INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **30**: 179–191 (1999)

NEW TECHNOLOGIES FOR ADVANCED THREE-DIMENSIONAL OPTIMUM SHAPE DESIGN IN AERONAUTICS

ALAIN DERVIEUX[a,*], STÉPHANE LANTERI[a], JEAN-MICHEL MALÉ[a], NATHALIE MARCO[a,1], NICOLE ROSTAING-SCHMIDT[a,2] AND BRUNO STOUFFLET[b]

[a] *INRIA, Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, France*
[b] *Dassault Aviation, DGT-DEA, 78 Quai Dassault, F-92214 Saint Cloud Cedex, France*

SUMMARY

The analysis of complex flows around realistic aircraft geometries is becoming more and more predictive. In order to obtain this result, the complexity of flow analysis codes has been constantly increasing, involving more refined fluid models and sophisticated numerical methods. These codes can only run on top computers, exhausting their memory and CPU capabilities. It is, therefore, difficult to introduce best analysis codes in a shape optimization loop: most previous works in the optimum shape design field used only simplified analysis codes. Moreover, as the most popular optimization methods are the gradient-based ones, the more complex the flow solver, the more difficult it is to compute the sensitivity code. However, emerging technologies are contributing to make such an ambitious project, of including a state-of-the-art flow analysis code into an optimisation loop, feasible. Among those technologies, there are three important issues that this paper wishes to address: *shape parametrization*, *automated differentiation* and *parallel computing*. *Shape parametrization* allows faster optimization by reducing the number of design variable; in this work, it relies on a hierarchical multilevel approach. The sensitivity code can be obtained using *automated differentiation*. The automated approach is based on software manipulation tools, which allow the differentiation to be quick and the resulting differentiated code to be rather fast and reliable. In addition, the *parallel algorithms* implemented in this work allow the resulting optimization software to run on increasingly larger geometries. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: Euler flows; optimum shape design; automated differentiation; shape parametrization; parallel computing

1. THE OPTIMIZATION PROBLEM IN HAND

The optimization problem in hand is the following:

* Correspondence to: INRIA, Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, France.
[1] Projet SINUS.
[2] Projet SAFIR.

Contract/grant sponsor: Program GENIE of The French Ministry of Research

CCC 0271–2091/99/100179–13$17.50

Problem 1.1
Find γ_0 such that $j(\gamma_0) \leq j(\gamma)$, $\forall \gamma$, under the constraint that some equation is satisfied, i.e. $\Psi(W(\gamma), \gamma) = 0$.

To be more precise, interest is in finding an optimal shape (γ_0) of a three-dimensional geometry under the constraint that the *Euler* equations for compressible flows ($\Psi(W(\gamma), \gamma) = 0$) are satisfied, where $W(\gamma)$ is the flow in the domain Ω whose boundary contains γ. The appropriate Lagrangian L for Problem 1.1 is defined by:

$$L(\gamma, W, \Pi) = J(\gamma, W) + \langle \Psi(\gamma, W), \Pi \rangle, \tag{1.1}$$

where Π denotes the generalized Lagrange multiplier associated with L. Derivations of (1.1) with respect to γ, W and Π yield the following equations:

$$\nabla_w \Psi \cdot \Pi = -\nabla_W J, \tag{1.2a}$$

$$\Psi(W(\gamma), \gamma) = 0, \tag{1.2b}$$

$$\nabla_\gamma J + \langle \nabla_\gamma \Psi, \Pi \rangle = 0. \tag{1.2c}$$

Equation (1.2b) defines the constraint, (1.2a) is the adjoint state equation and (1.2c) is the optimality condition for γ_0. As stated earlier, the direct state W solution of (1.2b) for a given γ is the steady Euler flow computed in the overall domain Ω. In order to take into account possible complex geometries, such as complete aircrafts, the Euler equations are solved using a mixed finite element/finite volume method designed to operate on fully unstructured meshes.

2. SOLVING FOR THE DIRECT STATE

2.1. Mathematical model and approximation methods

Let $\Omega \subset \mathbb{R}^3$ be the flow domain of interest and Γ be its boundary. The boundary Γ is partitioned into a wall boundary Γ_w and a far-field boundary Γ_∞: $\Gamma = \Gamma_w \cup \Gamma_\infty$. The shape γ is a subset of the total boundary Γ. Let $\vec{n}$ denote the outward unit normal at any point of Γ. The conservative law form of the equations describing three-dimensional Euler flows is given by:

$$\frac{\partial W}{\partial t} + \vec{\nabla} \cdot \vec{\mathscr{F}}(W) = 0, \qquad W = (\rho, \rho \vec{U}, E)^T, \qquad \vec{\nabla} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right)^T, \tag{2.3}$$

where $\vec{\mathscr{F}}(W)$ is the vector of convective fluxes whose components are given by:

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

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 and p denotes the pressure that is obtained using the perfect gas state equation $p = (\gamma_p - 1)(E - \frac{1}{2} \rho \| \vec{U} \|^2)$, where γ_p is the ratio of specific heats ($\gamma_p = 1.4$ for air). The flow domain Ω is assumed to be a polyhedral bounded region of $\mathbb{R}^3$. Let $\mathscr{T}_h$ be a standard tetrahedrization of Ω. A vertex of the mesh is denoted by S_i, and the set of its neighboring vertices by $N(i)$. At each vertex S_i, a control volume C_i is constructed as the union of local

contributions obtained from the set of tetrahedra attached to S_i; for a given tetrahedron T, the contribution to C_i is obtained by joining the barycenter, G, of T with the barycenters of the faces of T attached to S_i; the latter barycenters are also connected to the midpoints of the edges attached to S_i. The boundary of C_i is denoted by ∂C_i, and the unit vector of the outward normal to ∂C_i by $\vec{v}_i$. The union of all these control volumes constitutes a discretization of the domain Ω.

