DOMAIN DECOMPOSITION PRECONDITIONERS: THEORETICAL PROPERTIES, APPLICATION TO THE COMPRESSIBLE EULER EQUATIONS, PARALLEL ASPECTS

THÈSE N^{O} 2733 (2003)

PRÉSENTÉE À LA FACULTÉ SCIENCES DE BASE

SECTION DE MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Marzio SALA

dottore in ingegneria aerospaziale, Politecnico di Milano, Italie
et de nationalité italienne

acceptée sur proposition du jury:

Prof. A. Quarteroni, directeur de thèse
Prof. H. Deconinck, rapporteur
Prof. M. Deville, rapporteur
Dr A. Toselli, rapporteur
Prof. A. Valli, rapporteur

Lausanne, EPFL
2003

Abstract

The purpose of this thesis is to define efficient parallel preconditioners based on the domain decomposition paradigm and to apply them to the solution of the steady compressible Euler equations.

In the first part we propose and analyse various domain decomposition preconditioners of both overlapping (Schwarz) and non-overlapping (Schur complement-based) type. For the former, we deal with two-level methods, with an algebraic formulation of the coarse space. This approach enjoys several interesting properties not always shared by more standard two-level methods. For the latter, we introduce a class of preconditioners based on a peculiar decomposition of the computational domain. The domain is decomposed in such a way that one subdomain is connected to all the others, which are in fact disconnected components. A class of approximate Schur complement preconditioners is also considered. Theoretical and numerical results are given for a model problem.

In the second part we consider the application of the previous domain decomposition preconditioners to the compressible Euler equations. The discretisation procedure, based on multidimensional upwind residual distribution schemes, is outlined. We introduce a framework that combines non-linear system solvers, Krylov accelerators, domain decomposition preconditioners, as well as mesh adaptivity procedures. Several numerical tests of aeronautical interest are carried out in order to assess both the discretisation schemes and the mesh adaptivity procedures.

In the third part we consider the parallel aspects inherent in the numerical solution of the compressible Euler equations on parallel computers with distributed memory. All the main kernels of the solution algorithm are analysed. Many numerical tests are carried out, with the aim of investigating the performance of the domain decomposition preconditioners proposed in the first part of the thesis, in the applications addressed in the second part.

Version abrégée

L'objet de cette thèse est la définition de préconditionneurs parallèles performants basés sur une approche de décomposition de domaines ainsi qu'à leurs applications à la résolution des équations d'Euler stationnaires compressibles.

Dans la première partie de la thèse, nous proposons et analysons différents préconditionneurs de décomposition de domaines avec recouvrement (méthodes de type Schwarz) et non-recouvrement (méthodes basées sur le complément de Schur). Pour les premiers, nous utilisons des méthodes à deux niveaux avec une formulation algébrique de l'espace grossier. Cette approche possède plusieurs propriétés intéressantes qui ne sont pas toujours mises en évidence par des méthodes à deux niveaux standards. Pour les seconds, nous introduisons une classe de préconditionneurs basés sur une décomposition particulière du domain de calcul. Nous considérons également une classe de préconditionneurs avec une méthode de Schur approchée. Nous donnons des résultats théoriques et numériques pour un problème modèle.

Dans la deuxième partie de la thèse, nous appliquons ces préconditionneurs de décomposition de domaines aux équations d'Euler compressibles. Nous décrivons la procédure de discrétisation basée sur des schémas distributifs de type *multidimensional upwind*. Nous introduisons un cadre qui combine des schémas non-linéaires, des solveurs itératifs de type Krylov, des préconditionneurs de décomposition de domaines ainsi que des maillages adaptatifs.

Dans la troisième partie de la thèse, nous considérons les aspects parallèles des préconditionneurs proposés dans la première partie de la thèse. Des résultats numériques sont présentés pour des cas test issus de l'aéronautique.

Acknowledgements

I am very much indebted to Prof. Alfio Quarteroni, who provided me the opportunity to carry our this thesis at the EPFL and directed this work. I must thank him for his careful supervision, his continuous and valuable support in every circumstance, his advice and numerous suggestions.

A special thank is due to Prof. Luca Formaggia, who has always been present and ready to give me suggestions and indications, and also for the patience he showed in explaining to me the subtilities of (parallel) programming.

I would like to thank Dr. Pénélope Leyland, who furnished part of the financial support, for the help she gave me at the beginning of my PhD thesis, and for kindly allowing me to use the adaptivity modules she has developed in the framework the European IDeMAS project.

Prof. Alberto Valli is particularly acknowledged for many helpful discussions, especially about nonoverlapping domain decomposition methods. Dr. Andrea Toselli is acknowledged for his precious suggestions about overlapping domain decomposition methods.

I would like to thank Prof. Herman Deconinck (the responsible of the IDeMAS project, under which part of the work of this thesis was conducted), and Prof. Michel Deville, who agreed to be members of the jury.

I would also like to thank all my colleagues and friends at the EPFL, who contributed to making this period a very pleasant one.

Last but not least, I would like to thank my parents, Carlo and Enrica, and my sister Elena, who have always encouraged me.

Part of the work of this thesis was conducted in the course of the European project IDeMAS (contract number BRPR-CT97-0591). The OFES (Office Fédérale de l'Education et de la Science) and the EPFL are acknowledged for the financial support.

Contents

I have no satisfaction in formulae
unless I feel their numerical magnitude.

Lord Kelvin (William Thomson)

1 Introduction

1.1 Motivation

In this thesis we consider parallel algorithms for the solution of the three dimensional steady compressible Euler equations (CEE), which describe the conservation of mass, momentum, and energy of inviscid flows.

The volume of literature of the last decade about the CEE and, more generally, about Computational Fluid Dynamics (CFD), is impressive. Important improvements have been achieved, due to the development of both computer hardware and numerical schemes. The first leads to powerful supercomputers that employ parallel architectures, while the second leads to efficient and robust numerical methods, that must be well-suited for the parallel design of the supercomputers.

To obtain a numerical solution of the CEE, the equations have to be discretized both in space and time. For space discretisation, there are four main classes of numerical methods: finite difference (FD) methods, based on the direct discretisation of the derivatives, finite element (FE) methods, based on the weak formulation of the equations, finite volume (FV) methods, based on the Gauss theorem on each cell (or volume), and spectral methods, which use high-order polynomial approximations.

The amount of literature about the CEE is impressive. Generally, for their solution, FV methods are used; see for instance [Lev02] and the references therein. More recently, some approaches based on FE procedures have been proposed. Among them, in this thesis we will consider *multidimensional upwind residual distribution* (MURD) schemes, also called *fluctuation splitting* schemes, which have been studied during the last 20 years. These schemes

were introduced by P.-L. Roe at the beginning of the 1980's and have been subsequently developed mainly by H. Deconinck and co-workers.

The starting point is a reinterpretation of 1D FV schemes using the concept of fluctuation. Considering the continuous piecewise-linear data representation classically used in FE methods rather than the discontinuous cell-wise constant reconstruction used in FV methods, the flux difference $(F_{i+1} - F_i)$ between two nodes $i+1$ and i is reinterpreted as the flux balance (or fluctuation or residual) over the interval (or element) $[i, i+1]$, which has to be distributed to the vertices of the element. This interpretation can be carried over to several dimensions, provided the fluctuation is now the flux balance over a simplex (triangle in 2D and tetrahedron in 3D) to which is associated a continuous piecewise-linear data representation. The multidimensional nature of the problem can be taken into account by distributing the residual to the vertices of the simplex rather than to the two neighbours of an edge.

MURD schemes 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, since only the nodes in the triangle are used in the evaluation of the fluctuation.

One of the most appealing aspect of MURD schemes is that they can be easily applied to unstructured grids. Nowadays, unstructured grids are an essential component for a successful use of numerical methods in industrial applications. In fact, these applications are often defined on complex shape domains. For such domains, the grid generation process is difficult and can be very time-consuming, unless completely unstructured grids are used. This is attractive as fully automatic unstructured tetrahedral grid generators have been developed. These generators can be directly coupled to CAD systems and incorporated within a unique software tool which can – at least in principle – provide a solution of the problem, without the intervention of the user, once the domain and the boundary conditions have been specified. Furthermore, unstructured grids can easily handle adaptation cycles. Grid adaptivity is particularly attractive in CFD, as it offers the possibility of resolving localised flow features in the most efficient and cost-effective manner.

Space discretisation is just one aspect of the numerical solution of the CEE. Their discretisation leads to a large system of non-linear equations, solved by iterative procedures, such as the Newton method or its variants. However, an exact Newton method is rarely optimal in terms of memory and CPU resources for large scale problems. Moreover, in general, a Newton method will not suffice, since initial estimates sufficiently near to the solution are not available. Therefore, the scheme is made more robust through a continuation scheme: a pseudo-time derivative term is added to the original non-linear system, whose root is now found as a steady-state solution of the time-dependent problem.

The continuation procedure requires, at each (pseudo)time level, the solution of a linear system whose matrix is the Jacobian (or a convenient approximation of it). Using a MURD scheme for the space discretisation, this matrix turns out to be sparse, non-symmetric, and particularly ill-conditioned, especially when the CFL number is "large."

For the solution of the linear system, one can use either direct or iterative solvers. The former are usually a better choice for small-size problems, since they are much less sensitive with respect to the conditioning of the problem. On the other hand, iterative solvers are commonly used in many applications, and in particular in CFD problems. Indeed direct solvers are prohibitively expensive for large-size problems. The substantially sequential algorithm makes efficient parallel implementations of direct solvers on modern parallel computers troublesome. Furthermore direct solvers require a lot of storage to perform the factorisation. As the dimension of the linear system increases, iterative solvers are very often the matter of choice.

Other reasons may suggest the use of iterative solvers. In an iterative solver, the system matrix A is not needed in explicit form but only a procedure to compute the matrix-vector product $A\mathbf{x}$ for a given vector $\mathbf{x}$ is required. Furthermore, prior knowledge about the solution can be introduced into the solution process via the initial approximation. Finally, iterative methods can be terminated when the desired accuracy is achieved. In fact, the linear system under consideration is the discrete version of a system with an infinite number of degrees of freedom (which itself is only a model of the real system). It does not make any sense to compute the solution of the linear system with accuracy higher than the discretisation error. Hence, they require only the employment of those resources (time, memory) which are necessary to obtain the required accuracy.

Unfortunately, the performance of iterative solvers depends strongly on the spectral properties of the linear system matrix, and in particular its condition number. If the matrix is ill-conditioned, one is forced to devise an efficient preconditioner. A preconditioner is an operator that transforms the initial linear system into another one having the same solution, but better conditioned and hence easier to solve.

Many constraints must be satisfied in the definition of the preconditioner. Firstly, it must be inexpensive to compute and apply, in terms of memory usage and computation time. If possible, it should be general-purpose. Even more importantly, it should be optimal, that is its performance should be independent of the problem size. Finally, since we are interested in parallel applications, it should be scalable, that is its performance should not degradate as the number of processors increases.

A class of preconditioners, well-suited for parallel computations, is 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, $i = 1, \ldots, M$, called subdomains, such that $\cup_{i=1}^{M} \overline{\Omega_i} = \overline{\Omega}$. Next, the original problem can be reformulated within each subdomain Ω_i, of smaller size. This family of subproblems is coupled one to another 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 $\partial\Omega_i \setminus \partial\Omega$.

Note that the term DD can embrace a large variety of numerical schemes. Each subdomain may refer to a different physical region, modelled with different physical or numerical models.

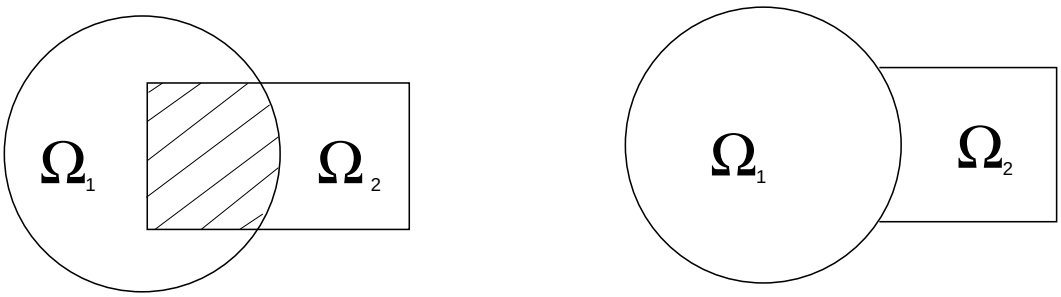


Figure 1.1: Example of overlapping (left) and non-overlapping (right) DD. Overlapping area is shaded.

This leads to the so-called heterogeneous DD; see [QV99, Chap. 8]. On the other hand, when each subdomain is modelled and discretized using the same equations and numerical model, one has homogeneous domain decomposition.

The class of homogeneous DD methods is rather large. A common classification divides these DD methods into two groups [CM94, SBG96, QV99].

In the first group, named after Schwarz, the computational domain is subdivided into *overlapping* subdomains (as depicted on the left of figure 1.1), 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. This method has been proposed by H.-A. Schwarz in 1870 to prove the existence of solution of elliptic PDEs on domain of complex shape, that is, a domain for which no exact solution of the problem is available [Sch70].

The second group uses *non-overlapping* subdomains (as shown on the right of figure 1.1). It is thus possible to decompose the unknowns into two sets: one formed by the unknowns on the interface between subdomains, the other formed by unknowns associated with nodes internal to the subdomains. At the algebraic level, one may then compute the Schur complement (SC) matrix by "condensing" the unknowns in the second set. The system is then solved by first computing the values of the interface unknowns and then solving the independent problems for the internal unknowns.

These DD methods are usually rather inefficient when used as solvers of the linear problem; however, they can be reformulated as efficient parallel preconditioners. Note that all the state-of-art DD preconditioners consist of *local* and *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. Should a preconditioner act only locally, then the iterative method that uses such a local preconditioner will have convergence rate that depends strongly on the number of the subdomains.

The global component is usually referred to as a "coarse space correction", since usually it is 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.

1.2 Thesis Outline

The main objective of this thesis is the investigation of parallel preconditioners based on the DD paradigm for problems defined on unstructured grids. Several new preconditioning techniques, for both overlapping and non-overlapping decompositions, are proposed.

A computational kernel we will frequently address is the FE problem

$$\begin{cases} \text{Find } u_h \in V_h \text{ such that:} \\ a\left(u_h, v_h\right) = f(v_h) \quad \forall v_h \in V_h, \end{cases} \tag{1.1}$$

$V_h(\Omega)$ being the space of FE function on a tessellation $\mathcal{T}_h$ of Ω, $a(\cdot,\cdot) : V_h \times V_h \to \mathbb{R}$ a bilinear function and $f : V_h \to \mathbb{R}$ a linear functional.

