Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2013, Article ID 694354, 16 pages
http://dx.doi.org/10.1155/2013/694354

Research Article

A Parallel Adaptive Newton-Krylov-Schwarz Method for 3D Compressible Inviscid Flow Simulations

Marzio Sala,[1] Pénélope Leyland,[2] and Angelo Casagrande[2]

[1] UBS Investment Bank, 8098 Zurich, Switzerland
[2] EPFL STI GR-SCI-IAG, Station 9, 1015 Lausanne, Switzerland

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

Received 27 April 2012; Accepted 19 December 2012

Academic Editor: MuDer Jeng

A parallel adaptive pseudo transient Newton-Krylov-Schwarz ($\alpha\Psi$NKS) method for the solution of compressible flows is presented. Multidimensional upwind residual distribution schemes are used for space discretisation, while an implicit time-marching scheme is employed for the discretisation of the (pseudo)time derivative. The linear system arising from the Newton method applied to the resulting nonlinear system is solved by the means of Krylov iterations with Schwarz-type preconditioners. A scalable and efficient data structure for the $\alpha\Psi$NKS procedure is presented. The main computational kernels are considered, and an extensive analysis is reported to compare the Krylov accelerators, the preconditioning techniques. Results, obtained on a distributed memory computer, are presented for 2D and 3D problems of aeronautical interest on unstructured grids.

1. Introduction

The aim of this paper is to provide an overview of the methods required for an efficient parallel solution of the compressible Euler equations (CEE) on unstructured 2D and 3D grids. The ingredients include space and time discretisation schemes for the underlying partial differential equations (PDEs), a nonlinear solver based on the Newton's method, and a parallel Krylov accelerator with domain decomposition preconditioners.

Nowadays, unstructured grids are of particular interest for industrial applications. With respect to structured grids, it is often easier to produce unstructured grids of good quality in domains of complex shape, especially if the unstructured grid generator can be coupled to a CAD system. This can provide—at least in principle—a fast process to solve the problem at hand, with minimal intervention of the user, once the geometry of the domain and the boundary conditions have been specified. However, such techniques imply an almost perfect geometric representation (CAD level) and automatic grid generation remains a challenge in computational engineering, especially for moving boundaries.

The framework presented here can be successfully applied to the solution of sets of PDEs problems discretised on unstructured grids. The focus in this paper is restricted to the solution of the CEE. In particular, the space discretisation technique considered is based on the so-called *multidimensional upwind residual distribution* (MURD) schemes; see for instance [1–5]. These schemes are supposed to render higher accuracy and less spurious numerical anomalies. This is due to the fact that they take into account the multidirectionality of the wave structure, rather than being based upon a summation of one-dimensional Riemann problems, [6, 7]. Firstly developed for a scalar advection equation, MURD schemes have been extended to homogeneous flux hyperbolic systems such as the Euler equations. They can be interpreted as Petrov-Galerkin finite element methods, with compact-stencil basis functions. This turns out to be of particular advantage for their parallel implementation, since both first-order and second-order in space schemes share the same communication pattern.

For compressible flow simulations, we will consider here the Euler equations. The steady state solutions are found as the steady-state of their time-dependent version. The time derivative is discretised using the backward Euler method.

This leads to a nonlinear system to be solved at each time step. Hence, for these cases, time accuracy is not an issue. In fact, the pseudo-time derivative is added as a mean to obtain a "good" initial iterate for the Newton's method (which can even diverge for initial iterates too far-away from the solution). Moreover, only few steps of the Newton's method are considered, and the linear system with the Jacobian matrix is solved only approximatively.

The Newton's method for the solution of this nonlinear system results in a large, ill-conditioned linear system with the system's Jacobian matrix. Direct methods appear to be too computationally expensive, especially in 3D. For our kind of applications, preconditioned iterative solvers of Krylov type are a prime choice and can also be associated with multigrid techniques, another challenging method, especially for compressible flow simulations. A possible way to derive the preconditioner is to adopt a domain decomposition approach [8, 9], and more in particular to use a Schwarz preconditioner. This means that the computational domain Ω is decomposed into M overlapping subdomains Ω_i, $i = 1, \ldots, M$, such that $\bigcup_{i=1}^{M} \overline{\Omega_i} = \overline{\Omega}$. Then, a Dirichlet problem is solved in each subdomain. Since typically each subdomain is assigned to a different processor, this leads to a parallel preconditioner. Further, a coarse level is added, resulting in an overall—although cheap—communication among the subdomains. Here, to construct the coarse level, we consider algebraic procedures based on the concept of aggregation.

The resulting framework is usually addressed to as the Newton-Krylov-Schwarz framework, a quite generic procedure that can be applied to nonlinear systems of PDEs [10–16]. The idea is to obtain (asymptotically) a second-order convergence in time through the Newton iterations and a reasonable parallelism thanks to the use of a Krylov solver complemented with a Schwarz-type preconditioner. The resulting algorithm will be addressed to as $\alpha\Psi$NKS.

For an efficient and scalable implementation of $\alpha\Psi$NKS methods, parallel codes have to be developed, extending their sequential counterparts. Data, like grid structures, the Jacobian matrix, state variable, and work vectors, must be partitioned among the processors in order to minimise the amount of intraprocessor communications. The definition of a distributed data structure, wellsuited for all the computational phases, is the first problem to be addressed. To that aim, all the main computational kernels of the $\alpha\Psi$NKS algorithm are analysed, and suitable algorithms are presented and implemented in order to make use of as much local data as possible, thus minimising communication. Note that robust and accurate methods are necessary to treat the large variety of regimes which may typically arise in aeronautics which rely on solving the Euler equations. The interest of these techniques is that many software packages, such as PetSc and ISL++, offer these functionalities in a user-friendly way.

The paper is organised as follows. Section 2 describes the numerical solution of the compressible Euler equations and the $\alpha\Psi$NKS scheme. In Section 3 the data structure used for parallelisation is introduced. Section 4 outlines the main computational kernels of the $\alpha\Psi$NKS scheme. Section 5 gives some numerical results obtained on a SGI Origin 3800 for academic test cases as well as typical full aircraft problems. Although this machine is nowadays replaced by Linux clusters, the scaling of the results for multiple processor clusters is significant. Indeed, all techniques and results described here scale in exactly the same way on Linux clusters under MPI. Finally, Section 6 outlines the conclusions.

2. Numerical Solution of the Compressible Euler Equations

The compressible Euler equations describing nonviscous and nonheat conducting compressible fluid flows in a domain $\Omega \subset \mathbb{R}^3$ can be written as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{F}_1(\mathbf{w})}{\partial x_1} + \frac{\partial \mathbf{F}_2(\mathbf{w})}{\partial x_2} + \frac{\partial \mathbf{F}_3(\mathbf{w})}{\partial x_3} = \mathbf{0}, \quad (1)$$

where $\mathbf{w} = (\rho, \rho u_1, \rho u_2, \rho u_3, \rho e)$ is the vector of conservative variables. For the applications of interest, (1) are posed on a bounded domain and completed by appropriate boundary conditions and initial condition $\mathbf{w}(t = 0) = \mathbf{w}_0$ (see for instance [17]).

In (1), ρ is the density, u_1, u_2, and u_3 are the x_1-, x_2-, and x_3-components of the velocity vector $\mathbf{v}$, and e is the specific total energy. The quantities $\mathbf{F}_i = (\rho u_i,\ \rho\mathbf{v} + \delta_{i,j} p,\ \rho u_i h)$, $i \in \{1, 2, 3\}$ are homogeneous functions of $\mathbf{w}$, called fluxes. Here $\delta_{i,j}$ is the Kronecker symbol, p is the (static) pressure, and h is the specific enthalpy. The gas is considered to be a calorically perfect gas [18, Chapter 2], therefore

$$e(T) = C_v T, \qquad h(T) = C_p T, \quad (2)$$

with C_v and C_p being the specific heat coefficient at constant volume and constant pressure, respectively.

The space discretisation of (1) is conducted using MURD schemes, which are based on the quasilinear form as

$$\frac{\partial \mathbf{w}}{\partial t} + G_1(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_1} + G_2(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_2} + G_3(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_3} = \mathbf{0}, \quad (3)$$

with

$$G_i(\mathbf{w}) = \frac{\partial \mathbf{F}_i}{\partial \mathbf{w}}. \quad (4)$$

The basis idea of MURD schemes is to compute the *residual* at element level, defined as

$$\mathbf{R}^{(T)}(\mathbf{w}) = \int_T \left\{ G_1(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_1} + G_2(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_2} + G_3(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial x_3} \right\} dT. \quad (5)$$

Then, the main idea is to redistribute this quantity to the nodes of the element, according to the choice of the scheme.

We will not describe MURD schemes, referring the reader to, for example, [19–24] and the references therein. Here, we just give a very broad overview of the schemes.

At their beginning, MURD schemes have been developed for scalar equations. The extension to system schemes is straightforward if the system is diagonalisable. In this case, scalar schemes can be used for each scalar equation of the diagonalised system. For nondiagonalisable systems, like the CEE, three main orientations are identified. One consists in the formal extension of the scalar scheme. This leads, for instance, to the system N-scheme [4]. A second orientation was introduced by Roe in 1986 [25] and consists in decomposing the initial residual as a sum of simple wave solutions. The last one, initially proposed by Deconinck and coworkers, see [26], is an elliptic/hyperbolic splitting, which decompose the CEE into an acoustic subsystem of three equations (for 3D problems), plus two advection equations, one of them being the entropy advection. The last approach is followed here. It was shown in [27] that this decomposition can lead to numerical methods with minimal spurious entropy production.

