psi(W_i^{n+1}) = 0 \quad , \quad i = 1, \cdots, N_V \tag{20}$$

where $\delta W_i^{n+1} = W_i^{n+1} - W_i^n$. In Eq. (20), an elementary convective flux is computed as :

$$\left\{ \begin{array}{ll} \tilde{\Phi}_{\mathcal{F}} & = \tilde{\Phi}_{\mathcal{F}}(W_i^n, W_j^n, W_i^{n+1}, W_j^{n+1}, \vec{\nu}_{ij}) \\ & \\ & = \Phi_{\mathcal{F}}(W_i^n, W_j^n, \vec{\nu}_{ij}) + \left[\frac{\partial \Phi_{\mathcal{F}}}{\partial W_i^n}\right] \delta W_i^{n+1} + \left[\frac{\partial \Phi_{\mathcal{F}}}{\partial W_j^n}\right] \delta W_j^{n+1} \end{array} \right. \tag{21}$$

A similar expression can be written for the computation of an elementary diffusive fluxe. Suppose now that the numerical flux function of Eq. (8) can be written as :

$$\left\{ \begin{array}{l} \Phi_{\mathcal{F}}(U, V, \vec{\nu}) = H_1(U, V, \vec{\nu})U + H_2(U, V, \vec{\nu})V \\ \\ H_1(U, U, \vec{\nu})U + H_2(U, U, \vec{\nu})U = \mathcal{A}(U, \vec{\nu}) \end{array} \right. \tag{22}$$

where the first of Eq. (22) is a tentative linearisation while the second one is a consistency property. For instance the Steger and Warming[22] flux-splitting is such that :

$$H_1(U, V, \vec{\nu}) = \mathcal{A}^+(U, \vec{\nu}) \quad , \quad H_2(U, V, \vec{\nu}) = \mathcal{A}^-(V, \vec{\nu})$$

In this study, we make use of a linearisation of Roe's numerical flux function (9).

The derivatives of $\Phi_{\mathcal{F}}$ may be very expensive to compute, therefore we introduce a simplified linearised version which make use of Eq. (22) :

$$\left\{ \begin{array}{lcl} \tilde{\Phi}_{\mathcal{F}} & = & \tilde{\Phi}_{\mathcal{F}}(W_i^n, W_j^n, W_i^{n+1}, W_j^{n+1}, \vec{\nu}_{ij}) \\ & & \\ & = & \Phi_{\mathcal{F}}(W_i^n, W_j^n, \vec{\nu}_{ij}) + \\ & & \\ & & H_1(W_i^n, W_j^n, \vec{\nu}_{ij})\delta W_i^{n+1} + H_2(W_i^n, W_j^n, \vec{\nu}_{ij})\delta W_j^{n+1} \end{array} \right. \tag{23}$$

The resulting scheme is in fact a modified Newton method where the exact Jacobians arising in Eq. (22) are replaced by simpler expressions. As a consequence, we cannot ensure that this formulation will yield a quadratically converging method for time steps tending to infinity.

The resulting Euler implicit time integration scheme is given in matrix form by :

$$P(W^n)\delta W^{n+1} = \left(\frac{I}{\Delta t^n} + J(W^n)\right) \delta W^{n+1} = \delta \hat{W}^n \tag{24}$$

where $J(W^n)$ denotes the approximate Jacobian matrix and $\delta \hat{W}^n$ is the explicit part of the linearisation of $\Psi(W^{n+1})$. The matrix $P(W^n)$ is sparse and has the suitable properties (diagonaly dominant in the scalar case) allowing the use of a relaxation procedure (Jacobi

or Gauss-Seidel) in order to solve the linear system of Eq. (24). Moreover, an efficient way to get second order accurate steady solutions while keeping the interesting properties of the first order upwind matrix is to use the second order elementary convective fluxes based on Eq. (10) in the right-hand side of Eq. (24).

3 Computational and Parallel Implementation Issues

3.1 Identification of the Main Computational Kernels

From the description of the proposed numerical solution algorithm, it follows that our fluid solver contains essentially two types of elementary computations, one based on the mesh edges (i.e. for the convective fluxes computation), and the other based on the mesh tetrahedra (i.e. for the diffusive fluxes computation). Both type of computations can be described as three-step sequences of the form *Gather/Compute/Scatter*.

3.1.1 Edge Based Computations

The evaluation of the second term of $< 1 >$ in Eq. (6) using the numerical flux function $\Phi_{\mathcal{F}}$ of Eq. (8) with the second order approximation outlined in Eq. (10) can be summarized as follows:

$$\begin{cases} H_{ij} &= \Phi_{\mathcal{F}}(W_{ij}, W_{ji}, \vec{\nu}_{ij}) \approx \int\limits_{\partial C_{ij}} \vec{\mathcal{F}}(W).\vec{\nu}_i d\sigma \\ H_{ji} &= -H_{ij} \end{cases} \tag{25}$$

where: $\vec{\nu}_{ij} = \int\limits_{\partial C_{ij}} \vec{\nu}_i d\sigma = \vec{\nu}_1 + \vec{\nu}_2$

Essentially, one-dimensional elementary convective fluxes are computed at the intersection between the control volumes C_i and C_j . Each elementary flux contributes to a flux balance at the boundary of the control volume C_i. This balance involves the accumulation over the set of neighboring vertices $K(i)$ of all computed fluxes. From the second of Eqs. (25), 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 $W_i = W(S_i)$, $W_j = W(S_j)$

Gather $\vec{\nabla}\tilde{W}_i = \vec{\nabla}\tilde{W}(S_i)$, $\vec{\nabla}\tilde{W}_j = \vec{\nabla}\tilde{W}(S_j)$

Compute $H_{ij} = \Phi_{\mathcal{F}}(W_{ij}, W_{ji}, \vec{\nu}_{ij})$

Scatter $\Phi_i = \Phi_i + H_{ij}$, $\Phi_j = \Phi_j - H_{ij}$

End Do

3.1.2 Tetrahedron Based Computations

In the last term $< 4 >$ of Eq. (6), the elementary diffusive flux $\vec{\mathcal{R}}_i(T)$ is constant in each tetrahedron T. Its evaluation requires accessing the values of the physical state W at the four vertices of tetrahedron T. The values of $R_x(T)$, $R_y(T)$ and $R_z(T)$ contribute to the diffusive fluxes at all four vertices of tetrahedron T. Clearly, the most efficient way for evaluating the diffusive fluxes is to loop over the list of the mesh tetrahedra and compute as follows:

For each tetrahedron $T_{ijkl} = \{S_i, S_j, S_k, S_l\}$ of $\mathcal{T}_h$ *Do*

Gather $W_i = W(S_i)$, $W_j = W(S_j)$, $W_k = W(S_k)$, $W_l = W(S_l)$

Compute $R_x(T)$, $R_y(T)$, $R_z(T)$

Scatter $\mathcal{V}_i = \mathcal{V}_i + \vec{\mathcal{R}}_i(T)$, $\mathcal{V}_j = \mathcal{V}_j + \vec{\mathcal{R}}_j(T)$, $\mathcal{V}_k = \mathcal{V}_k + \vec{\mathcal{R}}_k(T)$, $\mathcal{V}_l = \mathcal{V}_l + \vec{\mathcal{R}}_l(T)$

End Do

The evaluation of the half-upwind nodal gradient of Eq. (11) follows the same computational pattern described above.

3.2 Parallelisation Strategy

The parallelisation strategy adopted in this study has been already successfully applied in the two-dimensional case (see Fezoui *et al.*[11], Farhat and Lanteri[8]). Preliminary results in

the three-dimensional case are also presented in Loriot and Fezoui[17] and Fezoui *et al.*[13]. It combines mesh partitioning techniques and a message-passing programming model. The underlying mesh is assumed to be partitioned into several submeshes, each defining a subdomain. Basically the same "old" serial code is going to be executed within every subdomain. Modifications occured in the main time-stepping loop in order to take into account one or several assembly phases of the subdomain results, depending on the order of the spatial approximation and on the nature of the time advancing procedure (explicit/implicit). The assembly of the subdomain results can be implemented in one or several separated modules and optimized for a given machine. This approach enforces data locality, and therefore is suitable for all parallel hardware architectures.

The reader can verify that for the computations described herein, mesh partitions with overlapping simplify the programming of the subdomain interfacing module. However, mesh partitions with overlapping also have a drawback: they incur redundant floating-point operations. On the other hand, non-overlapping mesh partitions incur little redundant floating-point operations but induce additional communication steps. 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. In addition, special care must be taken in the treatment of the convective fluxes in the case of non-overlapping mesh partitions (because of the possible differences in the orientation of the interface edges which are not part of the original mesh but are instead constructed during a preprocessing phase of the parallel algorithm). In other words, the programming effort is maximized when considering non-overlapping mesh partitions. We refer to Farhat and Lanteri[8] for a comparison of these two approaches in the context of two-dimensional simulations. In the present study we will consider both one tetrahedra wide overlapping and non-overlapping mesh partitions for second order accurate explicit and implicit computations.

3.3 Parallel Algorithms

For an explicit time integration procedure and a one tetrahedra wide overlapping mesh partition, the main loop of the parallel fluid solver described herein goes as follows:

Repeat $step = step + 1$

Compute the local time steps
Compute the nodal gradients and the diffusive fluxes
Exchange the nodal gradients
Compute the convective fluxes
Update the physical states
Exchange the conservatives variables

Until $step = step_{max}$

In the above pseudo code, $step_{max}$ denotes the maximum number of time steps. For an implicit time integration procedure, the update phase is repaced by the following two-step solution procedure:

Forms the implicit matrix
For $srl = 1$ to $nsrl$ **Do**

Exchange the right-hand sides
Perform a Jacobi relaxation

End Do

where $nsrl$ denotes the number of Jacobi relaxations that need to be done in order to approximately solve the linear system arising at each time step. One can notice that a global solution strategy has been selected through the choice of the Jacobi method which is naturally parallelisable.

When a non-overlapping mesh partition is used, the explicit parallel algorithm becomes:

Repeat $step = step + 1$

Compute the local time steps
Compute the nodal gradients and the diffusive fluxes
Exchange the partially gathered nodal gradients
Compute the convective fluxes
Exchange the partially gathered nodal fluxes
Update the physical states

Until $step = step_{max}$

While for an implicit time integration procedure we obtain:

```
Forms the implicit matrix
Exchange the partially gathered diagonal blocks of the implicit matrix
For srl = 1 to nsrl Do

   Exchange the the partially gathered right-hand sides
   Perform a Jacobi relaxation

End Do
```

We can therefore expect a lower communication cost for an implicit computation using a one tetrahedra wide overlapping mesh partition as suggested by the additional communication step involving the diagonal blocks of the implicit matrix in the non-overlapping case. On the other part, the above pseudo-codes only show local communication steps at artificial submesh boundaries; indeed, global communication steps (reduction operations) are also necessary for the computation of the non-linear (time stepping loop) and linear residuals (linear system resolutions).

3.4 Automatic Mesh Partitioning

For the time integration procedures considered in this study, an automatic mesh partitioner should focus primarily on creating load balanced submeshes which induce a minimum amount of interprocessor communications. This can be achieved by using a two-step procedure. First, a fast and cheap partitioning scheme is used to derive an initial candidate; then, an optimisation process is performed in order to realise the stated goals. While the former step consists in a global operation (the overall mesh is concerned by this step), the latter mainly concentrate on those mesh components that are neighbors of the artificial submesh interfaces (local operation). Optimisation techniques that are used in this context include (among others) simulated annealing and the Kernighan-Lin algorithm. Mesh partitioning algorithms can exploit the mesh connectivity as it is the case for the Greedy algorithm (see Farhat and Lesoinne[7]). This algorithm basically bites into the mesh in order to construct every submesh. They can also be based on geometric informations such as the principal inertia directions of the mesh. In that case, a direction is first specified (one of the principal inertia directions). Next, the mesh vertices are projected (orthogonal projection) onto that direction. Finally, the projected nodes are sorted along that direction then collected to build the requested submeshes. Variations of the above algorithm can be obtained by applying the project, sort and collect paradigm in a recursive manner. More sophisticated algorithms are graph theory based such as the recursive graph bisection or the recursive spectral bisection algorithms (see Simon[21] for more details). We refer to Farhat and Lanteri[8] for an

evaluation of the influence of the mesh partitioning algorithm on two-dimensional parallel simulations.

In the present study, the computational mesh is partitioned in a preprocessing step. We have used two special purpose packages that implement several mesh partitioning algorithms : `MS3D` (a Mesh Splitter for 3D applications, a description of a two-dimensional version of this preprocessor with a set of experimental results may be found in Loriot[16] and Fezoui *et al.*[11]) for the construction of one tetrahedra wide overlapping mesh partitions, and `TOP/DOMDEC` (a software tool for mesh partitioning and parallel processing of CSM and CFD computations [9]) to generate non-overlapping ones.

4 Performance Results

4.1 Parallel Platforms Description

In this section, we discuss the parallel performance results obtained on various configurations of the `Intel Paragon`, the `Ibm SP2` and the `Cray T3D` parallel processors. We also give results from experiments performed on a network of workstations. The main characteristics of the targeted systems are briefly described below.