For an elliptic linear problem (like our model problem, see section 3.1) we have that $a(u_h, v_h)$ is a coercive bilinear form and u_h is the approximated solution, while for the CEE (see section 4.3) $a(u_h, v_h)$ should be regarded as the bilinear form expressing the Jacobian of the Euler system (after time and space discretisation) and u_h plays the role of the increment of the physical variables.

Chapter 2 gives some background on Krylov accelerators, how accelerators accommodate preconditioning, and outlines the main issues encountered in the definition of the preconditioner. The main classes of parallel preconditioners which are reported in the literature are addressed and compared.

Chapter 3 is mainly devoted to addressing the theoretical properties of DD preconditioners. The chapter is divided into two parts.

Part I, covered in section 3.2, focuses on overlapping Schwarz preconditioners. The main issue is the definition of the coarse space, an area of research that has been very prolific in recent years and is still so. The usual way to define the coarse matrix is based on the FE approximation of the variational problem, that is, the coarse operator is formed by discretising the problem at hand on a much coarser grid. However, the construction of a coarse grid and of the corresponding restriction and extension operators may be difficult or computationally expensive on general unstructured grids and arbitrary geometries. In addition, for a general, possibly non-convex domain, it is difficult to ensure that the boundary conditions are correctly represented on the coarse level. This way of defining the coarse space may be problematic also for the implementation details. To deal with a coarse grid may result in a much more complex – and therefore more difficult to validate and maintain – code.

To avoid these problems, we have resorted to a definition of the coarse space that does not require the explicit construction of a coarse grid. Starting from previous papers [PSFQ97,

JKMK00, LT00, SF01], we propose a general framework to build a coarse operator by using a procedure that operates directly on the matrix entries, based on the concept of aggregation. We consider a set of assumptions on the coarse basis functions. Theorems 3.2.4 and 3.2.5 represent the main results of this section. The theory here developed extends previous results presented in the literature. The influence of the geometrical properties of the aggregates and of the subdomains, and the influence of the smoother, have been kept separate in the final bound. This allows a better appreciation of the role of the three different phases of our smoothed aggregation algorithm. Numerical experiments are reported to illustrate the performance of the proposed methods.

In Part II, covered in section 3.3, the solution of the SC system is discussed. The section opens with a review of some results for SC-based methods and the preconditioners presented in literature. We analyse in detail the differences between element-oriented (EO) and vertex-oriented (VO) decompositions. In the former, the computational domain is decomposed so that each element of the grid belongs to a different domain, while in the latter the decomposition acts on the vertices of the grid, so that each vertex belongs to a different subdomain. The SC systems arising from the two approaches are clearly different. We present a new preconditioner arising from the VO decomposition, whose spectral properties are defined in proposition 3.3.3, as well as a class of preconditioners obtained from the approximate SC systems. Proposition 3.3.3 gives a theoretical bound for this latter class of preconditioners, whose idea is to approximate the computationally expensive part of the SC (i.e., the solution of the internal Dirichlet problems), and to use this approximate solver as a preconditioner for the unreduced system. The preconditioner obtained is numerically proved to be robust and shows good parallel properties. Preconditioners based on approximate Schur complement methods are also presented.

Chapter 4 introduces the CEE and their spatial discretisation, based on MURD schemes. In their current state, MURD schemes can be used for complex aerodynamic flows, both in the subsonic and transonic regime. 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 the definition and implementation of a scalable preconditioner and of a parallel grid adaptation technique, to improve the solution accuracy whilst optimising the computational resources.

For the solution of the large, sparse, non-symmetric linear system arising from the Newton method, we use Krylov techniques. In order to control the number of Krylov iterations and to obtain a scalable algorithm, we precondition them with DD-based preconditioners, of Schwarz or Schur type. The resulting framework, called ΨNKS, can deliver asymptotic rapid convergence through pseudo-transient continuation and inexact Newton methods, reasonable parallelisation for an implicit method through a Krylov solver, and high per-processor performance through attention to distributed memory and cache locality, especially through Schwarz (or Schur) preconditioners. The idea is to obtain (asymptotically) second-order convergence in time through the Newton iterations and a reasonable parallelism

due to the use of a Krylov solver complemented with a Schwarz-type preconditioner; see also [NAWK95, CKV97, GMTK00, GKKS01]. Combined with grid adaptation techniques this also offers a way to improve the quality of the solution whilst optimising the use of computational resources. The resulting framework is called $\alpha\Psi$NKS. To the author's knowledge, this chapter – and the corresponding paper [SL02] – represents the first application of a complete $\alpha\Psi$NKS scheme to the CEE using MURD schemes.

Numerical results for aeronautical applications are reported. Several 2D and 3D cases, of aeronautical interest, have been extensively studied: a NACA0012 profile, two full aircrafts (Falcon and X29), and a typical test case for aeronautical codes, the ONERA M6 wing.

In **Chapter 5**, we analyse the $\alpha\Psi$NKS framework from the point of view of its final implementation on a parallel computer. An efficient implementation of this framework requires a careful analysis of all its main steps. A distributed data structure, capable of handling in an efficient way all the main operations required by the $\alpha\Psi$NKS code, must be defined.

Several issues are addressed: how to implement the local solver, the data exchange, and the construction of the coarse components of the preconditioners. Also, the good parallel properties of the upwind methods and of the preconditioners proposed in chapter 3 are shown.

At the end of the chapter, numerical results using these preconditioners are presented for aerodynamic applications. The numerical results, obtained on distributed memory computers, using MPI as a message passing library [For95], confirm the applicability of the proposed methods to the compressible Euler equations.

1.3 Notation

Throughout this thesis, we will adopt the following notation. Capital italic Roman letters denote matrices and bold lowercase Roman letters denote vectors. The (i,j) component of the matrix A is denoted by $A_{i,j}$ or by $A(i,j)$, and the $i-$th component of the vector $\mathbf{v}$ is denoted by v_i or by $\mathbf{v}(i)$. Calligraph letters indicate linear operators. $\lambda_i(A)$ indicates the $i-$th eigenvalue of A, $\sigma(A)$ the set of all eigenvalues of A, and $\varrho(A)$ the spectral radius of A:

$$\varrho(A) = \max_{i=1,\dots,n} \{|\lambda_i(A)|\} .$$

In particular,

$$\lambda_{max}(A) = \varrho(A), \quad \lambda_{min}(A) = \min_{i=1,\dots,n} \{|\lambda_i(A)|\} .$$

We shall use $\langle \mathbf{x}, \mathbf{y} \rangle$ or $\mathbf{x} \cdot \mathbf{y}$ to denote the Euclidean inner product between two vectors $\mathbf{x}$ and $\mathbf{y}$, while $\| \cdot \|$ will denote the Euclidean norm.

For a symmetric positive definite matrix A, the symbol $\kappa(A)$ denotes the condition number:

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}, \tag{1.2}$$

while for a general matrix A the condition number is

$$\kappa(A) = |||A||| \cdot |||A^{-1}|||,$$

$|||\cdot|||$ being a matrix norm [QSS00].

For a matrix P symmetric and positive definite, we shall indicate using a slight change of notation the condition number of the preconditioned problem $\kappa(P^{-1}A)$ as

$$\kappa(P^{-1}A) = \frac{\lambda_M}{\lambda_m},$$

where λ_M and λ_m are two positive constants such that

$$\lambda_m \langle P\mathbf{x}, \mathbf{x}\rangle \leq \langle A\mathbf{x}, \mathbf{x}\rangle \leq \lambda_M \langle P\mathbf{x}, \mathbf{x}\rangle, \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

This definition follows from the identity $\sigma(P^{-1}A) = \sigma(P^{-1/2}AP^{-1/2})$.

Finally, we introduce here the definition of the adjacency graph of the matrix A. An (undirected) graph G is a pair (V, E), where V is a finite set of points called vertices and E is a finite set of edges. An edge $e \in E$ is an unordered pair (v, w), where $v, w \in V$. An edge indicates that vertices v and w are connected. By definition, the adjacency graph of A is a graph $G = (V, E)$, whose vertices are in the index set $V = \{1, 2, \ldots, n\}$ and whose edges are defined as $(i, j) \in E$ if and only if $A_{i,j} \neq 0$.

2

Setting the Ground for the Solution of Linear Systems

THE TERM ITERATIVE method refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear systems at each step.

One can distinguish between two different aspects of the iterative solution of a linear system. The first one in the particular acceleration technique for a sequence of iterations vectors, that is a technique used to construct a new approximation for the solution, with information from previous approximations. This leads to specific iteration methods. The second aspect is the transformation of the given system to one that can be more efficiently solved by a particular iteration method. This is called *preconditioning*. A good preconditioner improves the convergence of the iterative method, sufficiently to overcome the extra cost of its construction and application. Indeed, without a preconditioner the iterative method may even fail to converge in practice.

In this chapter, iterative methods for the parallel solution of large sparse linear systems are presented. The main lines of iterative methods are recalled, and the main aspects of parallel preconditioning are introduced.

2.1 Generalities on Iterative Methods

Iterative methods provide the solution to a linear system of type

$$A\mathbf{u} = \mathbf{f}, \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$, $\mathbf{u}$ and $\mathbf{f} \in \mathbb{R}^n$, as the limit of a sequence $\{\mathbf{u}_m\}$. Typical situations in which iterative methods are involved are large sparse problems, or band matrix problems, when the band itself is sparse, making decomposition algorithms difficult to implement.

Classical iterative methods are generally analysed in term of splittings of the matrix A. Given a non-singular matrix P, the splitting $A = P - N$ defines an iterative method as follows: starting from a tentative solution $\mathbf{u}_0$, compute $\mathbf{u}_{m+1}$ solving the linear system in P

$$P\mathbf{u}_{m+1} = N\mathbf{u}_m + \mathbf{f}, \tag{2.2}$$

or, equivalently,

$$\mathbf{u}_{m+1} = B\mathbf{u}_m + \mathbf{c}, \tag{2.3}$$

where $B = P^{-1}N$ is the iteration matrix and $\mathbf{c} = P^{-1}\mathbf{f}$. P should be an "inexpensive" matrix. For example, in the Jacobi method, P is the diagonal part of A, and in the Gauss-Seidel method, P is the lower part of A; see for instance [Var00, Chapters 3 and 4].

Since B does not depend on the iteration count m, these iterative methods are called *stationary*. The method (2.3) converges for any $\mathbf{u}_0$ if and only if $\rho(B) = \rho(I - P^{-1}A) < 1$.

Note that (2.2) can be rewritten in an equivalent form as

$$P\left(\mathbf{u}_{m+1} - \mathbf{u}_m\right) = \mathbf{r}_m,$$

where $\mathbf{r}_m \equiv \mathbf{f} - A\mathbf{u}_m$ is the *residual* of the linear system at the $m-$th time step. This leads to the class of Richardson methods; see [Axe94, Saa96b, QSS00] for more details. Richardson methods contain all the main features of modern iterative methods: the scheme is composed by matrix-vector products, vector-vector products, and vector updates, plus the solution of a linear system in P, the *preconditioning matrix*.

In modern terms, classical iterative methods are seldom used in practice, whereas they are often employed as preconditioners of a more performing iterative procedure like the projection methods, covered in the following section.

2.1.1 Projection Methods

The idea of projection methods is to extract an approximate solution from a well-chosen subspace of $\mathbb{R}^n$. This subspace, say $\mathcal{V}_m$, will be of dimension m (generally $m \ll n$); therefore the candidate solution will be found out by imposing m constraints. A typical way of describing these constraints is to impose m (independent) orthogonality conditions. In general, given an approximate solution $\mathbf{u}_m$, the residual $\mathbf{r}_m$ is constrained to be orthogonal to m linearly independent vectors, which defines another subspace, say $\mathcal{W}_m$, of dimension m. $\mathcal{W}_m$ is called the subspace of constraints.

In all generality, we may define a projection method for the solution of (2.1) as a process which finds as approximate solution $\mathbf{u}_m$ by imposing the Petrov-Galerkin condition that $\mathbf{u}_m$

belongs to the affine space $\mathbf{u}_0 + \mathcal{V}_m$ and that the new residual vector be orthogonal to $\mathcal{W}_m$:

$$\begin{cases} \text{Find } \mathbf{u}_m \in \mathbf{u}_0 + \mathcal{V}_m \text{ such that} \\ \mathbf{f} - A\mathbf{u}_m \perp \mathcal{W}_m. \end{cases} \tag{2.4}$$

To obtain a matrix representation of problem (2.4), let $V_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ be a $n \times m$ (real) matrix whose column vectors form a basis of $\mathcal{V}_m$ and, similarly, $W_m = [\mathbf{w}_1, \ldots, \mathbf{w}_m]$ be a $n \times m$ (real) matrix whose column vectors form a basis of $\mathcal{W}_m$. Writing the approximate solution as

$$\mathbf{u}_m = \mathbf{u}_0 + V_m \mathbf{y}_m, \quad \mathbf{y}_m \in \mathbb{R}^m, \tag{2.5}$$

the orthogonality condition leads to the system of equations

$$W_m^T (\mathbf{f} - A\mathbf{u}_0 - AV_m \mathbf{y}_m) = \mathbf{0},$$

and therefore

$$\mathbf{u}_m = \mathbf{u}_0 + V_m \left(W_m^T A V_m\right)^{-1} W_m^T \mathbf{r}_0.$$

Clearly, this approximate solution $\mathbf{u}_m$ is defined only when the matrix $W_m^T A V_m$ is non-singular, which, for arbitrarily chosen spaces $\mathcal{V}_m$ and $\mathcal{W}_m$ is not guaranteed to be true even though A is non-singular. It can be shown that $W_m^T A V_m$ is non-singular for any basis of V_m and W_m of $\mathcal{W}_m$ and $\mathcal{V}_m$, respectively, is A is positive definite and $\mathcal{W}_m = \mathcal{V}_m$, or if A is non-singular and $\mathcal{W}_m = A\mathcal{V}_m$; see, for instance, [Saa96b, Chapter 5].