MURD schemes are targeted to the solution of advection-dominated systems on unstructured grids because they are not constructed by concentrating on any particular direction of the grid. An advantage is that, at least for scalar equations, one can construct a fully second-order accurate scheme on triangular grids with a very compact stencil: only the nodes in the triangle are used in the evaluation of the fluctuation. Note that these schemes can be interpreted as Petrov-Galerkin finite-element schemes, see [28, Section 4.1].

The spatial discretisation of (1) results in a system of nonlinear equations of type

$$S\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{6}$$

with a convenient initial condition $\mathbf{U}(t = 0) = \mathbf{U}_0$. In (6), S is a nonsingular (lumped) mass matrix. In particular, we seek for the the steady-state solution of system (6), which will be one of the roots of

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}. \tag{7}$$

Note that the Newton method applied directly to system (7) does not suffice, since an initial iterate sufficiently near to the root is generally not available. This is particularly true for complex flows containing shock waves and contact discontinuities, like the ones considered in this paper. Framing the steady-state equation into a time-dependent setting constitutes a sort of continuation process, which can be considered as an attempt to widen the domain of convergence of Newton method, or as a procedure to obtain sufficiently close starting points.

System (6) is discretised using a backward Euler scheme. Starting from a given $\mathbf{U}_0$, the solution at the pseudotime step $k = 1, 2, \ldots$ is found by solving the nonlinear system of equations

$$S\frac{\mathbf{U}_k - \mathbf{U}_{k-1}}{\delta_k} + \mathbf{R}(\mathbf{U}_k) = \mathbf{0} \tag{8}$$

or, equivalently,

$$S\delta_k^{-1}\mathbf{U}_k - S\delta_k^{-1}\mathbf{U}_{k-1} + \mathbf{R}(\mathbf{U}_k) = \mathbf{0}, \tag{9}$$

where δ_k is the time increment at step k.

A strategy for the definition of δ_k is needed. As a general rule, one should keep the time step small until all the main flow features are well resolved, then large time steps may be taken near to obtain superlinear or quadratic convergence of Newton's method. In this paper we have adopted the so-called exponential rule [29], that is,

$$\delta_k = \min\left\{\delta_0 \times (\sigma)^k, \mathrm{CFL}_{\max}\right\}, \tag{10}$$

where δ_0 is the initial time step and $\sigma > 1$ is a prescribed growing factor. Other strategies have been proposed in the literature, like the so-called switched evolution relation rule (see [29]) or the expert rule (see [30]).

A Newton iteration at time level k for the solution of (9) would then read

$$\begin{aligned}\mathbf{U}_{k,l} = \mathbf{U}_{k,l-1} &+ \left[S\delta_k^{-1} + \frac{\partial\mathbf{R}}{\partial\mathbf{U}}(\mathbf{U}_{k,l-1})\right]^{-1} \\ &\times \left(-S\delta_k^{-1}\mathbf{U}_{k,l-1} - \mathbf{R}(\mathbf{U}_{k,l-1}) + S\delta_k^{-1}\mathbf{U}_{k-1}\right)\end{aligned} \tag{11}$$

with $l = 0, 1, \ldots$ being the index associated to the Newton procedure, and $\mathbf{U}_{k,0} = \mathbf{U}_{k-1}$. The Newton iteration should stop when either a certain tolerance level is reached, or after a fixed number of iterations. Although it is possible in principle to take more Newton correction iterates, convergence results from [31] show that, for steady-state computations, quadratic convergence can be eventually achieved using only one step of process (11), provided that a suitable time-step evolution strategy for δ_k is found. Therefore, in the following we are supposed to take just one Newton iteration. This leads to the following linear problem:

$$\mathbf{U}_k = \mathbf{U}_{k-1} + \left[S\delta_k^{-1} + \frac{\partial\mathbf{R}}{\partial\mathbf{U}}(\mathbf{U}_{k-1})\right]^{-1}(-\mathbf{R}(\mathbf{U}_{k-1})) \tag{12}$$

that can be written, dropping the index k, as

$$A\mathbf{u} = \mathbf{f}, \tag{13}$$

where $A \in \mathbb{R}^{n\times n}$ and $\mathbf{u}, \mathbf{f} \in \mathbb{R}^n$. More precisely,

$$\begin{gathered}A = \left[S\delta_k^{-1} + \frac{\partial\mathbf{R}}{\partial\mathbf{U}}(\mathbf{U}_{k-1})\right], \\ \mathbf{u} = (\mathbf{U}_k - \mathbf{U}_{k-1}), \\ \mathbf{f} = -\mathbf{R}(\mathbf{U}_{k-1}).\end{gathered} \tag{14}$$

A preliminary analysis of the memory requirements and the computational time required by for direct *LU* techniques suggests the use of an iterative solver of Krylov type for the solution of (13).

The problem with iterative methods is that their convergence rate depends on the spectral properties of the coefficient matrix, in particular the condition number $\kappa(A) = \|A\| \cdot \|A^{-1}\|$, where $\|\cdot\|$ is a matrix norm. To obtain a more favourable condition number, one can solve the modified system $P^{-1}A\mathbf{x} = P^{-1}\mathbf{f}$, where P is a nonsingular matrix. P should approximate A^{-1} as closely as possible, so that

$\kappa(P^{-1}A) \ll \kappa(A)$, while still being reasonably cheap to compute both in memory and CPU time requirements. Also, it should be *optimal,* that is, $\kappa(P^{-1}A)$ does not depend on the problem dimension, and *scalable,* that is, $\kappa(P^{-1}A)$ does not depend on the number of processor used. Indeed, one of the main aims of a preconditioner is to transform the system into a system such that the matrix $P^{-1}A$ is close to an identity matrix.

Many parallel preconditioners have been proposed in the literature. Here, we have resorted to preconditioners based on the *domain decomposition* (DD) approach. The basic idea of DD methods is to decompose the computational domain Ω into M smaller parts Ω_i, $i = 1, \dots, M$, called subdomains, such that $\bigcup_{i=1}^{M} \overline{\Omega_i} = \overline{\Omega}$. Next, the original problem can be reformulated within each subdomain Ω_i, of smaller size. This family of subproblems is coupled one to the other through the values of the unknown solution at subdomain interface. This coupling is then removed at the expense of introducing an iterative process which involves, at each step, solutions on the Ω_i with additional interface conditions on $\partial\Omega_i \setminus \partial\Omega$.

We will focus on overlapping domain decomposition methods, also known as *Schwarz methods*. In these methods, the computational domain is subdivided into *overlapping* subdomains, and local Dirichlet-type problems are then solved on each subdomain. The communication between the solutions on the different subdomains is here guaranteed by the overlapping region.

These DD methods are usually rather inefficient when used as solvers of the linear problem; however, they can be reformulated as efficient parallel preconditioners. To express the preconditioner, we need to fix some notation. Let $R_i : \Omega \to \Omega_i$ be a rectangular matrix which returns a vector defined on the nodes internal to Ω_i from a global vector defined on Ω, $A_i = R_i A R_i^T$ the local matrix, and finally $B_i = R_i^T A_i^{-1} R_i$ the correction on subdomain Ω_i. Using this notation, the one-level (additive) Schwarz method can be written as

$$P_S^{-1} = \sum_{i=1}^{M} B_i. \tag{15}$$

This preconditioner is particularly well suited for the class of problems considered here. It is quite easy to implement and has reduced memory requirements. However, since P_S acts only locally, its scalability is hindered by the weak coupling between far away subdomains. Its performances degrade rapidly as the number of subdomains increases.

This behaviour is typical of all one-level preconditioners, which, by construction, are composed only by local corrections on the subdomains. In fact, all the state-of-art DD preconditioners consist of local correction and a global components. The local part, acting at the subdomain level, captures the strong couplings that appear between neighbouring subdomains, while the global part provide an overall—although inexpensive—communication among the subdomains. The global component is usually referred to as a "coarse space correction," since it is usually defined on a space that is coarse with respect to the fine space containing the solution. The complexity of this auxiliary problem is much lower than that of the original problem, and its role is to diffuse information among the subdomains. In an analogous manner to multigrid methods, this coarse space is used to correct the "smooth" part of the error, whereas the local preconditioner is used to dump the "high-frequency" part of the error.

In all generality, the global correction term has the form $B_0 = R_0^T A_0^{-1} R_0$, where A_0 corresponds to the discretisation of the original PDE problem on a coarse space V_0, and R_0 is the restriction operator from the fine space to the coarse space. The resulting preconditioner reads as

$$P_{C,\text{add}}^{-1} = B_0 + P_S^{-1} = \sum_{i=0}^{M} B_i. \tag{16}$$

This preconditioner is fully additive since all the corrections on the subdomains and on the coarse space are added together. Alternatively, an hybrid preconditioner can be used as

$$P_{C,\text{hybrid}}^{-1} = P_S^{-1} + B_0 - P_S^{-1} A B_0. \tag{17}$$

Roughly speaking, preconditioner (17) is called hybrid because the corrections on the subdomains are treated in an additive way (the term $P_S^{-1} + B_0$), as well as in a multiplicative way (the term $P_S^{-1} A B_0$). For more details, the reader is referred to [9].