`Intel Paragon` : like its predecessors, the prototypical `Touchstone Delta` system and the `iPSC-860`, the `Intel Paragon XP/S` is a distributed memory computer. Its architecture supports Multiple Instruction Multiple Data stream (MIMD) and most notably Single Program Multiple Data (SPMD) styled applications. The `Intel Paragon` processing nodes are arranged in a two-dimensional rectangular grid. The system contains compute nodes, service nodes and I/O nodes. Compute nodes are used for execution of parallel programs; service nodes offer the capabilities of a UNIX system including compilers and program development tools thus making a traditional front-end computer unnecessary; I/O nodes are interfaces to mass storage and Local Area Networks (LANs). The data network is constructued on the basis of Mesh Routing Chips (MRCs) which are connected by high-speed channels. The network has a theoretical bidirectional bandwith of 175 Mb/s. Each processing node consists of two `i860XP` RISC processors. One of them is working as the *application processor*, the other one as the *message processor*. They share a common memory. The purpose of the *message processor* is to relieve the *application processor* from the overhead work related to message-passing. The *message processor* sends and receives messages from the other nodes via a network interface. The `i860XP` processor runs at 50 Mhz. It has two pipes for floating-point operations, an adder and a multiplier, each able to provide a result (64 bit IEEE arithmetic) at every cycle. This adds up to 75 Mflop/s peak performance for a single `i860XP` processor.

`Cray T3D` : the `Cray T3D` massively parallel processing system is a scalable MIMD system with a physically distributed, globally addressable memory. The `Cray T3D` compute

nodes are comprised of two processing elements each with a CPU, local memory and a MCU (Memory Control Unit). These processing elements are coupled by an extremely fast interconnection network with a transfer rate at 300 Mb/s per data channels. The data channels are bidirectional and independent in each of the three dimensions x, y and z. The `Cray T3D` network topology is a three-dimensional torus, i.e. a three-dimensional grid with wrap-around connections. This topology ensures short connection paths as well as a high bisection bandwith. The `Cray T3D` system employs the `DEC Alpha` processor which operates at a clock speed of 150 Mhz (6.6 ns). This superscalar single chip processor can initiate a floating-point operation, a load or a store, or an integer operation in one cycle. Thus a nominal peak performance of 150 Mflop/s is achieved for 64 bit IEEE format. The `Cray T3D` system is tightly coupled to a `Cray Y-MP` or a `Cray C90` system where development and compilation activities, as well as submition of jobs take place.

`Ibm SP2` : blending the best attributes of workstations and high performance computers, the `Ibm 9076 SP2` is designed to provide a parallel computing environment while being effective for serial workloads, for both batch and interactive applications. Based on the `Ibm` RISC System/6000 66 Mhz `POWER2` microprocessor, the `SP2` can be configured with *thin* or/and *wide* nodes. The primary differences between *thin* and *wide* nodes are 64Kb versus 256 Kb data cache size, 64 bit versus 256 bit memory bus width and 128 bit versus 256 bit processor to data cache bus width. Because of these architectural differences, codes may not achieve the same performance on *thin* and *wide* nodes even though both are based on the same microprocessor. The 66 Mhz `POWER2` microprocessor is capable of 266 Mflop/s peak performance (64 bit IEEE format). The `SP2` offers a great flexibility concerning the implemented communication strategy. Indeed, users are able to connect to early every network they may have installed. Offering broader connectivity and higher communication speeds, the Micro Channel adapters of the `SP2` include among others HIPPI, FDDI and ATM. Based on a technology developed by `Ibm`, the High-Performance Switch (HPS) is designed to provide efficient communication between the `SP2` nodes. It is designed such that the point-to-point inter-processor communication time is independent of the relative location of the communicating nodes. A connection between any two nodes supports a peak bi-directional bandwith of 40 Mb/s with a hardware latency of 500 ns (up to 80 nodes) and 875 ns (80-512 nodes). An important characteristic of the switch is that the bisectional bandwith is designed to scale linearly to thousands of nodes, with an essentially constant latency per connection. Hence, the necessary balance between inter-processor communication speed and the total system compute power is retained as the number of processor nodes is increased.

4.2 Programming Languages and Environments

We emphasize the fact that essentially the same code is used on each of the above platforms the main differences being the following :

- on the Intel Paragon, communication steps are performed using the NX native communication library;
- on the Ibm SP2, the MPL native communication library is used;
- the Cray optimized PVM library is used on the Cray T3D.

Computations have also been performed on a group of Dec 3000/600 workstations. In that case, communication steps are performed using PVM version 3.3, the tested workstations being part of a larger Ethernet based local network. The Dec 3000/600 workstation is based on a DEC Alpha chip. The programming language is Fortran 77. The compilation options that have been used are -Mr8 -Mr8intrinsics -O4 -Knoieee on the Intel Paragon, -O1 on the Cray T3D and -qdpc -qstrict -qarch=pwr2 -qautodbl=dblpad -O3 on the Ibm SP2.

Timing measures concern the main parallel loops as depicted in the previous section. All performance results reported herein are for 64-bit arithmetic; on the other part, the redundant floating-point operations are not accounted for when evaluating the Mflop rate. Unless stated otherwise, the reported CPU times always refer to the maximum of the individual processor measures. In the following tables N_p is the number of involved processors (submeshes) while "Loc Comm" and "Glb Comm" respectively denote the local (send/receive at artificial submesh boundaries) and global communication times. In the case of local communication operations, the corresponding measures include the time spent in packing and unpacking message buffers. On the other part, explicit synchronisation points have been inserted prior to entering the update phases at the artificial submesh boundaries; this allow a precise timing of pure local communication operations without taking into account the idle times due to computational load imbalance.

As a covention, we shall use the terms "linear iteration" and "linear residual" when referring to the linear system solution phase in the implicit formulation (24); for the non-linear iteration (the pseudo-time step iteration) the corresponding terms will be "non-linear iteration" and "non-linear residual".

4.3 External Flow Around an ONERA M6 Wing

We consider here the Euler flow around an ONERA M6 wing. The angle of attack is set to 3.06° and the free stream Mach number to 0.84; parallel solutions for this test case are presented in Morano and Mavriplis[19] and in Johan *et al.*[14]. Five meshes with increasing sizes have been generated. Their characteristics are summarized in Tab. 1 where N_V denotes the number of vertices, N_T the number of tetrahedra.

4.3.1 Computational Scalability For Increasing Size Problems

Parallel computational scalability (in contrast to numerical scalability) is evaluated for problems where the subdomain size is fixed, and the total size is increased with the number of

Table 1: Five meshes and their characteristics for an `ONERA M6` wing

MESH	N_V	N_T
M1	2203	10053
M2	15460	80424
M3	31513	161830
M4	63917	337604
M5	115351	643392

processors. Note that because we are dealing with unstructured meshes, some slight deviations are inevitable.

We begin by computations performed on the `Intel Paragon` using the explicit predictor/corrector time advancing procedure (18)-(19). Tab. 2 summarizes the results obtained with overlapping and non-overlapping mesh partitions. Timing measures have been obtained for 100 iterations of the predictor/corrector scheme. Fig. 2 gives a graphic interpretation of the evolution of the total CPU time versus the number of processors. As the number of processors increase, one observe a 16.5% degradation of the scalability when using overlapping mesh partitions; this figure reduces to 10.5% for non-overlapping mesh partitions. In the former case, the communication time represents 2% of the total CPU time while it increases to 4% in the latter case.

Table 2: Explicit Euler computations on the `Intel Paragon` : `NX` communication library
CPU for 100 iterations of the predictor/corrector scheme
Increasing problem size and *overlapping* (upper) and *non-overlapping* (lower) mesh partitions

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	8	233.5 s	50	3.0 s	0.1 s
M3	16	261.5 s	89	4.0 s	0.15 s
M4	32	266.5 s	180	4.5 s	0.2 s
M5	64	272.0 s	334	5.0 s	0.25 s

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	8	210.5 s	55	4.5 s	0.1 s
M3	16	221.5 s	105	7.0 s	0.15 s
M4	32	226.5 s	212	8.0 s	0.2 s
M5	64	232.5 s	389	9.0 s	0.25 s

We proceed with scalability issues for the implicit time advancing procedure. Computations are now performed on the `Cray T3D` and `Ibm SP2` systems. Note that, for a given

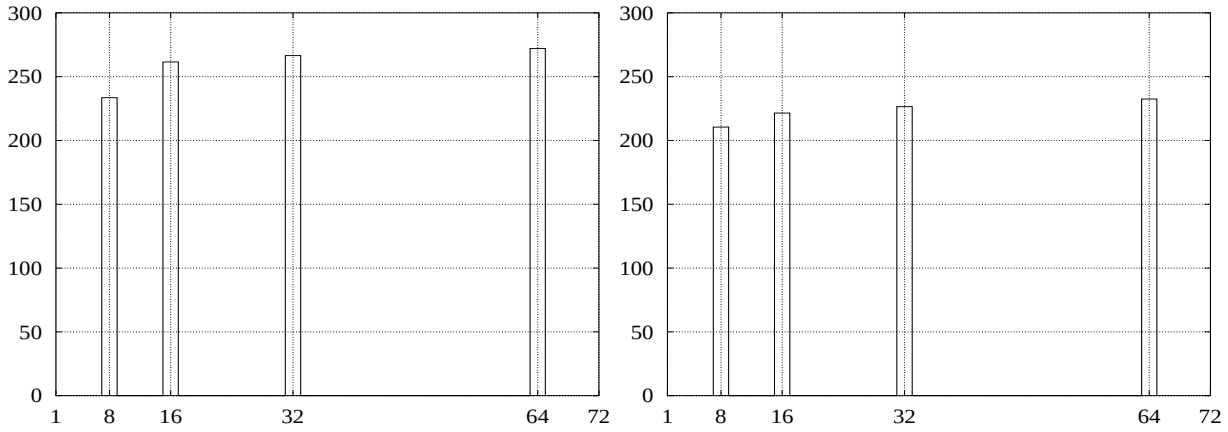

Figure 2: Scalability issues for explicit computations on the `Intel Paragon`
CPU (in s) for 100 iterations versus #processors
Comparsion between *overlapping* (left) and *non-overlapping* (right) mesh partitions

problem size, we have used half the number of processors on the `Ibm SP2` than on the `Cray T3D`; the results are however in favor of the former system. At each time step, we use 18 Jacobi relaxations for the approximate solution of the linear system resulting from (24). Timing measures are given in Tab. 3 to 4 for 10 non-linear iterations. A graphic interpretation of the evolution of the total CPU time versus the number of processors is given on Fig. 3 for the `Cray T3D` and 4 for the `Ibm SP2`.

Table 3: Implicit Euler computations on the `Cray T3D` : `PVM` communication library
CPU for 10 iterations of the implicit linearised scheme
Increasing problem size and *overlapping* (upper) and *non-overlapping* (lower) mesh partitions

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	8	44.0 s	101	1.1 s	0.05 s
M3	16	48.5 s	185	1.6 s	0.1 s
M4	32	50.0 s	370	1.9 s	0.2 s
M5	64	52.5 s	655	2.5 s	0.45 s

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	8	38.0 s	117	1.9 s	0.05 s
M3	16	40.5 s	222	2.8 s	0.1 s
M4	32	42.0 s	440	2.9 s	0.2 s
M5	64	43.5 s	791	5.0 s	0.5 s

The parallel solution algorithm based on non-overlapping mesh partitions demonstrate very good scalability properties on both parallel systems. In that case, one can notice that the communication time represents 7.5% of the total CPU time on the `Ibm SP2` using 32 processors for mesh *M5*; on the `Cray T3D` the timing measures give 7% on 32 processors with mesh *M4* and 11.5% on 64 processors with mesh *M5*. In addition, the Mflop rates for one processor, deduced from the figures obtained with $N_p = 4$ and non-overlapping mesh partitions, are 15 Mflop/s on the `Cray T3D` and 55 Mflop/s on the `Ibm SP2` i.e. respectively 10% and 20% of the theoretical peak rates.

Table 4: Implicit Euler computations on the `Ibm SP2` : `MPL` communication library
CPU for 10 iterations of the implicit linearised scheme
Increasing problem size and *overlapping* (upper) and *non-overlapping* (lower) mesh partitions

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	4	22.2 s	200	0.7 s	0.08 s
M3	8	24.2 s	372	1.0 s	0.1 s
M4	16	26.3 s	703	1.1 s	0.15 s
M5	32	26.6 s	1294	1.3 s	0.18 s

MESH	N_p	CPU	Mflop/s	Loc Comm	Glb Comm
M2	4	20.3 s	220	0.9 s	0.05 s
M3	8	20.8 s	432	1.0 s	0.07 s
M4	16	21.0 s	880	1.3 s	0.1 s
M5	32	21.2 s	1623	1.6 s	0.15 s

4.3.2 Steady State Computations

We consider here steady state computations for the test case selected above. A partial view of mesh *M1* is shown on Fig. 5; Fig. 6 and 7 visualise the steady Mach lines for meshes *M3* and *M5*; the latter figure clearly shows the λ-shock pattern on the wing surface.