In modern iterative methods, the subspaces $\mathcal{V}_m$ and $\mathcal{W}_m$ are Krylov subspaces, that is subspaces spanned by vectors of the form $\mathbb{P}_m(A)\mathbf{v}$, where $\mathbb{P}_m(A)$ is a polynomial of degree m in A, and $\mathbf{v}$ is a vector. The Krylov subspace $\mathcal{V}_m$ may be defined as

$$\mathcal{K}_m = \mathcal{K}_m(A; \mathbf{r}_0) = \text{span}\left\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{m-1}\mathbf{r}_0\right\}.$$

The main classes of Krylov subspace accelerators fall into three categories, depending on the choice of $\mathcal{W}_m$:

- *The Ritz-Galerkin approach* ($\mathcal{W}_m = \mathcal{V}_m$): construct $\mathbf{u}_m$ for which the residual is orthogonal to the current space. When A is symmetric positive definite, this choice minimises the A norm of the error $\mathbf{u}_m - \mathbf{u}$. An example is the Conjugate Gradient (CG) method of Hestenes and Steifel [HS52]. See also [GO89] for an overview of CG methods;

- *The minimum residual approach* ($\mathcal{W}_m = A\mathcal{V}_m$): identify $\mathbf{u}_m$ which minimises the residual norm $\|\mathbf{f} - A\mathbf{u}_m\|_2$. Examples are the GMRES method of Saad and Schultz and its equivalents [SS86, Saa93, VV94];

- *The Petrov-Galerkin approach*: find $\mathbf{u}_m$ so that the residual is orthogonal to some other $m-$dimensional space. A possible choice is $\mathcal{W}_m = \mathcal{K}_m(A^T, \tilde{\mathbf{r}}_0)$, where $\tilde{\mathbf{r}}_0$ is a vector non-proportional to $\mathbf{r}_0$. This choice was designed for non-symmetric problems, and may provide a short-term recurrence relations for the Krylov subspace bases. Examples are Bi-CG method of Fletcher [Fle76], the CGS method of Sonneveld [Son89], the Bi-CGSTAB method of van der Vorst [vdV92], and the QMR method of Freund and Nachtigal [FN91], and the TFQMR method of Freund [Fre93].

As a general rule, we can say that if the matrix is symmetric positive definite, the CG method is the best choice, while for non-symmetric problems usually GMRES is the most stable of all methods. An appealing property of GMRES if that, in exact arithmetic, the method cannot breakdown[1] or, more precisely, it can only breakdown when it delivers the exact solution. However, GMRES it is also the most expensive of the presented method in terms of memory. A restarted GMRES can be employed to reduce memory requirements, at the price of possibly non-convergence of the scheme. This restarted GMRES is usually called GMRES(k), where k is the maximal dimension of the Krylov subspace. Bi-CGSTAB and TFQMR are a compromise between stability and required memory, while the convergence behaviour of CGS is normally too erratic to be used for our problems.

Many other methods have been proposed in literature, like FOM, ORTHOMIN, ORTHODIR, ORTHORES, MINRES, SYLMMLQ. For an exhaustive presentation the reader is addressed to [BBC+94, Saa96b, Gv97, Sv00].

2.2 Definition of the Preconditioner

The convergence of iterative methods depends on the spectral properties of the linear system matrix. The basic idea is to replace the original system (2.1) by

$$P^{-1}A\mathbf{u} = P^{-1}\mathbf{f}$$

(left-preconditioning), or by

$$AP^{-1}P\mathbf{u} = \mathbf{f}$$

(right-preconditioning), using a linear transformation P^{-1}, called preconditioner, in order to improve the spectral properties of the linear system matrix. In general terms, a preconditioner is any kind of transformation applied to the original system which makes it easier to solve.

It is not clear who coined the term "preconditioning" first. According to Golub and O'Leary [GO89], it may have been Turing. In the paper of 1952 by Lanczos [Lan52] the notion of polynomial preconditioning is clearly defined: "*The construction of the inverse matrix is equivalent to a linear transformation which transforms the given matrix into the unit matrix. The unit matrix can be conceived as the extreme case of a well-conditioned matrix whose*

[1] A breakdown occurs when a division by zero is encountered in the algorithm.

eigenvalues are all 1. We will ask for much less if we merely demand the transformation of the original system whose dispersion is moderate."

In a modern perspective, the general problem of finding an efficient preconditioner is to identify a linear operator P with the following properties:

1. P is a good approximation of A is some sense. Although no general theory is available, we can say that P should act so that $P^{-1}A$ is near to being the identity matrix and its eigenvalues are clustered within a sufficiently small region of the complex plane;

2. P is efficient, in the sense that the iteration method converges much faster, in terms of CPU time, for the preconditioned system. In other words, preconditioners must be selected in such a way that the cost of constructing and using them is offset by the improved convergence properties they permit to achieve;

3. P or P^{-1} can take advantage of the architecture of modern supercomputers, that is, can be constructed and applied in parallel environments.

It should be stressed that computing the inverse of P is not mandatory; actually, the role of P is to "preconditioning" the residual $\mathbf{r}_m$ through the solution of the additional system $P\mathbf{z}_m = \mathbf{r}_m$. This system $P\mathbf{z}_m = \mathbf{r}_m$ should be much easier to solve than the original system. Note that some recent preconditioning techniques aim to compute P^{-1} directly instead of P (see section 2.3.2).

The choice of P varies from "black-box" algebraic techniques which can be applied to general matrices to "problem dependent" preconditioners which exploit special features of a particular class of problems. Although problem dependent preconditioners can be very powerful, there is still a practical need for efficient preconditioning techniques for large classes of problems. Between these two extrema, there is a class of preconditioners which are "general-purpose" for a particular – although large – class of problems. These preconditioners are sometimes called "grey-box" preconditioners, since the user has to supply few information about the matrix and the problem to be solved.

Note that all the accelerators presented in section 2.1.1 require the preconditioner to be constant at each iteration. Other methods have been developed to consider non-constant preconditioners by different authors. Axelsson and Vassilevski [AV91] have proposed a Generalised Conjugate Gradient (GENCG) with variable preconditioning, Saad [Saa93] has proposed a scheme very similar to GMRES, called Flexible GMRES (FGMRES), and Var der Vorst and Vuik have published a GMRESR scheme [VV94].

FGMRES has probably received most attention, mainly because it is easy to implement: only the update direction in GMRES has to be preconditioned, and each update direction may be preconditioned differently. This means that only one line in the GMRES algorithm has to be adapted. The GENCG and GMRESR are slightly more expensive in terms of memory requirements and computations overhead per iteration step. In exact arithmetic the two methods produces the same results; however, GMRESR required only one matrix-vector

product per iterations. In GMRESR the residual vectors are preconditioned, and if this gives a further reduction then GMRESR does not breakdown. This give more control on the method with respect to FGMRES, even if this latter scheme is probably slightly more efficient.

2.2.1 Incomplete LU Preconditioners

A broad class of effective preconditioners is based on incomplete factorisation of the linear system matrix [MvdV77, Saa96b], and it is usually indicated as ILU. The ILU-type preconditioning techniques lie between direct and iterative methods and provide a balance between reliability and numerical efficiency.

The preconditioner is given in the factored form $P = \tilde{L}\tilde{U}$, with $\tilde{L}$ and $\tilde{U}$ being lower and upper triangular matrices. Solving with P involves two triangular solutions.

The incomplete LU factorisation of a matrix A can be described as follows. Let $A_0 = A$. Then, for $k = 2, \ldots, n$, we have

$$A_{k-1} = \begin{pmatrix} B_k & F_k \\ E_k & C_k \end{pmatrix}.$$

Thus, we can write the $k-$step of the Gaussian elimination in a block form as

$$A_{k-1} = \begin{pmatrix} I & 0 \\ E_k B_k^{-1} & I \end{pmatrix} \begin{pmatrix} B_k & F_k \\ 0 & A_k \end{pmatrix},$$

where $A_k = C_k - E_k B_k^{-1} F_k$. If B_k is a scalar, then we have the typical point-wise factorisation, otherwise we have a block factorisation. Pivoting, if it is necessary, can be accomplished by reordering A_k at every step.

To make the factorisation incomplete, entries as dropped in A_k, i.e. the factorisation proceeds with

$$\tilde{A}_k = A_k - R_k$$

where R_k is the matrix of dropped entries.

Dropping can be performed by position, for example, dropping those entries in the update matrix $E_k B_k^{-1} F_k$ that are not in the pattern of C_k. This simple ILU factorisation is known as ILU(0). Although effective, in some cases the accuracy of the ILU(0) may be insufficient to yield an adequate rate of convergence. More accurate factorisations will differ from ILU(0) by allowing some *fill-in*. The resulting class of methods is called ILU(f), where f is the level-of-fill. A level-of-fill is attributed to each element that is processed by Gaussian elimination, and dropping will be based on the level-of-fill. The level-of-fill should be indicative of the size of the element: the higher the level-of-fill, the smaller the elements. A simple model can be employed to justify to effectiveness of this approach is detailed in [Saa96b, Section 10.3.3].

Alternative dropping techniques can be based on the numerical size of the element to be discarded. Numerical dropping strategies generally yield more accurate factorisations with the

same amount of fill-in than level-of-fill methods. The general strategy is to compute an entire row of the $\tilde{L}$ and $\tilde{U}$ matrices, and then keep only the biggest entries in a certain number. In this way, the amount of fill-in is controlled; however, the structure of the resulting matrices is undefined. These factorisations are usually referred to as ILUT, and a variant which performs pivoting is called ILUTP.

Many other variants have been presented in literature; see for instance [Axe94, Saa96b, Cv97]. Latest developments concern multilevel methods, like ILUM [Saa96a] or BILUM [SZ99], or ARMS [SS02]. These approaches try to combine the "general-purpose" approach of ILU-type preconditioners with the superior convergence rate of multilevel methods, and, in particular, of (algebraic) multigrid methods (see section 2.3.3), using some idea from DD methods.

Remark 2.2.1. *As for complete factorisations, ILU methods depend on the order in which the variables are eliminated. Thus, the ordering of a matrix plays an important role in determining the quality of the ILU preconditioner. In [DM80] an extensive study is reported, although limited to symmetric matrices. In [BSvD99] the authors found numerically that the reverse Cuthill-McKee (RCM) achieved the best results for the majority of the test cases. Even when RCM was not the optimal, it still gave good results, that is, similar results to those provided by the best reordering.*

2.2.2 Preconditioners Induced by Iterative Procedures

Iterative methods can be used to construct preconditioners. This is especially true for stationary methods of section 2.1. Whether these methods are used in conjunction with a Krylov accelerator, they can effectively improve their convergence ratio.

Let us consider, for example, the use of the preconditioned CG method to accelerate a linear iterative solver in the form

$$\mathbf{u}_{m+1} = P^{-1}N\mathbf{u}_m + P^{-1}\mathbf{f}$$

with P symmetric and definite. This process converges provided that

$$\|P^{-1}N\|_A = \|I - P^{-1}A\|_A = q < 1.$$

By simple computations, it is possible to show that the condition number of the matrix $P^{-1}A$ can be bounded by

$$\kappa(P^{-1}A) \leq \frac{1+q}{1-q}.$$

Therefore, the application of preconditioned CG method with preconditioner P^{-1} yields a superior convergence factor, as

$$\frac{\sqrt{\kappa(P^{-1}A)}-1}{\sqrt{\kappa(P^{-1}A)}+1} \leq \frac{\sqrt{\frac{1+q}{1-q}}-1}{\sqrt{\frac{1+q}{1-q}}+1} \leq \frac{1-\sqrt{1-q^2}}{q} \leq q \quad \forall q \in (0,1).$$

Therefore, for a given splitting $A = P - N$, the CG method, with P as a preconditioner, can be used to effectively accelerate (2.3).

2.3 Parallel Preconditioners

Here we briefly discuss the main classes of parallel preconditioners proposed in the literature. We start with polynomial preconditioners in section 2.3.1; then, we briefly outline sparse approximate inverses (SPAI) preconditioners in section 2.3.2, multigrid methods in section 2.3.3. DD preconditioners, which are the main subject of this thesis, will be covered in details in chapter 3.

2.3.1 Polynomial Preconditioners

A first class of (parallel) preconditioners is based on polynomial preconditioners. The main motivation for considering polynomials in A is that the matrix-vector product is often more parallelisable than other parts of the solver like, for instance, the vector-vector product. Hence, a preconditioner which makes use of matrix-vector product only (plus local vector update) should be expected to have interesting parallel properties. The main problem is to find effective low degree polynomials $\mathbb{P}_k(A)$, so that the iterative solver can be applied to $\mathbb{P}_k(A)A\mathbf{u} = \mathbb{P}_k(A)\mathbf{f}$.

One approach for obtaining polynomial preconditioner is to use low order terms of a Neumann expansion of $(I - B)^{-1}$, if A can be written as $A = I - B$ (with $\rho(B) < 1$). More general polynomial preconditioners have also been proposed, and they are usually shifted Chebyshev polynomials over intervals that are estimated from the iteration parameters of a few steps of the unpreconditioned solver, or from other spectral information.

2.3.2 Sparse Approximate Inverse Preconditioners (SPAI)

Rather than constructing an approximation P of A, an alternative is to build $M = P^{-1}$ which approximates A^{-1} directly. When M is available, no inversion or solve operations will be required on the preconditioning stage. Preconditioners based on this technique are called "explicit" preconditioners, while those based on the former are called "implicit."

To approximate A^{-1} is a difficult task since this matrix is in general dense, and moreover the construction should be done in parallel. One possible way to operate is to minimise the Frobenius norm of the residual matrix $||I - AM||_F$. An important feature of this objective function is that it can be decoupled as the sum of squares of the 2-norm of the n individual column:

$$||I - AM||_F^2 = \sum_{j=1}^{n} ||\mathbf{e}_j - A\mathbf{m}_j||_2^2, \tag{2.6}$$

in which $\mathbf{e}_j$ and $\mathbf{m}_j$ are the $j-$th column of the identity matrix I and of matrix M, respectively. Hence, one can minimise the individual functions $||\mathbf{e}_j - A\mathbf{m}_j||_2^2$. This approach has been

proposed by Grote and Huckle [GH97]. Of course, the exact inverse will be found if no restriction is placed on M. Usually, a sparsity pattern (that is, the set of non-zero indexes) for M is prescribed (or other dropping strategies as in ILU preconditioners), or minimisation is performed using an iterative method like GMRES, implemented with sparse matrix-vector and sparse vector-vector operators.

A notable disadvantage of this approach is that it is difficult to assess in advance whether or not the resulting matrix will be non-singular. An alternative is to seek a two-sided approximation, i.e. a pair of L and U respectively lower triangular and upper triangular, which attempt to minimise the objective function

$$\|I - LAU\|_F^2.$$

Gould and Scott [GS98] indicate that the SPAI preconditioner may be a good alternative to ILU, but it is more expensive to compute both in terms of time and storage, at least if computed sequentially. Thus, the use of SPAI preconditioner can be of interest when used for several right-hand sides.

References about SPAI methods can be found in [BMT96, BT98, Zha98, Cho00]. For a comparative study of various SPAI preconditioners we refer to [BT99].