The key element to obtain a scalable and efficient preconditioner is the proper definition of the coarse space. The general approach is to discretise the original problem on a coarse grid. However, the construction of the coarse grid and of the corresponding restriction operator R_0 can be difficult or computationally expensive for problems defined on unstructured grids in domain of complex shape, as typical in aeronautical applications. For this reason, here we consider agglomeration procedures to construct the coarse matrix for a two-level Schwarz preconditioner. The procedure is completely algebraic and very wellsuited for unstructured grids.

We now detail the procedure used to define the coarse space V_0, referring the reader for more details to [28, 32]. Our approach is based on the concept of *aggregation* [33–35]. On each subdomain Ω_i we build $n_{0,i}$ *aggregates* $\vartheta_{i,s}$, $i = 1, \dots, M$, $s = 1, \dots, n_{0,i}$. An aggregate is a set of contiguous vertices, see a 2D examples in Figure 1. The value $n_0 = \sum_{i=1}^{M} n_{0,i}$ represents the dimension of the coarse space. This means that we split the degrees of freedom n_0 among the M subdomains, for example, simply setting $n_{0,i} = n_0/M$. Finally, inside Ω_i, each aggregate is associated to a vector $\boldsymbol{\eta}_{s,i}$, $s = 1, \dots, n_{0,i}$, whose elements are built following the rule

$$\boldsymbol{\eta}_{s,i}(k) = \begin{cases} 1 & \text{if } k \text{ belongs to } \vartheta_{s,i} \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

The algorithm uses a simple procedure to build the operators R_0 and R_0^T and to form the coarse matrix as

$$A_0 = R_0 A R_0^T. \tag{19}$$

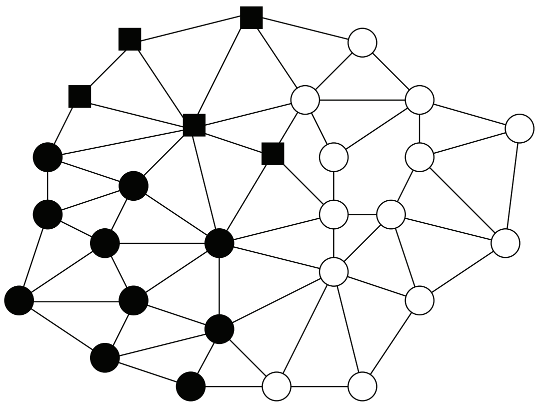

Figure 1: Examples of aggregates for a 2D domain. The nodes marked with "■," "○", and "●" belong to three different aggregates.

For a comparison between the definition of the coarse space using a coarse grid and using aggregation, the reader is referred to [36], or [28, Chapter 3], where the authors compare the two approaches for a model problem.

3. Definition of the Distributed Data Structure

The development of a scalable parallel $\alpha\Psi$NKS code requires the definition of an efficient distributed data structure. This data structure must efficiently handle the preconditioning phase, which is probably the most important aspect of the $\alpha\Psi$NKS solver, as well as all the other computational kernels of the scheme.

Throughout our description, we make the following assumptions.

(i) The parallel computer is of multiple instruction multiple data (MIMD) architecture.

(ii) The parallel computer is assumed to use distributed memory architecture. In fact, this is not a restriction at all since it can be emulated on almost every type of parallel computers including shared memory computers.

(iii) Communications are achieved using a message passing interface, like MPI [37]. This is not a limitation since message passing libraries are currently available on almost every type of architecture.

We also suppose that the starting grid, of moderate size, say, up to a million of elements for an inviscid solution, has been generated on a sequential computer. More refined grids will be obtained by the means of parallel adaptation techniques, as described in paper [38].

The first step is the partition of the grid among the M processors. To that aim, we have used the *k-way* graph partitioning algorithms [39]. Given a graph $G = (V, E)$ with $|V| = n$, we partition V into k subsets $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \emptyset$, $|V_i| = n/k$ on average, and $\bigcup_i V_i = V$, and the number of edges of E whose incident vertexes belong to different subsets is minimised. A k-way partition is commonly represented by a partition vector $\mathbf{p}$ of length n, such that for every vertex $v \in V$, $P[v]$ is an integer between 1 and k, indicating the partition to which vertex v belongs to. Given a partition P, the number of edges whose incident vertexes belong to different subsets i is called the *edge-cut* of the partition.

The k-way algorithm can be used to decompose the direct graph or the dual graph of the grid, leading to vertex-oriented (VO) or element-oriented (EO) decompositions. In the former, each vertex of the grid is assigned to a different processor, while in the latter the decomposition is at element wise. A numerical comparison between the two approaches for the compressible Euler equations using explicit time-marching scheme can be found in [40], while results concerning implicit time-marching schemes can be found in [41].

We have followed a VO decomposition. As a result of the VO decomposition, each vertex of the computational grid is assigned to a unique processor, whereas there exist elements shared by more processors. These elements are called *cut elements*—since their edges have been intersected—and they form a layer of elements that actually belong to more than one processor. The cut elements are stored on all processors which have at least one node of the element in their update set. For example, in Figure 2 the shaded elements are stored on both processor P_1 and processor P_2. The nodes assigned to processor i form its *update* set, whose size is n_i. Clearly, $\sum_{i=1}^{M} n_i = n$, n being the global dimension of the problem. The nodes belonging to the subdomain, but not to the update set, are called *external* nodes. The size of this set is $n_{E,i}$.

From the point of view of the parallel application, grid generation and graph partitioning result in M input files [40]; each input file contains a list of elements (grid connectivity and spatial coordinates of each vertex) and a list of the (physical) boundary nodes, grouped in the so-called *patches*. Each patch corresponds to a part of the boundary, like, for instance, the nacelle, the wing, or the symmetry plane, on which the same boundary conditions are imposed. The grid is represented in terms of element-based data structure. Also, a local-to-global mapping must be provided, to enable communication among the processors.

After convergence of the $\alpha\Psi$NKS solver, each processor writes the solution corresponding to its update set as a part of the global data. Often, these partitioned data are

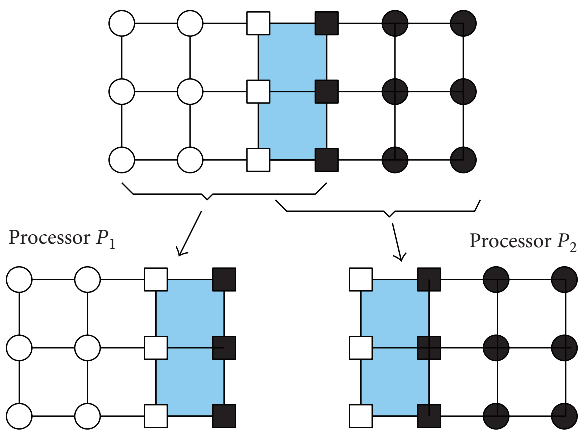

FIGURE 2: Vertex-oriented decomposition. Example of the overlapping stripe and respective mapping onto processors p and q, update and external nodes. For processor P_1, "○" represents the internal nodes, "□" the border nodes, and "■" the external nodes. For processor P_2, "●" represents the internal nodes, "■" the border nodes, and "□" the external nodes.

merged into a single file for global solution analysis and visualisation. However, certain packages also work directly on the partitioned solution and can also monitor the solution. We will not consider automatic visualisation issues in the following, since we have used stand-alone software tools.

4. $\alpha\Psi$NKS Solver Main Operations Analysis

The main computational kernels of our solver are as follows:

(1) construction of the distributed matrix;

(2) vector updates and AXPY operations;

(3) vector-vector products;

(4) matrix-vector products;

(5) application of the preconditioner.

In this section we analyse these operations.

Let us focus on the *construction of the distributed matrix*. As typical, the finite-element matrix is constructed by looping over all the elements of the local grid, to compute the elemental matrix $A^{(T)}$, corresponding to each element T. On processor i, the contribution of $A^{(T)}$ will be considered only if the corresponding row is contained in the update set of i. This means that A is constructed in a distributed way, that is, each row of A is stored on a unique processor. In an analogous way, every vector, say $\mathbf{u}$, is distributed: each processor will store only $\mathbf{u}_i$, the vector composed by the components of $\mathbf{u}$ which correspond to its update set, and of size n_i.

An *AXPY operations* (i.e., $\mathbf{u} \leftarrow \mathbf{u} + \alpha\mathbf{v}$ for $\mathbf{u}$ and $\mathbf{v}$ two given vectors and α a given real number) are done as follows. First, one has to verify that all processors have the same value of α; then, for all the elements of the local vector $\mathbf{u}_i$, one do $\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha\mathbf{v}_i$. Apart from the synchronisation of the value of α (step which is often avoided), no communication occurs in vector updates and AXPY operations.

The *vector-vector product* $\beta = \langle \mathbf{u}, \mathbf{v}\rangle$ on processor i reads as follows: (1) compute the local value $\beta_i = \langle \mathbf{u}_i, \mathbf{v}_i\rangle$; (2) sum up the local contributions of all the processors: $\beta = \sum_{i=1}^{M} \beta_i$. Note that communication is required only in the second step. Before computing the final value of β, all processors must have finished the computations in the first step. For this reason, vector-vector product always act, in a parallel environment as *synchronisation points* and require, global communications.

In the *matrix-vector product* $\mathbf{y} = A\mathbf{x}$, a processor i computes only the elements of $\mathbf{y}$ in its update set, that is, in $\mathbf{y}_i$. Hence, the local parallel matrix-vector product reads as

$$\mathbf{y}_i = A_i\mathbf{x}_i + A_{E,i}\mathbf{x}_{E,i}, \tag{20}$$

where $\mathbf{x}_i$ and $\mathbf{x}_{E,i}$ are the values of $\mathbf{x}$ corresponding to its update and external set, respectively, A_i is the submatrix representing the influence of update nodes on themselves, while $A_{E,i}$, the action of external nodes on update ones.