The spatial discretization method adopted here uses a finite volume upwind formulation. Briefly, for each control volume C_i associated to a vertex S_i, one has to solve for n *one-dimensional Riemann problems* at the control volume boundary, n being the number of neighbors of S_i. The spatial accuracy of the Riemann solver depends on the accuracy of the interpolation of the physical quantities at the control volume boundary. For first-order accuracy, the values of the physical quantities at the control volume boundary are taken equal to the values in the control volume itself. In this case, the *dependency domain* of one edge is reduced to the two points sharing this edge. Extension to second-order accuracy can be performed via a 'monotonic upwind scheme for conservative laws' (MUSCL) technique of Van Leer [1]. It consists of giving the Riemann solver a better interpolated value, taking into account some gradient of the physical quantities. One can use a *finite element gradient* (P_1–Galerkin) computed on a particular tetrahedron, or an *averaged nodal gradient*, which is taken as a particular average of the finite element gradients on the set of tetrahedra sharing a given vertex. In the first case, one has to choose the candidate tetrahedron carefully, but the *dependency domain* of one edge is of fixed size (six points in two space dimensions, eight points in three space dimensions). In the second case, the *dependency domain* is of variable size, depending on the local structure of the mesh. The technique employed for finding the right element is known as *upstream/downstream element* and consists in choosing the tetrahedron that has non-empty intersection with the line containing the current edge (see Figure 1).

Integrating Equation (2.3) over C_i yields:

$$\iiint_{C_i} \frac{\partial W}{\partial t}\,\mathrm{d}\vec{x} + \iiint_{C_i} \vec{\nabla}\cdot\vec{\mathscr{F}}(W)\,\mathrm{d}\vec{x} = 0. \tag{2.4}$$

Integrating Equation (2.4) by parts leads to:

$$\iiint_{C_i} \frac{\partial W}{\partial t}\,d\vec{x} + \sum_{j\in N(i)} \int_{\partial C_{ij}} \vec{\mathscr{F}}(W)\cdot\vec{v}_i\,\mathrm{d}\sigma \qquad \langle 1\rangle$$
$$+ \int_{\partial C_i\cap\Gamma_w} \vec{\mathscr{F}}(W)\cdot\vec{n}_i\,\mathrm{d}\sigma + \int_{\partial C_i\cap\Gamma_\infty} \vec{\mathscr{F}}(W)\cdot\vec{n}_i\,\mathrm{d}\sigma = 0 \qquad \langle 2\rangle, \tag{2.5}$$

where $\partial C_{ij} = \partial C_i \cap C_j$. A first-order finite volume discretization of $\langle 1\rangle$ goes as follows:

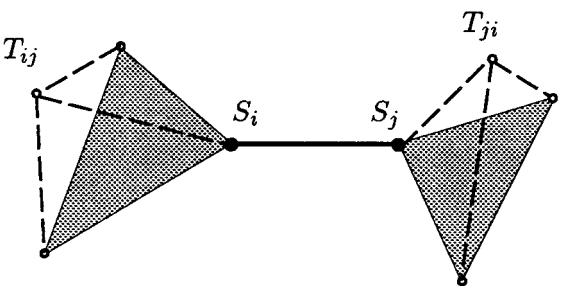

Figure 1. Definition of the upstream/downstream elements attached to one edge.

$$\langle 1 \rangle = W_i^{n+1} - W_i^n + \Delta t \sum_{j \in N(i)} \Phi_{\mathscr{F}}(W_i^n, W_j^n, \vec{v}_{ij}), \tag{2.6}$$

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

$$\Phi_{\mathscr{F}}(W_i, W_j, \vec{v}_{ij}) \approx \int_{\partial C_{ij}} \vec{\mathscr{F}}(W) \cdot \vec{v}_i \, \mathrm{d}\sigma, \quad \vec{v}_{ij} = \int_{\partial C_{ij}} \vec{v}_i \, \mathrm{d}\sigma. \tag{2.7}$$

Upwinding is introduced here in the computation of Equation (2.7) by using the Van Leer vector flux splitting (see [2] for more details), thus computing $\Phi_{\mathscr{F}}$ as:

$$\Phi_{\mathscr{F}}(W_i, W_j, \vec{v}_{ij}) = \vec{\mathscr{F}}^{+}(W_i, \vec{v}_{ij}) + \vec{\mathscr{F}}^{-}(W_j, \vec{v}_{ij}). \tag{2.8}$$

