Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2012, Article ID 317359, 14 pages
doi:10.1155/2012/317359

Research Article

Parallel Mesh Adaptive Techniques Illustrated with Complex Compressible Flow Simulations

Pénélope Leyland,[1] Angelo Casagrande,[1] and Yannick Savoy[2]

[1] *EPFL STI GR-SCI-IAG, Station 9, 1015 Lausanne, Switzerland*
[2] *APCO Technologies, Chemin de Champex 10, CH-1860 Aigle, Switzerland*

Correspondence should be addressed to Angelo Casagrande, angelo.casagrande@epfl.ch

Received 27 April 2012; Accepted 13 August 2012

Academic Editor: Antonio Munjiza

The aim of this paper is to discuss efficient adaptive parallel solution techniques on unstructured 2D and 3D meshes. We concentrate on the aspect of parallel a posteriori mesh adaptation. One of the main advantages of unstructured grids is their capability to adapt dynamically by localised refinement and derefinement during the calculation to enhance the solution accuracy and to optimise the computational time. Grid adaption also involves optimisation of the grid quality, which will be described here for both structural and geometrical optimisation.

1. Introduction

The accuracy of a numerical simulation is strongly dependant on the distribution of grid points in the computational domain. For this reason grid generation remains a topical task in CFD applications. Prior knowledge of the flow solution is usually required for a grid to be efficient, that is, matching the features in the flow field with appropriate grid resolution. This, however, may not be available, requiring human intervention in analysing the results of an initial solution, going back to the preprocessing stage, and taking an educated guess at how the mesh should be modified. Alternatively, a generally fine grid over most parts of the domain is generated to obtain a relatively good solution. Both of the above cases however, require excessive time, effort, and computational resources.

Let us consider the case with manual intervention by the user. This step can be automated by adaptation, whereby the flow solution is analysed automatically, following some predefined criteria, and the grid resolution adjusted to the problem. The use of such techniques allows for computationally precise distribution of grid points (rather than eye precision) and for extremely reduced user intervention, thus addressing the time and effort issues. It also resolves problems related to computational time and costs, as the adapted grid can have fewer overall points, with similar resolution in areas of interest, than an unadapted fine mesh.

Grid enrichment (h-refinement) is used here; that is, the density of grid points is increased in regions in order to minimise the space discretisation error [1].

In this method the mesh topology is drastically changed, as nodes are added and removed in order to capture flow features and at the same time reduce the computational load in areas where the solution is sufficiently smooth. Therefore it is particularly suitable for unstructured grids, where the structure can undergo significant changes.

The criteria for refinement and derefinement can be based on solution-based criteria and/or error estimation criteria. Grid enrichment may be further divided into two main streams, grid remeshing and grid subdivision. We will be using the second method, with the grid being divided into smaller elements where necessary. New nodes are added to edges that are identified for refinement, and in turn the cells are divided. Therefore, it is easy to see how the use of unstructured grids can be particularly beneficial. The advantage of this method is its speed and efficiency.

Drawbacks of this technique are the complex data structure and most often, the lack of information of the underlying geometry on the bounding surfaces.

This technique can be approached in two different ways, with a hierarchical framework which saves parent-child relationship between cells at every step and a non-hierarchical approach which discards the history of the original mesh during the filiation of successive grids. The method adopted here is completely nonhierarchical [2], since higher quality meshes can be achieved. This is due to the greater flexibility gained from the omission of the original macromesh, which in turn allows the use of high performance structural optimisation algorithms. With the use of efficient (de)refinement techniques, this approach is well suited for transient problems or for producing coarse grids to be used in multigrid algorithms. In fact, the resulting grids are almost equivalent to those obtained by remeshing, with much less computational time required.

The paper is organised as follows. Grid adaption is treated in detail in Section 2. Numerical results are shown in Section 3. Parallel performance aspects are discussed in Section 4. Finally, Section 5 outlines the conclusions.

2. Parallel Grid Adaptation Techniques

The algorithm outlined in [3] allows the solution of the problem, resulting from a space discretisation on a given grid, say $\mathcal{T}_h^{(0)}$. Once a preliminary solution of $\mathcal{T}_h^{(0)}$ has been obtained, one or several steps of grid adaptation cycles are considered in order to improve the solution accuracy and to optimise the computational resources. This means that, after convergence on $\mathcal{T}_h^{(0)}$, the grid is adapted one or several times. At adaptation cycle i, the solution $\mathbf{U}^{(i)}$ corresponding to the solution of the pseudotransient problem on the grid $\mathcal{T}_h^{(i-1)}$ is projected on the grid $\mathcal{T}_h^{(i)}$ and then used as a starting solution for the problem

$$S\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{1}$$

discretised on $\mathcal{T}_h^{(i)}$, with S a nonsingular (lumped) mass matrix.

Note that different space discretisation techniques can be used at different grid adaptation cycles. In general, we start with first-order schemes on nonadapted grids, then we turn to second-order schemes on adapted grids.

The goal of grid adaptation is to increase the accuracy of the solution process by locally enforcing the h-adaptivity using smaller discretisation elements. This process tends to uniformly equidistribute the local error η_h throughout the grid, $\mathcal{T}^h$.

The first step in a grid adaptation algorithm is therefore to locally evaluate criteria corresponding to the solution error estimate and mark out the zones to be modified in order to minimise globally the error. The criteria used should be as close as possible to the error estimations of the underlying discretisation scheme, taking as adaptation criteria functions of the current solution field (local error). There are several derivations of the adaptation criteria. One way is to evaluate the error based on the form of the original equations, a priori, which is a challenging task for nonlinear complex systems such as the hyperbolic-elliptic system of the Euler equations. Another is to evaluate an optimisation procedure based on derivatives. For the grid adaptation procedures developed here, a strategy based on an a-posteriori error estimate where the computed residual of the solution is used to define the error [4, 5] has proved to be robust and precise for inviscid flows and for tracking discontinuities.

Strictly speaking, all the error estimation-based criteria require a complete formulation of the underlying discretisation scheme of the non-linear system that is used to model the physical problem. For finite element discretisations, there are several rules for assuring admissibility, conformality, and regularity of the geometrical properties of the grid elements [6]. Also, the various forms of the adaptation criteria can be exactly deduced from the discretised system which assures accuracy and stability. This is applicable for model problems and even the incompressible Navier-Stokes equations. For numerical schemes for hyperbolic problems, and even more so for the compressible Euler or Navier-Stokes systems, the exact formulation of the discretisation schemes is still incomplete, especially for equivalent finite volume-type schemes, where the error estimation can be obtained by duality arguments on model problems. For the multidimensional upwind schemes used in the present work, this is still an open issue. In [7], some advances are made in this direction. A special mention must be made on recent works of discontinuous Galerkin methods, which allow deep and complex mathematical background and hence render "exact" error estimates [8, 9]. However, these techniques are not the concern here. The choice of an adaptation criterion starts with the study of the partial derivative operators of the underlying equations and hence reflects the physical phenomena. It is hence logical that physical criteria enter into the criterion. The criteria used here are mostly based on physical quantities evaluated on the solution u_h and the residual R_h. Another concern is the regularity of the grid, which also is a function of the physics coming from the equations. Indeed, anisotropic grid adaptation techniques, for example, for boundary layer adaptation, are based on working on the regularity of the grid within a certain metric coming from the equation system and detecting the dominating directionality [10–12]. Here, isotropic grid adaptation is required as the fundamental properties of the system are of wave nature. Regularity and grid optimisation strategies are developed using techniques based on error redistribution and spring analogy techniques.

Adaptation requires in all cases a local error estimate per grid cell, $\eta(T_k)$, where $\mathcal{T}_h = \bigcup_k T_k$, ponderated by some tolerance levels:

$$\max_{T_k} \frac{\eta(T_k)}{\widetilde{\eta}(T_k)} = \delta. \tag{2}$$

Here, $\widetilde{\eta}(T_k)$ can be the average on the neighbours of T_k. If the ratio $\eta(T_k)/\widetilde{\eta}(T_k) > \delta$, then T_k is to be refined.

In this work local estimates are all based on a posteriori criteria, which require a solution on the starting grid. Then,

grid refinement and coarsening operations are performed, taking as adaptation criteria functions of the current solution field (local error) and geometrical properties of the current grid (optimisation).

These criteria are used to perform both grid refinement and derefinement operations at first. Then, an optimisation step follows, based on geometrical properties of the current grid, followed by repartition, reordering, and renumbering. These phases are now detailed.

2.1. Adaptation Criteria. The physical adaptation criteria adopted in this work are based on flow quantities such as density, Mach number, pressure, and entropy, as are also error estimators, but differ in the simplicity of their construction. In fact these use directly the physical quantities mentioned. A first method is to take the difference between the values at the nodes of a segment and use its absolute value as an indicator for the adaptation process. Although this may seem as a very crude way of identifying flow features, it is very effectively applied to the grid enrichment method mentioned earlier. Another method employed is the undivided gradient along an edge [13]:

$$\varepsilon = \frac{\partial u}{\partial x} h, \tag{3}$$

which discretely can be written as

$$\varepsilon = \Delta u. \tag{4}$$

From this it can be clearly seen that the value of ε, which should approximate the error, decreases as the mesh size h becomes smaller.