From (20), the steps required to perform the parallel matrix product are (1) the exchange of boundary data, that is, the update the value of the external nodes $\mathbf{x}_{E,i}$; (2) the computation of the the local matrix-vector product $\mathbf{y}_i = A_i\mathbf{x}_i$; (3) the computation of the external matrix-vector product $\mathbf{y}_i \leftarrow \mathbf{y}_i + A_{E,i}\mathbf{x}_{E,i}$.

Steps (1) and (2) can be overlapped. Note that an efficient implementation of step (1) is particularly important to obtain a scalable matrix-vector product. As a general rule, data to be sent to a processor should be packed and sent using only one MPI call to avoid latency problems occurring within the network. However, since the grid discretisation is unstructured, it is not trivial to determine the owner of an external set and, moreover, all the nodes belonging to a certain processor are not necessarily stored contiguously in the vector $\mathbf{x}_{E,i}$. This operation of packing data into a buffer creates contiguous messages from irregularly spaced data structure and requires a preprocessing of the matrix, plus the definition of a certain number of data structures.

We want to underline the fact that for both first-order and second-order MURD schemes, the computational stencil of a generic vertex i is compact, that is, it only depends on information from vertexes that are one edge-distance away in the grid structure. Therefore, the Jacobian matrix will have nonzero elements only for the contributions corresponding to the first-order neighbouring nodes even for second-order schemes. This simplifies considerably the communication and allows the use of the same data structure for first-order and second space discretisation schemes.

Finally, let us consider the *application of the Schwarz preconditioner*. Using the VO decomposition outlined in Section 3, it is particularly easy to implement a one-level Schwarz preconditioner P_S with minimal overlap, that is, with an overlap among the subdomains of one element only. Referring again to Figure 2, the overlapping region among the subdomains is represented by the elements in shaded colour.

On processor i, the computation of the preconditioned residual $\mathbf{z} = P_S^{-1}\mathbf{r}$ simply reads as

$$\mathbf{z}_i = A_i^{-1}\mathbf{r}_i. \tag{21}$$

Matrix A_i is already stored and formed in the local memory of processor i. Note that Schwarz preconditioners with wider overlaps will require communication. In general, the optimal amount of overlap is clearly a compromise between the effectiveness of the preconditioner and its computational complexity. For this class of problems, we have found it convenient to use the minimal overlap version. This is equivalent to a block-Jacobi procedure, where the blocks have size n_i. An ILU(0) or BILU(0) factorisation (using a reverse Cuthill-McKee reorder) of the block matrices gives satisfactory results (in terms of CPU times) with respect to other "richer" incomplete factorisations [41]. In fact, compared with other variants, ILU(0) is computationally fast and memory efficient.

Now, let us focus on the two-level Schwarz preconditioner described in Section 2, with a coarse space based on aggregation. We show that this preconditioner makes use of information at subdomain level. To obtain a representation of matrix R_0, we order the set of nodes putting before the nodes in Ω_1, and then the nodes of the domain subdomains in the other subdomains. Analogously, we order the degree of freedom of V_0. As a result, $R_0^T \in \mathbb{R}^{n_0 \times n}$ has the following block structure:

$$R_0^T = \begin{bmatrix} \boldsymbol{\eta}_{1,1}^T & \mathbf{0} & \cdots & \mathbf{0} \\ \boldsymbol{\eta}_{2,1}^T & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \mathbf{0} & \cdots & \mathbf{0} \\ \boldsymbol{\eta}_{n_{0,1},1}^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\eta}_{1,2}^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \boldsymbol{\eta}_{n_{0,2},2}^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\eta}_{1,p}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\eta}_{n_{0,p}}^T \end{bmatrix}. \tag{22}$$

Note that each vector $\boldsymbol{\eta}_{s,i}$ can be constructed at the subdomain level, using only the local information about the grid of Ω_i (hence no communications). Very often, the graph of the local matrix A_i can furnish the information required to produce the aggregates, and henceforth grid data are not required by the algorithm. This is notably the case of finite-element codes with $\mathbb{P}^1$ basis function, as used in this paper. Note moreover that the vectors $\boldsymbol{\eta}_{s,i}$ are not explicitly required by the algorithm. We use one integer vector $\mathbf{p}$, so that p_i represents the (local) degree of freedom associated to the aggregate $\vartheta_{s,i}$. The vector $\mathbf{p}$ will be of size $n_i + n_{E,i}$. First, communications occur to exchange the actual values of $\mathbf{p}_{E,i}$ among the processors. Then, in the aggregation procedure to compute the coarse matrix as in (19), no communication will be required to compute, on each subdomain, the rows of A_0 corresponding to the degree of freedom of V_0 contained in that subdomain.

5. Numerical Results

In this section the $\alpha\Psi$NKS solver presented previously is validated. The underlying Euler code we have used is called THOR, developed within the BRITE-EURAM project IDeMAS. The basic code was provided by the project coordinator, the von Karman Institute VKI (Belgium), and includes contributions from the partners, DASSAULT AVIATION (France), ALENIA (Italy), DASA/EADS (Germany), INRIA (France), and EPFL (Switzerland). THOR involves several high-level numerical schemes aiming to produce highly accurate multidimensional upwinding techniques, see [5]. The graph partitioning is based on the public domain software METIS [42] and its parallel counterpart ParMETIS [43]. The linear systems involved are solved using the library AZTEC [44], developed at the Sandia National Laboratories (USA). The Schwarz algorithms and the Krylov solvers have been implemented by the authors.

The numerical results have been obtained on a SGI-Origin 3800, equipped with 128 MIPS R14000 500 Mhz processors. Each processor has 512 Mbytes of RAM, with 32 Kbytes and 8 Kbytes of first level and second level cache memory, respectively. This type of machine can be replaced by Linux cluster-type machines, the results being significant for both types of architectures. The parallel communication paradigm used is MPI [37].

Three test cases taken of aeronautical interest have been selected to illustrate the variation of the performance with the various parameters within the algorithm and validate the full $\alpha\Psi$NKS cycle.

In all the numerical tests, P_S indicates the one-level Schwarz preconditioner, with ILU(0) factorisations and minimal overlap among the subdomains. $P_{C,\text{hybrid}}$ denotes the two-level Schwarz preconditioner, as detailed in Section 2. The local dimension of the coarse space (i.e., the number of aggregates assigned to each processor) is n_0/M. The coarse problem is solved redundantly on each processor, using a direct LU decomposition.

5.1. *Falcon Aircraft at $M_\infty = 0.45$ and $\alpha = 1^\circ$.*

We have studied the influence of the Krylov accelerator on a small 3D test case, namely, a Falcon aircraft with a computational

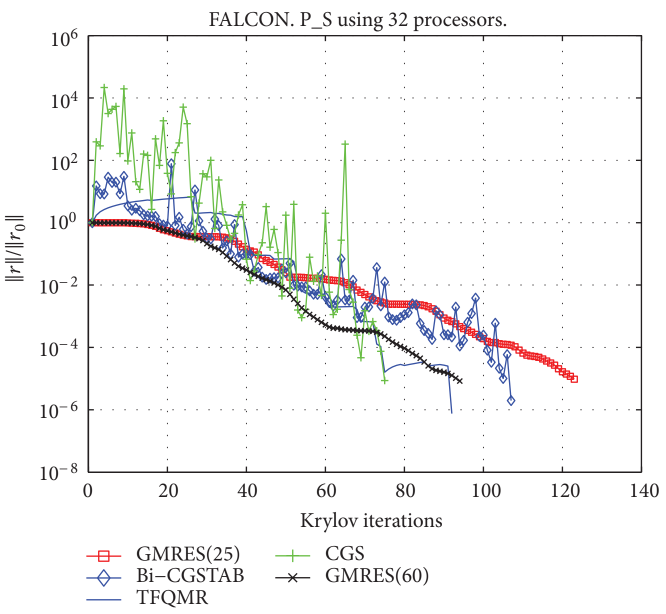

Figure 3: FALCON aircraft. Convergence history at the 10th time level using 32 processors and various Krylov accelerators.

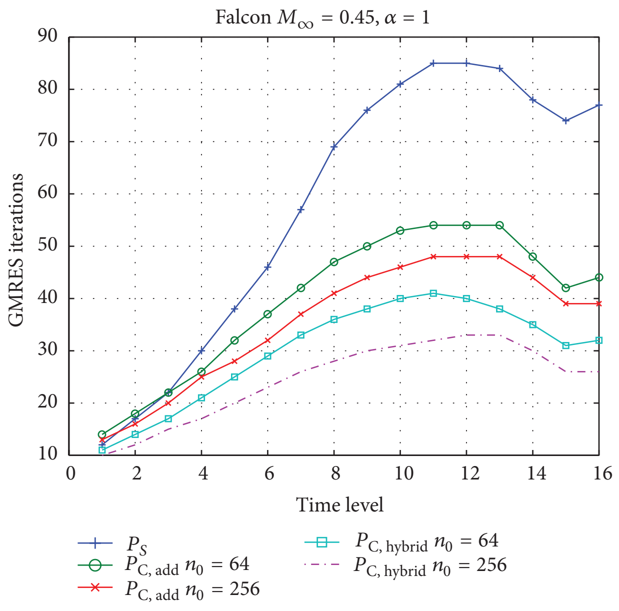

Figure 4: FALCON aircraft. Iterations to converge at each time iterations, using P_S, $P_{C,\text{add}}$ and $P_{C,\text{hybrid}}$, and 16 processors. n_0 denotes the dimension of the coarse space for the aggregates.

grid composed of 45387 nodes and 255944 elements, with flight conditions of $M_\infty = 0.45$ and $\alpha = 1^\circ$. This test case is representative of a complete aeronautical configuration of an industrial airplane, with body, wing, and nacelles. This is the only subsonic test case. The mesh is quite coarse, and we have used this test case to validate the Krylov accelerators and the basic properties of the preconditioners (see also [23, 45] for a more extensive analysis on this subject).