Following the MUSCL technique [1], second-order accuracy is achieved in Equation (2.8) via a piecewise linear interpolation of the states W_{ij} and W_{ji} at the interface between C_i and C_j:

$$\tilde{W}_{ij} = \tilde{W}_i + \frac{1}{2}(\vec{\nabla}\tilde{W})_i \cdot \vec{S_i S_j}, \qquad \tilde{W}_{ji} = \tilde{W}_j - \frac{1}{2}(\vec{\nabla}\tilde{W})_j \cdot \vec{S_i S_j}, \tag{2.9}$$

where $\tilde{W} = (\rho, \vec{U}, p)^T$. An *averaged nodal gradient* $(\vec{\nabla}\tilde{W})_i$ is obtained by averaging the P_1–Galerkin gradients computed on each tetrahedron of C_i. On the other hand, the *upstream/downstream element* approach consists of computing the P_1–Galerkin gradient on each of the T_{ij} and T_{ji} candidate tetrahedra. Finally, the van Albada limitation procedure is introduced in the interpolation (2.9) in order to preserve the monotony of the approximation (see [3] for details). The second term $\langle 2 \rangle$ of Equation (2.5) includes the contributions of the boundary conditions and is evaluated as follows:

- *Wall boundary*: when considering a non-viscous flow (solution of the Euler equations) the condition $\vec{U} \cdot \vec{n} = 0$ is introduced, either directly or via a transpiration condition (see Section 4), in the computation of the corresponding boundary term of Equation (2.5).
- *Inflow and outflow boundaries*: at these boundaries, a *plus–minus* flux splitting is applied between far-field uniform data and interior values.

A remark that is important to the present work is that the above discrete formulations for internal and boundary convective flux computations as well as for the MUSCL interpolation are differentiable. Indeed, the flux vector splitting, the interpolation, the limitation procedure and the transpiration condition are differentiated with respect to the flow variable W and to the co-ordinates of the geometry. This is detailed in Section 5.

2.2. 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 Equation (2.5)), the following semi-discrete fluid flow equations are obtained:

$$\mathrm{vol}(C_i) \frac{\mathrm{d}W_i}{\mathrm{d}t} + \Psi(W)_i = 0, \quad i = 1, \ldots, N_V, \tag{2.10}$$

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

$$\Psi(W)_i = \sum_{j \in N(i)} \Phi_{\mathscr{F}}(W_{ij}, W_{ji}, \vec{v}_{ij}) + \int_{\partial C_i \cap \Gamma} \vec{\mathscr{F}}(W) \cdot \vec{n}_i \, \mathrm{d}\sigma. \tag{2.11}$$

Applying a first-order linearization to the flux $\Psi(W^{n+1})$ yields the Newton-like formulation [4]:

$$\left(\frac{\mathrm{vol}(C_i)}{\Delta t^n} + J(W^n)\right)(W^{n+1} - W^n) = -\Psi(W^n), \tag{2.12}$$

where $J(W^n)$ denotes the associated Jacobian matrix. At each time step, the above linear system is approximately solved using Jacobi relaxations.

3. MULTILEVEL ALGORITHMS FOR SHAPE OPTIMIZATION

Described here are the application of hierarchical basis concepts for building multilevel algorithms for the optimization of complex three-dimensional shapes. The resulting multilevel parametrization strategy allows the numerical treatment of shapes with a large number of control parameters.

3.1. Multilevel parametrization for 3D shapes

Described here is a multilevel approach for the parametrization applied to the optimization of an aircraft geometry in a three-dimensional Euler flow; the parametrized shape is then a three-dimensional surface and flow calculations are performed on an unstructured tetrahedral mesh. The construction of a multilevel parametrization [5] of this shape will rely on a volume-agglomeration principle [6]. Given the dual discretization of the initial mesh (the finer level), a coarser level is obtained by grouping or agglomerating neighboring control volumes. This process is applied as many times as necessary to obtain the expected grid level hierarchy. The surface is assimilated to a manifold Σ, which is supposed smooth enough. The discretized surface Σ_h is nothing more than a triangulation of γ, which is part of the boundary Γ of the overall computational domain. A deformation of Σ_h is noted $\delta\Sigma_h$. A new configuration of Σ_h is computed as $\mathscr{L}\mathscr{P}\mathscr{P}^*\mathscr{L}^*\delta\Sigma_h$ where $\mathscr{P}$ is a canonical prolongation operator from a coarse level to a fine one; $\mathscr{P}^*$ denotes a restriction operator that is computed as the transpose of the prolongation operator $\mathscr{P}$, $\mathscr{P}^*$ takes a quantity defined on a fine level and projects it on a coarser level; $\mathscr{L}$ and $\mathscr{L}^*$ (the transpose of $\mathscr{L}$) are smoothing operators that are introduced in order to guaranty the V-regularity of the discretized shape (where V is an appropriate functional space); refer to [6] for more details on this construction.