Various modifications to this method have been developed, such as inclusion of local mesh length scale:

$$\varepsilon = \Delta u \cdot \Delta x. \tag{5}$$

This leads to a more effective refinement criterion [14], as the simple form of (4) remains approximately constant in the vicinity of shock waves, due to the steepened shock wave profile as the mesh is refined and the jumps remaining relatively constant. The drawback is a heavier weight of larger cells than smaller ones because of the additional length scale, even in regions of smooth flow, leading to global refinement. Although these criteria have been successfully employed, they are not optimal. This is due to the tendency of excessively refining the mesh.

In fact the so-called "physical" adaptation criteria correspond to exact mathematical error estimators. In [5], it is shown that for the linear advection diffusion problems, the evaluation of the jump of a characteristic variable across an edge is equivalent to the evaluation of the discrete H^1 norm of the solution. The relation between physical and mathematical criteria is therefore very close.

Two adaptation criteria used herein are based on physical criteria. Let us consider a solution field U that has been evaluated over the entire mesh for the selected criterion function [15]. A low- or high-pass filter is then applied on the solution field, which we will then denote as $\hat{U}$. For each segment, the function $f = f(\hat{U})$ is computed, where $f(\hat{U})$ is one of the following:

(i) the difference of $\hat{U}$ between the vertices of the segment

$$\left| \hat{U}_a - \hat{U}_b \right|. \tag{6}$$

(ii) the gradient of $\hat{U}$ between the vertices of the segment

$$\frac{\left| \hat{U}_a - \hat{U}_b \right|}{l_{ab}}, \tag{7}$$

where a and b are the end nodes of the segment and l_{ab} is the segment length. For the 2D case only, the choice also includes

(i) the upwind flux of $\hat{U}$ through the segment,

(ii) the downwind flux of $\hat{U}$ through the segment.

Further control on the field is obtained through the use of a filter F that removes part of the segments from the field. This can be applied in two ways, as an offset value, or as a cut-off value (Figure 1). In the first case each node of the mesh is examined and the following is applied:

$$\hat{U} = \max\left(0, \hat{U} - F\right). \tag{8}$$

In the second case, the above changes to

$$\hat{U} = \min\left(F, \hat{U}\right). \tag{9}$$

The refinement criterion is then built with $\overline{f}$, the mean value of f, over the grid. The segment i is then marked, for splitting in the case of refinement or for remaining in the case of the derefinement step, if

$$f_i \geq \mathcal{F}\overline{f}, \tag{10}$$

where $\mathcal{F}$ is a factor used to set the criterion as a function of the mean value $\overline{f}$ (Figure 2). In other words $\mathcal{F}$ is a coefficient that multiplies the mean value of the segment field, and all segments with higher values are marked.

2.2. Mesh Refinement and Coarsening. The adaptation procedures developed here apply for general shaped elements in 2 or 3 dimensions. They are based on the concept of an element built up as an agglomerate of a cell and its neighbours, called a shell. The filters work on the shells, as the structural optimisation procedures that are described below. The first step is the local refinement and coarsening step, which requires evaluation of the criteria and successive marking out of the interior properties of the shells.

Initially, the algorithm tests all the segments of the existing grid and decides whether a new node has to be created on each segment or not. Usually, a low-pass and a high-pass filter are applied to the solution fields. These filter operations render an error estimation in an appropriate norm, and also detect discontinuities. By filtering the

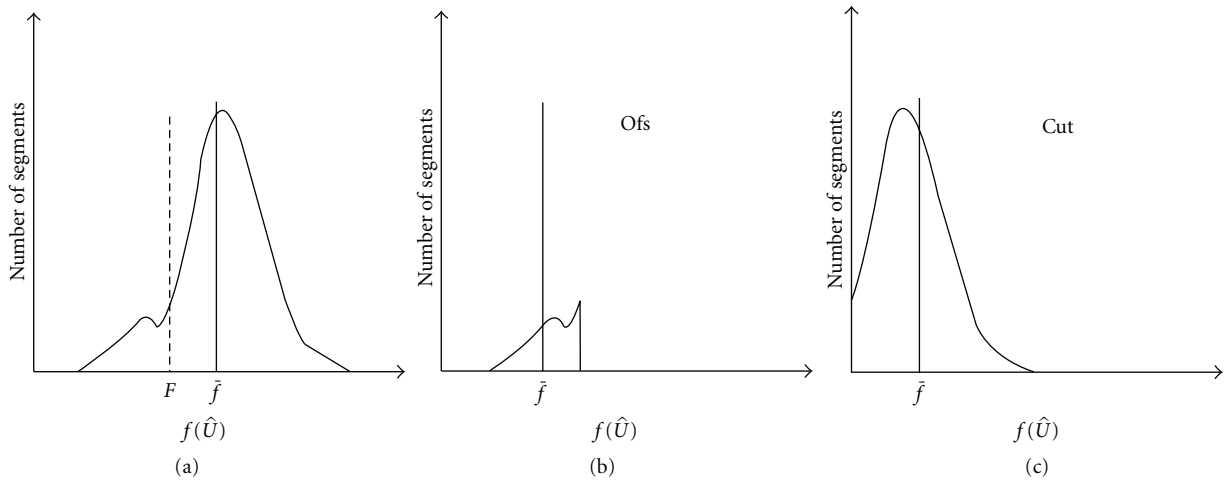


Figure 1: Graphical representation of the filter F and its effect.

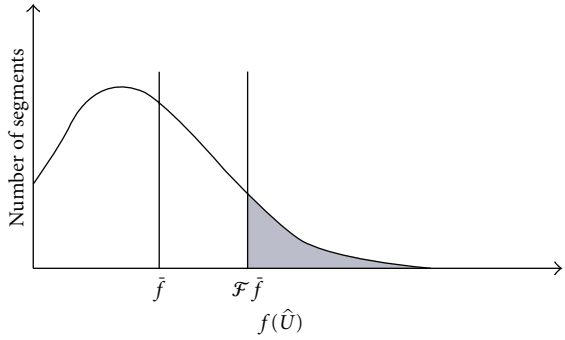


Figure 2: Graphical representation of factor $\mathcal{F}$ and $\overline{f}$ values used to limit segment marking.

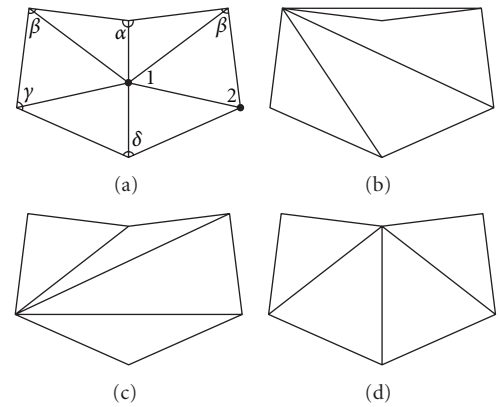


Figure 3: Segment collapsing with shell control: (a) initial shell, (b) shell inversion collapse, (c) collapse with element distortion, and (d) best collapse available.

gradient of the pressure or the Mach number, for instance, a local densification and stretching of the grid is applied. Also criteria based on the original geometry of the grid are used to optimise the grid structure. These criteria will depend on factors such as absolute segment length, related to neighbouring cells, and so forth.

Then, in order to minimise the number of nodes and elements and to improve the geometrical quality of the grid, a grid coarsening algorithm is employed. The procedure is essentially as the one outlined for grid refinement. As a result, a set of nodes to be deleted is found. These nodes are removed using an edge collapsing procedure (see Figure 3).

The low- or high-pass filter which is applied on the solution field u_h becomes a certain function $\hat{u}_h$. This can be, for example, the difference of $\hat{u}_h$ between the two vertexes of the segment. Let f_i be the value of $\hat{u}_h$ on the segment i. The refinement is performed on the segment i if $f_i \geq \mathcal{F}\overline{f}$, or it is kept unchanged otherwise. The factor $\mathcal{F}$ is used to set the criterion as a function of the mean value $\overline{f}$. This filtering process acts as a cutoff to the *a posteriori* error, by limiting its domain of influence to only a bandwidth of values.

This segment collapsing method used herein, developed by Savoy and Leyland [15], consists in building a shell around the node marked for removal with its surrounding elements. Let us consider the shell created around node 1 in Figure 3(a). Vertex 2 will not be considered as it is also marked, whilst at all other vertices the inner curvature angle will be calculated. The next step consists in collapsing one of the inner segments in order to delete the centre node. The choice will fall onto the segment that connects the centre node to the neighbouring node with the greater angle associated to it, in this case α. Note that with this technique the risk of cell inversion (Figure 3(b)) and element distortion (Figure 3(c)) is minimised, resulting in improved mesh quality. However, shell volume conservation is also checked, in order to prevent accidental element inversion from happening.