Slip boundary conditions are imposed on the airplane. Since the problem is symmetric, as well as the boundary conditions, the mesh is represented by half a sphere, where we have imposed far-field and symmetry-plane boundary conditions. Referring to equation (10), we have used $\text{CFL}_0 = 1.0$, $\sigma = 2$, and $\text{CFL}_{\max} = 10^6$. For this test case, only the first-order N-scheme for systems was used, since the aim of this test case was to validate the choice of the Krylov accelerator.

Figure 3 reports the norm of the residual (scaled by the norm of the initial residual) versus the linear system solver iterations, for the 10th time step. The accelerators considered are, Bi-CGSTAB, CGS, TFQMR, GMRES(25), and GMRES(60). The preconditioner is P_S.

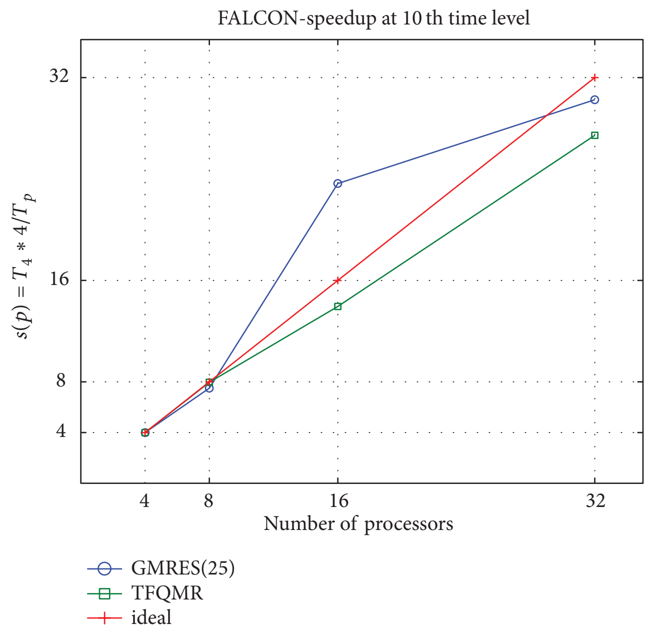

FIGURE 5: FALCON aircraft. Relative speedup curves taken at the 10th time level.

Figure 4 reports the iterations to converge to a tolerance on the relative residual of 10^{-5} at each time iteration, using P_S, $P_{C,\text{add}}$, and $P_{C,\text{hybrid}}$. From the figure, it is evident that the CFL number has a strong influence on the iterations to converge. In fact, as the CFL number increases, the linear system becomes more skew-symmetric and less diagonal-dominant, and hence more iterations are required to converge to the desired accuracy. Note that, for low values of the CFL number, the number of iterations to converge is almost the same with one-level and two-level methods. This may suggest to adapt the preconditioner for the problem to be solved, introducing the coarse problem only for certain values of the CFL number.

For any combination of solver and preconditioner, convergence of the linear system solver is reached at any time step. GMRES is slightly faster than the other methods; it takes more iterations to converge, but the single iteration is faster, since it requires only one matrix-vector product, instead of two as in the other methods. However, it is quite sensitive to the dimension of the Krylov space, especially as the number of processor used in the computation increases. The convergence history at the 10th time step (when the CFL number is 10240) reveals that TFQMR has a quite erratic convergence, while the one provided by CGS is (unexpectedly) quite regular. Bi-CGSTAB performs quite well but it is still erratic. The differences among the Krylov accelerators are particularly evident as the number of processors grows. This may reflect the fact that the Schwarz preconditioners with no coarse correction worsen their performances with many subdomains.

As a result of this analysis, GMRES appear to be the most interesting among the iterative schemes for our class of problems. This confirms what was already pointed out in [23]. The only drawback of GMRES is represented by the memory requirements. Therefore, if it is the case to use Krylov subspaces of "low" dimension, one may consider other iterative schemes, like TFQMR or Bi-CGSTAB.

The results also show a reasonable scalability as illustrated in Figure 5. The whole convergence process is autoregulating; however, in general, 20–200 linear system iterations are performed 4 to 16 time levels as illustrated in Figure 6.

5.2. X29 Aircraft at $M_\infty = 0.75$ and $\alpha = 5^\circ$. The next test case is a X29 experimental aircraft, flying at almost transonic speed. It represents the experimental Grumman Model 712, designated X-29A by the USAF, a single-seat jet aircraft fitted with a wing mounted at the rear of the fuselage, swept forward at 35°, and with canards just behind the cockpit. The flight conditions are $M_\infty = 0.75$, $\alpha = 5^\circ$. Equation (10) is used, with $\text{CFL}_0 = 0.2$, $\sigma = 2$, $\text{CFL}_{\max} = 1000$.

The initial grid is composed by 136767 nodes and 726713 elements. The first-order N-scheme for systems was used on the nonadapted grid, while a second-order blended scheme was used on the adapted grids. In all the computations, the Jacobian matrix is formed starting from first-order space approximation to save computational time, because the evaluation of second order distribution residual is more time consuming with respect to first-order method.

In Table 1, the cost of the following adaptation sequence is reported. The complete solution is required at least 8 hours on 16 processors, and less than one hour on 64 processors. This result is mainly due to the specific architecture of the SGI-Origin, which allows a processor to use (when required) the memory of another processor. This is notably the case of the X29 test case when run with only 16 processors, due to the size of the problem; while by using 64 processors, then only the local processor memory was used.

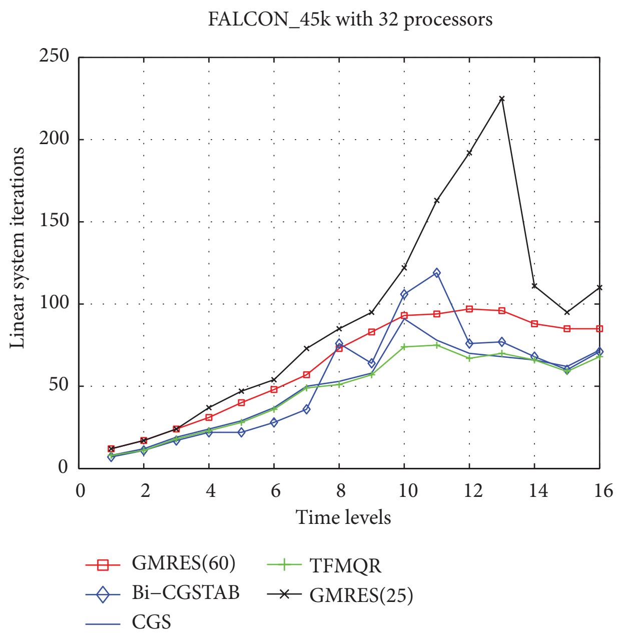

Figure 6: FALCON aircraft. Comparison of number of iterations per time level for 32 processors.

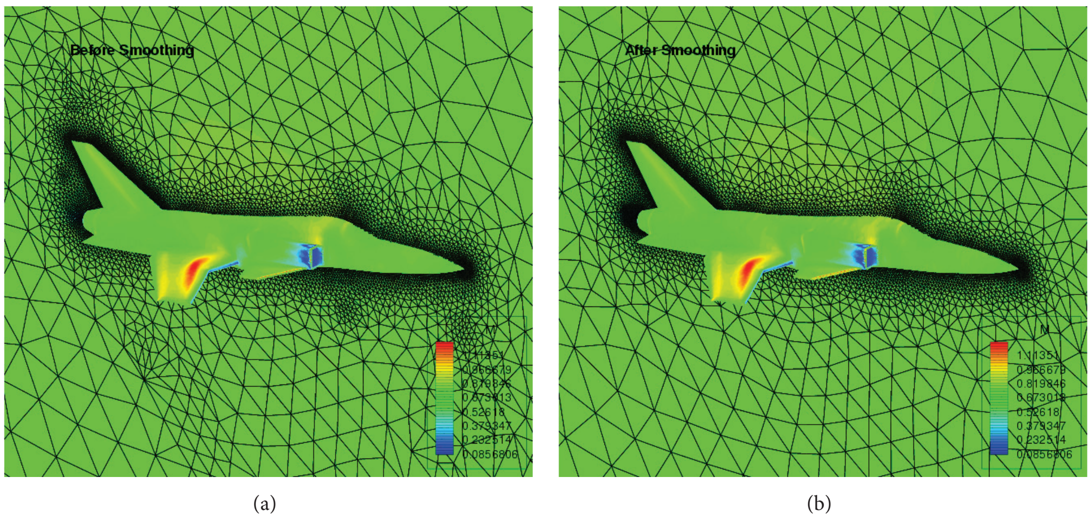

Figure 7: X29 aircraft. Before and after optimisation and smoothing. In particular, the symmetry plane grid on the left picture shows an excessive neighbour number elements. This is no longer present in the right picture, after optimisation and smoothing.