The smoothing operator $\mathscr{L}$ is an average weighted by a scalar product of normals:

$$(\mathscr{L}\vec{x})_i = (1-\theta)\vec{x}_i + \theta\frac{\displaystyle\sum_{j\in N(i)\cup\{i\}} w_{ij}\vec{x}_{ij}}{\displaystyle\sum_{j\in N(i)\cup\{i\}} w_{ij}}, \tag{3.13}$$

where w_{ij} are the weights defined by:

$$w_{ij} = \max[\mathscr{A}(i)\cdot\mathscr{A}(j)\cdot(\vec{n}_i\cdot\vec{n}_j), 0], \quad \|\vec{n}_i\| = 1 \quad \forall i, \tag{3.14}$$

with $\mathscr{A}(i) = \mathrm{vol}(C_i)$ and where $N(i)$ represents the set of neighboring control volumes of C_i and θ is the smoothing parameter.

3.2. The multilevel algorithms: gradient and one-shot variants

The multilevel gradient approach is introduced in [5] and relies on the following algorithm:

Begin Multilevel algorithm
  For [each cycle n_c]
  Do
    For [each level n_l, $1 \le n_l \le n_l^{\max}$]
    Do
      Compute state and adjoint state
      Compute gradient $G(W, \Pi)$
      Compute $\gamma^{n_l+(n_l^{\max}-1)n_c} = \gamma^{n_l+(n_l^{\max}-1)n_c-1} - \rho \mathscr{L}_{n_l} \mathscr{P}_{n_l} \mathscr{P}^*_{n_l} \mathscr{L}^*_{n_l} G$
    End
  End
End Multilevel algorithm

where $\mathscr{L}_{n_l}$ and $\mathscr{P}_{n_l}$ are the smoothing operators defined in the previous section, according to level n_l, and where $G(W, \Pi)$ is a function of two variables W and Π, which is equal to $j'(\gamma)$ only if the conditions $W = W(\gamma)$ (state equation) and $\Pi = \Pi(\gamma)$ (adjoint state equation) are verified. The parameter ρ is either fixed or obtained as the result of a one-dimensional search algorithm (steepest descent version).

This algorithm results in a descent-type one when $G(W, \Pi)$ is exactly equal to $j'(\gamma)$ and it is referred to as a *multilevel gradient method* [6]; conversely, when W and Π are obtained by applying only a few iterations of state equation and adjoint state equation iterative solution (i.e. in the case of approximate state and adjoint state solutions), the G is not the gradient of j, but aims to converge towards $j'(\gamma)$ when the whole loop is converging; this is referred to as a *one-shot multilevel method* (according to [7]) for solving the optimality system of the optimization problem.

4. OVERALL OPTIMIZATION LOOP

The application of a shape optimization loop involves the repeated rezoning of the mesh in order to account for the modification of the shape. In the present work, inspired by the approach used in [8], considered in a first phase is the option of representing the shape modification by applying a transpiration-type boundary condition. In practice, this means that a new configuration of the shape is defined with respect to the initial discretization of the aircraft skin as a perturbation simulated by transpiration (see [9]), referred in the sequel as the 'transpired perturbation'. Let γ denote the perturbed shape, γ_0 being the initial configuration of γ; γ is obtained by deforming γ_0 along its normal $\vec{n}^{\gamma 0}$. Briefly recalling the principles for applying this kind of boundary condition in the context of Euler flows: if $\vec{n}^{\gamma}$ is the normal associated to the perturbed shape γ, then applying a transpiration-type boundary condition on the shape is equivalent to computing the appropriate part of the first integral of the term $\langle 2 \rangle$ of Equation (2.5) as (with $q_i = \vec{V}_i \cdot (\vec{n}_i^{\gamma} - \vec{n}_i^{\gamma 0})$):

$$\int_{\partial C_i \cap \gamma} \vec{\mathscr{F}}(W) \cdot \vec{n}_i \, \mathrm{d}\sigma = q_i W_i + p_i (0, n_x^{\gamma 0}, n_y^{\gamma 0}, n_z^{\gamma 0}, q_i)^T. \tag{4.15}$$

This approximation has proved to be accurate and robust enough for rather large perturbations of the boundary. The overall method is essentially made of three loops (Figure 2). In the general case, the external loop is a remeshing loop in which a new three-dimensional mesh is obtained from a new configuration of the shape at each optimization iteration (in that case, the transpired perturbation approach is no longer necessary); here, this external loop is not taken