The procedure works in both two and three dimensions and is very efficient in the first case. Efficiency is somewhat reduced in the 3D case due to a large number of constraints imposed during the marking, especially when avoiding

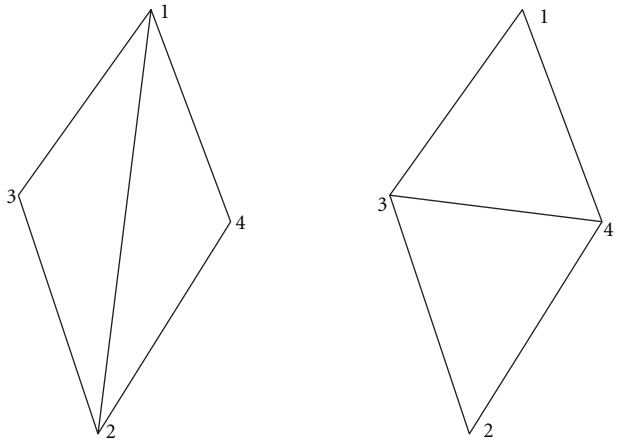


Figure 4: Two-dimensional edge swapping.

element inversion, which in turn does not allow to remove a large amount of nodes.

2.3. Optimisation Techniques. Mesh quality and precision of the underlying discretisation are highly dependent on the shape of elements and shells just described. Therefore an equilibrium state would be desirable in the cells. This is achieved by equilateral triangles in 2D and equilateral-type tetrahedra in 3D. However, the mesh obtained after the refinement and coarsening steps will be far from this desired equilibrium state. This is due to the different local node density and strong variations between element sizes and nodes angles. The number of node neighbours may also differ dramatically between vertices. In order to overcome these problems arising from the previous steps, the mesh must be optimised. This is done in several ways that may be grouped into two major strategies:

(1) structural optimization:

  (a) diagonal swapping,
  (b) edge collapsing,

(2) geometrical Optimisation:

  (a) spring analogy,
  (b) boundary smoothing,
  (c) inverted elements.

2.3.1. Structural Optimisation. In this step the mesh is analysed and modified in function of the number of node neighbours $\mathcal{N}_i$. Following the Delaunay criterion [16], where the optimal element should be equilateral, $\mathcal{N}_{\text{opt}}$ is then related to the number of equilateral elements needed to fill the area around the node. In the two-dimensional case, $\mathcal{N}_{\text{opt}} = 6$ and can be easily calculated by considering $\pi/3$, in the Euclidean metric, as the optimal node angle. For the three-dimensional case, the spherical angle of the tetrahedron at each vertex is considered and the number of neighbouring elements calculated. The Euler-Descartes relation is then used to find the number of neighbouring nodes, leading to $13 < \mathcal{N}_{\text{opt}} < 14$ (for further details see [7, 15]).

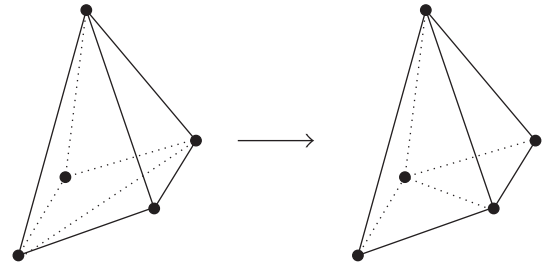


Figure 5: Three-dimensional face swapping.

Diagonal Swapping. This consists in swapping the internal edge of two neighbouring triangles, as shown in Figure 4, for the two-dimensional case.

The procedure is carried out to reduce the number of node neighbours $\mathcal{N}_i$ when this is greater than $\mathcal{N}_{\text{opt}}$. This is done by checking $\mathcal{N}_i$ on all vertices implicated in the operation. In particular the swapping is performed if the following conditions are satisfied:

$$\mathcal{N}_3 + \mathcal{N}_4 + 2 < \mathcal{N}_1 + \mathcal{N}_2 \tag{11}$$

or

$$\begin{gathered} \mathcal{N}_3 + \mathcal{N}_4 + 2 \\ \max(\mathcal{N}_3, \mathcal{N}_4) + 1 < \max(\mathcal{N}_1, \mathcal{N}_2). \end{gathered} \tag{12}$$

The three-dimensional case requires more effort and attention, as the swap implies a face swapping, leading to complete remeshing of the shell built with the elements surrounding the deleted segment. The volume conservation must also be checked in order to avoid cell inversions during the shell remeshing. An example of a face swap is shown in Figure 5.

Edge Collapsing. This intervention is done when $\mathcal{N}_i < \mathcal{N}_{\text{opt}}$, and although the method is similar to the one shown in Section 2.2 the scope is completely different. As for the swapping, the collapsing criteria are applied to the segments. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the node neighbour numbers for the two vertices of the given segment, with $\mathcal{N}_1 \leq \mathcal{N}_2$. The collapsing is done by deleting the node which corresponds to $\mathcal{N}_1$. The collapsing is performed if

$$\mathcal{N}_2' \leq \mathcal{N}_2 \quad \text{or} \quad \mathcal{N}_2' \leq \mathcal{N}_{\text{opt}}, \tag{13}$$

where $\mathcal{N}_2'$ is the node neighbours' number resulting from the collapsing. It can be deduced from $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{M}$ the number of cells surrounding the segment:

$$\mathcal{N}_2' = \mathcal{N}_1 + \mathcal{N}_2 - \mathcal{M} - 2. \tag{14}$$

These criteria are valid in both two and three dimensions. An example of edge collapsing in 2D is shown in Figure 6.

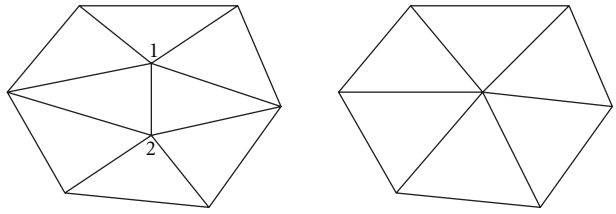


Figure 6: Edge collapsing in 2D.

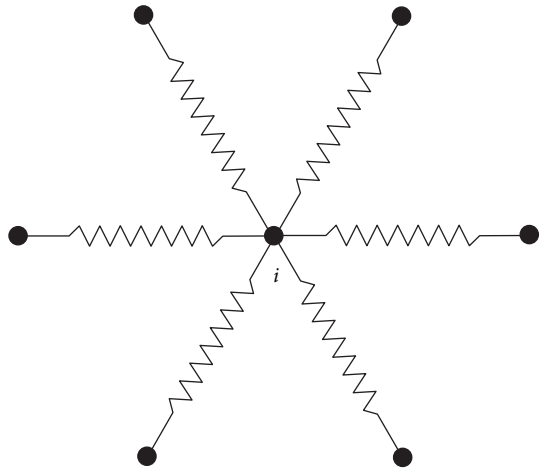


Figure 7: Spring analogy: springs replacing segments.

2.3.2. Geometrical Optimisation. The goal of this step is to modify the mesh without changes to the global data structure. This is achieved primarily by means of node displacement, based on spring analogy. However, other techniques must be applied to ensure a better handling of the node displacement. Node neighbours' number, for example, will be employed again for adjusting the spring stiffness. Particular care will be given to nodes lying on the bounding geometry and avoiding element inversion.

Spring Analogy. This technique has been heavily developed for moving mesh algorithms [17–19], for instance. We have adapted these concepts, to the present strategy of parallel mesh adaptation. Here each segment in the mesh is replaced by an elastic spring (Figure 7). The objective is then to minimise the deforming energy of the overall elastic system. This will result in the force **F** at node i obtained using Hooke's law:

$$\mathbf{F}_i = \sum_{j\in k(i)} \alpha_{ij}\left(\mathbf{x}_j - \mathbf{x}_i\right) = \mathbf{0}, \tag{15}$$

where $k(i)$ represents the set of node neighbours of vertex i, with size $\mathcal{N}_i$, and α_{ij} denotes the spring stiffness of the segment joining node i with neighbour j. Hence the equilibrium position at coordinates $\mathbf{x}_i$ can be expressed as

$$\mathbf{x}_i = \frac{\sum_{j\in k(i)} \alpha_{ij}\mathbf{x}_j}{\sum_{j\in k(i)} \alpha_{ij}}, \tag{16}$$

which can be resolved using a Jacobi iterative scheme.