Calculation of the parallel speed-up

We can write the total run time T_1^s of a serial program running on a single processor node as composed of two parts :

$$T_1^s = T_{serial}^s + T_{parallel}^s$$

where T_{serial}^s and $T_{parallel}^s$ respectively denote the serial execution time of the non-parallelisable and the parallelisable part of the application. If we assume an ideal situation where the parallel part can be parallelised by distributing equal shares among the processor nodes,

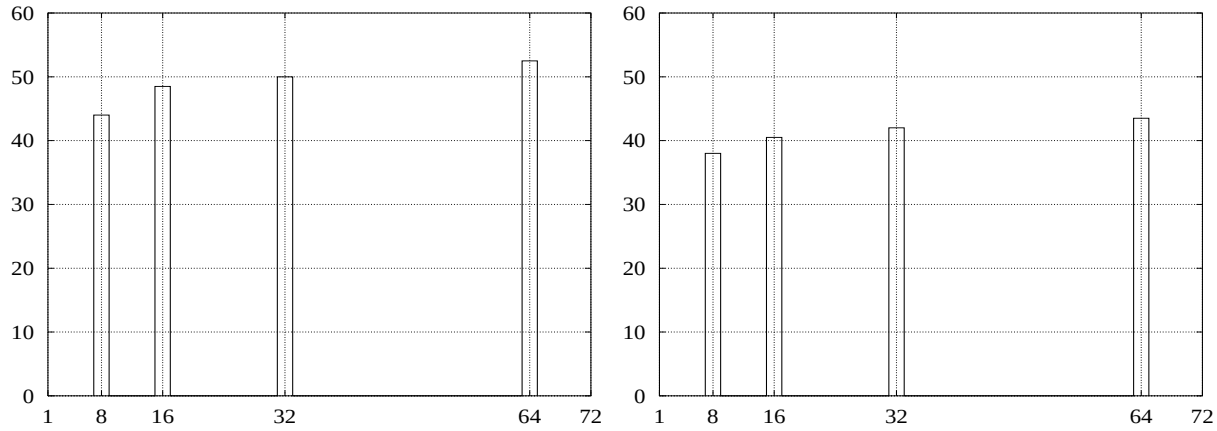

Figure 3: Scalability issues for implicit computations on the `Cray T3D`
CPU (in s) for 10 iterations versus #processors
Comparsion between *overlapping* (left) and *non-overlapping* (right) mesh partitions

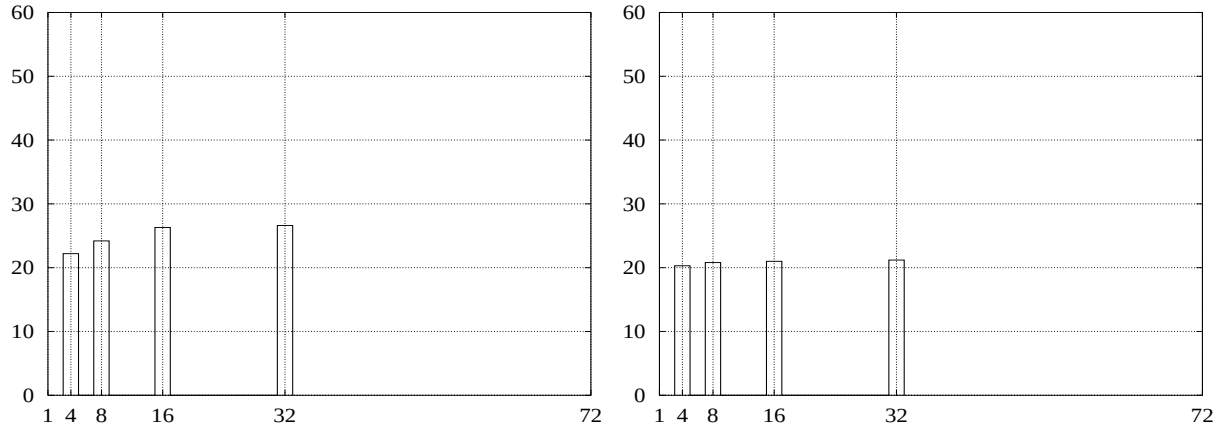

Figure 4: Scalability issues for implicit computations on the `Ibm SP2`
CPU (in s) for 10 iterations versus #processors
Comparsion between *overlapping* (left) and *non-overlapping* (right) mesh partitions

we can expect the total execution time $T^p_{N_p}$ for the parallel program running on N_p processors to be :

$$T^p_{N_p} = T^p_{serial} + T^p_{parallel} = T^s_{serial} + \frac{T^s_{parallel}}{N_p}$$

The parallel speed-up $S(N_p)$ can be expressed as :

$$S(N_p) = \frac{T^s_1}{T^p_{N_p}}$$

The execution time of the parallel part $T^p_{parallel}$ can be split into two terms :

$$T^p_{parallel} = T^p_{comp} + T^p_{comm} = \frac{T^s_{parallel}}{N_p} + T^p_{comm}$$

where T^p_{comp} and T^p_{comm} respectively denote the computation time and the communication time of the parallel part of the application. We obtain the following expression for the parallel speed-up :

$$S(N_p) = \frac{T^s_1}{T^s_{serial} + \frac{T^s_{parallel}}{N_p} + T^p_{comm}} = \frac{T^s_1}{(1-p)T^s_1 + \frac{pT^s_1}{N_p} + T^p_{comm}}$$

However, for large size problems, the total run time T^s_1 of the serial program running on a single processor node is generally not available because of memory limits. It is therefore necessary to devise another expression of the parallel speed-up which doesn't make use of T^s_1. While doing so, we assume that the application under consideration is such that $p = 1$ so that $T^s_{serial} = 0$ (which is a realistic assumption for the SPMD programming model we are using here) and $T^s_{parallel} = T^s_1$. We will write the total execution time T^s_1 as :

$$T^s_1 = N_p \times T^p_{comp}$$

and the parallel speed-up takes the following form :

$$S(N_p) = \frac{T^s_1}{T^p_{N_p}} = \frac{N_p \times T^p_{comp}}{T^p_{comp} + T^p_{comm}} = N_p \times \left(\frac{1}{1 + \frac{T^p_{comm}}{T^p_{comp}}}\right)$$

This expression will be used in the sequel in order to evaluate the speed-up of the parallel algorithms based on overlapping and non-overlapping mesh partitions.

Computations with mesh *M3*

We first present results of computations that have been performed on an Ethernet based local network of `Dec 3000/600` workstations used in a non-dedicated mode. The implicit time advancing procedure is used with 36 Jacobi relaxations for the approximate solution

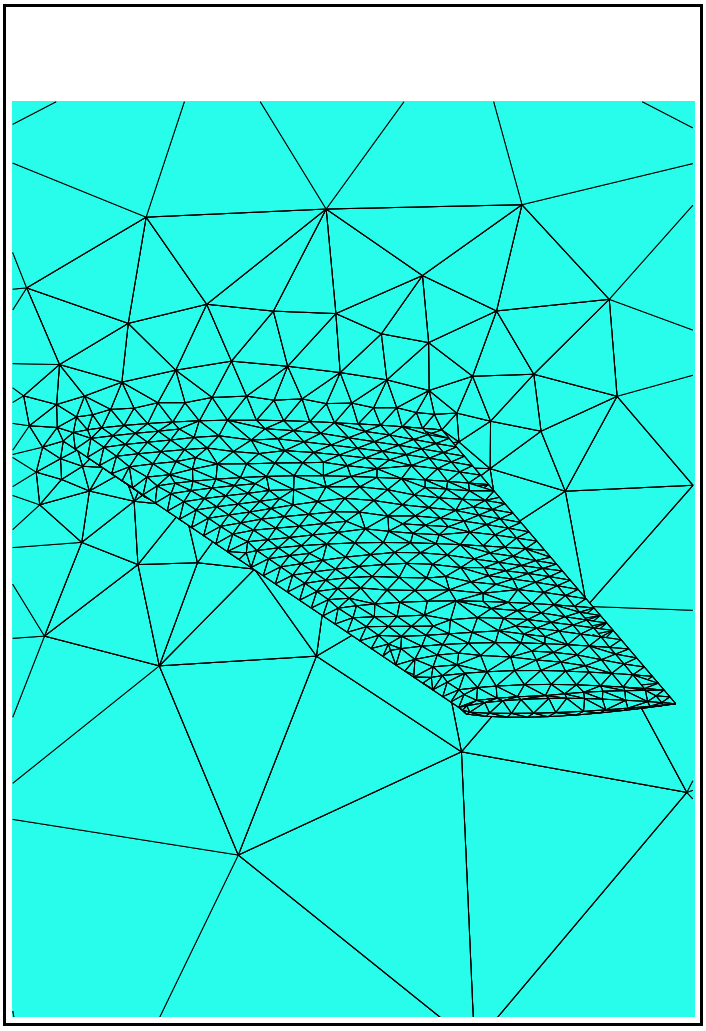

Figure 5: Mesh on the skin of on an `ONERA M6` wing (mesh *M1*)

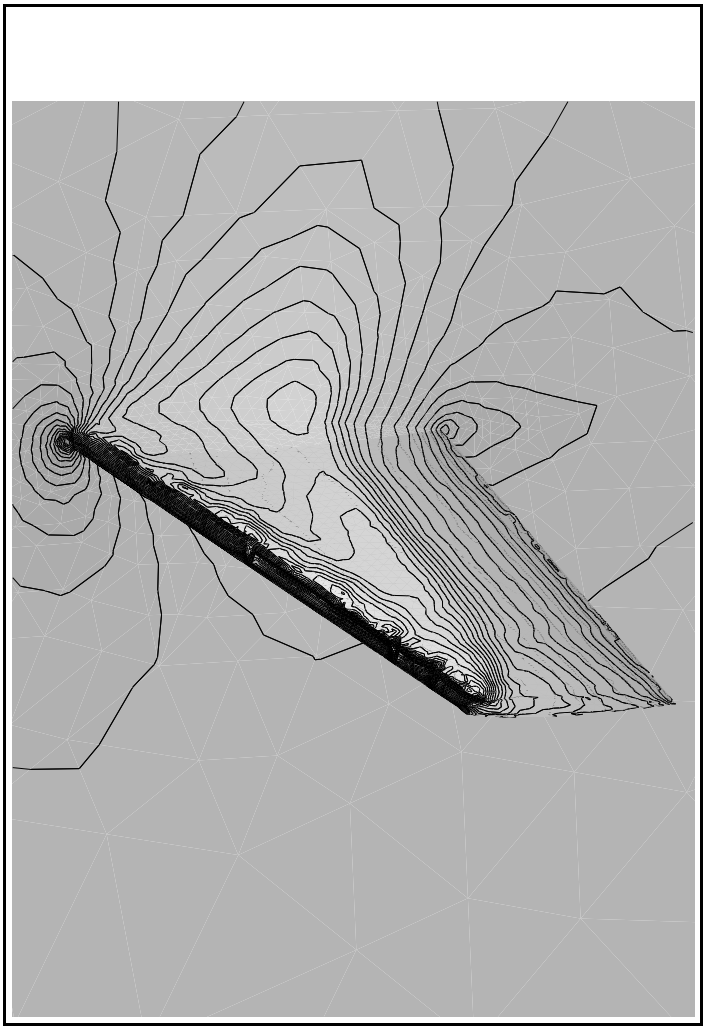

Figure 6: Steady Mach lines on an `ONERA M6` wing (mesh *M3*)

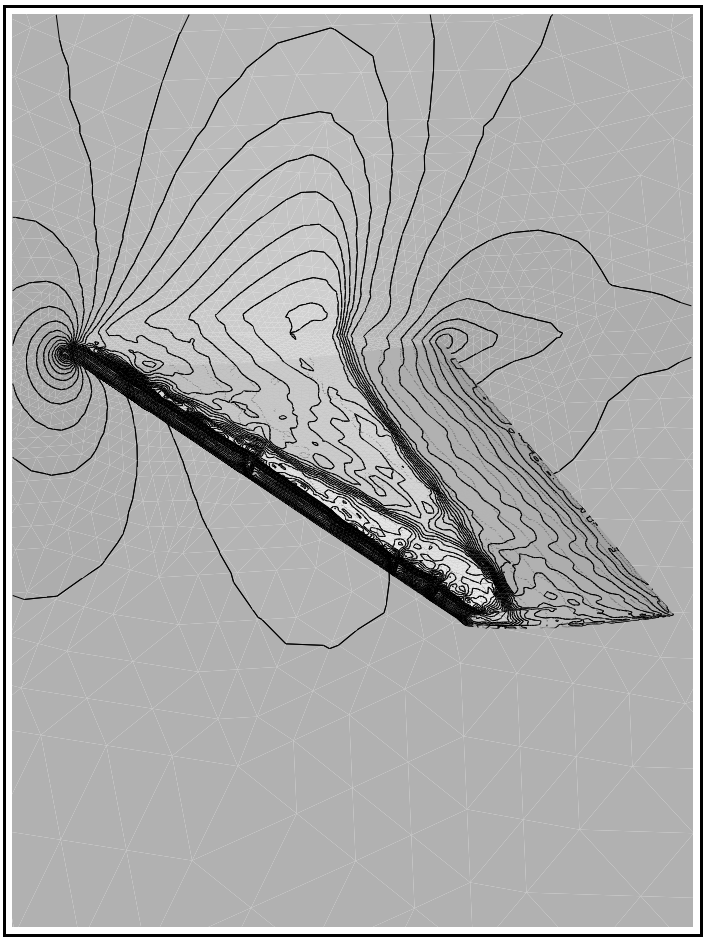

Figure 7: Steady Mach lines on an `ONERA M6` wing (mesh *M5*)

of the linear system resulting from (24). The pseudo-time step is computed according to the law CFL=4 $\times$ *it* where *it* denotes the current non-linear iteration. The steady state solution (initial normalized non-linear residual divided by 10^6) has been obtained after 89 pseudo-time steps. Fig. 8 below depicts the non-linear and linear convergence curves. Tab. 5 compares the total CPU times for parallel solution algorithms based on overlapping ("Size = 1") and non-overlapping ("Size = 0") mesh partitions. We also report in this table the minimum and maximum values of the local and global communication times. We obtain the expected behavior with this kind of computing platform. Even though the communication times represent only a small percentage of the total CPU times, the influence of the exchange phase becomes more important as the number of workstations increases. However, a 20% Mflop/s rate is approximately obtained on one workstation which suggests that with a faster interconnexion network this type of computing platform can represent a serious alternative to purely parallel systems. For comparison purposes, we give in Table 6 the results obtained on 16 processors of the `Intel Paragon`.