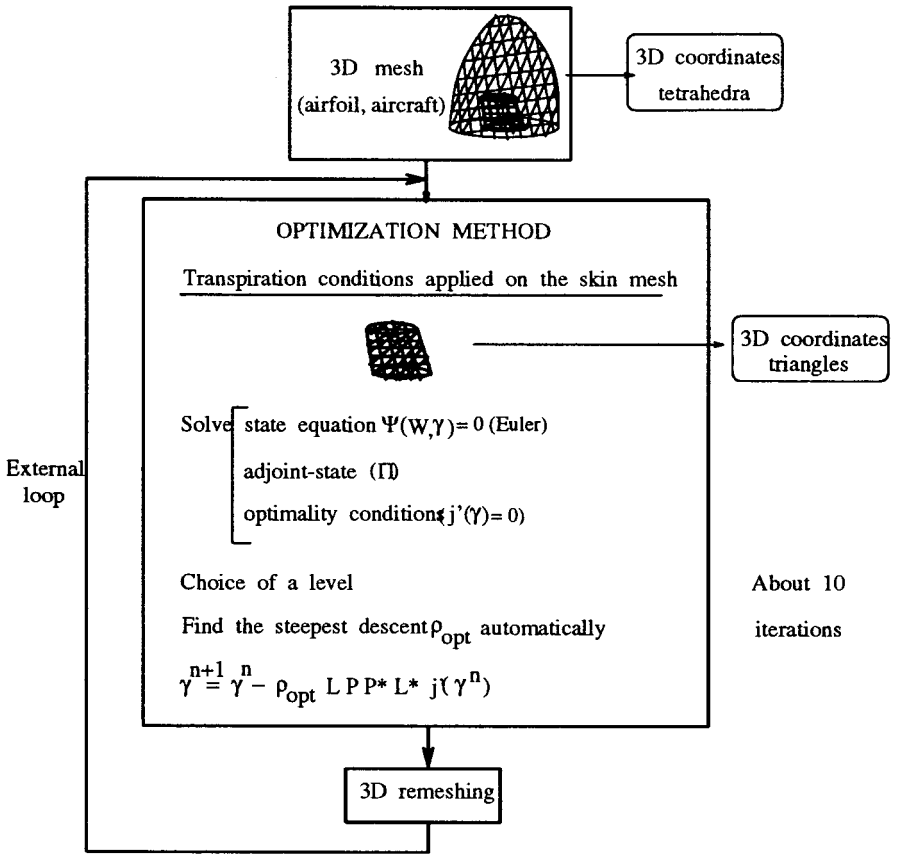

Figure 2. Organization of the optimization loop.

into account. The middle loop is a gradient optimization loop in which the control variable is the transpired perturbation; this loop involves the evaluation of the gradient of the cost functional through an adjoint state calculation. The most internal loop is the one-dimensional local research of the steepest descent parameter ρ_{opt}.

5. SOLVING FOR THE ADJOINT STATE

5.1. Automatic differentiation

Several derivatives are needed to perform the above class of computation. Several of them have been generated using an automatic differentiation tool, Odyssée[3]. This choice has some (big) advantages, and some (small) drawbacks. The advantages are clear: the tool is powerful, it saves a lot of time in human calculus, coding, error tracking; as it works by program transformation, the source code of the function is used; as the tool is automatic, one can expect it to produce reliable code. However, it may be necessary to adapt the source code in

[3] Odyssée is developed in the SAFIR project, which is a joint project between the University of Nice-Sophia Antipolis, INRIA Sophia Antipolis and CNRS.

some way, or to change the function algorithm; the generated code will more than likely not be optimal in terms of computer resource use.

5.2. A quick introduction to the Odyssée system

The automatic differentiation system Odyssée takes as input a Fortran 77 program and a set of variables with respect to which the program must be differentiated. Odyssée computes the differentiated code by transformation of the original one, and the output is a new Fortran 77 program. Odyssée system has been developed as a tool-kit of software components written in a strongly typed functional language, CAML-LIGHT [10]. This language is particularly well-suited for tree traversals and treatments.

From the user's point of view, the Odyssée system can be used through a graphic interface or a command language. Two modes of differentiation are available. In the forward mode, either a directional derivative (the linear tangent map) or the Jacobian matrix can be computed. In the reverse mode, the code computing the linear cotangent map is produced. Odyssée can perform multiprocedural differentiation, in the sense that it can differentiate a 'head' subroutine and all the procedures called within this 'head' subroutine. As only the forward mode has been used, for the purpose of computing the Jacobian matrix, focus will be on this particular feature. When computing the Jacobian matrix, a subroutine call in the original program leads to a call to the subroutine computing the derivatives with respect to the inputs of the called subroutine, and a matrix product between this matrix and the Jacobian matrix existing just before the call. These two rules gives the derivatives of the variables modified by the call with respect to the variables of the main subroutine. The second item is what makes computing the Jacobian code difficult; the data structure holding the 'matrix' may as well be a fourth-order tensor, and the 'matrix product' is in fact a generalized tensor–tensor product; Odyssée has to determine the appropriate number of nested loops, the appropriate indices, and these portions of generated code are generally CPU consuming. The advantage of this algorithm is code reusability; if the same procedure is called several times with different formal parameters, there is no need to differentiate it more than once. Specific details on Odyssée and on interprocedural differentiation may be found in [11] and an example of application in [14].

5.3. Using Odyssée

The automatic differentiator Odyssée has been used extensively to compute the various gradients needed in the different steps of the algorithm. Focus here will be on the differentiation of $\Psi(\gamma, W)$ to obtain $\nabla_W \Psi$. This matrix is then transposed 'on the fly' during the linear system solution.