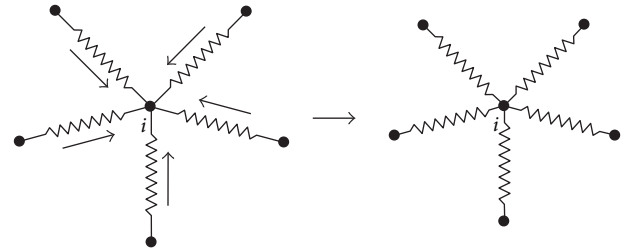


Figure 8: Springs' movement based on node neighbours: springs contracting.

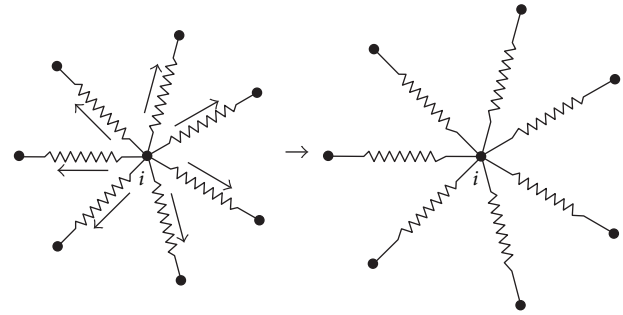


Figure 9: Springs' movement based on node neighbours: springs expanding.

Spring Stiffness. Node neighbours' number is once again very useful for mesh optimisation. In fact, if spring stiffness α_{ij} were to be set to one in order to produce equilateral elements, the following would occur:

(i) if $\mathcal{N}_i < \mathcal{N}_{\text{opt}}$, $k(i)$ move towards i (Figure 8);

(ii) if $\mathcal{N}_i > \mathcal{N}_{\text{opt}}$, $k(i)$ move away from i (Figure 9).

To partially avoid this problem, the following weight function can be used to determine the spring stiffness:

$$\alpha_{ij} = \alpha_j = \max\left[1, \mathcal{N}_{\text{opt}} + \mathcal{A}\left(\mathcal{N}_i - \mathcal{N}_{\text{opt}}\right)\right]. \tag{17}$$

This relates the spring stiffness to $\mathcal{N}_j$, the number of neighbours for the node $j \in k_i$. It also introduces the smoothing lineal factor $\mathcal{A}$, which is set manually.

Boundary Nodes. Nodes lying on the geometric boundaries have to be moved with caution (if moved at all). Whatever the method used for positioning the node on the underlying geometry, a sufficient node density must be guaranteed within critical regions where the boundary curvature is large. This can be achieved by maintaining boundary nodes with a new spring joining the reference point $\tilde{\mathbf{x}}$ and the new position $\mathbf{x}^{n+1}$. The stiffness β_i of this new spring is then chosen as a function of the local maximum boundary curvature. The resulting force is then calculated as

$$\mathbf{F}_i = \sum_{j\in k_\Gamma(i)} \alpha_j\left(\mathbf{x}_j - \mathbf{x}_i\right) + \beta_i(\tilde{\mathbf{x}}_i - \mathbf{x}_i) = \mathbf{0}, \tag{18}$$

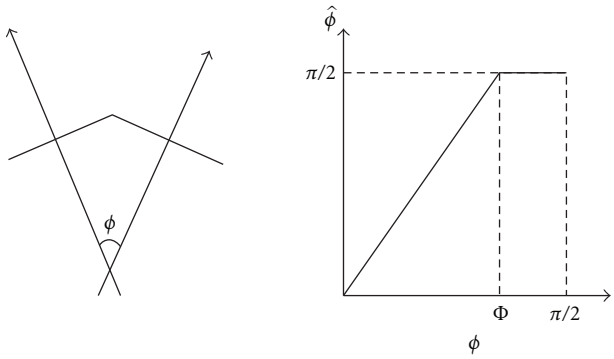


Figure 10: Curvature angle and filter.

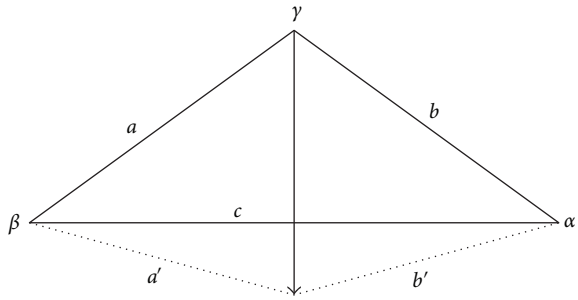


Figure 11: Inverted elements' *snap-through*.

where $k_\Gamma(i)$ represents the subset of $k(i)$ which contains all the node neighbours located on the boundary. The following formulation may then be obtained substituting $\tilde{\mathbf{x}}$ by $\mathbf{x}^n$:

$$\mathbf{x}_i^{n+1} = \frac{\sum_{j\in k_\Gamma(i)} \alpha_j \mathbf{x}_j^n + \beta_i \mathbf{x}_i^n}{\sum_{j\in k_\Gamma(i)} \alpha_j + \beta_i}. \tag{19}$$

The stiffness of the new spring is then defined as a function of the curvature angle ϕ. This angle is first filtered such that the node displacement is restricted, especially when it exceeds a given value Φ (Figure 10):

$$\beta_i = \mathcal{B}\left(\frac{1}{\cos^2\hat{\phi}} - 1\right) \quad \text{with } \mathcal{B} \geq 0, \tag{20}$$

where $\mathcal{B}$ is a user-defined boundary stiffness factor.

Inverted Elements or Torsional Springs. This is a major issue, [18], which needs to be controlled thoroughly, as it causes loss in overall volume mesh conservation. It may occur when a vertex crosses over the opposite face of the element, which inverts the cell volume. This phenomenon, called *snap-through*, is shown in Figure 11 and is prone to happening on the boundary when this moves. The configuration shown has a low energy as the springs a and b rotate. To remedy this, the segment spring analogy is used together with initially rigid mesh boundaries, then semitorsional springs are placed in the corner between adjacent edges, that is, the stiffness of segment c is divided by the angle between segments a and b. As the sum of the angles is equal to π, the stiffness is approximately unchanged if the triangle is equilateral. For deformed elements instead, the vertex angles that are closer to 0 or π become rigid.

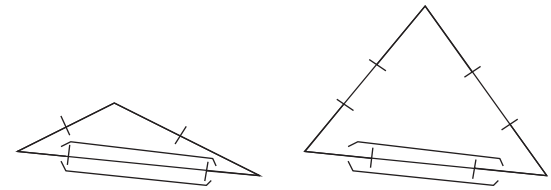


Figure 12: Cell inversion *stops*.

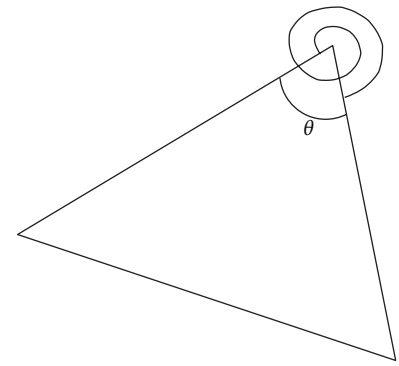


Figure 13: Torsional spring.

Cell inversion may also occur inside the heart of the mesh. A method to avoid this can be devised by setting critical cells rigid, with segment springs working in only one direction, rendering a relaxation of the elements. The vertex movement is then made free if it increases the element quality, which means that the introduced segment springs work like *stops* (Figure 12).

To determine the stiffness of the segment spring when it acts as a *stop*, the angular deformation energy of the cells is computed. A torsion spring is set at the opposite angle of each cell surrounding the segment (Figure 13):

$$\mathbf{C} = \mathcal{C}\left(\frac{1}{\sin^2\theta} - 1\right) \quad \text{if } \sin^2\theta < \sin^2\Theta, \tag{21}$$

where Θ is a filter value and $\mathcal{C}$ a user-defined torsion stiffness factor. It allows to take into account only the most critical angles. The maximum of the torsion spring for a given segment is then converted to a segment spring using the following relation:

$$\gamma = \frac{1}{\delta}\mathbf{C} = \frac{\mathcal{C}}{\delta}\left(\frac{1}{\sin^2\theta} - 1\right), \tag{22}$$

where δ is the distance between the segment and the opposite vertex in 2D and the opposite edge in 3D.

The nonisotropic behaviour of the *stops* causes the problem to be nonlinear. A time advancing strategy must be implemented, with the *stops* relaxing during the evolution of the procedure. The force applied on the node i is then given by

$$\mathbf{F}_i = \sum_{j\in k(i)} \alpha_j \left(\mathbf{x}_j - \mathbf{x}_i\right) + \sum_{j\in k(i)} \gamma_{ij}(\tilde{\mathbf{x}}_i - \mathbf{x}_i) = \mathbf{0}, \tag{23}$$

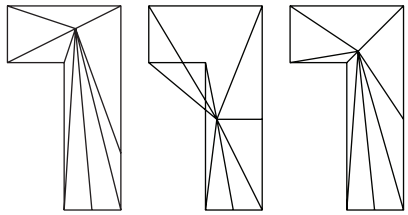


Figure 14: Torsion spring effect: (a) initial grid, (b) cell inversion, and (c) torsion spring.

which leads to the following formulation:

$$\mathbf{x}_i^{n+1} = \frac{\sum_{j\in k(i)} \alpha_j \mathbf{x}_j^n + \sum_{j\in k(i)} \gamma_{ij} \mathbf{x}_i^n}{\sum_{j\in k(i)} \alpha_j + \sum_{j\in k(i)} \gamma_{ij}}. \tag{24}$$

Finally the effect of the torsion spring is shown in Figure 14.