2.3.3 Multigrid Methods

A very important class of methods, mainly developed during the 80's, is the class of multigrid methods (MG) as solution methods (for both linear and non-linear systems).

Multigrid methods require the operator to be defined on a sequence of coarser grids, an iterative method that evolves the solution (called a smoother) and interpolation operators that transfer information between the grids. The principle behind the algorithm is that the high-frequency errors can be efficiently solved on the fine grid, while the low-frequency are treated on the coarser one, where there frequencies manifest themselves as high-frequencies. Geometric multigrid (GMG) methods cannot be applied without the existence of an underlying grid (this is their major limitation). This led to the development of algebraic multigrid method (AMG), initiated by Ruge and Stüben [RS85]. In AMG, both the matrix hierarchy and the prolongation operators are constructed just from the stiffness matrix. Since the automatic generation of a grid-hierarchy for GMG and especially the proper assembly of all components would be a very difficult task for unstructured problems, the automatic algebraic construction of a virtual grid is a big advantage.

More recently, a variant called AMGe has been proposed in [BCF$^+$00]. AMGe assumes that the element matrices is available, and then determines local representations of the smooth error components which provide the basis for constructing effective interpolation operators.

The literature about multigrid methods is astonishing. We refer the reader to Brandt [Bra86], Wesseling [Wes92], and Hackbusch [Hac94]. Recent developments can be found in [CFvH$^+$00, Stü01]. As regards CFD, see, for instance, [SLD92, VM94, Mav96, Mav98,

Mav00, CL00, Mav02]. Results concerning MURD schemes can be found within the IDeMAS project [IDe01].

Remark 2.3.1. *A comparison between multigrid methods and CG methods with preconditioning based on domain decomposition ideas is given in [HJ98] for non-linear boundary value problems arising from finite element discretisation of magnetic field computations and solid mechanics problems. The authors found the DD methods more scalable than the others, even if the geometric multigrid preconditioners has appeared as the more appropriate in term of CPU times.*

2.4 Concluding Remarks

Although the past years have seen the development of a very lively area of research on iterative methods, probably their evolution has come to a (temporary) end. The main challenge faced now is the solution of very large systems for which iterative methods are often the only way to obtain a solution. For these problems the choice of a good preconditioner can affect the computational time more substantially than the choice of the Krylov accelerator itself. In fact, for a good preconditioner there is not much difference between the various Krylov subspace methods. Clearly, this shifts the attention to the identification and construction of good preconditioners.

It is very difficult to construct efficient preconditioners for a given class of problems, and even more if these also have to be parallel.

The first attempts in the definition of parallel preconditioners aimed at bringing minimal changes to existing preconditioners, therefore, ILU-based preconditioners were made parallel using an approach – termed "level-scheduling" or "wavefront" approach, which tried to unravel parallelism from the forward and backward solves. However, these techniques were soon viewed as having a limited potential, since ILU-based preconditioners are also very sequential in nature.

For this reason many researchers tried to develop parallel preconditioners based on new methods, or to reinterpret old schemes (like DD methods). Polynomial preconditioners have a performance relative to existing alternatives that was not too convincing, especially when using a small number of processors. Moreover, it is difficult to find a good polynomial in the case of indefinite matrices. As a result, current interest for these kinds of preconditioners is almost vanished, and there has been a shift of focus toward methods like MG, DD, and SPAI.

Despite some partial success, SPAI are still in their infancy, especially for their application in parallel environments.

As regards MG methods, we can say that, in general, they can be extremely efficient when they work – far more efficient than preconditioned Krylov methods. However, their efficiency relies essentially on the inter-level restriction and prolongation operators, the choice of which will vary for one application to the next. Full MG efficiency can only be achieved for problems associated with certain types of partial differential equations defined on regularly structured

domains which have sufficient regularity. Though AMG methods have been designed to deal with more general problems, the success of such methods and the type of problems solved are still limited, and more in general it is difficult to define multigrid algorithms which are capable of efficiently solving general unstructured sparse linear systems.

On the contrary, DD techniques have been successful for large class of problems. From the theoretical point of view, scalable and optimal algorithms can be defined, as described in chapter 3. From the point of view of the final application, instead, the DD approach may be seen not only as an effective way to define parallel preconditioner, but also as an approach to decompose the data among the processors. In chapter 5, we analyse all the main computational kernels of a solution algorithm for the compressible Euler equations, the Newton-Krylov-Schwarz (or -Schur) framework, and we show how DD can lead to an efficient data structure, which cluster on each processor the data corresponding to a given subdomain, and then use these data in a computationally and mathematically efficient way.

3

Theoretical Properties of Domain Decomposition Preconditioners

THE THEORETICAL PROPERTIES of domain decomposition preconditioners can be extensively analysed for model elliptic problems. Well-understood and powerful mathematical theories can be developed for this model problem. As a result, sharp bounds on the condition number can be derived. This is true for both overlapping and non-overlapping methods.

First, the model problem is detailed in section 3.1. Then, the remaining part of the chapter is divided into two parts. Section 3.2 is devoted to overlapping (Schwarz) methods, and in particular to the definition of a coarse space based on the concept of aggregation. Section 3.3 considers non-overlapping (Schur complement) based methods for element-oriented and vertex-oriented decompositions.

3.1 Model Problem

Let Ω be a bounded domain in $\mathbb{R}^d, d = 2, 3$, with a piecewise smooth boundary $\partial\Omega$. We will consider the model problem

$$\left\{ \begin{array}{rcll} \mathcal{L}w & = & f & \text{in } \Omega \\ w & = & g & \text{on } \partial\Omega \end{array} \right. \tag{3.1}$$

where

$$\mathcal{L}v = -\sum_{r,s=1}^{d} \frac{\partial}{\partial x_r} \left(\alpha_{r,s} \frac{\partial v}{\partial x_s} \right)$$

with the coefficient matrix $\alpha_{r,s}$ uniformly positive definite and bounded. Problem (3.1) can be restated in terms of an equivalent variational Galerkin problem:

$$\begin{cases} \text{Find } u \in V \text{ such that} \\ a(u,v) = F(v) \quad \forall v \in V \end{cases} \tag{3.2}$$

where $V = H_0^1(\Omega)$,

$$\begin{aligned} a(u,v) &= \int_\Omega \sum_{r,s=1}^{d} \alpha_{r,s} \frac{\partial u}{\partial x_s} \frac{\partial v}{\partial x_r}, \\ F(v) &= (f,v) - a(\mathcal{R}g, v), \end{aligned}$$

where $\mathcal{R}g$ is a prolongation of g and $u = w - \mathcal{R}g$; see [QV94]. Throughout this chapter, we will always suppose that $a(\cdot,\cdot)$ is symmetric and coercive.

The finite element discretisation reformulates problem (3.2) as problem (1.1), where V_h is a finite-dimensional subspace of V and

$$u_h(\mathbf{x}) = \sum_{i=1}^{n} u_i \varphi_i(\mathbf{x}),$$

with $n = \dim(V_h)$. Here and in the following, $\{\varphi_i, i = 1, \ldots, n\}$ represents the set of standard (compact-stencil) finite element basis functions of V_h, and h the maximum diameter of the elements of the triangulation $\mathcal{T}_h$ of the shape-regular domain Ω.

The corresponding algebraic formulation of problem (1.1) gives rise to equation (2.1), where $\mathbf{u} = [u_1, \ldots, u_n]$ is the vector of unknowns, $f_i = F(\varphi_i)$, and the entries of A, called the stiffness matrix, are given by

$$A_{i,j} = a(\varphi_j, \varphi_i) = \sum_{r,s=1}^{d} \int_\Omega \alpha_{r,s} \frac{\partial \varphi_i}{\partial x_s} \frac{\partial \varphi_j}{\partial x_r} d\Omega.$$

3.2 Part I: Schwarz Preconditioners

Schwarz methods move from partitioning the domain Ω into overlapping subdomains Ω_i. The solution on the original domain, belonging to the space V_h, is obtained by an iterative procedure involving at each step solutions on subspaces V_i, defined on the subdomains Ω_i. To obtain scalable methods, a further space V_0 is considered. V_0 represents a discretisation of the original problem on the complete domain, although its dimension is remarkably smaller than that of V_h. Usually, V_0 is defined by means of a grid which is coarser than the grid used to define V_h. For our model problem, this may lead to a scalable and optimal preconditioner – depending on the definition of the overlap among the subdomains.

The definition of the coarse grid can be difficult or computationally expensive for 3D unstructured grids on complex-shape domains. To overcome this difficulty, we have considered

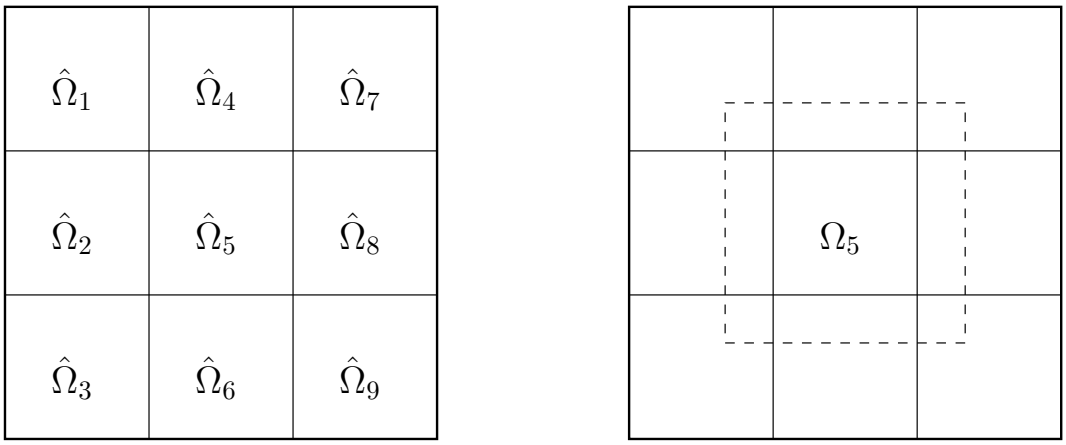


Figure 3.1: Subdivision of a 2D rectangular region Ω partitioned into 9 non-overlapping regions $\hat{\Omega}_i$ (on the left), and the extended subdomain Ω_5 (on the right).

aggregation (or *agglomeration*) procedures, an algebraic approach which does not demand for a geometrical definition of the grid. First, the differential problem is discretized on the fine grid, giving rise to a linear problem with a matrix, say A. Then, the matrix corresponding to the discretisation of the differential operator on the coarse grid is built starting from the entries of A.

Part I is organised as follows. Sections 3.2.1 and 3.2.2 introduce the one-level Schwarz preconditioner. In section 3.2.3 we present the aggregation procedure. Some bounds for the resulting preconditioners and their proofs are given in sections 3.2.4, 3.2.5, 3.2.6, and numerical results are reported in section 3.2.7. Finally, section 3.2.8 draws some concluding remarks.

3.2.1 One-level Schwarz Preconditioners

The simplest Schwarz preconditioner is the one-level preconditioner, defined as follows. We divide Ω into M subdomains $\{\Omega_i\}_{i=1}^M$ such that $\cup_{i=1}^M \overline{\Omega_i} = \overline{\Omega}$ and with an overlap width of the order of at least $\delta = \xi h$, $\xi \in \mathbb{N}^+$. ($\xi = 1$ corresponds to *minimal overlap*, that is, in that case we mean that the overlap among the subdomains is of one element.) From an algorithmic point of view, we can think to subdivide Ω into M non-overlapping subdomains $\hat{\Omega}_i$, and then extend each of them to Ω_i adding all layers of elements in Ω within a distance δ from $\hat{\Omega}_i$. See figure 3.1 for an illustration of a 2D rectangular region Ω partitioned into 9 non-overlapping regions $\hat{\Omega}_i$ (on the left) and an extended subdomain (on the right).

By notation, let H be the linear size of the subdomains, $H = \max\{\mathrm{diam}(\Omega_i)\}$. We assume that all subdomains have comparable size, i.e., $C_1 H \leq \mathrm{diam}(\Omega_i) \leq C_2 H$, with C_1 and C_2 two positive constants independent of H.

Consider the variational formulation of the model problem (1.1) and its corresponding

discrete part (2.1). V_h is the standard finite element space spanned by the basis functions associated to the nodes in $\Omega \setminus \partial\Omega_i$. Let $V_i \subset V_h$ be the finite element space spanned by the basis function associated to the nodes in $\Omega_i \setminus \partial\Omega_i$, and let n_i be the dimension of V_i. Moreover, let $\mathcal{R}_i$ be the restriction operator $V_h \to V_i$ and R_i the associated $n_i \times n$ matrix. R_i^T is the injection map, more precisely R_i^T is a $n \times n_i$ matrix whose action extends by zero a vector with nodal values in $\Omega_i \setminus \partial\Omega_i$.

It is now possible to define the local bilinear forms

$$a_i(\cdot,\cdot) : V_i \times V_i \to \mathbb{R}, \quad i = 1, \dots, M$$

which are in general an approximation of $a(\cdot,\cdot)$ (yet often it coincides with a). Should $a(\cdot,\cdot)$ be coercive, then $a_i(\cdot,\cdot)$ is coercive too. This guarantees that the corresponding algebraic matrix A_i is non-singular. In the case of $a_i(\cdot,\cdot) = a(\cdot,\cdot)$, the algebraic counterpart of $a_i(\cdot,\cdot)$ is the matrix

$$A_i = R_i A R_i^T.$$

Let the non-singular symmetric matrix $\tilde{A}_i$ be either A_i itself or an approximation of it. Then, the one-level Schwarz preconditioner can be written as

$$P_S^{-1} = \sum_{i=1}^{M} \underbrace{R_i^T \tilde{A}_i^{-1} R_i}_{B_i}. \tag{3.3}$$

Remark 3.2.1. *Non-symmetric preconditioners can be obtained using different restriction and prolongation operators. A typical choice consists of using $R_{i,0}^T$ instead of R_i^T, where $R_{i,0}^T$ represents the restriction operator for the minimal overlap case. This preconditioner is called RAS [CS99, FS01].*

Preconditioner 3.3 is not scalable: its convergence rate deteriorates as the number of subdomains increases (i.e., H decreases), as stated by the following theorem.

Theorem 3.2.1. *Let $\tilde{A}_i = A_i$. Then, there exists a positive constant C independent of H and h (but possibly dependent on the operator coefficients) such that*

$$\kappa(P_S^{-1} A) \leq C \frac{1}{H\delta}.$$

Proof. See [DW90]. □

3.2.2 Block-Jacobi Preconditioners

In the case of minimal overlap, the one-level Schwarz preconditioner may be seen as a block-Jacobi preconditioner, whose blocks have size n_i. In this particular case, it is possible to derive a bound for the condition number which does not rely on the existence of the bilinear form $a(\cdot,\cdot)$.