5.4. Original code transformations

From the above description (Section 2), it is clear that the computation of $\Psi(\gamma, W)$ for the entire mesh involves lots of scatter–gather operations (between points and edges, elements and edges, etc.) These operations are implemented via array indirect-addressing. Thus, while the domain dependency for one edge is rather small, it is evaluated *only* at run-time. A static analysis of the original code shows that the domain dependency for one edge is virtually equal to the whole domain. This confuses code manipulation tools and that is why it should be expected that Odyssée generate maximal dependencies when computing the Jacobian matrix. This problem is critical. Small typical meshes have at least 10000 points and 70000 edges in three space dimensions and the local flux on one edge depends on very few (8) points. If

maximal dependencies are generated, a very large matrix—approximately $70000 \times 10000 = 700000000$ entries—of which $700000000 - 70000 \times 8 = 699440000$ (99.92%) will be zero, each entry consisting of a 5×5 matrix of 8 bytes floating-point numbers. This leads to a storage requirement of approximately 130 Gigabytes, for the sole matrix, while the minimal storage needed is only about 106 Megabytes for second-order accuracy, and only about 26 Megabytes for first-order. This is vital for being able to run the code to address this problem, and it has been done by carefully rewriting the original code computing Ψ (γ, W).

5.5. Eliminating spurious dependencies

The main idea is to separate computation and access to memory. The global sum of local fluxes is still computed in one big loop on the mesh edges. However, the computation of the local fluxes no longer involves direct accesses to the global structure; instead, a local structure is filled before and passed to the local flux procedure. The local flux procedure only depends on the local structure, which is of *fixed* small size. Local flux procedures are differentiated with Odyssée, and a structure very similar to the rewritten algorithm is implemented to compute the complete derivative. Resolving all indirections before the call to the function to be automatically differentiated leads to little extra CPU cost (5%) because of the **CALL** overhead, but ensures minimal memory occupation because dependencies are minimal.

5.6. Parallelization strategy

The parallelization strategy adopted in this study combines domain 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. When this parallelization strategy is applied to the direct state calculation (i.e. the Euler flow solver), modifications occurred 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). This approach enforces data locality, and therefore, is suitable for all parallel hardware architectures. For the partitioning of the unstructured mesh, two basic strategies can be considered. The first one is based on the introduction of an overlapping region at subdomain interfaces and is well-suited to the mixed finite volume/element formulation considered herein. Mesh partitions with overlapping have a main drawback: they incur redundant floating-point operations. The second possible strategy is based on non-overlapping mesh partitions and incur no redundant floating-point operations. While updated nodal values are exchanged between the subdomains in overlapping mesh partitions, partially gathered quantities are exchanged between subdomains in non-overlapping ones. It has been the experience of the authors that both the programming effort and the performances are maximized when considering non-overlapping mesh partitions [12]. In the present study, one-tetrahedron wide overlapping mesh partitions will be considered for second-order-accurate implicit computations.

Concerning the overall optimization loop, the above parallelization strategy is straightforwardly extended to the adjoins state calculation (i.e. the construction of the adjoint state Jacobian matrix and the linear solution of the resulting system using Jacobi relaxations). Finally, the only part that remains sequential (i.e. duplicated as it is on each processor) concerns the shape parametrization; we note that this part is concerned with a small data subset (i.e. the set of control points) compared with the complete computational data set (i.e. the tetrahedral mesh). In order to be able to do that, additional communication steps take

place at each optimization iteration in order to distribute the pressure values as well as the displacement values associated to the set of control points.

6. OPTIMIZATION OF A FALCON JET GEOMETRY

The aim of this section is to demonstrate the application of the proposed shape optimization strategy to an industrially relevant problem. The geometry under consideration is the one of a Falcon aircraft. The three-dimensional computational mesh contains 45387 nodes and 255944 tetrahedra; the Falcon skin mesh consists of 2992 nodes (i.e. control points) and 5838 triangles. The Odyssée tool has been applied to Van Leer flux routines in order to generate the code for the first-order adjoint state Jacobian matrix. It is only a preliminary result, in the sense that the memory needed to store the matrix is not too important in this case. The direct state (i.e. the steady Euler flow computed for a free-stream Mach number equal to 0.85) has been calculated using the second-order-accurate MUSCL formulation based on the *averaged nodal gradient approach*. It is noted that the *upstream/downstream element* formulation will be used in future works for computing a second-order-accurate adjoint state Jacobian matrix; this means that the Odyssée tool will also be applied to the routines yielding the MUSCL interpolation.

The multilevel parametrization of the skin mesh uses four levels that are visited using a full V-cycle strategy. The second to fourth levels contain 873251 and 76 nodes (i.e. control points) respectively. Firstly, the *multilevel gradient method* has been applied. For this purpose, the direct state is solved using the implicit time advancing scheme (2.12), the steady state being considered as obtained for a value of the absolute non-linear residual equal to 10^{-6} using a maximum of 100 time steps (non-linear iterations); a variable CFL number strategy has been used such that the pseudo-time step increases proportionally to the inverse of the non-linear residual; for each time step, the tolerance for the approximate solution of system (2.12) has been fixed to 10^{-2}. Finally, the adjoint state is obtained using Jacobi relaxations on the adjoint state system; as the accuracy of the solution is crucial to the obtaining of a correct value of the functional gradient, a large number of relaxations are necessary in order to solve this system at a tolerance that has been fixed to 10^{-8}. A *one-shot multilevel method* has thus be applied. In this case, a maximum of 15 time steps have been used for the direct state solution (except for the first optimization iteration, for which this figure has been kept to 100). For the adjoint state solution, a maximum of 100 Jacobi relaxations have been used (except for the first optimization iteration, for which 2000 Jacobi relaxations have been used).