We note that smoothing and stretching algorithms are global, and hence the parallelisation of the these algorithms requires a global renumbering of all the grid nodes. This means that each node in the overlapping region will be assigned to the update set of a unique processor before the smoothing and stretching can be performed. Apart from the renumbering phase, the communication required to solve (16) by the Jacobi method is essentially the same of the parallel matrix-vector product outlined in [3].

2.4. Parallelisation of Grid Adaptation Techniques. In order to perform the grid adaption procedures on a parallel computer, two approaches can be followed. The first one is the *master-slave* approach, in which a processor is responsible for the management of the grid data (and in general of I/O routines). In [20, 21], the authors have discussed the limitations and demonstrate the relative performances of unstructured calculations using the master-slave approach. Once a preliminary solution has been obtained, it is gathered from the slave processors to the master processor, where sequential grid adaptation is started. Then, the grid is partitioned using a graph partitioning algorithm and redistributed among the processors. This approach works quite well for "small" grid, and in general steady-state problems. For evolutionary problems and/or intensive calculations, a *no-master* approach is required, in which the completely parallel dynamic grid adaptation algorithm takes place entirely on the network of processors. This is precisely the approach we have followed. More detailed description of the no-master approach may be found in [22].

The paradigm we have adopted is based upon the concept of nonhierarchical grid adaptation; that is, the successive grids do not remember their original affiliation; see [23, 24]. This allows high flexibility and quality for the different stages of adaptation as the grid at a certain time does not rely on the background macrogrid. Hence, radical changes and optimisation are possible. Also, efficient automatic dynamic adaptation, which is particularly interesting for following evolving or transitional phenomena, is facilitated. These concepts can also be developed for general element types when based on a concept of generalised elements consisting of the group of nearest neighbours called "shells," [16, 25]. This non-hierarchical technique, associated with an optimisation, produces similar grids to those obtained by the regriding. The drawbacks reside in the complexity of programming and the coherent reprojection on the geometry definitions defining the boundary surfaces.

Note that the incorporation of parallel grid adaptation within the solution process requires load balancing partitioning techniques to obtain well-balanced subdomains. This introduces other algorithmic concepts such as parallel sorting and renumbering techniques.

The grid adaptation techniques are applied globally throughout a prepartitioned mesh and require careful renumbering and reordering internally per processor (local) and globally of the addresses of the entities, cells, shells, faces, edges, nodes, and so forth, in order that the adaptation renders a global mesh that is in turn re-partitioned again. All this is dynamic and needs to have the partitioning procedure as an integral part of the adaptation procedure.

The parallelisation of the refinement leads to the tracking of nodes created on an updated segment which are considered as a new border (interface) node. For coarsening, the principle that when attempting to delete a border node, a border segment must be chosen was applied.

The parallelisation of the structural changes is one of the hardest points, especially for the choice of overlapping partitions. For these reasons the swapping and collapsing work most efficiently on the internal segments. However, diagonal swapping or face swapping is still straightforward across partition interfaces. Collapsing is often harder to control.

The smoothing procedures do not modify significantly the internal mesh topology; the parallelisation is hence straightforward as long as a coherent numbering of the nodes, segments, faces, and cells is employed.

2.5. Repartitioning. From the point of view of parallel computing, the grid adaptation procedure may result in an unbalanced distribution of the workload among the processors. Hence, the workload for each subdomain may be different, and this can produce an inefficient parallel performance. Effectively, the worker with the largest workload can delay the process. In fact, the starting domain decomposition was obtained to balance the workload for the initial grid (i.e., the same number of nodes on each subdomain and the minimum number of cut elements), whereas the adaptation algorithm could have generated many nodes on some subdomains (leading to more computing resources on the corresponding processors) and may have derefined in subdomains given to other processors (thus requiring less computational resources). Therefore, the computational domain is repartitioned dynamically within the parallel adaptation procedure using a parallel graph partitioning algorithm. For these purposes, the library ParMETIS [26] can be used dynamically within the source code, as well as home-made partitioners as in [22].

2.6. Reorder and Renumber. To complete the parallel adaptation procedure, fast efficient multiple renumbering techniques are necessary for the grid entities: elements, segments, faces, and nodes. For this MPI library routines are called explicitly and a fast dynamic binomial search tree to sort during renumbering procedures is implemented based on a balanced binomial search tree algorithm AVL (Adelson-Velskii and Landis) [27]. Consequently all the vectors and matrices used in the code may have to be reallocated in memory. In particular, the data structure for the parallel matrix-vector product must be recomputed.

An AVL tree is a dynamically balanced binary search tree that is height H balanced. Height balanced means that for every node in the tree, the heights of the left and right subtrees differ by at most one. The height of a tree is the number of nodes in the longest path from the root to any leaf. The implementation is as a recursive structure of interlinked nodes. The difference in height H between different branches is kept minimal by imposing that pairs of such subtrees of every node differ in height by at most 1. As it is a binary tree for n nodes, $H \leq n \leq 2^H - 1$ where the extrema correspond to a balanced tree.

When a new node is inserted into the tree, it appears at the root, then moves along the branches until it finds an attachment to the tree. Once the node is inserted, the tree balance is checked. If no imbalance is found, another node is inserted and the process continues. If an imbalance is found, the heights of some nodes are fixed and the process repeated. When a node is deleted, the root becomes unbalanced. The lookup is performed to balance again.

Lookup, insertion, and deletion operations are of $O(\log n)$, where n is the number of nodes in the tree, when the tree is balanced. Search steps $S(n)$ needed to find an item are bounded by $\log(n) \leq S(n) \leq n$.

3. Numerical Results

In order to assess the various functionalities of the techniques in place, a few test cases have been carried out in both two and three dimensions. For the two-dimensional case, the transonic flow over a NACA 0012 airfoil is used. For the three dimensional case we consider the supersonic flow over a forward wedge and different flow regimes for a concept aircraft. For all test cases, a parallel, unstructured grid, Euler solver THOR [3] was used.

3.1. NACA0012 Airfoil at $M_\infty = 0.80$ and $\alpha = 1°$. For this standard 2D test, the starting, nonadapted grid is composed of 2355 nodes. A first-order solution is computed on this grid; then, 4 steps of adaptation are performed, in order to improve the solution quality. The adaptation criteria are based on the density. Figure 15 shows the evolution of the grid, whose final size is of 3831 nodes and 7471 elements. The shock positions on the windward and leeward side are stabilised by the adaptation procedure. Note that the final grid is identical to the original one, except within the shock regions, where nodes have been added. The MURD scheme employed was a blended second-order scheme based on a strategic switch between the Lax-Wendroff and the PSI schemes (see [28] for further details).

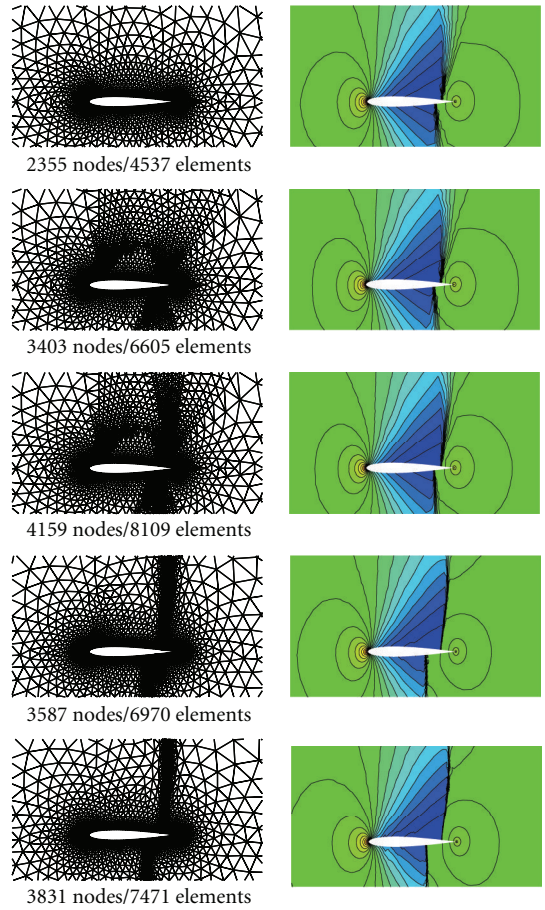


Figure 15: Evolution of the NACA0012 grid and the corresponding solution field (density) for $M_\infty = 0.80$ and $\alpha = 1°$.