Table 5: Implicit Euler computations on the `ONERA M6` wing with mesh *M3*
Computations on a network of `Dec 3000/600` workstations : `PVM` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
2	1	3107.0 s	37	12.0 s	12.8 s	2.6 s	5.0 s	2
	0	2858.0 s	40	11.0 s	12.5 s	2.6 s	4.6 s	2
3	1	2067.0 s	56	11.3 s	19.3 s	3.6 s	6.2 s	2.95
	0	1986.0 s	58	28.0 s	30.0 s	2.8 s	7.2 s	2.95

Table 6: Implicit Euler computations on the `ONERA M6` wing with mesh *M3*
Computations on the `Intel Paragon` : `NX` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
16	1	1783.0 s	65	23.0 s	50.0 s	2.5 s	3.0 s	15.5
	0	1278.0 s	90	16.0 s	31.0 s	2.5 s	3.0 s	15.5

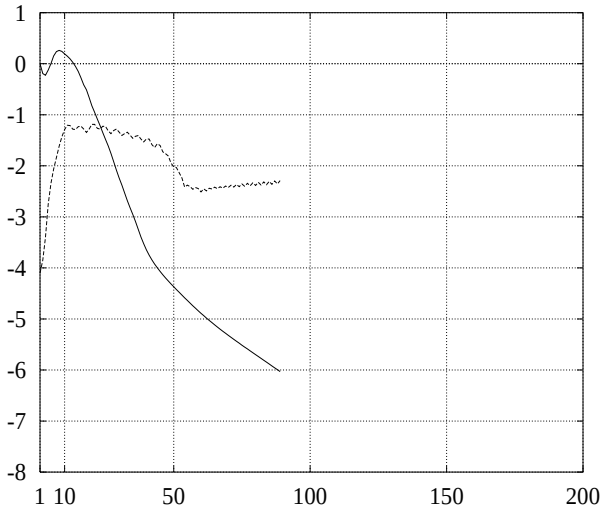

Figure 8: Implicit Euler computations on the `ONERA M6` wing with mesh *M3*
Non-linear (straight line) and linear (dashed line) convergence

Computations with mesh *M5*

For the finest mesh tested, the steady state solution (initial normalized residual divided by 10^6) has been obtained after 196 non-linear iterations (the CFL law is still given by CFL=$4 \times it$ while the number of Jacobi relaxations is equal to 36). Fig. 9 below depicts the non-linear and linear convergence behaviors. Tab. 7 to 9 compare the total CPU times for parallel solution algorithms based on overlapping and non-overlapping mesh partitions.

Table 7: Implicit Euler computations on the `ONERA M6` wing with mesh *M5*
Computations on the `Intel Paragon` : `NX` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
64	1	4348.0 s	223	45.5 s	98.0 s	8.0 s	9.5 s	62.5
	0	2819.0 s	344	57.0 s	126.0 s	8.0 s	10.0 s	61
128	1	2824.0 s	345	25.0 s	66.0 s	9.0 s	11.5 s	124
	0	1572.0 s	619	36.0 s	103.0 s	9.5 s	12.0 s	118

We first remark that the estimated speed-up are always better when using overlapping mesh partitions. This behavior is simply explained by the fact that between computations with overlapping and non-overlapping mesh partitions the pure computational times de-

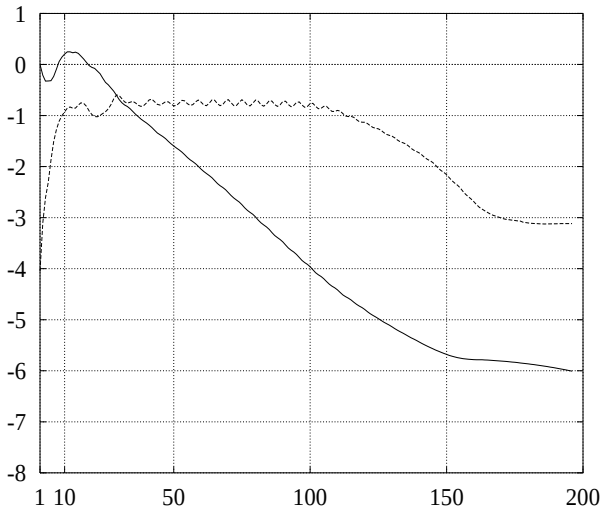

Figure 9: Implicit Euler computations on the `ONERA M6` wing with mesh *M5*
Non-linear (straight line) and linear (dashed line) convergence

Table 8: Implicit Euler computations on the `ONERA M6` wing with mesh *M5*
Computations on the `Cray T3D` : `PVM` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
64	1	1464.0 s	664	30.0 s	85.5 s	14.5 s	17.5 s	60
	0	1138.0 s	855	38.0 s	153.0 s	14.5 s	16.0 s	54
128	0	602.0 s	1615	24.5 s	107.0 s	28.0 s	31.0 s	99

Table 9: Implicit Euler computations on the `ONERA M6` wing with mesh *M5*
Computations on the `Ibm SP2` : `MPL` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
				Min	Max	Min	Max	
32	1	726.0 s	1340	29.0 s	43.0 s	4.0 s	5.0 s	30
	0	570.0 s	1705	33.0 s	50.0 s	3.5 s	4.5 s	29
64	1	444.0 s	2190	19.0 s	35.0 s	5.5 s	6.0 s	58
	0	316.0 s	3077	24.0 s	51.0 s	4.5 s	5.5 s	55

crease while the communication times increase in such a way that the ratio between the two figures is not favorable to an improved speed-up. Indeed, the pure computational times for overlapping mesh partitions include redundant floating-point operations which means that larger global problems are actually solved in this case. This suggests that techniques for overlapping communication steps and purely computational ones should be investigated in order to improve the performances of the parallel algorithm based on non-overlapping mesh partitions.

For the case $N_p = 64$ the total communication times for non-overlapping partitions ("Size = 0") are equal to 130.0 s on the `Intel Paragon`, 169.0 s on the `Cray T3D` and 56.5 s on the `Ibm SP2`. These figures respectively represent 5%, 15% and 18% of the corresponding total CPU times; this explains the higher speed-up figures obtained on the `Intel Paragon`.

Combined computational/numerical scalability assessment

Mesh *M5* is approximately four times larger than mesh *M3*. Here, we make a simple assessment of the combined computational/numerical scalability properties of the parallel algorithms under consideration. In order to do so, we compare the required numbers of non-linear iterations and total CPU times for steady state computations using these two meshes on the `Intel Paragon` system. Tab. 10 summarizes the obtained results. We recall that the CFL law is given by CFL=$4 \times it$ and the number of Jacobi relaxations is fixed to 36. Switching from mesh *M3* to mesh *M5*, the number of non-linear iterations is approximately doubled (the effective ratio is 2.2); it is seen that the corresponding CPU time ratio for overlapping mesh partitions is 2.4 while for non-overlapping mesh partitions it is equal to 2.2. Again the best scalability properties are obtained with the parallel algorithm based on non-overlapping mesh partitions.

Table 10: Implicit Euler computations on the `ONERA M6` wing
Computations on the `Intel Paragon` : `NX` communication library
Combined computational/numerical scalability assessment

Size	MESH	N_p	# it	CPU	CPU/it
1	*M3*	16	89	1783.0 s	20.0 s
	M5	64	196	1278.0 s	22.2 s
0	*M3*	16	89	4348.0 s	14.3 s
	M5	64	196	2819.0 s	14.4 s

4.4 External Flow Around a `FALCON` Jet

The viscous flow ($Re = 1000$) around a complete configuration of a commercial `FALCON` aircraft is computed for a Mach number of 0.85, an angle of attack equal to 1° and a yaw angle

equal to 1°. Two meshes have been generated; their characteristics are summarized in Tab. 11 below. Fig. 11 visualises the triangulation on the skin of mesh *F1*. The implicit time advancing procedure (24) is used to search for the steady state solutions of this external flow. At each non-linear iteration, the pseudo-time step is computed according to the law CFL=MAX(it, 50) where it denotes the current non-linear iteration while the residual tolerance for the linear system solution was fixed to 10^{-5}. A maximum of 72 Jacobi relaxations has been used.

Table 11: Two meshes and their characteristics for a `FALCON` jet

MESH	N_V	N_T
F1	30514	163732
F2	231036	1309856

Computations with mesh *F1*

For mesh *F1*, the convergence (initial normalized residual divided by 10^6) has required 126 non-linear iterations. Fig. 10 below depicts the non-linear and linear convergence behaviors. The steady pressure lines on the skin of mesh *F1* are shown on Fig. 12.

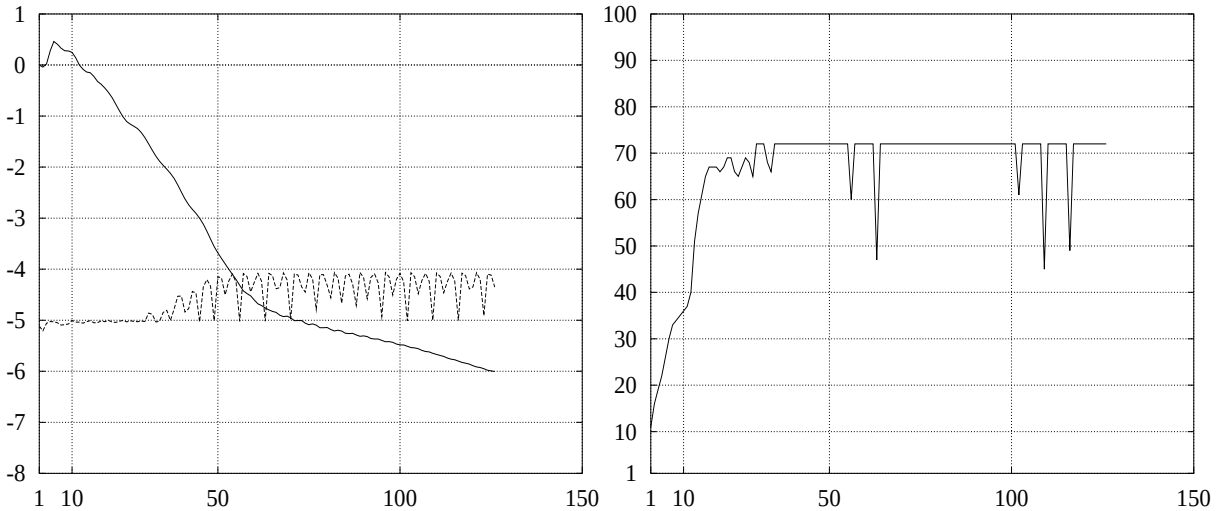

Figure 10: Implicit Navier-Stokes computations on the `FALCON` aircraft with mesh *F1*
Left figure : non-linear (straight line) and linear (dashed line) convergence
Right figure : effective number of Jacobi relaxations

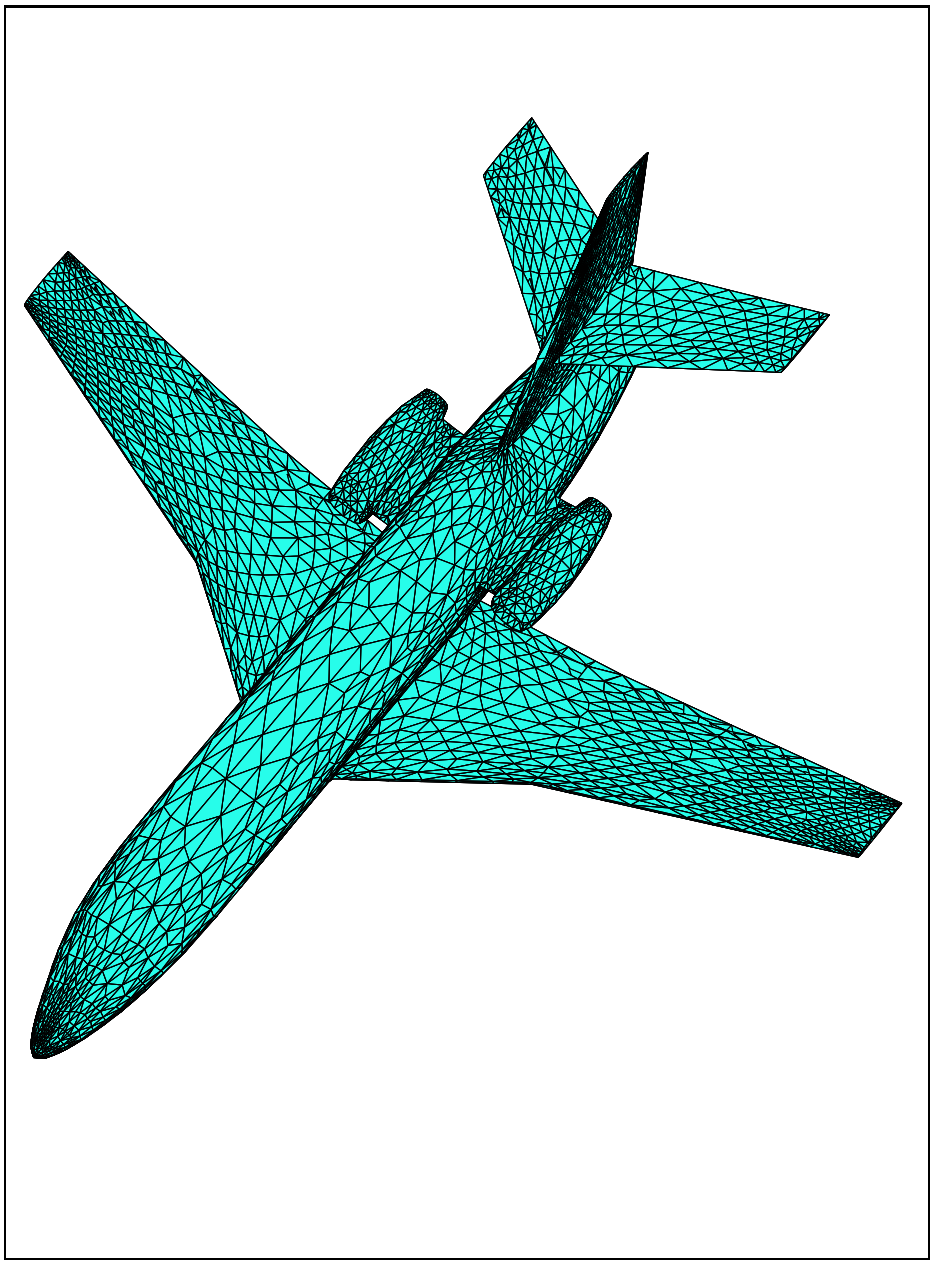

Figure 11: Mesh on the skin of the `FALCON` jet (mesh *F1*)