The $\alpha\Psi$NKS cycle is as follows. Starting from an initial grid $\mathcal{T}^{(0)}$, 2 steps of refinement and derefinement and structural optimisation gave the grid $\mathcal{T}^{(1)}$. Then smoothing optimisation is performed on $\mathcal{T}^{(1)}$ to obtain $\mathcal{T}^{(2)}$. Then on $\mathcal{T}^{(2)}$ large scale, refinement and optimisation gave a final grid of 1.47 million nodes and 8.7 million elements, $\mathcal{T}^{(3)}$, which is a very interesting performance test case.

The effect of the smoothing optimisation can be seen in Figure 7. Figure 8 details the solutions on these first refinements. The mesh refinement criteria taken here are segment length and Mach gradients and are detailed in [38].

A comparison of iterations to converge and CPU times is reported in Table 2 for the first 14 iterations of the $\alpha\Psi$NKS procedure. In the table, iters indicates the sum of Krylov iterations at each backward Euler time step, and CPU time the sum of the (elapsed) CPU times (in seconds) to solve the Jacobian linear system. More detailed results about one-level and two-level Schwarz preconditioners are reported in Table 3. As previously noted, the CFL number has a strong influence on the convergence. One may note that $P_{C,\text{hybrid}}$ performs better than $P_{C,\text{add}}$. For small problems, large dimension of V_0 have to be preferred, while bigger problems required a fine tuning of n_0/M to obtain the best numerical results.

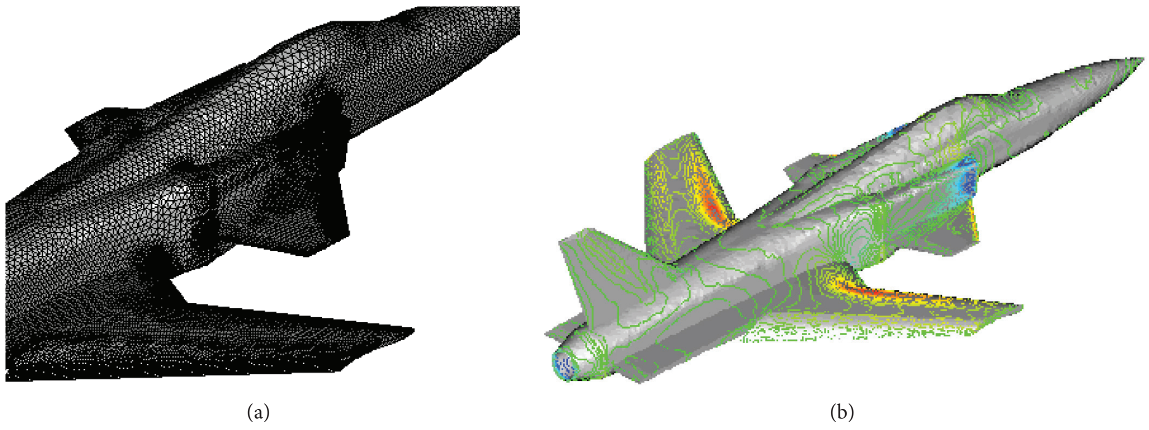

Figure 8: X29 aircraft. Zoom over part of the refinement and Mach number distribution on the body for the grid $\mathscr{M}_1$.

Table 1: X29 aircraft. Time for successive adaptations using 8 and 16 processors.

First cycle $\mathscr{T}^{(1)}$ 2 cycles	$M = 8$	78 secs.	Grid derefinement/refinement
		690 secs.	Regularisation and optimisation
		132 secs.	Overheads
Final Grid $\mathscr{T}^{(3)}$ 2 cycles	$M = 16$	854 secs.	Grid derefinement/refinement
		2860 secs.	Regularisation and optimisation
		1002 secs.	Overheads

Table 2: X29 aircraft. Comparison of P_S and $P_{C,\text{hybrid}}$ using different number of processors for the non-adapted grid, is made up of 136767 nodes and 726713 elements. Both CPU-time and iterations refer to the first 14 time iterations of the backward Euler method. For the coarse correction $n_0/M = 4$.

	$M = 8$	$M = 16$	$M = 32$	$M = 64$
	iters			
P_S	270	310	369	347
$P_{C,\text{hybrid}}$	—	199	210	171
$(P_S - P_{C,\text{hybrid}})/P_{C,\text{hybrid}} * 100$	—	55.7	75.1	102.9
	CPU-time			
P_S	4995.1	1461.2	451.6	202.9
$P_{C,\text{hybrid}}$	—	1253.3	434.3	189.1
$(P_S - P_{C,\text{hybrid}})/P_{C,\text{hybrid}} * 100$	—	16.58	3.98	7.29

It is worth to note that the efficiency of the $P_{C,\text{hybrid}}$ preconditioner clearly outperforms P_S in terms of iterations, while as concerns CPU-times there is an improvement of about 10%. This is due to the following facts:

(i) $P_{C,\text{hybrid}}$ requires a matrix-vector product to be performed, and this is computationally expensive;

(ii) the solution of the coarse problem requires intensive communications among theprocessors.

Table 3: X29 aircraft. Comparison among different preconditioners for different values of the CFL number. 16, 32 and 64 processors have been used in the computations.

Time Level	CFL	M	Prec	iters	CPU-time
10	102.4	32	P_S	29	33.26
14	1000.0	32	P_S	84	54.79
10	102.4	64	P_S	33	15.57
14	1000.0	64	P_S	90	25.28
10	102.4	16	$P_{C,\text{hybrid}}, n_0/M = 4$	16	96.69
14	1000.0	16	$P_{C,\text{hybrid}}, n_0/M = 4$	51	117.64
10	102.4	32	$P_{C,\text{hybrid}}, n_0/M = 4$	16	31.38
14	1000.0	32	$P_{C,\text{hybrid}}, n_0/M = 4$	39	37.93
10	102.4	64	$P_{C,\text{hybrid}}, n_0/M = 1$	20	13.53
14	1000.0	64	$P_{C,\text{hybrid}}, n_0/M = 1$	79	23.46
10	102.4	64	$P_{C,\text{hybrid}}, n_0/M = 4$	15	14.77
14	1000.0	64	$P_{C,\text{hybrid}}, n_0/M = 4$	35	19.45
10	102.4	64	$P_{C,\text{hybrid}}, n_0/M = 8$	15	12.93
14	1000.0	64	$P_{C,\text{hybrid}}, n_0/M = 8$	35	16.86
10	102.4	64	$P_{C,\text{hybrid}}, n_0/M = 16$	15	14.05
14	1000.0	64	$P_{C,\text{hybrid}}, n_0/M = 16$	35	18.82

5.3. ONERA M6 Wing at $M_\infty = 0.84$ and $\alpha = 3.06^\circ$. The last test case is an ONERA M6 wing at $M_\infty = 0.84$ and $\alpha = 3.06^\circ$. It represents a ONERA M6 wing, a widely used test case for 3D flows from subsonic to transonic regimes. The selected transonic case is $M_\infty = 0.84$ and $\alpha = 3.06$, which results in several shock waves over the wing. In the flow-field, a shock originates at approximatively 80% of the wing span. A second leading-edge shock is caused by the 30% sweep angle of the M6 wing. Near the tip of the wing, a lambda-shock structure is observed.

The first-order N-scheme for systems is used to obtain the solution on the first (nonadapted) grid, while the second-order blended scheme is used for the adapted grids. As for the X29 test case, the Jacobian matrix is formed using the first-order scheme.

The far field is represented by a half sphere with a radius of 12.5 root-chord length. The $\alpha\Psi$NKS procedure has been

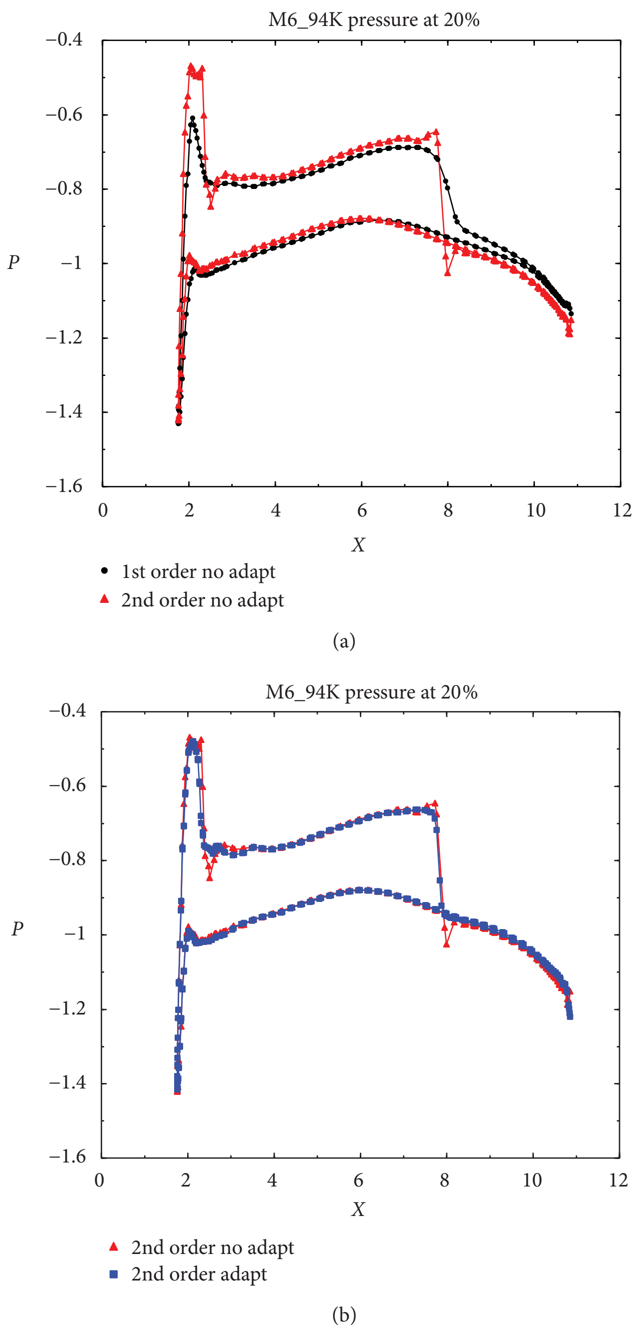

Figure 9: ONERA M6 wing. Pressure coefficient at 20% of wing span, using first-order and second-order schemes on a nonadapted grid (a), and second-order schemes on nonadapted and adapted grid (b).