3.2. Three Dimensional Forward Wedge at $M_\infty = 2.0$. In the second test case we present is a 3D forward wedge. We start from a rather coarse, hand-made grid, while the final adapted grid is composed by 80629 nodes and 480442 elements. The adaptive module is then used to refine the grid according to the physics of the solution field. This test case is interesting since despite its simple geometry, it presents shock reflections of different strength, which are to be captured by the adaptative procedure.

The successive grids and their corresponding solutions are presented in Figure 16. After each adaptation step, based on the gradient of the density, the number of nodes is roughly multiplied by a factor of 4.

The reasonably good quality of the last grid requires a large number of smoothing iterations. It is indeed essential to proceed carefully to avoid any element inversion. The solution scheme chosen is the standard N-scheme, which is a first-order scheme. Note that even if the starting grid is too coarse to permit an acceptable solution, adaptivity allows to obtain a solution which clearly captures the complex physics of the this problem.

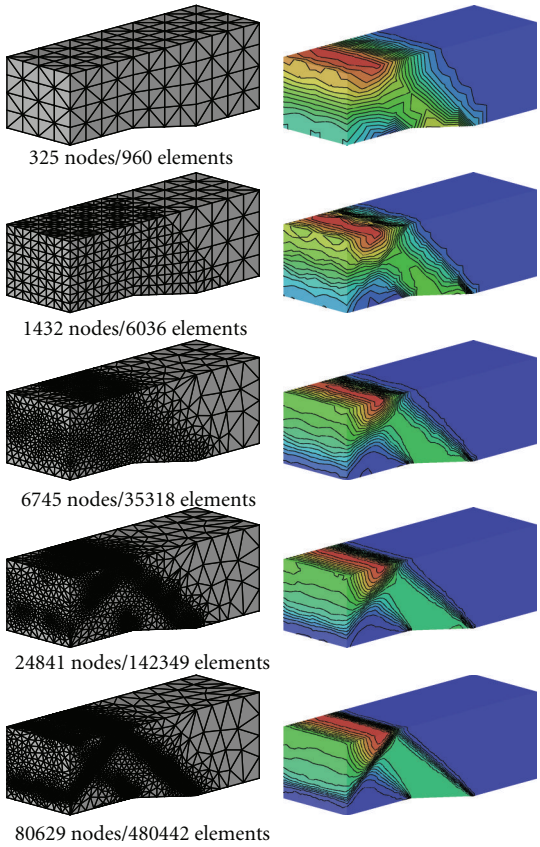


Figure 16: Evolution of the 3-dimensional wedge and the corresponding solution field (density) for $M_\infty = 2.0$.

3.3. Concept Aircraft. The second, three-dimensional test case is represented by a concept aircraft, Smartfish [29]. The interest of the geometry in this work is the extremely changing and complex form of the airplane, which poses a challenge for the grid generation and adaptation.

Here we present the results of some adaptations with different initial grid sizes and adapted with different physical criteria. The tests are carried out at transonic Mach numbers and with nonzero angles of attack. In particular we first test a very coarse grid for this type of problem, with 274 899 elements and 48 481 nodes. The first adaptation is done with respect to the change in gradient of the Mach number, with two adaptation cycles. The mesh is heavily refined (Figure 17) but only along the leading edge and not much over the wing.

Moving onto a denser initial grid (742 294 elements and 129 865 nodes), the difference in the Mach number along a segment is considered. Here the adaptation gives a better result (Figure 18), mainly because of the better solution to which it was adapted, due to the finer starting mesh.

Finally a relatively fine grid was used to start the process (1 772 861 elements and 314 913 nodes). The initial conditions are of Mach number 0.9 and angle of attack of 4°. The grid is refined well in the area of the shock, above and below, as shown in Figure 19.

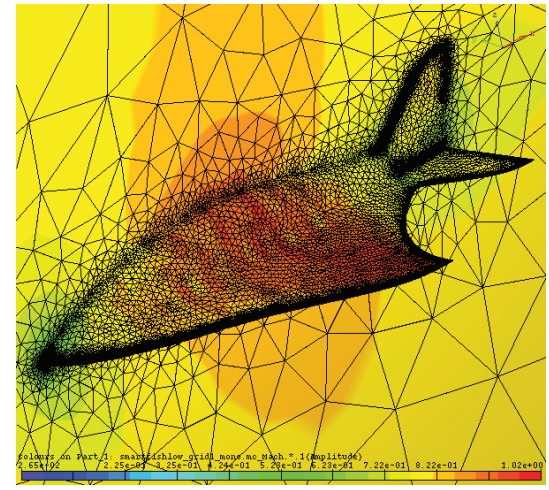


Figure 17: Adapted coarse grid with respect to the Mach gradient. 2 adaptation cycles only with refinement at the Mach number 0.8 and angle of attack 2°, 6 569 277 elements and 1 098 081 nodes.

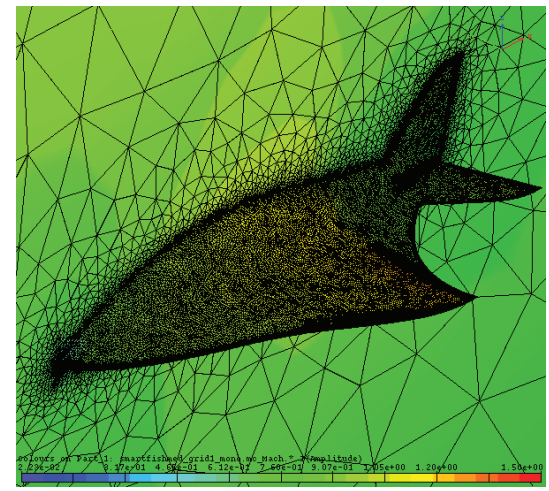
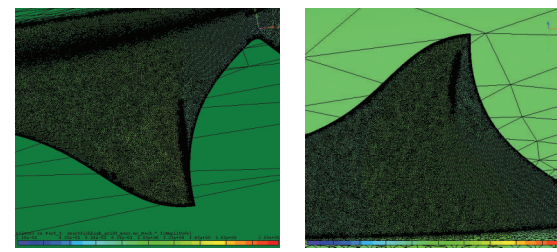


Figure 18: Adapted coarse grid with respect to the Mach difference. 1 adaptation cycle with refinement and derefinement at the Mach number 0.9 and angle of attack 4°, 1 795 794 elements and 302 723 nodes.



(a) (b)

Figure 19: Adapted finer grid with respect to the Mach difference. 4 adaptation cycles at the Mach number 0.9 and angle of attack 4°, 3 303 715 elements and 555 347 nodes. Upper side of the wing and lower side.

Table 1: Final grid sizes for different number of processors after 4 adaptation steps. 2D NACA 0012 airfoil.

Processors	N_nodes	N_elements	N_surf_el
1	35 130	69 615	645
2	35 082	69 527	637
4	34 335	68 031	639
8	34 839	69 035	643

Table 2: Final grid sizes for different number of processors after 3 adaptation steps and after 1 adaptation step for 64 and 128 restarting from 32 final solution and grid. 3D wedge.

Processors	N_nodes	N_elements	N_surf_el
8	1 118 350	6 721 070	50 714
16	1 123 326	6 753 716	50 963
32	1 130 577	6 801 332	51 073
64	3 843 209	23 302 721	85 652
128	3 842 653	23 290 638	85 641

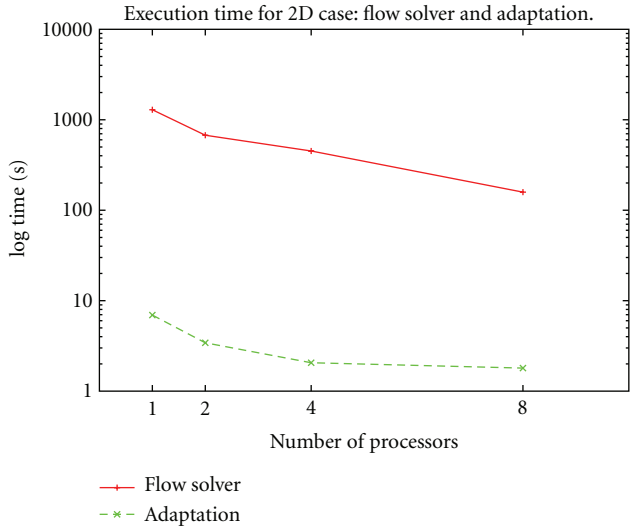


Figure 20: Execution time for flow solver and adaptation process on multiple processors. 2D NACA 0012, four adaptation steps.