Let ω be the set of vertices of the graph associated to A, and $\omega_i, i = 1, \ldots, M$, non-overlapping sets of ω (that is, a node of ω is included in only one set ω_i). We indicate by $W_i = \text{span}\{\mathbf{e}_j, j \in \omega_i\}$ where $\mathbf{e}_j$ is the canonical $j-$th basis vector of W. By construction, $W = W_1 \times W_2 \times \ldots \times W_M$. The block-Jacobi preconditioner P_{BJ} can be formally defined as

$$P_{BJ}^{-1} = \sum_{i=1}^{M} R_i^T C_i^{-1} R_i.$$

Here, R_i the projector from W into the subspace W_i and $C_i \in \mathbb{R}^{n_i \times n_i}$ are square matrices.

Remark 3.2.2. *The Schwarz preconditioner of section 3.2.1 may be seen as an extension of the BJ preconditioner. In fact, to each subdomain Ω_i we can associate a set ω_i composed by the nodes internal to the subdomain. Hence, the BJ preconditioner corresponding to the non-overlapping sets ω_i may actually be seen as a minimal overlap among the subdomains Ω_i (see figure 3.2 and section 3.3.2).*

Let A be partitioned consistently with W_i:

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,M} \\ \vdots & \ddots & \vdots \\ A_{M,1} & \cdots & A_{M,M} \end{bmatrix}. \tag{3.4}$$

Then, the preconditioner P_{BJ} can be written as

$$P_{BJ} = \begin{bmatrix} C_1 & & 0 \\ \vdots & \ddots & \vdots \\ 0 & & C_M \end{bmatrix}. \tag{3.5}$$

In the following, we will assume that A is a symmetric positive definite matrix, as well as the $C_i, i = 1, \ldots, M$.

Property 3.2.1. *We suppose that, for $i, j = 1, \ldots, M$, there exist three constants α_i, β_i, and $\varepsilon_{i,j}$ such that*

a. $\alpha_i \langle A_{i,i}\mathbf{x}_i, \mathbf{x}_i\rangle \leq \langle B_i\mathbf{x}_i, \mathbf{x}_i\rangle \leq \beta_i \langle A_{i,i}\mathbf{x}_i, \mathbf{x}_i\rangle \quad \forall \mathbf{x}_i \in W_i$;

b. $\langle A_{i,i}\mathbf{x}_i, \mathbf{x}_i\rangle \leq \varepsilon_{i,j} \langle A_{j,j}\mathbf{x}_j, \mathbf{x}_j\rangle \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in W_i \times W_j.$

Property 3.2.1.a defines the quality of the approximation B_i of $A_{i,i}$, while property 3.2.1.b evaluates if the matrix A is properly block-wise scaled.

The following lemma holds.

Lemma 3.2.1. *The condition number of $P_{BJ}^{-1}A$ can be bounded as follows:*

$$\kappa\left(P_{BJ}^{-1}A\right) \leq \max_i \left\{ \left(\frac{\beta_i}{\alpha_i} \left(1 + \frac{1}{2}(1 + \max_j \varepsilon_{i,j})(M-1) \right) \right) \frac{\lambda_{max}(A_{i,i})}{\lambda_{min}(A)} \right\}$$

where α_i, β_i, and $\varepsilon_{i,j}$ are the constants of the property 3.2.1.

Proof. To find a bound for the condition number of $P_{BJ}^{-1}A$, we have to characterise the extreme values of the generalised eigenvalue problem

$$\lambda P_{BJ}\mathbf{x} = A\mathbf{x}\ , \quad \mathbf{x} \in W,$$

i.e., the extrema of the Rayleigh quotients

$$RQ(P_{BJ}, A) = \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T P_{BJ}\mathbf{x}}, \quad \mathbf{x} \in W. \tag{3.6}$$

Let's consider first the upper bound of $RQ(P_{BJ}, A)$. Because of the block-structure (3.4), each vector $\mathbf{x} \in W$ can be written as $[\mathbf{x}_1, \ldots, \mathbf{x}_M]$, with $\mathbf{x}_k \in W_k$. We have:

$$\begin{aligned}
\frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T P_{BJ}\mathbf{x}} &= \frac{\sum_i \left\{\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \sum_{j\neq i}\mathbf{x}_i^T A_{i,j}\mathbf{x}_j\right\}}{\sum_k \mathbf{x}_k^T B_k \mathbf{x}_k} \\
&\leq \frac{\sum_i \left\{\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \sum_{j\neq i}\left(\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \mathbf{x}_j^T A_{j,j}\mathbf{x}_j\right)/2\right\}}{\sum_k \mathbf{x}_k^T B_k \mathbf{x}_k} \\
&\leq \frac{\sum_i \left\{\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \frac{1}{2}(M-1)\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \frac{1}{2}\sum_{j\neq i}\mathbf{x}_j^T A_{j,j}\mathbf{x}_j\right\}}{\sum_k \mathbf{x}_k^T B_k \mathbf{x}_k} \\
&\leq \frac{\sum_i \left\{[1 + (M-1)/2]\,\mathbf{x}_i^T A_{i,i}\mathbf{x}_i + \frac{\max_j \varepsilon_{i,j}}{2}(M-1)\mathbf{x}_i^T A_{i,i}\mathbf{x}_i\right\}}{\sum_k \mathbf{x}_k^T B_k \mathbf{x}_k} \\
&\leq \max_{i=1,\ldots,M}\left\{\frac{1 + (M-1)/2 + \max_j \varepsilon_{i,j}(M-1)/2}{\alpha_i}\right\}.
\end{aligned}$$

Thus,

$$\lambda_{max}(P_{BJ}^{-1}A) \leq \max_i \left\{\frac{1}{\alpha_i}\left(1 + \frac{1}{2}\left(1 + \max_j \varepsilon_{i,j}\right)(M-1)\right)\right\}. \tag{3.7}$$

For the lower bound, we have

$$\begin{aligned}
\lambda_{min}(P_{BJ}^{-1}A) = \min_{\mathbf{x}\in W}\frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T P_{BJ}\mathbf{x}} &\geq \frac{\lambda_{min}(A)\mathbf{x}^T\mathbf{x}}{\sum_i \mathbf{x}_i^T B_i \mathbf{x}_i} \\
&\geq \frac{\lambda_{min}(A)}{\max_i\{\beta_i \lambda_{max}(A_{i,i})\}}.
\end{aligned}$$

Thus, the resulting bound for the condition number is

$$\begin{aligned}
\kappa(P_{BJ}^{-1}A) &\leq \frac{\max_{i,j}\left\{\frac{1}{\alpha_i}\left(1 + \frac{1}{2}(1 + \max_j \varepsilon_{i,j})(M-1)\right)\right\}}{\lambda_{min}(A)} \max_i\{\beta_i\}\,\lambda_{max}(A_{i,i}) \\
&\leq \max_i\left\{\frac{\beta_i}{\alpha_i}\left(1 + \frac{1}{2}(1 + \max_j \varepsilon_{i,j})(M-1)\right)\right\}\frac{\lambda_{max}(A_{i,i})}{\lambda_{min}(A)}.
\end{aligned} \tag{3.8}$$

□

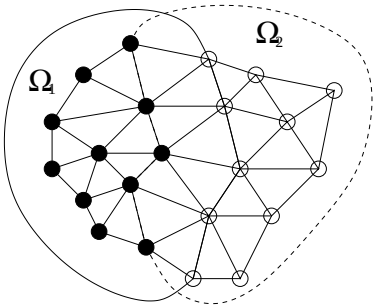


Figure 3.2: A minimal-overlap two-subdomain partition. ω_1 contains all the '•' nodes, and ω_2 the '○' nodes. Clearly, in this case $\omega_1 \cap \omega_2 = \emptyset$. To $\omega_i, i = 1, 2$, we can associate a subdomain Ω_i, composed by all the elements of the triangulation of Ω having a node in ω_i. Therefore, the two subdomains Ω_1 and Ω_2 share an overlap of 1 element.

Although not sharp (because of the generality of our assumptions), the estimation given by lemma 3.2.1 states that the condition number of $P_{BJ}^{-1}A$ grows linearly with the number of blocks used (which might not be of the same size) and the ratio $\lambda_{max}(A_{i,i})/\lambda_{min}(A)$. Note that the above result can be sharpened in the case of two subdomains [Axe94, Chapter 9].

3.2.3 Two-level Schwarz Preconditioners

The non-scalability of the one-level Schwarz preconditioner of section 3.2.1 is due to the fact that information are exchanged only locally through the overlapping regions, while for elliptic problems the domain of dependence is global. The Green function is non-zero throughout the whole domain, and some way of transmitting global information is needed to make the algorithm scalable.

In a two-level Schwarz preconditioner, a further correction term B_0 is added to the corrections on the subdomains, obtaining

$$P_{S,C,add}^{-1} = B_0 + P_S^{-1} = \sum_{i=0}^{M} B_i, \tag{3.9}$$

where $B_0 = R_0^T A_0^{-1} R_0$ is the coarse level correction. A_0 corresponds to the solution of the original variational problem in the space V_0, which is "coarse" in the sense that it contains a limited number of degrees of freedom so to make the "exact" inversion of A_0 computationally acceptable – if n_0 is the dimension of the coarse space, one has $n_0 \ll n$.

Since $\text{rank}(R_0^T A_0^{-1} R_0) = n_0$, B_0 has a large null space. Therefore, it cannot be directly used as a preconditioner: any component of the error which lies in the null space of $R_0^T A_0^{-1} R_0$ would never be corrected. Therefore, the formulation of equation (3.9) completes B_0. Note that B_0 reduces only the components of the error that can be represented on the coarse space, that is, the low frequencies, whereas P_S corrects the high frequencies.

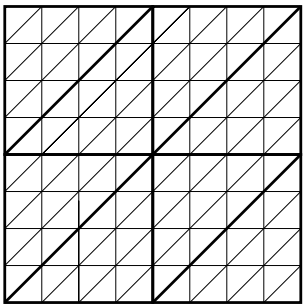


Figure 3.3: Example of coarse mesh using a structured grid for a 2D domain. Triangles of the fine mesh are in fine line, while the bold lines identify the triangles of the coarse mesh.

Remark 3.2.3. *An alternative two-level method adopts the following three-steps Richardson method*

$$\begin{aligned} \mathbf{u}^{n+1/3} &= \mathbf{u}^n + B_0\mathbf{r}^n \\ \mathbf{u}^{n+1/2} &= \mathbf{u}^{n+1/3} + P_S^{-1}\mathbf{r}^{n+1/3} \\ \mathbf{u}^{n+1} &= \mathbf{u}^{n+1/2} + B_0\mathbf{r}^{n+1/2}, \end{aligned}$$

where $\mathbf{r}^n$ *and* $\mathbf{u}^n$ *are respectively the residual and the approximate solution at the* $n-$*step of the Richardson method. The corresponding preconditioning matrix can be formally written as*

$$P_{S,C,hybrid}^{-1}A = I - (I - B_0A)(I - P_SA)(I - B_0A) \tag{3.10}$$

This approach is very close to the balancing Neumann-Neumann method for non-overlapping decompositions [Tal94]; see also section 3.3.3.

Clearly, the roles of B_0 *and* P_S *can be interchanged. Thus, a further preconditioner can be derived, which reads*

$$P_{S,C,hybrid2}^{-1}A = I - (I - P_SA)(I - B_0A)(I - P_SA)$$

This can be regarded as a multigrid approach, where P_S *can be seen as a smoother, while* B_0 *corrects the error on the coarse-level grid.*

The spectral properties (and the parallel performances) of two-level Schwarz preconditioners will depend strongly on the definition of the coarse space V_0. There are virtually unlimited choices of the coarse grid correction that may be used. Convergence of the entire scheme will depend on the particular interpolation and coarse grid operator used. When possible, the coarse space V_0 may be itself embedded into V_h. Usually, the coarse operator represents the discretisation of the continuous problem on a very coarse mesh. This is the common case, for instance, of structured grids like the one depicted in figure 3.3. In this case, the following result holds.

Theorem 3.2.2. *Consider the additive two-level overlapping Schwarz method, where the overlap is uniform of width $\mathcal{O}(\delta)$, the coarse grid space V_0 corresponds to the finite-element functions on elements of $\mathcal{O}(H)$, and $V_0 \subset V_h$. Then*

$$\kappa(P_{S,C,add1}^{-1} A) \leq C \left(1 + \frac{H}{\delta}\right), \tag{3.11}$$

where C is a constant independent of h, H and δ.

Proof. See [SBG96]. □

In a more general setting, with a coarse space which is not embedded in the fine space, it is possible to prove the following theorem.

Theorem 3.2.3. *For the additive two-level overlapping Schwarz method, when the overlap is uniform of width $\mathcal{O}(\delta)$, the coarse grid space V_0 corresponds to the finite-element functions on elements of $\mathcal{O}(H)$, $V_0 \not\subset V_h$ and $V_0 \not\subset H_0^1(\Omega)$, we have:*

$$\kappa(P_{S,C,add2}^{-1} A) \leq C \left(1 + \frac{H}{\delta}\right)^2, \tag{3.12}$$

where C is a constant independent of h, H and δ.

Proof. See [CSZ96]. □

With this approach, the fine and the coarse grid need not to be identical. One important constraint is that the closure of the coarse grid must cover any portion of the fine grid boundary for which Neumann boundary conditions are given. Multilevel extensions are presented in [CZ96, CGZ99].

Remark 3.2.4. *To build up the stiffness matrix on the coarse grid, one has two possibilities. The first one is to assemble the coarse grid exactly in the same way as on the fine level. This can be easily implemented in simple cases, whereas it is more difficult if the coefficient exhibits rapid variation since in that case data must be scaled down to the coarse grid. The second possibility is to build up the coarse matrix using the method called the Galerkin approximation,*

$$A_0 = R_0 A R_0^T. \tag{3.13}$$

This can be performed using matrix multiplication techniques.

Theorems 3.2.2 and 3.2.3 rely on the definition of a coarse grid. For structured meshes, it is relatively easy to find such a coarse grid. For unstructured meshes, it is not always a trivial task to define the interpolation operator from the fine mesh to the coarse mesh. Moreover, the implementation of this operator can be difficult or computationally expensive, especially for 3D computations.

Note that in many applications where the programmer has complete control of the software, generation of a (nested) coarse grid can be straightforward. One simply begins with

an appropriate FE coarse grid, and then refines it one or several times to obtain the fine grid. This fine grid will automatically inherit the shape regularity of the coarse one. When the coarse grid is defined in this manner, calculations of the interpolation and restriction operators is straightforward and inexpensive.