Here, we are also interested in assessing the influence of the partitioning algorithm on the performances of the parallel solution algorithm. In order to do so we concentrate ourselves on the case of non-overlapping mesh partitions obtained with the `TOP/DOMDEC` software tool [9]. Mesh partitions have been obtained using the Greedy (GRD) algorithm (see Farhat and Lesoinne[7]) and the recursive spectral bisection (RSB) algorithm (see Simon[21]). Tab. 12 below reports the measured partitioning times for mesh *F1*. For the RSB algorithm the size of the computational space has been set to 15000. Fig. 13 to 16 visualise both the

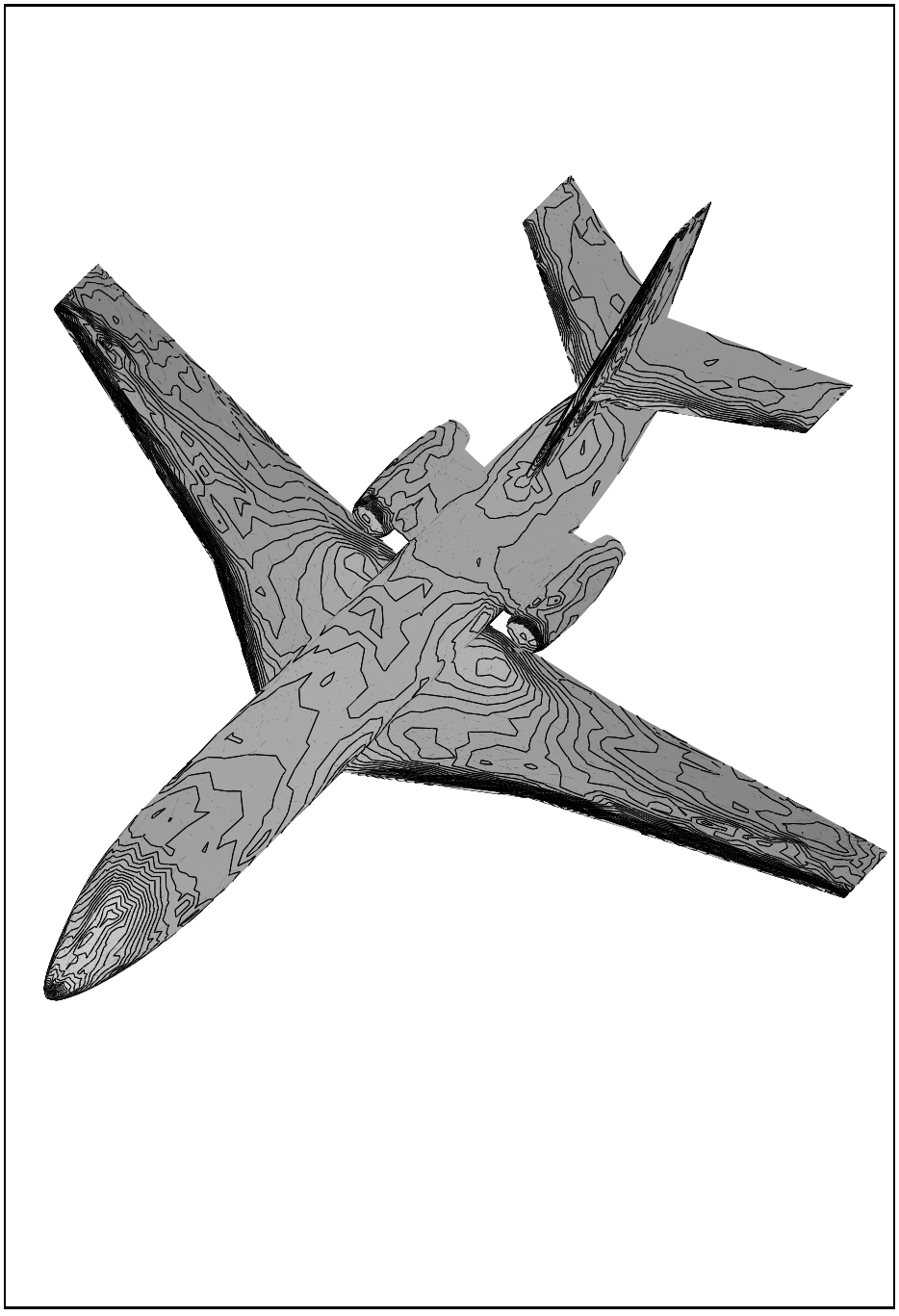

Figure 12: Steady pressure lines on the skin of the FALCON jet (mesh *F1*)

obtained repartition of the computational load (in terms of the number of vertices, elements and edges) and the corresponding repartition of the communication load (in terms of the number of neighboring submeshes and the maximal size of messages). The RSB algorithm is clearly much more expensive than the GRD one, but, as suggested by the cost of the optimisation phase, the initial partition is of better quality with regard to the repartition of the computational load and the total interface size. We note that a recent work by Barnard and Simon[1] has lead to a more efficient implementation of the RSB algorithm.

Table 12: Partitioning times for the `FALCON` aircraft using mesh *F1*

Algo	Decomp Time	Optim Time
GRD	10.9 s	189.2 s
RSB	512.3 s	112.2 s

In `TOP/DOMDEC`, the user is asked to define the type of computational load he wants to privilege in the optimisation phase. In the present case, the computational load part of the objective function was only taking care of the element-wise operations. It is seen that both partitioning strategies have been able to produce an optimal element-wise load balance. However, the edge-wise load balance is clearly not following the element-wise one due to the fully unstructured nature of the underlying meshes.

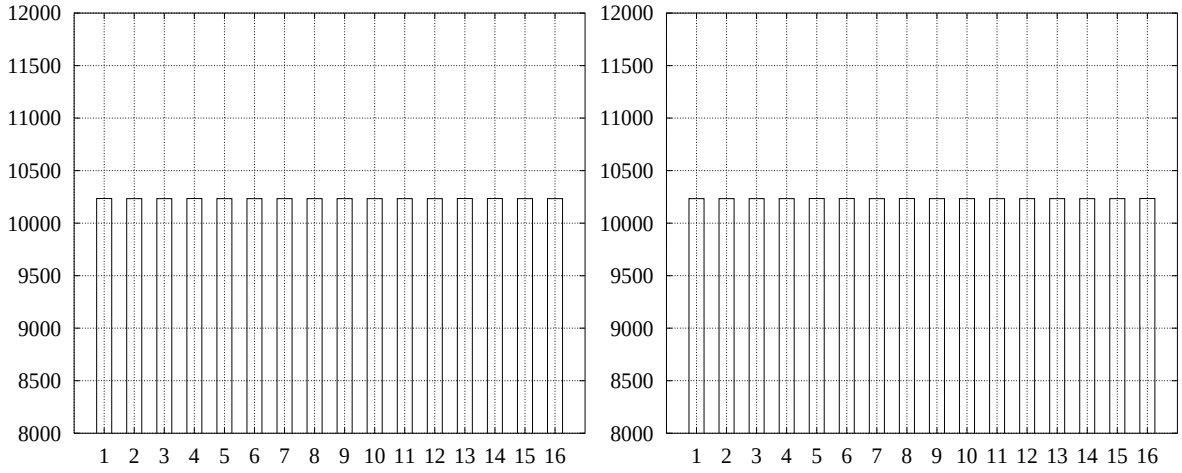

Figure 13: Mesh partitioning issues using non-overlapping partitions
Element-wise load balance
Left : Greedy algorithm (Min = 10233/Max = 10234)
Right : Recursive Spectral Bisection algorithm (Min = 10233/Max = 10234)
16 submeshes non-overlapping partition of a `FALCON` aircraft (mesh *F1*)

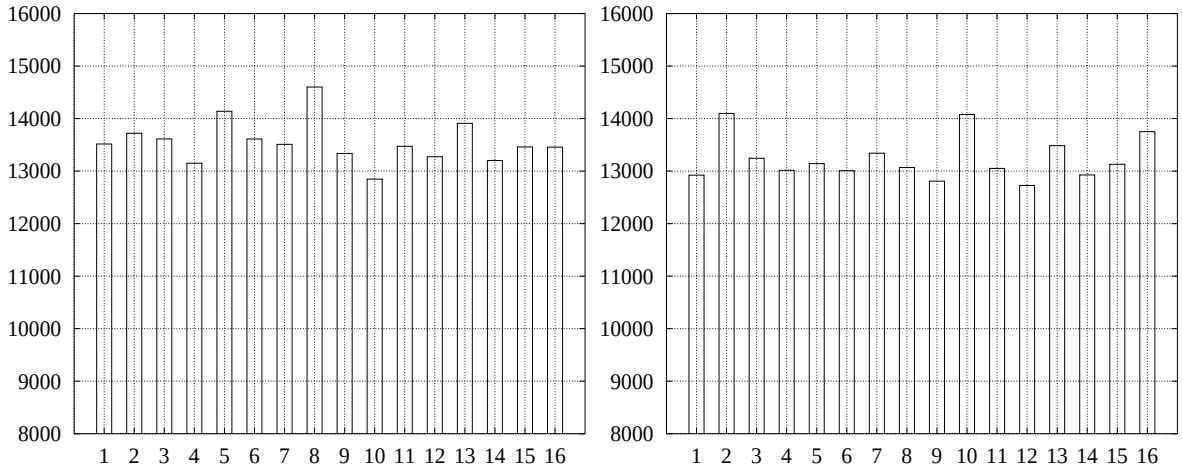

Figure 14: Mesh partitioning issues using non-overlapping partitions
Edge-wise load balance
Left : Greedy algorithm (Min = 12848/Max = 14601)
Right : Recursive Spectral Bisection algorithm (Min = 12727/Max = 14098)
16 submeshes non-overlapping partition of a `FALCON` aircraft (mesh *F1*)

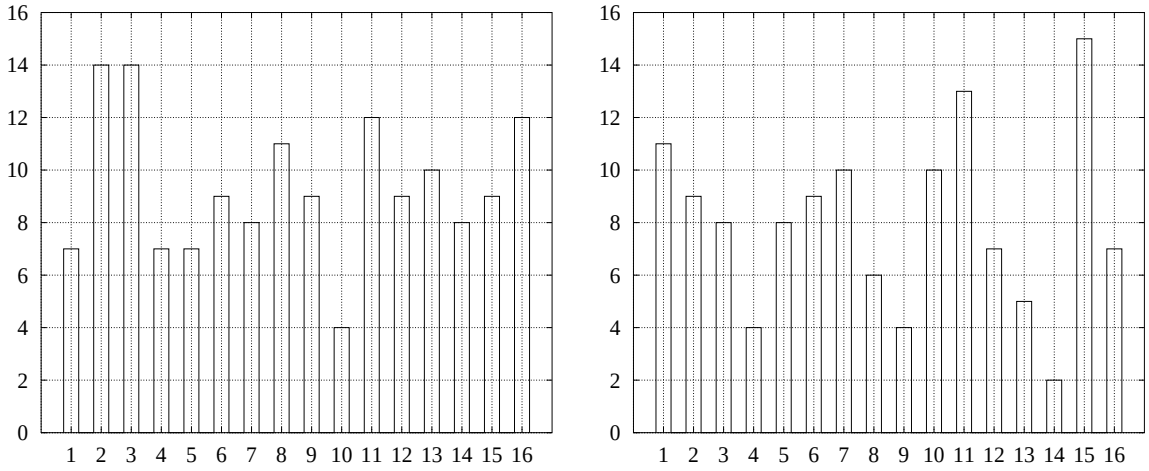

Figure 15: Mesh partitioning issues using non-overlapping partitions
Number of neighboring submeshes for each submesh
Left : Greedy algorithm (Min = 4/Max = 14)
Right : Recursive Spectral Bisection algorithm (Min = 4/Max = 15)
16 submeshes non-overlapping partition of a `FALCON` aircraft (mesh *F1*)

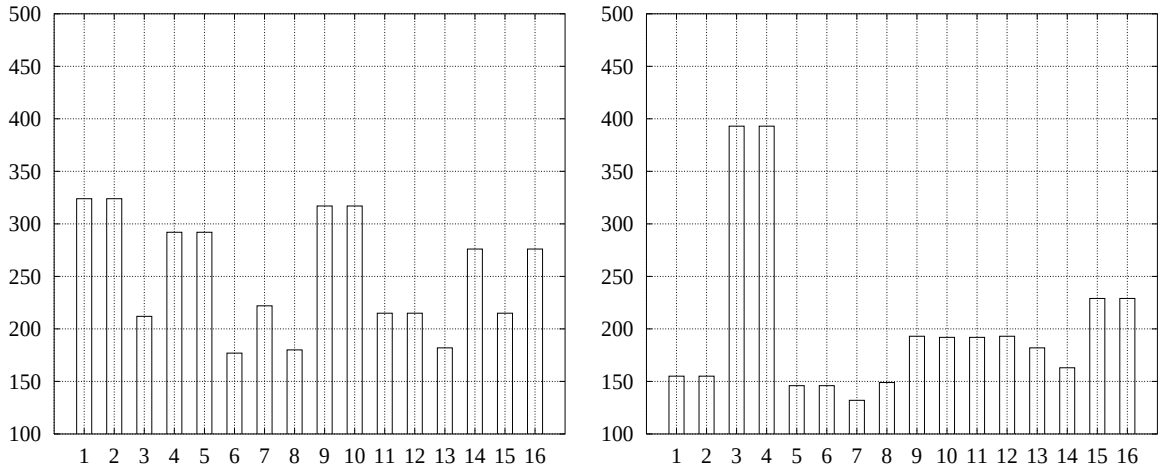

Figure 16: Mesh partitioning issues using non-overlapping partitions
Maximum message length for each submesh
Left : Greedy algorithm (Min = 177/Max = 324)
Right : Recursive Spectral Bisection algorithm (Min = 155/Max = 393)
16 submeshes non-overlapping partition of a `FALCON` aircraft (mesh *F1*)

The RSB algorithm seems to produce a partition with improved communication load. This is actually confirmed by the experiment as it can be seen on Tab. 13 below. The overlapping mesh partition has been obtained using the recursive inertia bisection (RIB) algorithm implemented in `MS3D`[16] and a heuristic iterative optimiser is used in order to obtain a good repartition of the computational load. Five steps of the simulated annealing algorithm have been used in order to improve the initial decompositions.