4. Performance Aspects

In order to verify the performance of the adaptive procedures, some of the test cases presented in the previous sections have been re-run. First a 2D NACA 0012 airfoil is used to test the code on the Linux cluster (*Pleiades* [29]) used for the computations, as well as measuring the CFD code and the adaptation parts elapse time. In particular the nodes used for the computations reported here are biprocessor, bicore. Then various 3D test cases are used to measure the total time of the adaptation process with respect to the CFD time and the breakdown of the adaptation process stages.

4.1. 2D Results. In order to test the charge of the adaptation process, with respect to the total time of the CFD computation, a same NACA 0012 airfoil case was executed with a different number of processors. In particular the adaptation process was run with refinement and coarsening procedures, adaptation with respect to the Mach gradient, 20 optimisation cycles (swapping and collapsing), and 20 smoothing cycles. Four adaptation steps were carried out; hence from an initial grid of 2 355 nodes, 4 537 triangular elements, and 173 boundary faces, the flow solver is run and the solution adapted in turn four times, and a final solution obtained from the final adapted grid. The final grid characteristics for the computations with different number of processors are reported in Table 1. In Figure 20 instead we report the total computational execution time of the flow solver and that of the adaptation process. As we can notice, the total time of adaptation can be considered negligible compared to the solution time, being at most less than 1.2% of the total CFD computation time.

4.2. 3D Results. In a similar way to that of the 2D case above, we first compare the execution time of the flow solver and that of the adaptation with a different number of processors. The test is carried out with the 3D wedge example, with an initial mesh of 306 415 tetrahedral elements, 54 370 nodes, and 12 906 boundary faces. The adaptation process was run with refinement only, with respect to the difference in density on the segments, 30 optimisation passes, and 10 smoothing cycles. Three adaptation steps were performed, each one after obtaining a partial solution with the flow solver, and a final solution obtained from the final adapted grid. This was carried out for 8, 16, and 32 processors as shown in Table 2 with the final adapted grids characteristics. The final mesh and solution obtained with 32 processors was then used for a single adaptation step, using 64 and 128 processors. In this last case only one solution step is required, as for the adaptation, since the grid is immediately adapted to the solution obtained with the previous computation.

Here the computations start from 8 processors, rather than 1, due to memory requirements for the grid obtained after the third adaptation step. Figures 21 and 22 show the computational times plotted for the three adaptation steps and for the restart, on a different number of processors. Once

Table 3: Step-by-step grid sizes for different number of processors after 1, 2, and 3 adaptation steps, and after 1 adaptation step for 64 and 128 restarting from 32 final solution and grid. 3D wedge.

Steps	Processors	*N*_nodes	*N*_elements	*N*_surf_el
Step 1	4	140 180	811 918	20 902
	8	140 129	811 258	20 868
	16	140 133	810 941	20 915
	32	140 108	810 176	20 941
Step 2	4	362 718	2 145 897	31 844
	8	364 891	2 158 249	31 728
	16	365 409	2 161 629	31 911
	32	367 912	2 176 392	31 923
Step 3	8	1 118 350	6 721 070	50 714
	16	1 123 896	6 756 786	50 931
	32	1 131 649	6 806 732	50 949
Restart	64	3 843 363	23 310 788	85 648
	128	3 842 828	23 298 937	85 647

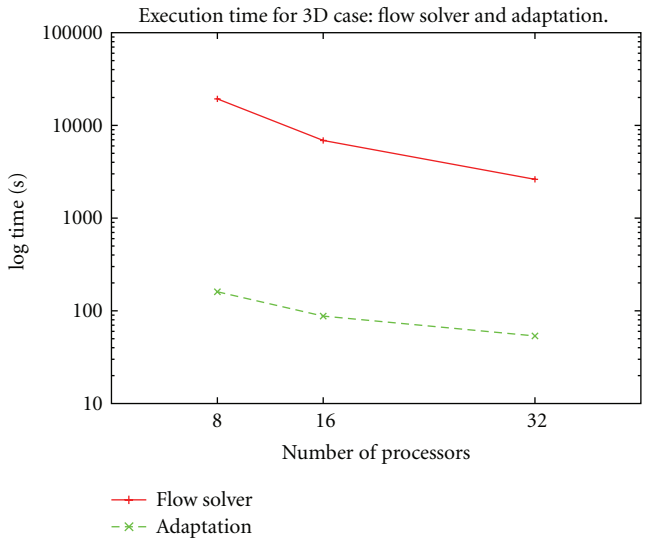


Figure 21: Execution time for flow solver and adaptation process on multiple processors. 3D wedge, three adaptation steps.

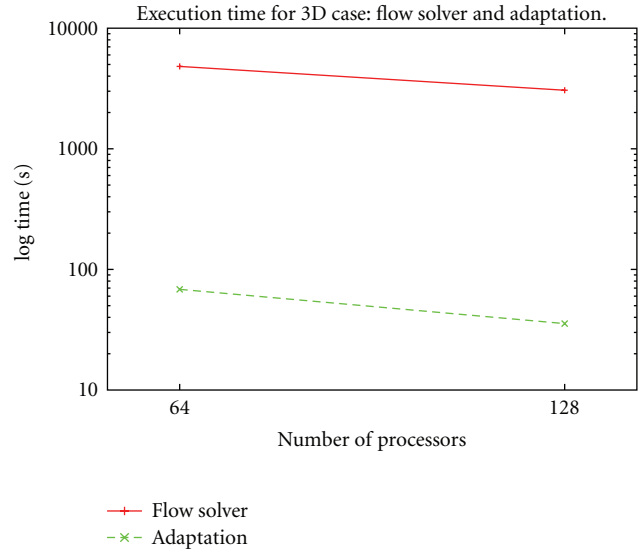


Figure 22: Execution time for flow solver and adaptation process on multiple processors. 3D wedge, one adaptation step after restart.

again the total adaptation time is negligible compared to that of the flow solver, reaching at most 2% of the total solution time.

4.3. Adaptation Execution Breakdown. Although the adaptation execution times are far less than the total computation, where the flow solver is accounted for, it is interesting to examine the various stages of the adaptation cycle and see the impact these have on the use of computational resources used. Therefore what follows is a breakdown of the adaptive cycle in three main blocks, with the refinement/derefinement and renumbering as a first block, swapping/collapsing and renumbering a second block, and smoothing being the third and last block.

The previous 3D wedge initial mesh was used to start a computation with two adaptation steps on 4 processors and a third adaptation step for 8, 16, and 32 processors. The reason for this choice is that it is not possible to run three adaptation steps on 4 processors, due to memory constraints. The final solution and mesh of this last computation were once again used as a starting point for an adaptation step carried out with 64 and 128 processors. Adaptation conditions were maintained the same as for the previous case. Grids for all steps and number of processors are given in Table 3. Figures 23 and 24 show the breakdown of the adaptation process time for the first two steps with multiple processors and Figure 25 shows that of the third step. Figure 26 instead shows the breakdown for the restarted case. As we can see from the above examples, the two optimisation procedures are far more time consuming than the refinement/derefinement block for the case where the difference physical criteria is chosen.

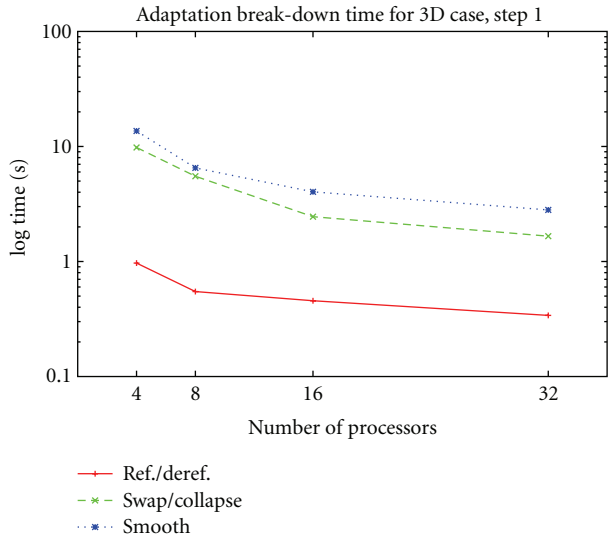


Figure 23: Execution time for adaptation blocks on multiple processors. 3D wedge, first adaptation step.

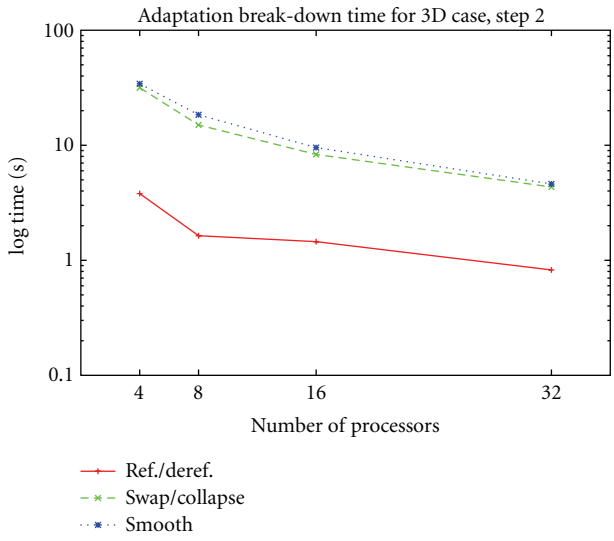


Figure 24: Execution time for adaptation blocks on multiple processors. 3D wedge, second adaptation step.