6.1. Numerical results

The challenge taken up was to find a geometry corresponding to a target pressure distribution. Therefore, the cost functional has been defined as $J(\gamma, W) = \int_\gamma (P(W) - P^t)^2 \, d\sigma$ where P^t denotes the target pressure distribution. The latter has been obtained by solving for the steady state solution using the initial geometry and then applying several iterations of the multilevel smoothing procedure on the associated pressure distribution. Convergence histories of the functional $J(\gamma, W)$ and of the l_2-norm of the cost functional gradient are depicted in Figure 3 for both optimization methods; these history curves are rather similar for both methods. Figure 4 gives the non-linear convergence, i.e. the convergence of the Euler calculations, across the optimization iterations for the *one-shot multilevel method*. The isolines of the functional gradient on the Falcon surface at the end of the first optimization iteration are visualized in Figure 5.

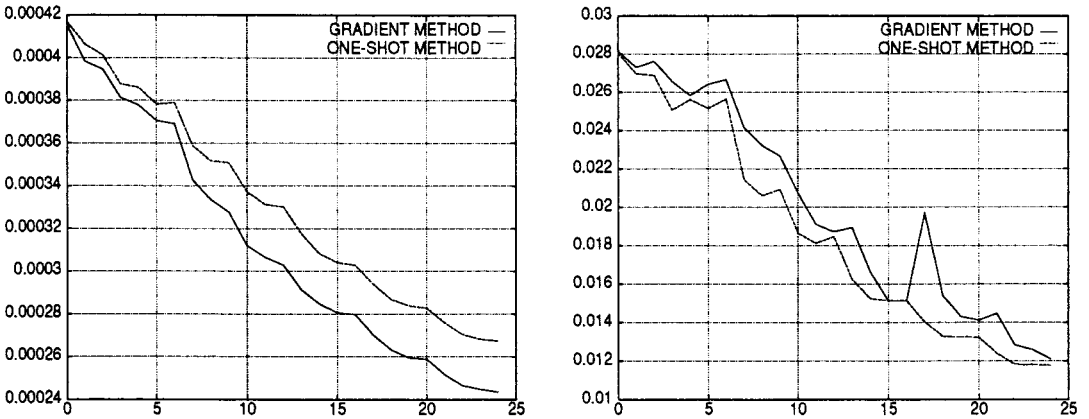

Figure 3. Cost functional $J(\gamma, W)$ vs. optimization iteration (left); l_2-norm of the cost functional gradient vs. optimization iteration (right).

6.2. Performance results

The above calculations have been performed on a six-node SGI POWER CHALLENGE ARRAY equipped with MIPS R8000/90 MHz processors. The communication steps are performed using the MPI library even though the architecture is a shared memory MIMD one. Performance results are gathered in Table I. All performance results reported herein are for 64-bit arithmetic. The total CPU times refer to the maximum of the individual processor measures; the other measures refer to both minimum and maximum values of the direct state and adjoint state solution times. The first striking out result is the much better performance of the one-shot computation. Secondly, most of the CPU time for both optimization methods is spent in the direct state solution. For the one-shot case, the total cost is not much larger that completely solving 6–8 times the state equation.

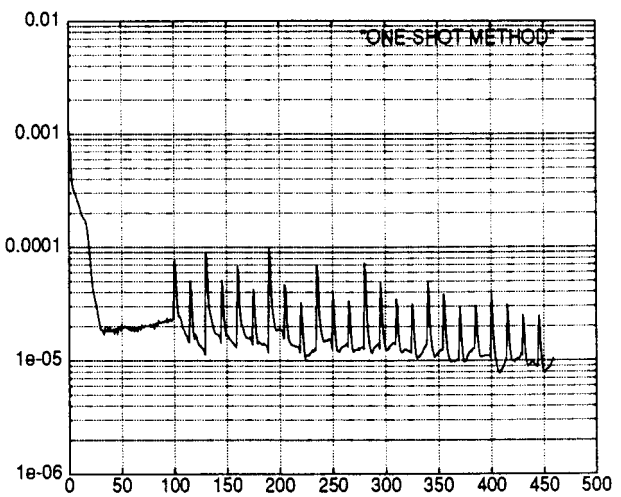

Figure 4. Direct state non-linear convergence in the one-shot multilevel method.

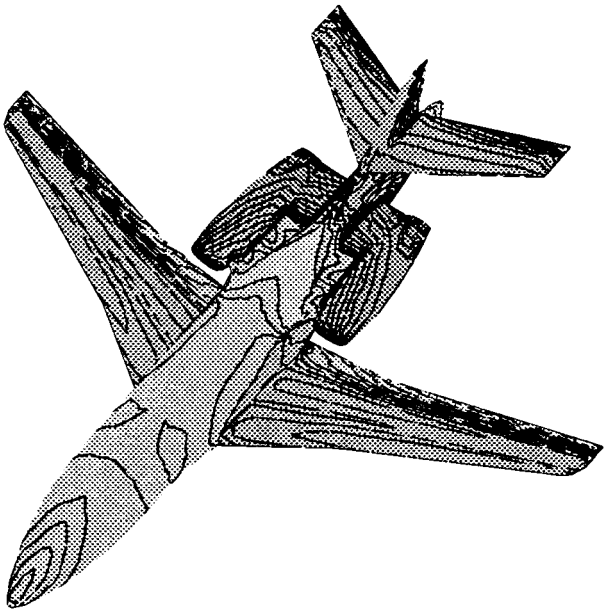

Figure 5. Isolines of the functional gradient on the Falcon surface (first optimization iteration).