Table 13: Implicit Navier-Stokes computations on the `FALCON` aircraft with mesh *F1*
Computations on the `Ibm SP2` : `MPL` communication library
Total CPU times for steady state computations

N_p	Size	Algo	CPU	Mflop/s	Loc Comm		Glb Comm		$S(N_p)$
					Min	Max	Min	Max	
16	1	RIB	440.0 s	734	17.0 s	25.0 s	3.5 s	4.0 s	14.9
	0	GRD	368.0 s	877	23.0 s	36.0 s	3.5 s	4.5 s	14.2
	0	RSB	350.0 s	922	18.0 s	24.0 s	3.0 s	4.0 s	14.7

Computations with mesh *F2*

For mesh *F2*, the convergence (initial normalized residual divided by 10^6) has required 172 non-linear iterations. Fig. 18 below depicts the non-linear and linear convergence curves. The steady pressure lines on the skin of mesh *F1* are shown on Fig. 17. Performance results are summarized in Tab. 14 and 15. On 128 processors of the `Ibm SP2` the steady flow field is computed in about 15 mn with a communication cost that represents 10% of the total CPU time. This figure reduces to less than 3% on 128 processors of the `Intel Paragon`.

Table 14: Implicit Navier-Stokes computations on the `FALCON` aircraft with mesh *F2*
Computations on the `Intel Paragon` : `NX` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Loc Comm		Glb Comm		$S(N_p)$
			Min	Max	Min	Max	
128	1	8246.0 s	75.0 s	238.0 s	14.5 s	18.5 s	124

Table 15: Implicit Navier-Stokes computations on the `FALCON` aircraft with mesh *F2*
Computations on the `Ibm SP2` : `MPL` communication library
Total CPU times for steady state computations

N_p	Size	CPU	Loc Comm		Glb Comm		$S(N_p)$
			Min	Max	Min	Max	
64	1	1437.0 s	64.0 s	102.0 s	11.0 s	14.0 s	59
128	1	843.0 s	34.0 s	69.0 s	13.0 s	18.0 s	115

4.5 Internal Flow Inside an Engine Diffusor

The problem under consideration here is the simulation of the flow inside a model diffusor of a jet engine. This part of the engine is surrounding the combustion chamber. The gas flow enters the diffusor from one injection input and, after an expansion, goes into the chamber through several ejection outputs. In practice the flow inside this geometry is subsonic and turbulent. In the present case we consider a laminar flow for which the Mach number is set to 0.3 and the Reynolds number to 300. Due to the viscous character of this flow, the limiting procedure (12) is not used.

Two meshes have been generated; their characteristics are summarized in Tab. 16 below. Fig. 19 visualises mesh *S1*. The implicit time advancing procedure (24) is used to search for

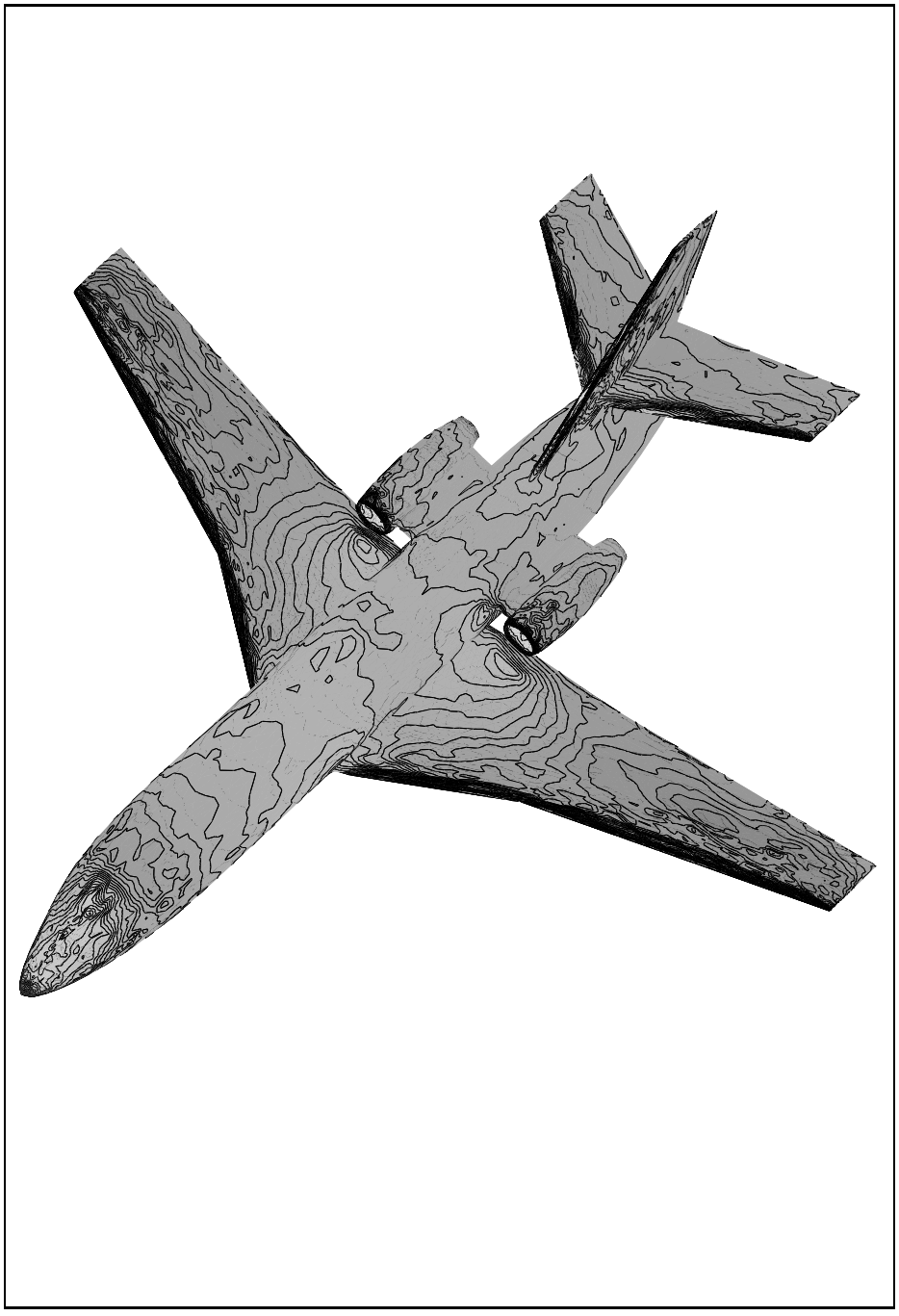

Figure 17: Steady pressure lines on the skin of the FALCON jet (mesh *F2*)

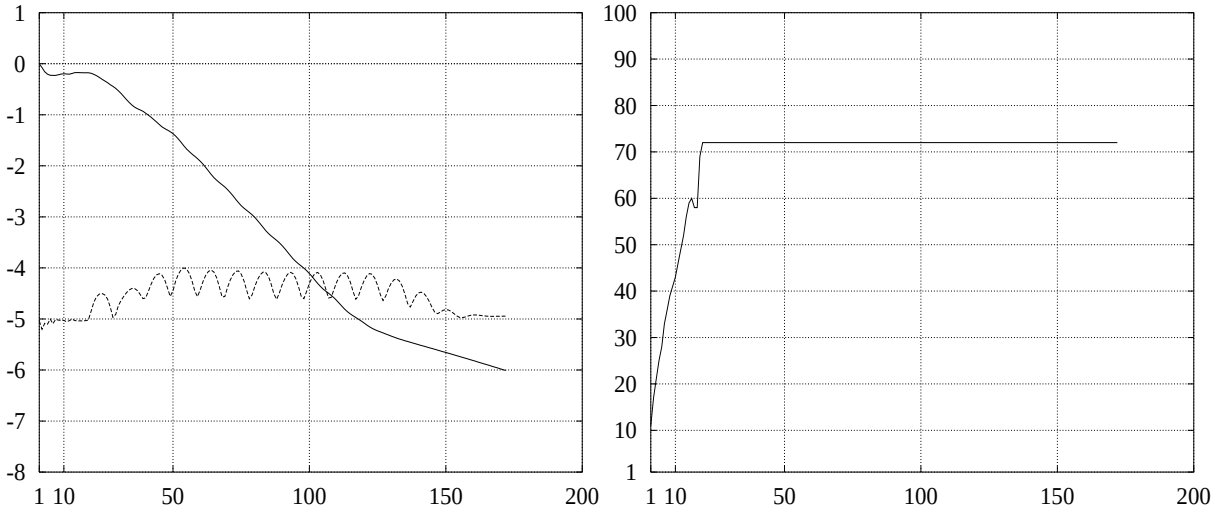

Figure 18: Implicit Navier-Stokes computations on the FALCON aircraft with mesh *F2*
Left figure : non-linear (straight line) and linear (dashed line) convergence
Right figure : effective number of Jacobi relaxations

the steady state solutions of this internal flow. At each non-linear iteration, the pseudo-time step is computed according to the law CFL=MAX(it, 100) where it denotes the current non-linear iteration while the residual tolerance for the linear system solution was fixed to 10^{-2}. A maximum of 36 Jacobi relaxations has been used.

Table 16: Two meshes and their characteristics for a model engine diffusor

MESH	N_V	N_T
S1	22515	114048
S2	165797	912384

Computations with mesh *S1*

Here, we are interesting in assessing the influence of the parameter β used in the computation of the nodal gradient (13), on the accuracy of the computed solution. We therefore consider two computations for which $\beta = 1/2$ and $\beta = 1/6$. For these two values of β, the convergence (initial normalized residual divided by 10^8) has respectively required 140 and 265 non-linear iterations. Fig. 23 below depicts the non-linear and linear convergence behaviors. Fig. 20 and 22 visualise the steady Mach lines in a selected cut-plane. The physical viscous effects are clearly dominating the computed flows; as a consequence, the influence of the upwinding parameter is somewhat weakened in this case even though some slight improvements are seen on the solution computed using the value $\beta = 1/6$ especially at the inflow

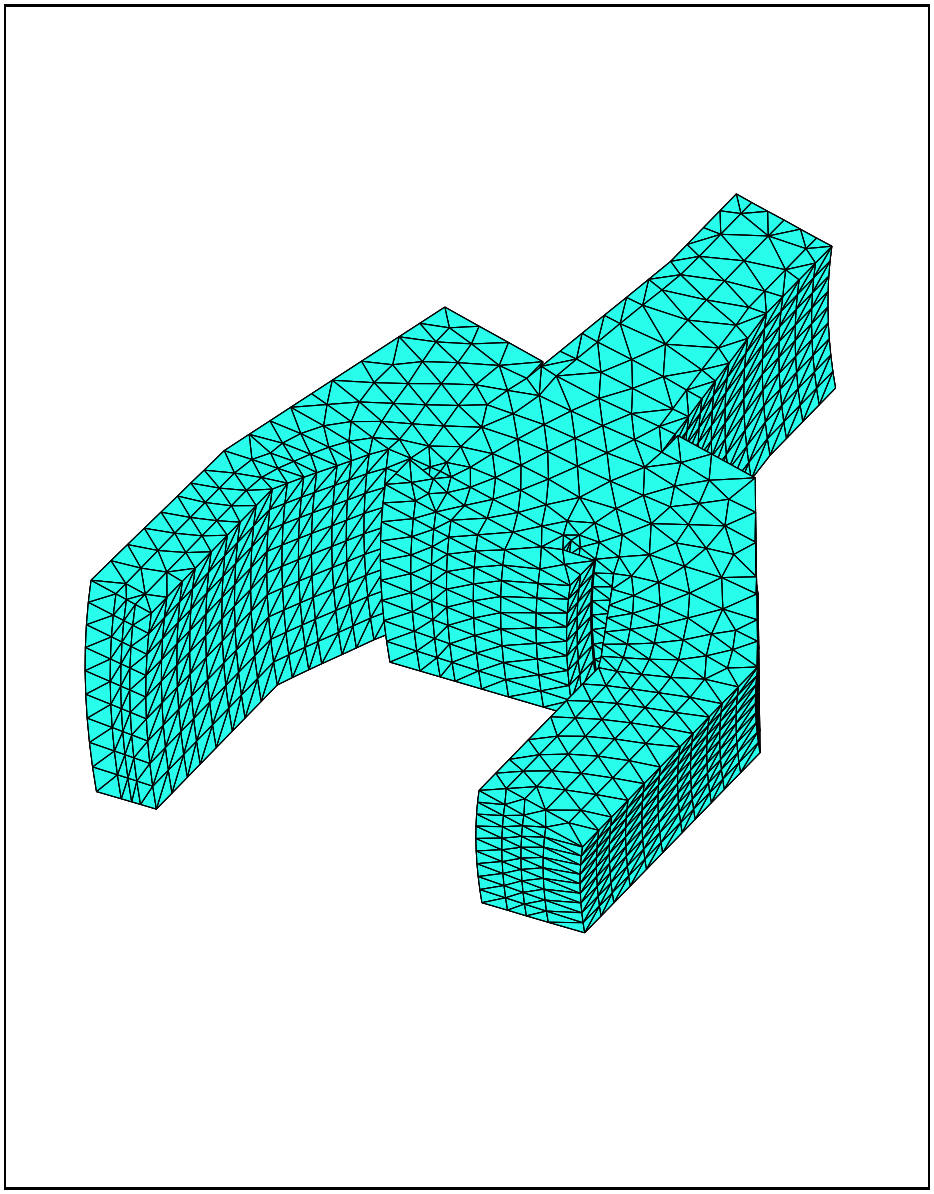

Figure 19: Mesh of the model engine diffusor (mesh *S1*)

boundary and in the neighbourhood of the vortices. Performance results are summarized in Tab. 17 and 18.