performed as follows. The starting nonoptimised grid is composed by 94493 nodes and 666569 elements, and an initial constant solution is used. Then, two steps of refining are performed. Each adaptation phase corresponds to at least two passes of adaptation (refinement coarsening and optimisation). The final grid (after two adaptation cycles) is composed by 585725 nodes and 3477090 elements.

The pressure coefficient at 20% of the wing span is reported in Figure 9. The picture on the left shows the improvement from first-order to second-order schemes. Second-order scheme captures the first shock wave, near the leading edge, with much greater precision than first-order schemes. The over- and under-shoots are sensibly reduced by using adapted grids, as one can notice from the picture on the right.

An analysis of the flow field on the upper part of the wing is reported in Figure 10, while Figure 11 reports the nonadapted grid and the adapted grid.

As regards the convergence of the $\alpha\Psi$NKS scheme, as one can seen from Figure 12 that in the case of 16 processors,

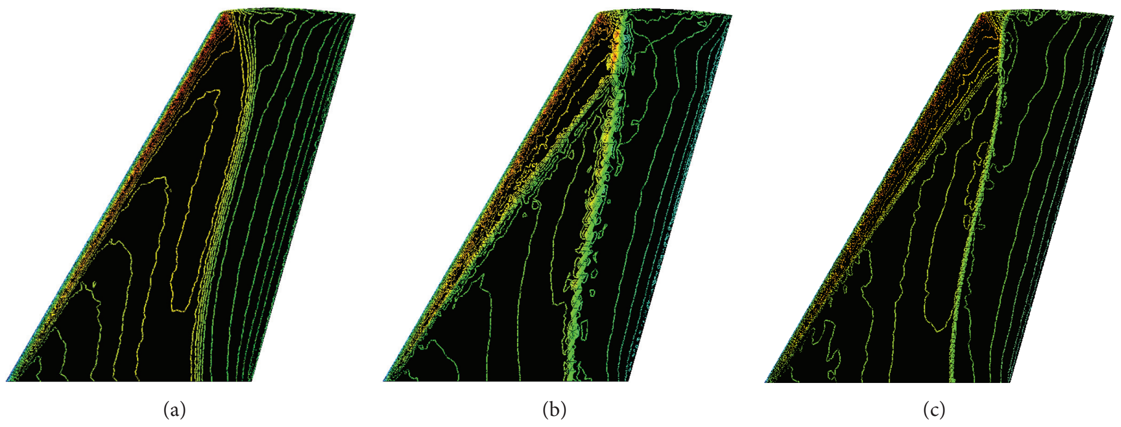

Figure 10: ONERA M6 wing. From left to right, Mach number contours for: a first-order solution and a second-order solution on the nonadapted grid, and a second-order solution on the adapted grid.

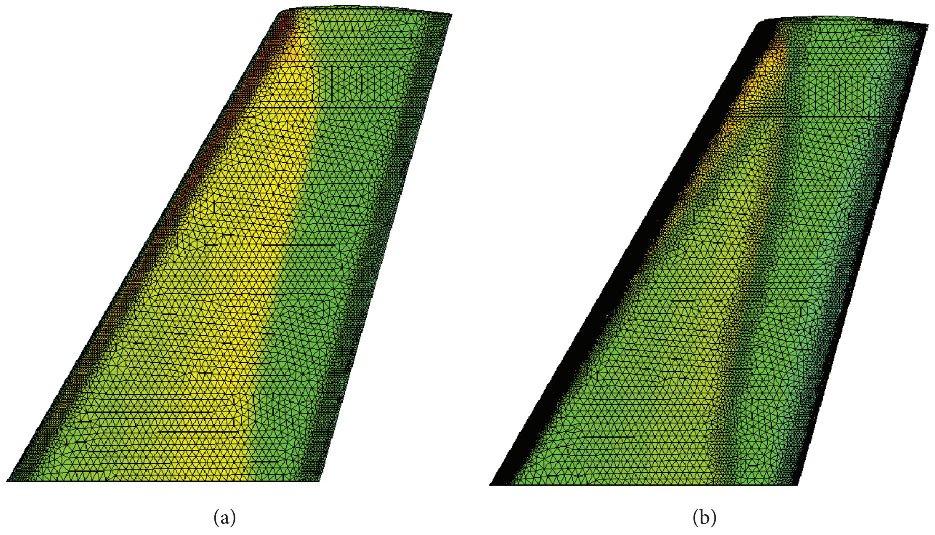

Figure 11: ONERA M6 wing. Nonadapted grid (a) and adapted grid (b).

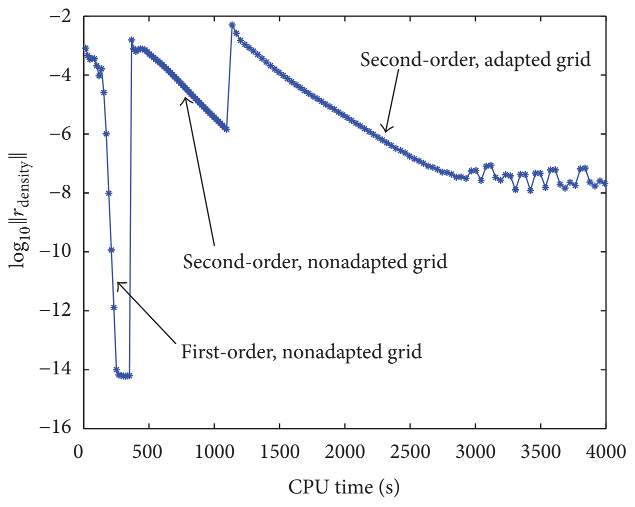

Figure 12: ONERA M6 wing. Convergence history of the $\alpha\Psi$NKS scheme. On the x-axes, CPU time, in seconds. On the y-axes, the logarithm of the (Euclidean) norm of the density residual.

the first-order N-scheme rapidly converges to steady-state solution very rapidly. In this phase, $\mathrm{CFL}_0 = 10$, $\sigma = 2.0$, $\mathrm{CFL}_{\max} = 10^6$, and analytical Jacobian are used. These initial iterations are needed to place the shocks in a more or less correct position, so that they do not have to move consistently when second-order schemes are employed. For second-order schemes, we have used $\mathrm{CFL}_0 = 0.2$, $\sigma = 2.0$, and $\mathrm{CFL}_{\max} = 1000$.

As already reported in [23, 46], the iterative convergence of second-order schemes is poor; basically, for nonadapted meshes, the L^2 density residuals stagnate around 2 to 3 orders of magnitude, where the residual is compared to the first iteration. The situation is slightly improved on adapted meshes, even if it was not possible to reduce the residual more than a certain factor, here about 10^{-5} with respect to the first iteration on the nonadapted grid.

6. Conclusions

Multidimensional upwind schemes have reached a certain degree of maturity for the solution of the steady-state compressible Euler equations on unstructured grids made up of triangles (in 2D) and tetrahedrons (in 3D). They can be used for complex aerodynamic flows, in the subsonic, transonic, and supersonic regime. Nevertheless, it is clear that the the full potential of the approach may be revealed only if these methods could effectively lead to scalable parallel codes. In particular, this requires, among others, the definition and the implementation of a scalable preconditioner and of a parallel grid adaption technique to improve the solution accuracy whilst optimising the computational resources. To that aim, we have here presented the adaptive pseudotransient Newton-Krylov-Schwarz framework.

Most parallel implementation of MURD schemes are based on one-level Schwarz preconditioner and do not include automatically grid adaptation procedure. This paper aims to present a complete framework which couples these two aspects with the characteristic aspects of MURD schemes.

The main kernels of this algorithm have been analysed. A data structure, based on a vertex-oriented decomposition of the grid, has been described. This data structure allows an efficient implementation of all the main computational kernels. Of particular importance is the definition of the preconditioner. One-level and two-level Schwarz preconditioners have been presented. The coarse matrix is constructed automatically and for any computational grid with no input from the user (except for the matrix A and the dimension of the coarse space). This simplicity has its roots mainly in the way the restriction and interpolation operators are defined. The results show an improvement of performance with respect to the one-level Schwarz method, which is the preconditioner usually adopted in the literature for multidimensional upwind residual distribution schemes.