5. Conclusions

In this paper mesh adaptation techniques based on physical phenomena are developed in a parallel environment. The various steps (refinement, coarsening, optimisation, smoothing, reordering, and renumbering) and their algorithms are described. The techniques are validated on extensive complex flow simulations. The parallel adaptation performances show the efficiency of the implementation of these methods.

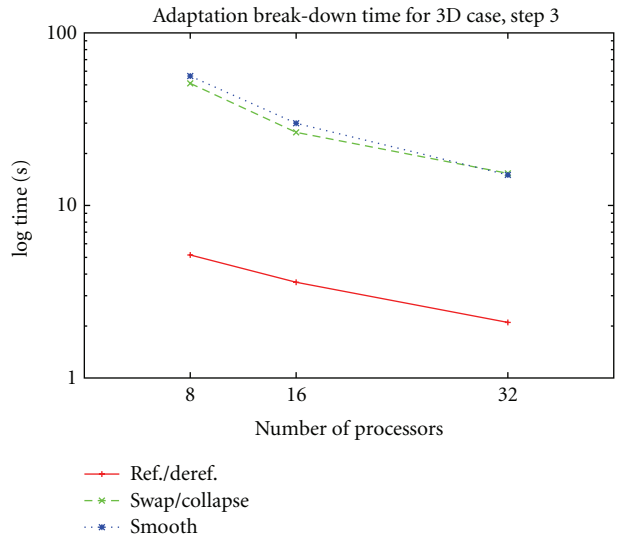


Figure 25: Execution time for adaptation blocks on multiple processors. 3D wedge, third adaptation step.

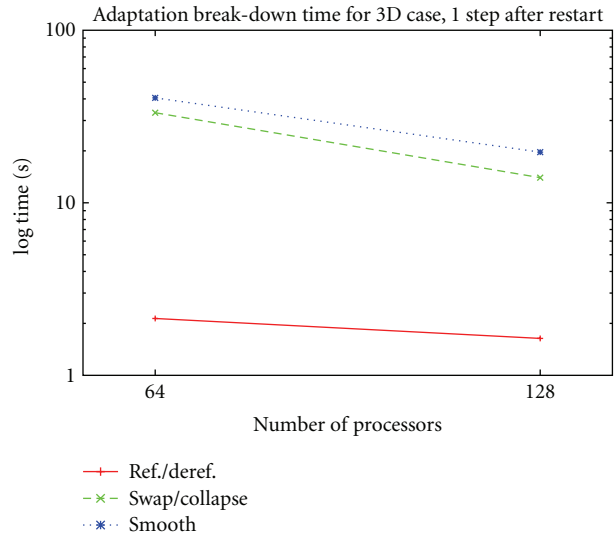


Figure 26: Execution time for adaptation blocks on multiple processors. 3D wedge, one adaptation step after restart.

Acknowledgments

The Swiss National Science Foundation (SNSF) and the Swiss Federal Office for Education and Science (OFES) are acknowledged for financial support.

References

[1] S. Z. Pirzadeh, "An adaptive unctructured grid method by grid subdivision, local remeshing, and grid movement," AIAA Paper 99-3255, 1999.

[2] R. Richter, *Schémas de Capture de Discontinuités en Maillage Non-Structuré avec Adaptation Dynamique: Applications aux écoulements de l'aérodynamique [Ph.D. thesis]*, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1993.

[3] M. Sala, P. Leyland, and A. Casagrande, "A parallel adaptive Newton-Krylov-Schwarz method for the 3D compressible flow simulations," *Modelling and Simulation in Engineering*. In press.

[4] K. Eriksson, C. Johnson, and J. Lennblad, "Error estimates and automatic time and space step control for linear parabolic problems," *SIAM Journal on Scientific Computing*, 1990.

[5] P. Leyland, F. Benkhaldoun, N. Maman, and B. Larrouturou, "Dynamical mesh adaptation criteria for accurate capturing of stiff phenomena in combustion," *International Journal of Numerical Methods in Heat and Mass Transfer*, 1993, (INRIA Report 1876).

[6] P. G. Ciarlet, "The finite element method for elliptic problems," *Classics in Applied Mathematics*, vol. 40, pp. 1–511, 2002.

[7] M. Sala, *Domain decomposition preconditioners: theoretical properties, application to the compressible euler equations, parallel aspects [Ph.D. thesis]*, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2003.

[8] B. Rivière, M. F. Wheeler, and V. Girault, "A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems," *SIAM Journal on Numerical Analysis*, vol. 39, no. 3, pp. 902–931, 2002.

[9] R. Hartmann and P. Houston, "Adaptive discontinuous Galerkin finite element methods for nonlinear hyperbolic conservation laws," *SIAM Journal on Scientific Computing*, vol. 24, no. 3, pp. 979–1004, 2003.

[10] W. G. Habashi, M. Fortin, J. Dompierre, M. G. Vallet, and Y. Bourgault, "Anisotropic mesh adaptation: a step towards a mesh-independent and user-independent cfd," in *Barriers and Challenges in Computational Fluid Dynamics*, pp. 99–117, Kluwer Academic, 1998.

[11] L. Formaggia and S. Perotto, "New anisotropic a priori error estimates," *Numerische Mathematik*, vol. 89, no. 4, pp. 641–667, 2001.

[12] T. Apel, *Anisotropic Finite Elements: Local Estimates and Applications. Advances in Numerical Mathematics*, Habilitationsschrift, Teubner, Germany, 1999.

[13] D. J. Mavriplis, "Unstructured mesh generation and adaptivity," Tech. Rep. TR-95-26, ICASE-NASA, 1995.

[14] G. Warren, W. K. Anderson, J. L. Thomas, and S. L. Krist, "Grid convergence for adaptive methods," AIAA Paper 91-1592, 1991.

[15] Y. Savoy and P. Leyland, "Adaptive module," Tech. Rep. TR5.1, IDeMAS, 2000.

[16] G. F. Carey, *Computational Grids: Generation, Adaptation and Solution Strategies*, Taylor & Francis, 1997.

[17] J. T. Batina, "Unsteady Euler airfoil solutions using unstructured dynamic meshes," *AIAA Journal*, vol. 28, no. 8, pp. 1381–1388, 1990.

[18] C. Degand and C. Farhat, "A three-dimensional torsional spring analogy method for unstructured dynamic meshes," *Computers and Structures*, vol. 80, no. 3-4, pp. 305–316, 2002.

[19] F. J. Blom and P. Leyland, "Analysis of fiuid-structure interaction by means of dynamic unstructured meshes," *Journal of Fluids Engineering*, vol. 120, no. 4, pp. 792–798, 1998.

[20] R. Richter and P. Leyland, "Distributed CFD using auto-adaptive finite element," in *ICASE/LaRC Workshop on Adaptive Grid Methods*, 1994.

[21] D. J. Mavriplis, "Three-dimension high-lift analysis using a parallel unstructured multigrid solver," Tech. Rep. TR-98-20, ICASE, 1998.

[22] P. Leyland and R. Richter, "Completely parallel compressible flow simulations using adaptive unstructured meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 184, no. 2–4, pp. 467–483, 2000.

[23] R. Richter, *Schémas de Capture de Discontinuités en Maillage Non-Structuré avec Adaptation Dynamique: Applications aux écoulements de l'aérodynamique*, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1993.

[24] R. Richter and P. Leyland, "Entropy correcting schemes and non-hierarchical auto-adaptive dynamic finite element-type meshes: applications to unsteady aerodynamics," *International Journal for Numerical Methods in Fluids*, vol. 20, no. 8-9, pp. 853–868, 1995.

[25] Y. Savoy and P. Leyland, "Parallel mesh adaptation for unstructured grids within the IDeMas project," Tech. Rep., IMHEF-DGM EPFL, 2000.

[26] G. Karypis and V. Kumar, "ParMETIS: parallel graph partitioning and sparse matrix ordering library," Tech. Rep. 97-060, Department of Computer Science, University of Minnesota, 1997.

[27] M. A. Weiss, *Data Structure and Algorithm Analysis*, The Benjamin Cummings Publishing Company, 1992.

[28] J. Bastin and G. Rogé, "A multidimensional fluctuation splitting scheme for the three dimensional Euler equations," *Mathematical Modelling and Numerical Analysis*, vol. 33, no. 6, pp. 1241–1259, 1999.

[29] R. Gruber and V. Keller, *HPC@ Green IT: Green High Performance Computing Methods*, Springer, 2010.