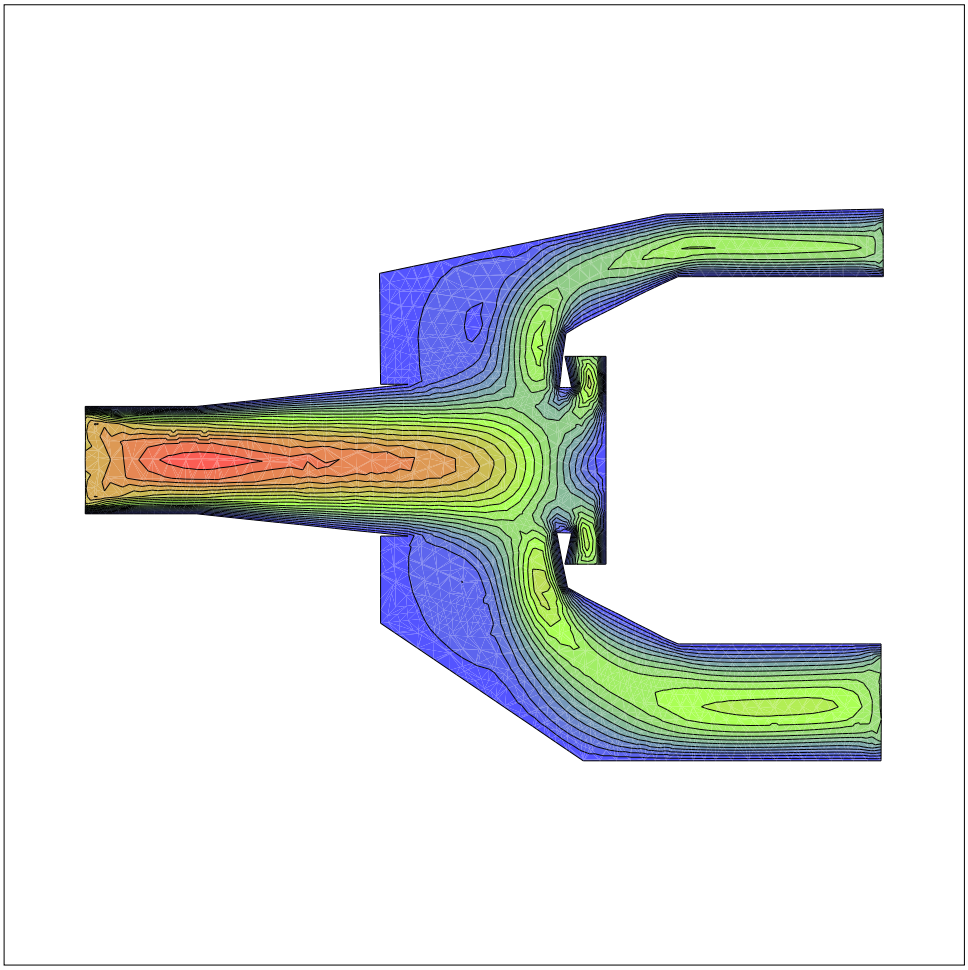

Figure 20: Viscous flow inside a model engine diffusor
Steady Mach lines ($\beta = 1/2$, mesh *S1*)

Table 17: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S1*
Computations on the `Intel Paragon` : `NX` communication library
Total CPU times for steady state computations

N_p	Size	β	CPU	Loc Comm		Glb Comm	
			Min	Max	Min	Max	
32	1	1/2	1374.0 s	7.0 s	24.0 s	4.5 s	5.5 s
		1/6	2603.0 s	13.5 s	44.5 s	8.5 s	10.5 s
32	0	1/2	939.0 s	13.0 s	30.0 s	5.0 s	6.0 s
		1/6	1777.0 s	25.0 s	56.5 s	9.0 s	11.0 s

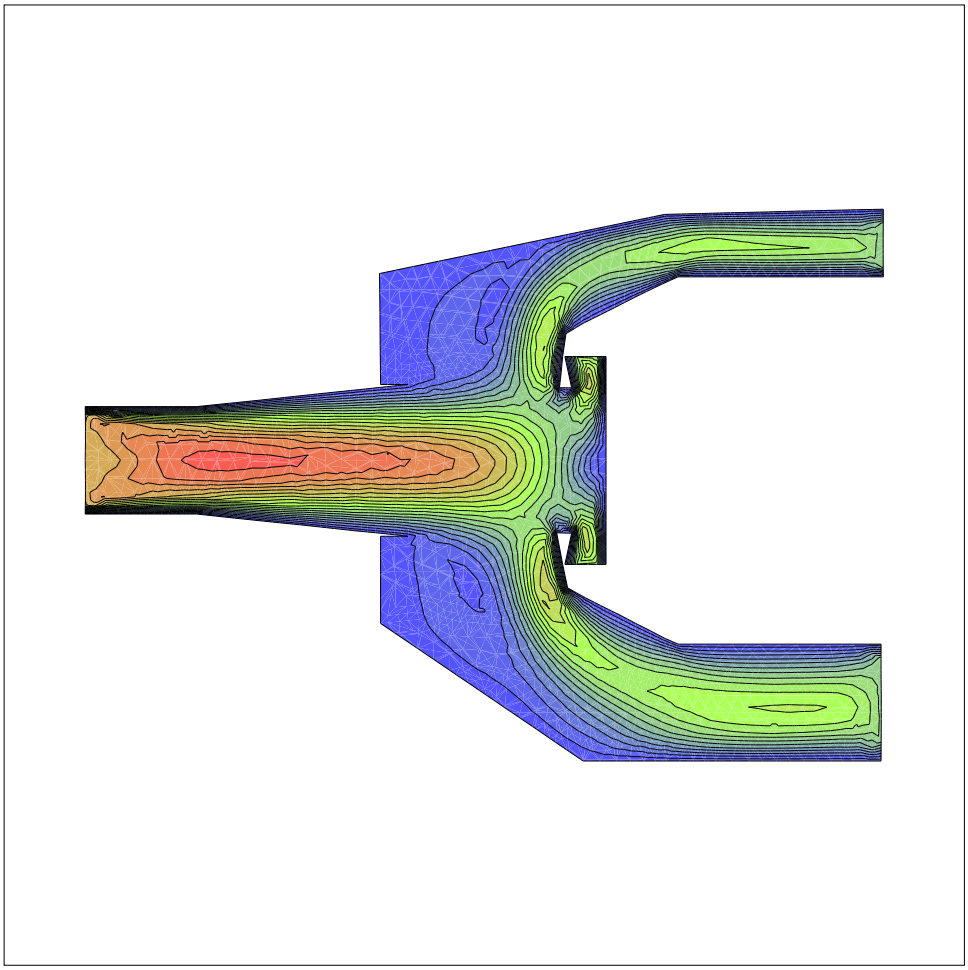

Figure 21:

Figure 22: Viscous flow inside a model engine diffusor
Steady Mach lines ($\beta = 1/6$, mesh *S1*)

Table 18: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S1*
Computations on the `Ibm SP2` : `MPL` communication library
Total CPU times for steady state computations

N_p	Size	β	CPU	Loc Comm		Glb Comm	
			Min	Max	Min	Max	
8	1	1/2	438.0 s	5.5 s	13.5 s	1.5 s	2.0 s
		1/6	836.0 s	12.0 s	26.0 s	3.5 s	4.0 s
8	0	1/2	363.0 s	8.0 s	16.0 s	1.5 s	2.0 s
		1/6	709.0 s	16.0 s	32.0 s	3.5 s	4.0 s

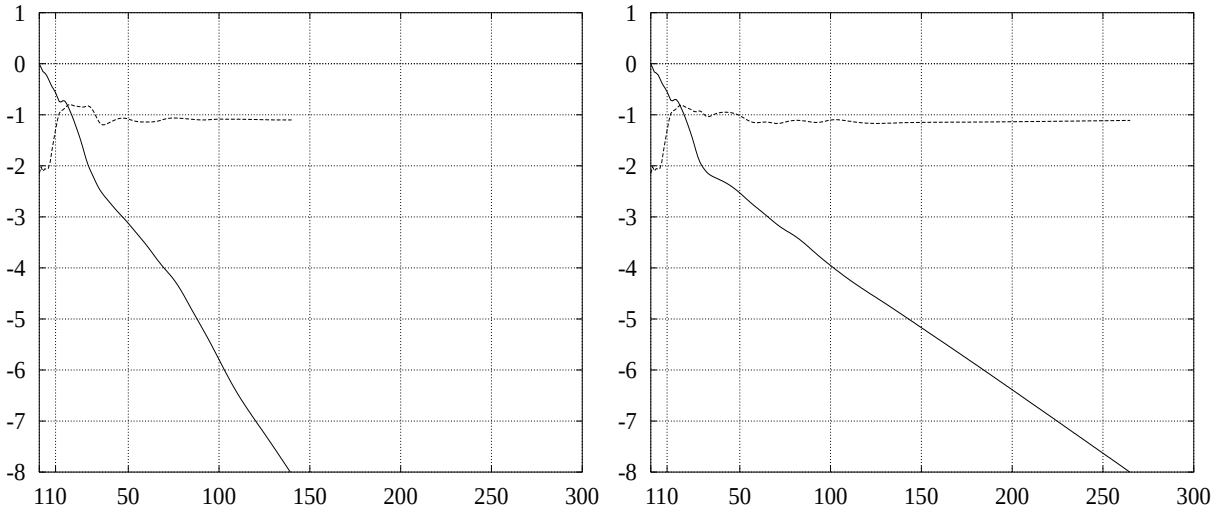

Figure 23: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S1*
Non-linear (straight line) and linear (dashed line) convergence
Left figure $\beta = \frac{1}{2}$ - right figure $\beta = \frac{1}{6}$

Computations with mesh *S2*

For the finest mesh, the value of the parameter β is set to 1/2. At each non-linear iteration, the pseudo-time step is still computed according to the law CFL=MAX(it, 100) where it denotes the current non-linear iteration. However, the residual tolerance for the linear system solution is now fixed to 10^{-1}. We consider two possibilities for the maximum number of Jacobi relaxations, denoted in the sequel as $nbrel_{max}$. First we set $nbrel_{max} = 36$ and the convergence (initial normalized residual divided by 10^6) is obtained after 290 non-linear iterations. Second we increase $nbrel_{max}$ to 72 and the convergence is obtained after 254 non-linear iterations. The linear convergence behavior together with the effective number of Jacobi relaxations are visualised on Fig. 24; the corresponding non-linear convergence curves are given on Fig. 25.