Numerical results, obtained on distributed memory computers, are presented for large scale computations in the sense of complexity and mesh adaptation on parallel machines. Results show that the conjunction of the Krylov-Schwarz approach with a grid adaptation procedure leads to an effective solution of the compressible Euler equations on unstructured grids. The same techniques apply to the Navier-Stokes system of equations solved by similar equivalent finite-volume techniques on unstructured grids [18, 47].

Acknowledgments

This paper summarises the activity of the authors, partially financed within the BRITE/EURAM European Union founded project IDeMAS (Contract no. BRPR-CT07-0591) and the Swiss National Science Foundation project no. 200020-112221. The Swiss Federal Office for Education and Science (OFES) is also acknowledged for financial support. The authors would like to thank Professor A. Quarteroni and Professor L. Formaggia for the useful discussions and help. VKI is acknowledged for the basic THOR code. The DIT/EPFL is acknowledged for the technical assistance.

References

[1] "Unstructured grid methods for advection dominated flows," In AGARD, R-987, 1992.

[2] H. Deconinck, K. Sermeus, and R. Abgrall, *Status of Multidimensional Upwind Residual Distribution Schemes and Applications in Aeronautics*, AIAA Paper, 2000.

[3] P. L. Roe, "Multidimensional upwinding—motivation and concepts," in *25th VKI Lecture Series on Computational Fluid Dynamics*, pp. 94–95, VKI, 1994.

[4] P. L. Roe and D. Sildikover, "Optimum positive linear schemes for advection in two and three dimension," Tech. Rep., ICASE, 1989.

[5] Brite/Euram (OFES N. 97.04271), "Industrial demonstration of accurate and efficient multidimensional upwinding algorithms for aerodynamic simulation IDeMAS," Tech. Rep., Von Karman Institute, Rhode Saint Genèse, Belgium, 2001.

[6] P. L. Roe, "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, vol. 43, no. 2, pp. 357–372, 1981.

[7] P. L. Roe, "Fluctuation and signals—a framework for numerical evolution problems," in *Numerical Methods For Fluid Dynamics*, K. W. Morton and M. J. Baines, Eds., pp. 219–257, Academic Press, 1982.

[8] A. Quarteroni and A. Valli, *Domain Decomposition Methods For Partial Differential Equations*, Oxford University Press, Oxford, UK, 1999.

[9] B. F. Smith, P. Bjorstad, and W. D. Gropp, *Parallel Multilevel Methods For Elliptic Partial Differential Equations*, Cambridge University Press, Cambridge, UK, 1996.

[10] X. C. Cai, D. E. Keyes, and V. Venkatakrishnan, "Newton-Krylov-Schwarz: an implicit solver for CFD," in *Proceedings of the 8th International Conference on Domain Decom-position Methods*, R. Glowinski, Ed., pp. 387–400, 1997.

[11] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, "High-performance parallel implicit CFD," *Parallel Computing*, vol. 27, no. 4, pp. 337–362, 2001.

[12] W. Gropp, D. Keyes, L. C. McInnes, and M. D. Tidriri, "Globalized Newton-Krylov-Schwarz algorithms and software

for parallel implicit CFD," *International Journal of High Performance Computing Applications*, vol. 14, no. 2, pp. 102–136, 2000.

[13] E. Nielsen, W. Anderson, R. Walters, and D. E. Keyes, "Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code," AIAA Paper 95–1733, 1995.

[14] D. A. Knoll and D. E. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[15] C. Y. Huang and F. N. Hwang, "Parallel pseudo-transient Newton-Krylov-Schwarz continuation algorithms for bifurcation analysis of incompressible sudden expansion flows," *Applied Numerical Mathematics*, vol. 60, no. 7, pp. 738–751, 2010.

[16] G. May, F. Iacono, and A. Jameson, "Efficient algorithms for high-order discretizations of the euler and navier-stokes equations," Tech. Rep., Stanford University, 2009.

[17] C. Hirsch, *Numerical Computation of Internal and External Flow*, vol. 2 of *Computational Methods for Inviscid and Viscous Flows*, John Wiley & Sons, New York, NY, USA, 1990.

[18] C. Hirsch, *Numerical Computation of Internal and External Flow*, vol. 1 of *Fundamentals of Numerical Discretization*, John Wiley & Sons, New York, NY, USA, 1989.

[19] H. Paillère, *Multidimensional upwind residual distribution schemes for the Euler and Navier-stokes equations on unstructured grids [Ph.D. thesis]*, Université Libre de Bruxelles, Brussels, Belgium, 1995.

[20] R. Struijs, *A multi-dimensional upwind discretization method for the Euler equations on unstructured grids [Ph.D. thesis]*, The University of Delft, Delft, The Netherlands, 1994.

[21] R. Struijs, H. Deconinck, and P. Roe, "Fluctuation Splitting Schemes for multidimensional convection problems: an alternative to finite volume and finite element methods," in *Computational Fluid Dynamics*, VKI LS, 1990.

[22] R. Struijs, H. Deconinck, and P. Roe, "Fluctuation splitting schemes for the 2D Euler equations," in *Computational Fluid Dynamics*, VKI LS, 1991.

[23] E. van der Weide, *Compressible flow simulations on unstructured grids using multi-dimensional upwind schemes [Ph.D. thesis]*, Université Libre de Bruxelles, Brussels, Belgium, 1998.

[24] T. Quintino, M. Ricchiuto, A. Csik, H. Deconinck, and S. Poedts, "Conservative multidimensional upwind residual distribution schemes for arbitrary finite elements," in *Computational Fluid Dynamics 2002*, S. Armfield, P. Morgan, and K. Srinivas, Eds., Springer, 2003.

[25] P. L. Roe, "Discrete models for the numerical analysis of time-dependent multidimensional gas dynamics," *Journal of Computational Physics*, vol. 63, no. 2, pp. 458–476, 1986.

[26] H. Deconinck, C. Hirsch, and J. Peuteman, "Characteristics decomposition methods for the multidimensional Euler equations," in *Proceedings of the International Conference on Numerical Methods in Fluid Dynamics*, vol. 264 of *Lecture Notes in Physics*, pp. 216–221, Springer, Berlin, Heidelberg, June 1986.

[27] 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.

[28] 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.

[29] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 400–408, 1982.

[30] D. Vanderstraeten, Á. Ćsík, and D. Roose, "An expert-system to control the CFL number of implicit upwind methods," Tech. Rep., Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium, 2000.

[31] C. T. Kelley and D. E. Keyes, "Convergence analysis of pseudo-transient continuation," *SIAM Journal on Numerical Analysis*, vol. 35, no. 2, pp. 508–523, 1998.

[32] M. Sala and L. Formaggia, "Algebraic coarse grid operators for domain decomposition based preconditioners," in *Parallel Computational Fluid Dynamics—Practice and Theory*, P. Wilders, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, Eds., pp. 119–126, Elsevier, Amsterdam, The Netherlands, 2002.

[33] L. Jenkins, T. Kelley, C. T. Miller, and C. E. Kees, "An aggregation-based domain decomposition preconditioner for groundwater flow," Tech. Rep. TR00-13, Department of Mathematics, North Carolina State University, 2000.

[34] C. Lasser and A. Toselli, "An overlapping domain decomposition preconditioner for a class of discontinuous Galerkin approximations of advection-diffusion problems," *Mathematics of Computation*, vol. 72, no. 243, pp. 1215–1238, 2003.

[35] L. Paglieri, A. Scheinine, L. Formaggia, and A. Quarteroni, "Parallel conjugate gradient with Schwarz preconditioner applied to fluid dynamics problems," in *Proceedings of the Parallel Computational Fluid Dynamics, Algorithms and Results Using Advanced Computer (CFD '97)*, P. Schiano, Ed., pp. 21–30, 1997.

[36] C. Lasser and A. Toselli, "Convergence of some two-level overlapping domain decomposition preconditioners with smoothed aggregation coarse spaces," Tech. Rep. TUM-M0109, Technische Universität München, 2001.

[37] "Message Passing Interface Forum. MPI: a message-passing interface standard," Tech. Rep., 1995.

[38] P. Leyland, A. Casagrande, and Y. Savoy, "Parallel mesh adaptive techniques illustrated with complex compressible flow simulations," *Modelling and Simulation in Engineering*, vol. 2012, Article ID 317359, 14 pages, 2012.

[39] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[40] 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.

[41] M. Sala and L. Formaggia, "Parallel Schur and Schwarz based preconditioners and agglomeration coarse corrections for CFD problems," Tech. Rep. 15, DMA-EPFL, 2001.

[42] G. Karypis and V. Kumar, "METIS: unstructured graph partitioning and sparse matrix ordering system," Tech. Rep., Department of Computer Science, University of Minnesota, 1998.

[43] 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.

[44] R. Tuminaro, M. Heroux, S. Hutchison, and J. Shadid, "Official Aztec user's guide: version 2. 1.," Tech. Rep. SAND99-8801J, Sandia National Laboratories, Albuquerque, Nm, USA, 1999.

[45] E. Issman, *Implicit solution strategies for compressible flow equations on unstructured grids [Ph.D. thesis]*, Université Libre de Bruxelles, Brussels, Belgium, 1997.

[46] R. Abgrall, "Toward the ultimate conservative scheme: following the quest," *Journal of Computational Physics*, vol. 167, no. 2, pp. 277–315, 2001.

[47] G. Degrez, "Implicit time-dependent methods for inviscid and viscous compressible flows, with a discussion of the concept of numerical dissipation," in *Computational Fluid Dynamics*, VKI Lecture Series, 2009.