7. CONCLUSIONS

An experiment, relying on the following ingredients, has been presented: (1) automated differentiation has been applied to several modules of the code, (2) the direct state solution is based on a parallel unstructured mesh Euler solver, and (3) use of a numerical mutilevel parametrization for dealing with a large number of shape unknowns. It has been observed that:

(1) State equation solving stays the dominant CPU concern; further efforts for faster solving are a priority, e.g. by using multigrid acceleration techniques [13];

(2) Odyssée has proven to be rather efficient, reliable and easy to use. Two points are to be noted. First, Odyssée-generated code is significantly slower than hand-coded derivatives, *when available* (a factor 5 in CPU time). The second point is that when complexity increases, it is usually extremely difficult or even impossible to hand-code exact derivatives. In these experiments, the whole process of rewriting crucial code and differentiating it with Odyssée has proven to be a faster alternative (a factor 30 in *human* time) with respect to hand-coding the derivatives;

Table I. CPU times for 25 optimization iterations

Method	Total optimization	Direct state solution		Adjoint state solution	
		Min	Max	Min	Max
Gradient	30 h, 50 min	26 h	30 h	2 h, 25 min	2 h, 30 min
One-shot	2 h, 30 min	1 h, 50 min	1 h, 52 min	38 min	40 min

Comparison between the *multilevel gradient method* and the *one-shot multilevel method*. Calculation on six nodes of a SGI POWER CHALLENGE ARRAY.

(3) Mutilevel parametrization is very easy to use; however, the main feature that should be improved is the tuning of pseudo-gradient step lengths.

Future works will concentrate on more complex approximations and models for which a matrix free adjoins state solver seems mandatory.

ACKNOWLEDGMENTS

This work has been partly supported by the French program GENIE of Ministry of Research. The authors thank Dassault Aviation for providing the Falcon geometry.

REFERENCES

1. B. Van Leer, 'Towards the ultimate conservative difference scheme V: a second-order sequel to Godunov's method', *J. Comp. Phys.*, **32**, 361–370 (1979).
2. B. Van Leer, 'Flux-vector splitting for the Euler equations', *Lectures Notes in Phys.*, **170**, 507–512 (1982).
3. L. Fezoui and A. Dervieux, 'Finite element non-oscillatory schemes for compressible flows', *Proc. 8th France–USSR–Italy Joint Symposium on Computational Mathematics and Applications*, Pavie, Italy, 1993.
4. L. Fezoui and B. Stoufflet, 'A class of implicit upwind schemes for Euler simulations with unstructured meshes', *J. Comp. Phys.*, **84**, 174–206 (1989).
5. F. Beux and A. Dervieux, 'A hierarchical approach for shape optimization', *Eng. Comp.*, **11**, 25–48 (1994).
6. N. Marco and A Dervieux, 'Multilevel parametrization for aerodynamical optimization of 3D shapes', *Tech. Rep. 2949*, INRIA, 1996.
7. S. Ta'asan, G. Kurovila and M. D. Salas, 'Aerodynamic design and optimization in one-shot', *AIAA Paper 92-0025*, *AIAA 30th Aerospace Sciences Meeting and Exhibit*, Reno, NV, 1992.
8. W.P. Huffman, R.G. Melvin, D.P. Young, F.T. Johnson, J.E. Bussoletti, M.B. Bieterman and C.L. Hilmes, 'Practical design and optimization in computational fluid dynamics', *AIAA Paper 93-3111*, *AIAA 24th Fluid Dynamics Conference*, Orlando FL, 1993.
9. G.D Mortchelewicz, 'Résolution des equations d'Euler tridimensionnelles instationnaires en maillages non-structurés', *La Recherche Aérospatiale*, **6**, 17–25 (1991).
10. X. Leroy, *The CAML-LIGHT System*, *Documentation and User's Manual*, release 0.7, INRIA Rocquencourt, Le Chesnay Cedex, France, 1995.
11. N. Rostaing, S. Dalmas and A. Galligo, 'Automatic differentiation in Odyssée', *Tellus*, **45A**, 558–568 (1993).
12. S. Lanteri, 'Parallel solutions of compressible flows using overlapping and non-overlapping mesh partitioning strategies', *Parallel Comput.*, **22**, 943–968 (1996).
13. M.H. Lallemand, H. Steve and A. Dervieux, 'Unstructured multigriding by volume agglomeration: current status', *Comput. Fluids*, **21**, 397–433 (1992).
14. B. Mohammadi, J.-M. Malé and N. Rostaing, 'Automatic differentiation in direct and reverse modes: application to optimum shapes design in fluid mechanics', in M. Berz, C. Bischof, G. Corliss and A. Griewank (eds.), *Computational Differentiation: Techniques*, *Applications and Tools*, SIAM, 1996.