Computations have been performed on 128 processors of the `Intel Paragon`. Tab. 19 details the performance results obtained with overlapping and non-overlapping mesh partitions including the minimum and maximum times for the approximate solution of the linear system (24) ; these times are given under the column "Matrix Inver". With non-overlapping mesh partitions, the linear system solution times are respectively representing 52% ($nbrel_{max} = 36$) and 62% ($nbrel_{max} = 72$) of the total CPU times. This result is not so surprising given the relatively low numerical efficiency of the Jacobi method. The use of a more sophisticated linear solver should clearly improve the above situation. This can be simply illustrated by comparing the performances of the Jacobi method used here with those obtained with a more efficient relaxation method such as the Gauss-Seidel method, in the context of a sequential

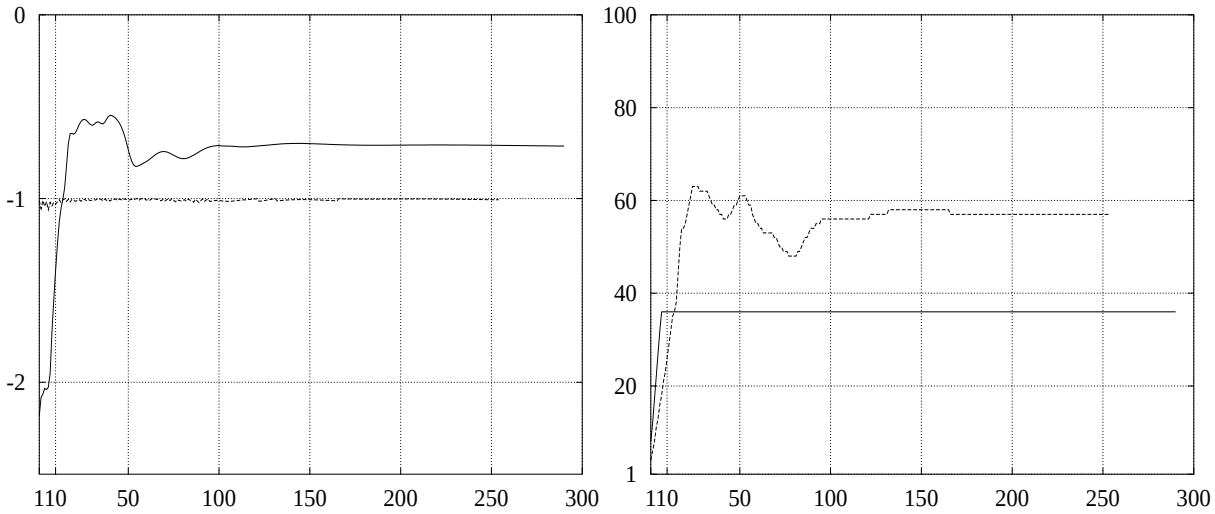

Figure 24: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S2*
Left figure : linear convergence
Right figure : effective number of Jacobi relaxations
Dashed line : $nbrel_{max} = 36$ - Straight line : $nbrel_{max} = 72$

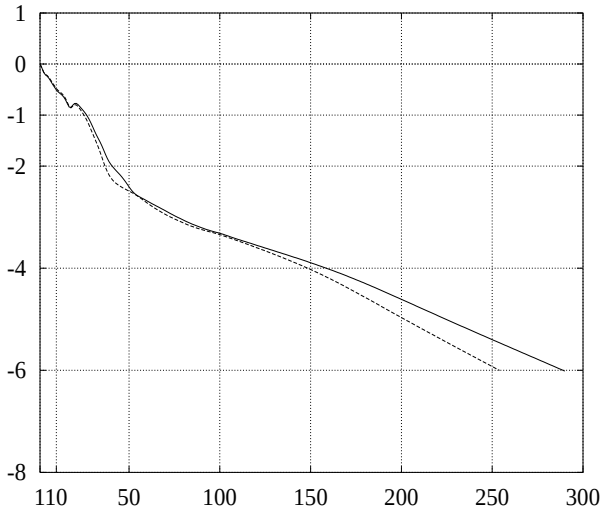

Figure 25: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S2*
Non-linear convergence
Dashed line : $nbrel_{max} = 36$ - Straight line : $nbrel_{max} = 72$

computation. In order to do so, we come back to the simulation of the external flow around an `ONERA M6` wing (the Mach number is equal to 0.84 and the angle of attack is set to 3.06°). We select mesh *M2* (15460 vertices and 80424 tetrahedra). The pseudo-time step is computed according to the law CFL=$4 \times it$ while the residual tolerance for the linear system solution is fixed to 10^{-3}. In each case (Jacobi and Gauss-Seidel methods), the convergence (initial normalized residual divided by 10^6) has required 87 non-linear iterations. The computations are performed on a `Dec 3000/700` workstation. Fig. 26 visualises the effective number of relaxations necessary to achieve the imposed residual tolerance while Tab. 20 summarizes the measures CPU times. These figures show a 20% reduction of the time spent in solving the linear systems. Despite the fact that the Gauss-Seidel relaxation method is not parallelisable (at least when it is used in the context of the approximate solution of linear systems based on sparse and irregular matrices), it can be used as the subdomain solver within an additive type of domain decomposition method such as the additive Schwarz algorithm studied by Cai[2]. Indeed, any other intrinsically sequential and efficient solver (often a direct elimination method) could play this role.

Table 19: Implicit Navier-Stokes computations on the model engine diffusor with mesh *S2*
Computations on the `Intel Paragon` : `NX` communication library, $N_p = 128$
Total CPU times for steady state computations

$nbrel_{max}$	Size	CPU	Loc Comm		Matrix Inver	
			Min	Max	Min	Max
36	1	5600.0 s	25.0 s	98.0 s	1867.0 s	2974.0 s
	0	3805.0 s	52.0 s	143.0 s	1880.0 s	2015.0 s
72	1	7071.0 s	32.0 s	128.0 s	2531.0 s	4697.0 s
	0	4324.0 s	64.0 s	178.0 s	2518.0 s	2697.0 s

Table 20: Implicit Euler computations on the `ONERA M6` wing with mesh *M2*
Computations on a `Dec 3000/700` workstation
Total CPU times for steady state computations

Solver	CPU	Matrix Inver
Jacobi	3995.0 s	3399.0 s
Gauss-Seidel	3293.0 s	2703.0 s

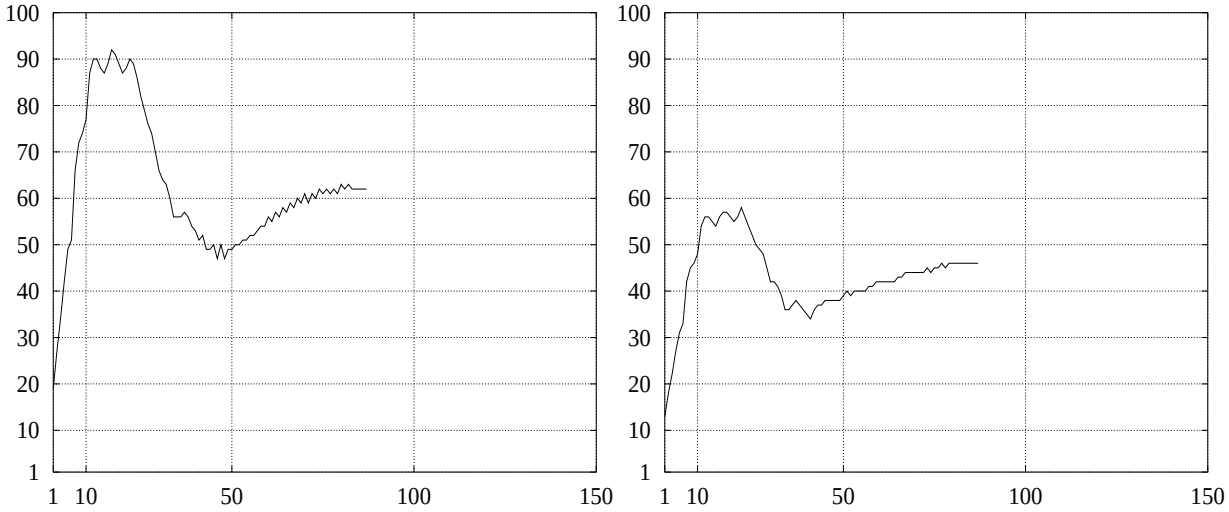

Figure 26: Implicit Euler computations on the `ONERA M6` wing with mesh *M2*
Effective number of relaxations for an imposed tolerance of 10^{-3}
Left figure : Jacobi relaxations
Right figure : Gauss-Seidel relaxations

5 Conclusion

In this paper we have obtained solutions of realistic three dimensional flows using a parallelisation strategy which combines mesh partitioning techniques and a message passing programming paradigm. Based on lessons drawn from previous works (Farhat and Lanteri[8], Fezoui and Lanteri[10], Fezoui, Lanteri and Loriot[13]) we have implemented and assessed two possible approaches : the first one makes use of overlapping mesh partitions; it contributes to minimize the programming effort on the original serial algorithm but is characterized by redundant arithmetic operations. The second approach uses non-overlapping mesh partitions and demands additional programming effort.

In the experiments considered here, the approach based on non-overlapping mesh partitions has always demonstrated better computational scalability properties. Despite the fact that this approach is also characterized by higher communication loads (additional communication steps, increased message sizes due to the exchange of partially gathered fluxes and gradient components instead of nodal physical values), the reduction of pure computational times through the elimination of redundant arithmetic operations plays the major role in this behavior. For all the tested message passing environments (`PVM`, `NX`, `MPL`), communication steps make use of blocking send calls. A substantial improvement in the treatment of this part of the parallel algorithms could be achieved by implementing strategies for overlapping communication and pure computational steps through the use of non-blocking send calls. Such strategies will be particularly interesting in the context of distributed simulations

over a network of workstations as it is suggested by the results presented in this paper for computations performed on this kind of computing platforms.

As stated in the introduction, we have considered here the parallelisation of a representative subset of an existing industrial code, `N3S-MUSCL`[3] . The complete software package which is able of simulating three-dimensional compressible reactive and turbulent flows, is currently ported on several distributed memory MIMD parallel systems using the `PARMACS` message passing environment. This porting activity is partially funded by the European Comission as part of the ESPRIT III/EUROPORT-1 project[4].

Acknowledgments

Performance results obtained on the `Ibm SP2` and on large configurations of the `Intel Paragon` have been provided by Pr. C. Farhat (*Center for Aerospace Structures*, University of Colorado at Boulder); the author addresses particular thanks to Pr. C. Farhat for having allowed the publication of these results in the present report, as well as for his constant help in the development of this work. Most of the results on small and medium configurations of the `Intel Paragon` have been obtained on the system located at Irisa/Inria Rennes with the help of Mr. H. Leroy. The author also wishes to thank Mr. J.-M. Fieni for giving him access to the `Cray T3D` located at the CEA in Grenoble. Finally, the author thanks Mr. M. Loriot from Simulog for his help in using the `MS3D` package.

References

[1] BARNARD S. and SIMON H., *Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems*, Concurrency: Practice and Experience, Vol. 6, pp. 101-117, (1994).

[2] CAI X.C., *An Additive Schwarz Algorithm for Nonselfadjoint Elliptic Equations*, Proceedings of the Third SIAM Symposium on Domain Decomposition Methods for Partial Differential Equations, Houston, Texas, March 1989, T.F. Chan, R. Glowinsky, J. Periaux and O.B. Widlund Eds., pp. 232-244, (1989).

[3] CHARGY D., *N3S-MUSCL : a 3D Compressible Navier-Stokes Solver*, User's Manual, Simulog (1993).

[4] DEGREZ G., GIRAUD L., LORIOT M., MICELOTTA A., NITROSSO B. and STOESSEL A., *Parallel Industrial CFD Calculations with N3S*, Proceedings of the High-Performance Computing and Networking Europe Conference, Milan, Italy, May 1995, B. Hertzberger and G. Serazzi Eds., Lecture Notes in Computer Science, Vol. 919, pp. 820-825, (1995).

[5] DONGARRA J.J., MEUER H.W. and STROHMAIER E., *TOP500 Supercomputer Sites*, (1994).

[6] FARHAT C., FEZOUI L. and 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).

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

[8] FARHAT C. and LANTERI S., *Simulation of Compressible Viscous Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performance Results*, Comp. Meth. in Appl. Mech. and Eng., Vol. 119, pp. 35-60, also INRIA Report No. 2154, (1994).

[9] FARHAT C., LANTERI S. and SIMON H., *TOP/DOMDEC : a Software Tool for Mesh Partitioning and Parallel Processing and Applications to CSM and CFD Computations*, Comput. Sys. in Engrg., (To Appear), (1995).

[10] FEZOUI L. and LANTERI S., *Parallel Upwind FEM for Compressible Flows*, Proceedings of the Parallel Computational Fluid Dynamics 91 Conference, Stuttgart, Germany, June 1991, K.G. Reinsch, W. Schmidt, A. Ecer, J. Hauser and J. Periaux Eds., Elsevier Science Publishers B.V., North Holland, pp. 149-163, (1992).

[11] FEZOUI L., LORIOT F., LORIOT M. and REGERE J., *A 2D Finite Volume/Finite Element Euler Solveur on a M.I.M.D. Parallel Machine*, Proceedings of the High Performance Computing II Conference, M. Duran and F. El Dabaghi Eds., Montpellier, France, Elsevier Science Publishers B.V., North Holland, pp. 283-294, (1991).

[12] FEZOUI L. and STOUFFLET B., *A Class of Implicit Upwind Schemes for Euler Simulations with Unstructured Meshes*, J. of Comp. Phys., 84, pp. 174-206, (1989).

[13] FEZOUI L., LANTERI S. and LORIOT M., *Strategies for Navier-Stokes solvers on MPP machines*, Proceedings of the International Workshop on Solution Techniques for Large-Scale CFD Problems, CERCA, Montréal, Québec, Canada, September 1994, W.G. Habashi Eds., to be published in a volume of Computational Methods in Applied Sciences, John Wiley & Sons, (1995).

[14] JOHAN Z., MATHUR K. K., JOHNSSON S. L. and 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).

[15] LANTERI S. and FARHAT C., *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).

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

[17] LORIOT M. and FEZOUI L., *A Parallel Compressible 3D Navier-Stokes Solver Using Unstructured Meshes*, CERMICS Report No. 93-19, (1993).

[18] MAVRIPLIS D.J., DAS R., SALTZ J. and VERMELAND R.E., *Implementation of a Parallel Unstructured Euler Solver on Shared and Distributed Memory Architectures*, ICASE Report No. 92-68, (1992).

[19] MORANO E. and MAVRIPLIS D.J., *Implementation of a Parallel Unstructured Euler Solver on the CM-5*, AIAA Paper No. 94-0755, AIAA 32nd Aerospace Sciences Meeting, Reno, Nevada, January 10-13, (1994).

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

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

[22] STEGER J. and WARMING R.F., *Flux Vector Splitting for the Inviscid Gas Dynamic with Applications to Finite-Difference Methods*, J. of Comp. Phys., 40, pp. 263-293, (1981).

[23] STOUFFLET B., PERIAUX J., FEZOUI L. and DERVIEUX A., *Numerical Simulation of 3D Hypersonic Euler Flows Around Space Vehicles Using Adapted Finite ELements*, AIAA Paper No. 87-0560, AIAA 25th Aerospace Sciences Meeting, Reno, Nevada, January 12-15, (1987).

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

[25] VAN LEER B., *Computational Methods for Ideal Compressible Flow*, von Karman Institute for Fluid Dynamics, Lecture series 1983-04, (1983).

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du 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