However, it is often neither possible nor practical to generate the fine grid from the coarse grid in a nested process. A possible approach is then to run the grid generating procedure twice, to generate two (uncorrelated and possibly non-nested) grids. The construction of the interpolation operators then becomes more difficult and expensive unless sophisticated algorithms are used to locate the fine grid nodes in the coarse grid. Another approach is to coarsen the given fine grid, finding a set of nodes and then building a tessellation of the domain. Note that for 2D domains is fairly easy to obtain a triangulation from a set of nodes, while for 3D domains is more difficult. Moreover, it is not evident that the resulting grid will be shape regular. Another important problem is the treatment of the boundary conditions, since the definition of the boundaries may be troublesome for the coarse domain.

To overcome these problems, we consider here aggregation procedures, which will be shown to be of particular advantage, especially when dealing with problems on unstructured grids. Aggregation procedures have been presented in literature by various authors. An aggregation technique was first introduced in 1951 by Leontief [Leo51, Chapter 9]. Here, it is written that *"in the case of products which comprise final demand, if enough is known to form a group of products whose use is strictly complementary, the whole group of products is sufficiently identified by one of its members. If the structure of final demand changes – and the use of the group of products is really complementary – such a change will affect the whole group in the same way. Therefore, no information is sacrificed by lumping them together. By the same token, products which are substitutable but have dissimilar production functions must separately identified since a change in their relative consumption will have different effects on the economy."* The justification to use this approach was the observation that *"[the production of] each industry [...] must be some kind of aggregates of products."* This approach has been extensively used in Economics; see [MS83] and the references therein.

In the domain of partial differential equations, smoothed aggregation techniques have been used, for example, for multigrid applications in [BV99, VBM01], where investigations of the smoothed aggregation properties have been reported. In the framework of domain decomposition methods, the focus is mainly on two-level methods. Results are presented in [PSFQ97] for the shallow water equations and 2D potential flows, in [FSQ94] for 3D potential flow computations, in [JKMK00] for groundwater flows, in [KMJK01] for multiphase flows, in [LT00] for discontinuous Galerkin approximation of advection-diffusion problems, and in [SF01, SF02] for 3D compressible Euler equations on unstructured tetrahedral grids.

The basic idea of aggregation is to adopt an algebraic setting to overcome the difficulties of the definition of a coarse grid. As suggested by equation (3.13), we first define the operator R_0, and consequently A_0 using two matrix-matrix products. The aggregation coarse space V_0 is defined such that each element of V_0 is actually composed by a linear combination of

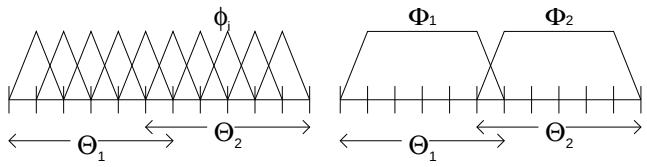


Figure 3.4: Fine-mesh and coarse-mesh functions in the case of two subdomains in one dimension. φ_i is a generic basis function for the fine space, whereas Φ_1 and Φ_2 are the two basis functions for the coarse space.

elements of V_h, and therefore $V_0 \subset V_h$. More precisely, we will group the nodes of the fine grid into aggregates ϑ_i. Then, the basis functions of the coarse space V_0 will be formed by summing the finite element basis functions of the nodes in each aggregate.

The procedure is as follows. First, the differential problem is discretized on the fine grid. Then, the matrix corresponding to the discretisation of the differential operator on the coarse space is build using the elements of the fine-grid matrix. The basis functions of the coarse space will be formed by summing the finite element basis functions of the nodes in each aggregate. The algorithm uses a simple procedure to build the restriction operators R_0 and the prolongation operator R_0^T and to form the coarse matrix as $A_0 = R_0 A R_0^T$.

The construction of R_0 is sometimes referred to as *smoothed aggregation* (SA). Its first step corresponds to define a restriction operator $\tilde{R}_0$ so that each aggregate will eventually correspond to a "degree of freedom" of the coarse operator. Typically, each aggregate is formed by the nodes belonging to a subdomain, although one may have more aggregates than subdomains, for instance by further subdividing each subdomain into a set of contiguous nodes. This decomposition is usually obtained using a graph partitioning algorithm. The number of aggregates n_0 represents the dimension of the coarse space V_0, since each aggregate will be given a unique coarse grid function.

The $n_0 \times n$ (real) matrix $\tilde{R}_0$ is constructed in such a way that each row corresponds to a node and each column to an aggregate. We will indicate with $\tilde{\vartheta}_i$ the set of nodes that form the non-smoothed aggregate i, $\mathcal{G}$ the set of nodes lying on $\partial\Omega$, $\tilde{\Theta}_j = \text{int}\left(\cup_{i\in\tilde{\vartheta}_j} \text{supp}(\Phi_i)\right)$, $\tilde{H}_0 = \max_{j=1,\dots,n_0} \text{diam}(\tilde{\Theta}_j)$, and $\tilde{\delta}_0$ the amount of overlap among the (non-smoothed) aggregates. A possible way to define the entries of $\tilde{R}_0$ is as follows:

$$\tilde{R}_0(i,j) = \begin{cases} 1 & \text{if } j \in \tilde{\vartheta}_i \\ 0 & \text{if } j \notin \tilde{\vartheta}_i \text{ or } j \in \mathcal{G}. \end{cases} \tag{3.14}$$

$\tilde{R}_0$ can be viewed as a simple grid transfer operator corresponding to piecewise constant interpolation. A one-dimensional example in the case of $\tilde{R}_0$ and with two subdomains is given in figure 3.4, while figure 3.5 reports a 2D example. Figure 3.6 represents graphically the basis functions of V_h and V_0 for a 2D case.

Remark 3.2.5. *In general, one lets each processor build the aggregate corresponding to its*

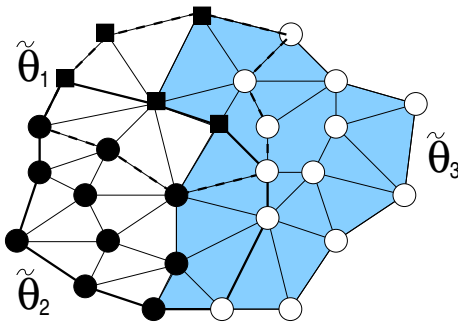


Figure 3.5: Examples of aggregation for a 2D domain. The nodes marked with '■', '○', and '•' belong to $\tilde{\vartheta}_1$, $\tilde{\vartheta}_2$, and $\tilde{\vartheta}_3$, respectively. The bold line defines $\tilde{\Theta}_1$, the dotted line $\tilde{\Theta}_2$, and the shaded background $\tilde{\Theta}_3$.

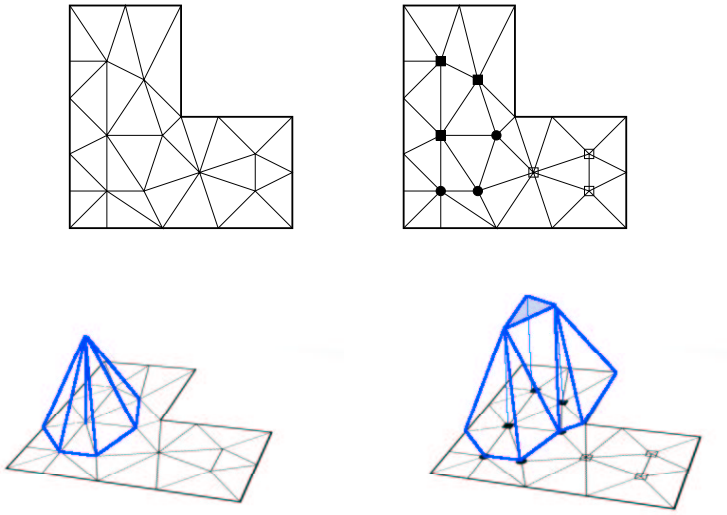


Figure 3.6: On the top, example of fine grid (on the left) and aggregates (on the right). On the bottom, basis function for the fine grid (on the left), and non-smoothed basis function for the coarse space (on the right).

piece of the grid (corresponding to the triangulation of the computational domain), ignoring the connections among subdomains. That is, each processor is assigned a subgrid of the entire grid (that corresponds to the triangulation of the subdomain given to that processor); then, a serial aggregation algorithm is used on each subgrid. This variant, sometimes referred to as decoupled aggregation, works quite well if the load balance of the grid data among the processors is satisfactory. In general, when the ratio between the number of nodes and the number of aggregation is large enough, like for the two-level methods here presented, the decoupled aggregation offers good partitioning. On the contrary, when large number of aggregation are required (like, for instance, in multilevel methods), decoupled aggregation may result in a somewhat irregular decomposition. In this case, it is usually worth to re-equilibrate the partitioning among the subdomains to minimise the dependency of the resulting algorithm on the subdomain decomposition; see [TT00] for discussions and results on parallel smoothed aggregation applied to multigrid methods.

The second step (that may be avoided) consists of applying a prolongation smoother $\tilde{S}_0$ to produce the final prolongation operator $R_0^T = \tilde{R}_0^T \tilde{S}_0$. Let $\tilde{\Phi}_i$ be the tentative coarse functions, as defined by $\tilde{R}_0$. This function will be smoothed out by increasing its support using the stencil of suitable polynomials in A. Then, the (smoothed) coarse space reads

$$V_0 := \text{span}\{\Phi_i, \quad i = 1, \dots, n_0\}, \quad \Phi_i = \tilde{S}_0 \tilde{\Phi}_i. \tag{3.15}$$

Typically, $\tilde{S}_0 = \mathbb{P}_k(D^{-1}A)$, where D is a suitable diagonal matrix, which can be chosen to be the diagonal part of A, and $\mathbb{P}_k$ a polynomial of degree k.

Possible choices for the smoother are a simple Richardson smoother [LT00, LT01]:

$$\tilde{S} = \tilde{S}_0(\varpi, k) = (I - \varpi D^{-1} A)^k, \tag{3.16}$$

where $\varpi > 0$ is a real parameter and $k \in \mathbb{N}_0$, or a recursive Richardson smoother [BV99, VBT99], or else a SPAI smoother [BGMR01]. For a comparison of these smoothers the reader is addressed to [LT00] or [LT01], where it is shown (experimentally) that the effect of the mentioned smoothers are quite similar. Hence, in the following we will focus only on smoother (3.16).

Note that, unless A has particular properties, the spectral radius of a non-damped Jacobi method is larger than one. In this case, ϖ must be chosen such that $\varpi < 2/\rho(D^{-1}A)$. The need of an estimation of the largest eigenvalue is common to other smoothers as well. However, it might be not too difficult to obtain an estimate of the largest eigenvalue. For example, if A is a $M-$matrix, the Gershgorin theorem can be applied to bound the spectral radius of A. If the matrix is scaled to have unit diagonal entries, this leads to the number '2' for the largest eigenvalue. For non $M-$matrices, it is possible to estimate the eigenvalue using a small number of Lanczos or conjugate gradient or power method iterations. It may be useful to scale the estimate by a small factor, because most computational methods give lower bounds to the largest eigenvalue.

Remark 3.2.6. *In the case of one aggregate per subdomain, a SA technique can be used to define the overlapping subdomains Ω_i. One possible way to proceed is as follows: one starts with a minimal-overlap decomposition $\tilde{\Omega}_i$, then a SA technique is applied to form V_0. Then, set $\Omega_i = \text{supp}\{\Phi_i\}, i = 1, \ldots, M$. The amount of overlap among the subdomains (and the aggregates) will depend on the degree of the smoother.*

Let us indicate with $\Theta_j = \text{int}\left(\cup_{i \in \vartheta_j} \text{supp}(\varphi_i)\right)$ the support of Φ_j, and

$$H_0 = \max_{j=1,\ldots,n_0} \text{diam}(\tilde{\Theta}_j), j = 1, \ldots, n_0.$$

We assume that the node aggregation has been carried out so that all Θ_j are connected subsets of Ω. In the following we will use the term aggregate for both ϑ_j and Θ_j, depending on the context.

3.2.4 Algebraic Interpretation

The aggregation procedure can be reinterpreted as an algebraic manipulation of matrix A. For the sake of generality, let us associate a (row) vector $\boldsymbol{\eta}_i \in \mathbb{R}^n$ to the aggregate $\Theta_i, i = 1, \ldots, n_0$, such that the R_0 matrix becomes

$$R_0 = \begin{bmatrix} \boldsymbol{\eta}_1^T \\ \vdots \\ \boldsymbol{\eta}_{n_0}^T \end{bmatrix}, \tag{3.17}$$

that is, $R_0(k, j) = \boldsymbol{\eta}_k(j)$. With this notation, a basis function of V_0 can be written as

$$\Phi_k = \sum_{j=1}^{n} R_0(k, j)\varphi_j, \tag{3.18}$$

while a generic function $u_0 \in V_0$ reads

$$u_0 = \sum_{k=1}^{n_0} u_{0,k}\Phi_k. \tag{3.19}$$

Then,

$$\begin{aligned} u_0 &= \sum_{k=1}^{n_0} u_{0,k} \sum_{j=1}^{n} R_0(k, j)\varphi_j \\ &= \sum_{j=1}^{n} \left(\sum_{k=1}^{n_0} u_{0,k} R_0(k, j) \right) \varphi_j \\ &= \sum_{j=1}^{n} \left(R_0^T \mathbf{u}_0 \right)_j \varphi_j, \end{aligned}$$

and this demonstrate that R_0^T is the natural choice for the operator which interpolates from the coarse space V_0 to the fine space V.

We are now ready to define the variational problem on coarse problem as

$$\begin{cases} \text{Find } u_0 \in V_0 : \\ a(u_0, v_0) = f(v_0) \ , \forall v_0 \in V_0, \end{cases} \tag{3.20}$$

$a(\cdot,\cdot)$ being the bilinear form associated to the considered problem on the fine grid and $f(\cdot)$ the corresponding linear functional (see section 3.1).

Using (3.19), problem (3.20) reads

$$a\left(\sum_{k=1}^{n_0} u_{0,k}\Phi_k(\mathbf{x}), \Phi_l(\mathbf{x})\right) = f\left(\Phi_l(\mathbf{x})\right) \quad l = 1, \dots, n_0.$$

Then, from equations (3.18) and (3.17) we derive

$$\begin{aligned} a\left(\sum_{k=1}^{n_0} u_{0,k}\sum_{j=1}^{n}\eta_{k,j}\varphi_j, \sum_{p=1}^{n}\eta_{l,p}\varphi_p\right) &= f\left(\sum_{p=1}^{n}\eta_{l,p}\varphi_p\right) \quad l = 1, \dots, n_0 \\ \sum_{k=1}^{n_0} u_{0,k}\sum_{j=1}^{n}\eta_{k,j}\sum_{p=1}^{n}\eta_{l,p}a\left(\varphi_j, \varphi_p\right) &= \sum_{p=1}^{n}\eta_{l,p}f\left(\varphi_p\right) \quad l = 1, \dots, n_0 \\ \sum_{k=1}^{n_0} u_{0,k}\sum_{j=1}^{n}\eta_{k,j}\sum_{p=1}^{n}\eta_{l,p}A_{j,p} &= \sum_{p=1}^{n}\eta_{l,p}f_p \quad l = 1, \dots, n_0. \end{aligned}$$

Hence, the element (k,l) of the stiffness matrix A_0 is given by

$$A_0(k,l) = \sum_{j=1}^{n}\eta_{k,j}\sum_{i=1}^{n}\eta_{l,i}A_{i,j},$$

and the right-hand side reads

$$f_0(l) = \sum_{p=1}^{n}\eta_{l,p}f_p.$$

In the case of minimal overlap, R_0 has a special form. We order the set of nodes starting from those in Ω_1, and then proceeding with those in $\Omega_2, \dots, \Omega_M$. Correspondingly, we order the degree of freedom of V_0. As a result, the prolongation matrix $R_0^T \in \mathbb{R}^{n\times n_0}$ has the

following block structure:

$$R_0^T = \begin{bmatrix} \boldsymbol{\eta}_{1,1}^T & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{\eta}_{2,1}^T & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{\eta}_{n_{0,1},1}^T & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\eta}_{1,2}^T & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \boldsymbol{\eta}_{n_{0,2},2}^T & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \boldsymbol{\eta}_{1,p}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \boldsymbol{\eta}_{n_{0,p}}^T \end{bmatrix}. \tag{3.21}$$

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.

3.2.5 Convergence Analysis of Abstract Schwarz Methods

To bound the condition number of (3.9) with V_0 given by (3.15), we use the abstract convergence theory for the Schwarz methods [DSW94]. This theory, defined for symmetric problems, centres around 3 parameters, $C_0, \rho(\epsilon)$ and ω later defined. The theory is quite general and does not rely on any specific definition of the coarse space V_0.

In this section we report the main lines of this convergence theory, following [SBG96]. Then, in section 3.2.6 we will make use of this abstract theory to prove theorem 3.2.4.

Consider again our model problem (1.1) and its discrete counterpart (2.1). As always in this thesis, we suppose to have M subspaces $V_i \subset V_h$, plus a subspace $V_0 \subset V_h$ which will play the role of coarse space, and therefore will be treated slightly differently from the other subspaces. We assume that there exists a set of interpolation operators $\mathcal{I}_i : V_i \to V_h$, and that a continuous and coercive bilinear form $a_i(\cdot,\cdot)$ that approximates $a(\cdot,\cdot)$ is defined on each V_i. Finally, let the projection-like operators $\tilde{\mathcal{T}}_i : V_h \to V_i$ defined as

$$a_i(\tilde{\mathcal{T}}_i u, v) = a(u, \mathcal{I}_i v) \quad \forall v \in V_i, i = 1, \dots, M \tag{3.22}$$

and the operators $\mathcal{T}_i : V_h \to V_h$ as $\mathcal{I}_i \tilde{\mathcal{T}}_i$. Algebraically, we can define a projection operator T_i as

$$T_i = R_i^T {A_i}^{-1} R_i A, \quad i = 0, \dots, M$$

where A_i is the algebraic operator (the matrix) associated to $a_i(\cdot,\cdot)$, which in the case where $a_i(\cdot,\cdot) = a(\cdot,\cdot)$ takes the form $R_i A {R_i}^T$. Note that if $\mathbf{u}$ is the solution of problem (2.1) associated with the FE approximation of (3.2), then

$$T_i \, \mathbf{u} = {R_i}^T {A_i}^{-1} R_i A \mathbf{u} = {R_i}^T {A_i}^{-1} R_i \, \mathbf{f}. \tag{3.23}$$

Thus, the result of $T_i\mathbf{u}$ depends solely on $\mathbf{f}$.

The idea is to define a new equation, using polynomials in T_i. This equation will have the same solution $\mathbf{u}$. Note that it is possible to calculate these polynomials since we already know how to compute $T_i\mathbf{u}$ from (3.23), as well as $T_i\mathbf{v}$ for any given $\mathbf{v}$. Let $\mathbb{P}(T_0, T_1, \dots, T_M)$ be any homogeneous polynomial in T_i, and let $\mathbf{g} = \mathbb{P}(T_0, T_1, \dots, T_M)\,\mathbf{u}$. The general abstract Schwarz method replaces the given linear equation $A\mathbf{u} = \mathbf{f}$ by the operator equation

$$\mathbb{P}(T_0, T_1, \dots, T_M)\,\mathbf{u} = \mathbf{g}, \tag{3.24}$$

which can be solved using, for instance, a Krylov subspace method. In this way we have replaced the (possibly) ill-conditioned operator A with a preconditioned operator that we can write as $\mathbb{P}(T_0, T_1, \dots, T_M) = BA$. The preconditioner B is therefore defined implicitly.

From the abstract formulation (3.24) we can obtain the two-level additive Schwarz method as defined by equation (3.9) setting

$$P^{-1}_{S,C,add}A = BA = T_0 + T_1 + \dots T_M. \tag{3.25}$$

Here, the preconditioner is implicitly defined by the sum of projections T_i onto the subspaces V_i. Since $T_i = B_iA$, we have

$$P^{-1}_{S,C,add}A = \sum_{i=0}^{M} T_i = \left(\sum_{i=0}^{M} B_i\right) A,$$

as previously defined by equation (3.9). This preconditioner is called *additive* since the projections onto the subspaces are simply added.

Remark 3.2.7. *Another noticeably case is the full multiplicative Schwarz method, that reads*

$$P^{-1}_{S,C,mult}A = (I - T_0)(I - T_1)\dots(I - T_M). \tag{3.26}$$

Although the multiplicative Schwarz method reveals to be better conditioned that the additive one, it is clear from (3.26) that the subspaces must be treated one after the other, in a sequential way, while in (3.25) one can consider each projector T_i separately and then simply add the results. Therefore, for the definition of parallel preconditioners (3.25) has to be preferred to (3.26).

Now we introduce the tools that are used in the convergence analysis of Schwarz methods. We start with the following lemma. This proof can be found, for instance, in [SBG96, Section 5.2] (see also pages 160 and 161 of the same book for further references about this lemma), and is reported here for the sake of completeness.

Lemma 3.2.2. *Define $\mathcal{T} = \sum_{i=0}^{M} \mathcal{T}_i$. Then, $\forall u_h \in V_h$,*

$$a(\mathcal{T}^{-1}u, u) = \min_{\substack{u_i \in V_i \\ u = \sum_{i=0}^{M} \mathcal{I}_i u_i}} \sum_i a_i(u_i, u_i). \tag{3.27}$$

Proof. First, we construct a particular decomposition of u which satisfies (3.27) exactly, and then we prove that

$$a(\mathcal{T}^{-1}u,u) \leq \sum_i a_i(u_i,u_i) \quad \forall u_i \in V_i, \ \sum_i \mathcal{I}_i u_i = u. \tag{3.28}$$

Let $u_i = \tilde{\mathcal{T}}_i \mathcal{T}_i^{-1} u$; then $\sum_i \mathcal{I}_i u_i = \left(\sum_i \mathcal{I}_i \tilde{\mathcal{T}}_i\right) \mathcal{T}^{-1} u = u$ and

$$\begin{aligned} \sum_i a_i(u_i,u_i) &= \sum_i a_i\left(\tilde{\mathcal{T}}_i \mathcal{T}^{-1}u, \tilde{\mathcal{T}}_i \mathcal{T}^{-1}u\right) \\ &= \sum_i a\left(\mathcal{T}^{-1}u, \mathcal{T}_i \mathcal{T}^{-1}u\right) \\ &= a(\mathcal{T}^{-1}u,u). \end{aligned}$$

Now, we prove (3.28) as follows:

$$\begin{aligned} a(\mathcal{T}^{-1}u,u) &= a(\mathcal{T}^{-1}u, \sum_i \mathcal{I}_i u_i) \\ &= \sum_i a(\mathcal{T}^{-1}u, \mathcal{I}_i u_i) \\ &= \sum_i a(\tilde{\mathcal{T}}_i \mathcal{T}^{-1}u, u_i) \\ \text{[by C-S]} \qquad &\leq \left[\sum_i a_i(\tilde{\mathcal{T}}_i \mathcal{T}^{-1}u, \tilde{\mathcal{T}}_i \mathcal{T}^{-1}u)\right]^{1/2} \left[\sum_i a_i(u_i,u_i)\right]^{1/2} \\ &= \left[\sum_i a(\mathcal{T}^{-1}u, \mathcal{T}_i \mathcal{T}^{-1}u)\right]^{1/2} \left[\sum_i a_i(u_i,u_i)\right]^{1/2} \\ &= a(\mathcal{T}^{-1}u,u)^{1/2} \left[\sum_i a_i(u_i,u_i)\right]^{1/2}. \end{aligned}$$

This yields (3.28). Here, C-S means that use has been made of the Cauchy-Schwarz inequality. □

The abstract convergence theory for symmetric problems centres around three parameters, defined in terms of the following three definitions.

Definition 3.2.1. *Let C_0 be the minimum constant such that for all $u_h \in V_h$ there exists a representation $u_h = \sum_{i=0}^M \mathcal{I}_i u_i, u_i \in V_i$ such that*

$$\sum_{i=0}^M a_i(u_i,u_i) \leq C_0^2 a(u_h,u_h). \tag{3.29}$$

Definition 3.2.2. *Let*

$$\epsilon_{i,j} = \inf_{u_i \in V_i, v_j \in V_j} \frac{|a(\mathcal{I}_i u_i, \mathcal{I}_j u_j)|}{a(\mathcal{I}_i u_i, \mathcal{I}_i u_i)^{1/2} a(\mathcal{I}_j u_j, \mathcal{I}_j u_j)^{1/2}}.$$

Note that $0 \leq \epsilon_{i,j} \leq 1$. *Then, let* $\rho(\epsilon)$ *be the spectral radius of the matrix* ϵ *whose entries are the* $\epsilon_{i,j}$,

Definition 3.2.3. *Let* ω *be the minimum constant such that*

$$a(\mathcal{I}_i u_i, \mathcal{I}_i u_i) \leq \omega a_i(u_i, u_i) \;\; \forall u_i \in V_i, i = 0, \ldots, M \tag{3.30}$$

where we assume that the $a_i(\cdot,\cdot)$ *are suitably scaled.*

Remark 3.2.8. *Definitions 3.2.1, 3.2.2 and 3.2.3 are usually inferred to as* assumptions*; see, for instance, [SBG96].*

For a linear operator $\mathcal{L}$, which is self-adjoint with respect to $a(\cdot,\cdot)$, we use the Rayleigh quotient characterisation of the extreme eigenvalues:

$$\lambda^{(a)}_{max}(\mathcal{L}) = \max_{u \neq 0} \frac{a(\mathcal{L}u, u)}{a(u,u)}, \quad \lambda^{(a)}_{min}(\mathcal{L}) = \min_{u \neq 0} \frac{a(\mathcal{L}u, u)}{a(u,u)}.$$

The condition number of $\mathcal{L}$ is then given by

$$\kappa(\mathcal{L}) = \frac{\lambda^{(a)}_{max}(\mathcal{L})}{\lambda^{(a)}_{min}(\mathcal{L})}.$$

In order to apply the preconditioned conjugate gradient method to the operator $\mathcal{T}$, it must be symmetric with respect to $a(\cdot,\cdot)$, that is

$$a(\mathcal{T}u, v) = a(u, \mathcal{T}v).$$

It easily seen that the operators $\mathcal{T}_i$ defined in (3.23) are self-adjoint in $a(\cdot,\cdot)$.

We can now provide a bound for the abstract additive Schwarz method in terms of the parameters in definitions 3.2.1, 3.2.2 and 3.2.3.

Lemma 3.2.3 (abstract Schwarz method). *The abstract additive Schwarz method satisfies*

$$\kappa(P^{-1}_{S,C,add} A) \leq \omega \left[1 + \rho(\epsilon)\right] C_0^2,$$

where $C_0, \rho(\epsilon)$ *and* ω *are the constants of definitions 3.2.1, 3.2.2 and 3.2.3, respectively. In particular,* $1/C_0^2$ *is a sharp lower bound on the smallest eigenvalue of* BA *and* $\omega\left[1 + \rho(\epsilon)\right]$ *is a bound on the largest eigenvalue of* BA.

Proof. The bound on the smallest eigenvalue follows immediately from lemma 3.2.2 and definition 3.2.1.

We now bound the largest eigenvalue. We first treat the subspaces $V_i, i = 1, \ldots, M$, while the space V_0 is treated separately. We have

$$
\begin{aligned}
a\left(\sum_{i=1}^{M} \mathcal{T}_i v, \sum_{j=1}^{M} \mathcal{T}_j v\right) &= \sum_{i,j=1}^{M} a(\mathcal{T}_i v, \mathcal{T}_j v) \\
\text{[by def. 3.2.1]} \qquad &\leq \sum_{i,j=1}^{M} \epsilon_{i,j} a(\mathcal{T}_i v, \mathcal{T}_i v)^{1/2} a(\mathcal{T}_j v, \mathcal{T}_j v)^{1/2} \\
&\leq \rho(\epsilon) \sum_{i=1}^{M} a(\mathcal{T}_i v, \mathcal{T}_i v) \\
\text{[by def. 3.2.2]} \qquad &\leq \omega \rho(\epsilon) \sum_{i=1}^{M} a_i(\tilde{\mathcal{T}}_i v, \tilde{\mathcal{T}}_i v) \\
&= \omega \rho(\epsilon) \sum_{i=1}^{M} a(v, \mathcal{T}_i v) \\
\text{[by C-S]} \qquad &= \omega \rho(\epsilon) a(v, \sum_{i=1}^{M} \mathcal{T}_i v) \\
&\leq \omega \rho(\epsilon) a(v, v)^{1/2} a\left(\sum_{i=1}^{M} \mathcal{T}_i v, \sum_{j=1}^{M} \mathcal{T}_j v\right)^{1/2}.
\end{aligned}
$$

Therefore

$$
\begin{aligned}
a([\sum_{i=1}^{M} \mathcal{T}_i]^2 v, v) &= a(\sum_{i=1}^{M} \mathcal{T}_i v, \sum_{j=1}^{M} \mathcal{T}_j v) \\
&\leq \omega^2 \rho^2(\epsilon) a(v, v). \qquad (3.31)
\end{aligned}
$$

Thus, the largest eigenvalue of $\sum_{i=1}^{M} \mathcal{T}_i$ is bounded by $\omega\rho(\epsilon)$. To estimate the largest eigenvalue of $\mathcal{T}_0$ we proceed as follows:

$$
\begin{aligned}
a(\mathcal{T}_0 v, \mathcal{T}_0 v) &\leq \omega a_0(\tilde{\mathcal{T}}_0 v, \tilde{\mathcal{T}}_0 v) \\
\text{[(3.22)]} \qquad &= \omega a(v, \mathcal{T}_0 v) \\
&\leq \omega a(v, v)^{1/2} a(\mathcal{T}_0 v, \mathcal{T}_0 v)^{1/2}.
\end{aligned}
$$

Thus,

$$
a(\mathcal{T}_0 v, \mathcal{T}_0 v) \leq \omega^2 a(v, v),
$$

which implies that

$$
a(\mathcal{T}_0 v, v) \leq \omega a(v, v). \qquad (3.32)
$$

Combining (3.31) with (3.32) completes the proof for the upper bound. □

3.2.6 Spectral Properties of SA Preconditioners

In this section we will apply lemma 3.2.3 to the aggregation coarse space. First, we need to make precise some geometrical information about the decomposition used. This is done in the following properties.

Property 3.2.2 (partition). *There are two constants C_1 and C_2 so that, for each aggregate $\Theta_i, i = 1, \dots, n_0$, we have:*

a. $\text{diam}(\Theta_i) \leq C_1 H_0, i = 1, \dots, n_0$;

b. the Lebesgue measure $|\Theta_i|$ of Θ_i satisfies $|\Theta_i| \geq C_2 H_0^d$;

c. the overlap among the aggregates is of order δ_0.

Property 3.2.2 states that the aggregates have diameter of comparable size H_0 and are shape-regular. The following property 3.2.3, instead, requires a certain regularity on the coarse space basis function Φ_i.

Property 3.2.3 (coarse space). *There exists a constant $C > 0$ such that the basis functions $\{\Phi_i\}$ of the coarse space satisfy*

a. $|\Phi_i|^2_{H^1(\Omega)} \leq C \dfrac{H_0^{d-1}}{\delta_0}$;

b. $||\Phi_i||^2_{L^2(\Omega)} \leq C H_0^d$;

c. There is a domain $\hat{\Omega} \subset \Omega$ such that $\sum_i \Phi_i(x) = 1$ for every $x \in \hat{\Omega}$ and $\sup_{x \in \Omega \setminus \hat{\Omega}} \text{dist}(x, \partial\Omega) \leq C H_0$.

We note that a non-negative function Φ_i, which is constant in the interior of Θ_i and decreases to zero with $\|\nabla \Phi_i\|_{L^\infty(\Omega)} \leq 1/\delta_0$ in a layer of width δ_0, satisfies the bounds 3.2.3.a and 3.2.3.b. In particular, the non-smoothed case defined by equation (3.14), corresponding to $\tilde{R}_0$, will bring to a set of functions $\{\Phi_i\}$ which satisfies property 3.2.3 with $\delta_0 = h$.

Let us define the operator $Q_0 : V_h \to V_0$ by

$$Q_0 u = \sum_{i=1}^{n_0} \alpha_i \Phi_i, \quad \alpha_i = \frac{1}{\gamma_i} \int_{\Theta_i} \eta_i(\mathbf{x}) u(\mathbf{x}) d\Omega,$$

where γ_i is a coefficient to be precised and $\eta_i(\mathbf{x})$ is a weighting function, whose support is at most Θ_i. We suppose $\eta_i(\mathbf{x}) \in L^2(\Theta_i)$. Using the Cauchy-Schwarz inequality we get

$$\sum_{i=1}^{n_0} \alpha_i^2 \leq \sum_{i=1}^{n_0} \frac{1}{\gamma_i^2} ||\eta_i||^2_{L^2(\Omega)} ||u||^2_{L^2(\Omega)} \tag{3.33}$$

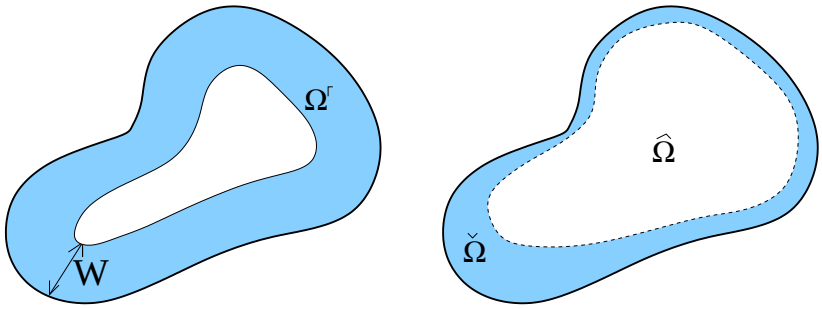


Figure 3.7: On the left, in shaded colour, Ω^Γ. On the right, in shaded colour, $\check{\Omega}$, and in white, $\hat{\Omega}$.

If we take η_i and γ_i such that $\frac{1}{\gamma_i^2}||\eta_i||^2_{L^2(\Theta_i)} \leq CH_0^{-d}$ (for example, $\eta_i = 1$ on Θ_i and $\gamma_i = |\Theta_i|$) and properties 3.2.2 and 3.2.3 hold, we finally get

$$\sum_{i=1}^{n_0} \alpha_i^2(u) \leq CH_0^{-d}||u||^2_{L^2(\Omega)}. \tag{3.34}$$

We can now prove the following lemma.

Lemma 3.2.4. *If definitions 3.2.2 and 3.2.3 are satisfied, then there exists a constant $C > 0$ independent of H_0, δ_0 and h such that $\forall u_h \in V_h$*

a. $||u - Q_0 u||^2_{L^2(\Omega)} \leq CH_0^2 |u|^2_{H^1(\Omega)}$,

b. $|Q_0 u|^2_{H^1(\Omega)} \leq C\frac{H_0}{\delta_0}|u|^2_{H^1(\Omega)}$.

Proof. The proof extends a similar proof proposed in [JKMK00]. It is split into 2 parts. First, we prove lemma 3.2.4 in $\Omega \setminus \hat{\Omega}$; then, in $\hat{\Omega}$, where $\hat{\Omega}$ is the domain introduced in property 3.2.3. Throughout the proof, C represents a generic positive constant independent of H, H_0, δ, and δ_0.

We consider first the L^2-estimate.

Let us set $\check{\Omega} = \Omega \setminus \hat{\Omega}$. Further, we define

$$\mathcal{B} = \left\{ i \ : \ \Theta_i \cap \check{\Omega} \neq \emptyset \right\}, \quad \check{\Xi} = \bigcup_{i \in \mathcal{B}} \Theta_i, \quad W = \sup_{x \in \check{\Omega}} \inf_{y \in \partial\Omega} \operatorname{dist}(x, y)$$

and set

$$\Omega^\Gamma = \{ x \in \Omega \mid \operatorname{dist}(x, \partial\Omega) \leq W \}.$$

Figure 3.7 graphically explains the domains $\check{\Omega}$ and Ω^Γ.

From property 3.2.2.c it follows that $W \leq CH_0$, and therefore the Poincaré inequality yields

$$\|u\|_{L_2(\check{\Omega})} \leq \|u\|_{L^2(\Omega^\Gamma)} \leq CH_0|u|_{H^1(\Omega^\Gamma)}. \tag{3.35}$$

The restriction of Q_0u onto $\check{\Omega}$ can be expressed as

$$(Q_0u)\,|_{\check{\Omega}} = \sum_{i\in\mathcal{B}} \alpha_i(u)\Phi_i(x)|_{\check{\Omega}}.$$

Further, let us set

$$\mathcal{N}_i = \{j \mid \Theta_j \cap \Theta_i \neq \emptyset\}. \tag{3.36}$$

We have

$$\begin{aligned}
\|Q_0u\|^2_{L^2(\check{\Omega})} &= \sum_{i\in\mathcal{B}} \sum_{j\in\mathcal{N}_i\cap\mathcal{B}} (\alpha_i(u)\Phi_i, \alpha_j(u)\Phi_j)_{L^2(\check{\Omega})} \\
&\leq \sum_{i\in\mathcal{B}} \sum_{j\in\mathcal{N}_i\cap\mathcal{B}} |\alpha_i(u)| \cdot |\alpha_j(u)| \cdot \|\Phi_i\|_{L^2(\Omega)}\|\Phi_j\|_{L^2(\Omega)} \\
\text{[by prop. 3.2.3.b]} \qquad &\leq CH_0^d \sum_{i\in\mathcal{B}} \sum_{j\in\mathcal{N}_i\cap\mathcal{B}} \frac{1}{2}\left(\alpha_i^2(u) + \alpha_j^2(u)\right) \\
&\leq CH_0^d \max\{\operatorname{card}\mathcal{N}_i\} \sum_{i\in\mathcal{B}} \alpha_i^2(u) \\
\text{[by (3.33)]} \qquad &\leq C\,\|u\|^2_{L^2(\check{\Xi})} \\
&\leq C\,\|u\|^2_{L^2(\Omega^\Gamma)}
\end{aligned}$$

Using the last inequality together with inequality (3.35) gives

$$\|u - Q_0u\|_{L^2(\check{\Omega})} \leq C\|u\|_{L^2(\check{\Omega})} \leq CH_0|u|_{H^1(\Omega^\Gamma)}.$$

In an analogous way, we can estimate 3.2.4.b in $\check{\Omega}$ exploiting property 3.2.3.a, obtaining

$$\begin{aligned}
|Q_0u|^2_{H^1(\check{\Omega})} &\leq CH_0^{-1}/\delta_0\,\|u\|^2_{L^2(\Omega^\Gamma)} \\
\text{[by (3.32)]} \qquad &\leq CH_0/\delta_0\,|u|^2_{H^1(\Omega^\Gamma)}.
\end{aligned}$$

Now, we turn our attention to $\hat{\Omega}$.

For every function $u \in V_h$, let us consider an extension u_E satisfying

$$\|u_E\|_{H^1(\mathbb{R}^d)} \leq \|u\|_{H^1(\Omega)}, \quad u_E = u \text{ in } \Omega.$$

For $j = 1, \ldots, n_0$, we define $\Theta_j^\star = \cup_{i\in\mathcal{N}_j}\Theta_i$ with $\mathcal{N}_i$ given by (3.36), and Ξ_j to be the ball circumscribing $\Theta_j^\star$. From property 3.2.2 is follows that

$$\operatorname{diam}(\Xi_j) \leq CH_0.$$

We now use the Friedrichs' inequality of the form

$$\|u\|_{L^2(\Xi_j)} \leq CH_0 \, |u|_{H^1(\Xi_j)} \ , \ \forall u \in \left\{ v \in H^1(\Xi_j) \mid \int_{\Xi_j} v dx = 0 \right\}. \tag{3.37}$$

For every $j = 1, \ldots, n_0$ we define

$$c_j = \int_{\Xi_j} u_E dx, \quad \bar{u}_j = u_E - c_j.$$

Then, the Friedrichs inequality holds for every $\bar{u}_j$. Due to property 3.2.3.c, for every $x \in \Theta_j \cap \hat{\Omega}$ it holds

$$\begin{aligned} (Q_0 u)(x) &= (Q_0 \bar{u}_j)(x) + Q_0 c_j \\ &= \sum_{i \in \mathcal{N}_j} \alpha_i(\bar{u}_j)\Phi_i(x) + c_j \sum_{i \in \mathcal{N}_j} \Phi_i(x) \\ \text{[by prop. 3.2.3.c]} \qquad &= (Q_0 \bar{u}_j)(x) + c_j. \end{aligned}$$

Therefore,

$$\begin{aligned} \|u - Q_0 u\|^2_{L^2(\hat{\Omega})} &\leq \sum_{i=1}^{n_0} \|u - Q_0 u\|^2_{L^2(\Xi_i \cap \hat{\Omega})} \\ &= \sum_i \|(I - Q_0)(\bar{u}_i + c_i)\|^2_{L^2(\Xi_i \cap \hat{\Omega})} \\ &= \sum_i \|(I - Q_0)\bar{u}_i\|^2_{L^2(\Xi_i \cap \hat{\Omega})} \\ &\leq 2 \sum_i \left(\|\bar{u}_i\|^2_{L^2(\Xi_i \cap \hat{\Omega})} + \|Q_0 \bar{u}_i\|^2_{L^2(\Xi_i \cap \hat{\Omega})} \right). \end{aligned} \tag{3.38}$$

Further,

$$\begin{aligned} \|Q_0 \bar{u}_i\|^2_{L^2(\Xi_i \cap \hat{\Omega})} &\leq \left\| \sum_{j \in \mathcal{N}_i} \alpha_j(\bar{u}_i)\Phi_j \right\|^2_{L^2(\Omega)} \\ &\leq \left(\sum_{j \in \mathcal{N}_i} |\alpha_j(\bar{u}_i)| \cdot \|\Phi_j\|_{L^2(\Omega)} \right)^2 \\ &\leq \text{card}\{\mathcal{N}_j\} \sum_{j \in \mathcal{N}_i} \alpha_j^2(\bar{u}_i) \, \|\Phi_j\|^2_{L^2(\Xi_i \cap \check{\Omega})} \\ \text{[by 3.2.3.b and (3.33)]} \qquad &\leq C \, \|\bar{u}_i\|^2_{L^2(\Xi_i \cap \check{\Omega})}. \end{aligned}$$

Substituting the last inequality into (3.38), using the Friedrichs inequality (3.37) and exploit-

ing the bounded intersections of balls $\{\Xi_i\}_{i=1}^{n_0}$, we get

$$\begin{aligned}\|u - Q_0 u\|^2_{L^2(\hat{\Omega})} &\leq C \sum_{i=1}^{n_0} \|\bar{u}_i\|^2_{L^2(\Xi_i)} \\ &\leq C H_0^2 \sum_{i=1}^{n_0} |\bar{u}_i|^2_{H^1(\Xi_i)} \\ &\leq C H_0^2 \, |u|^2_{H^1(\Omega)} .\end{aligned}$$

We have therefore proved lemma 3.2.4.a, since

$$\begin{aligned}\|u - Q_0 u\|^2_{L^2(\Omega)} &= \|u - Q_0 u\|^2_{L^2(\check{\Omega})} + \|u - Q_0 u\|^2_{L^2(\hat{\Omega})} \\ &\leq C H_0^2 \, |u|^2_{H^1(\Omega^\Gamma)} + C H_0^2 \, |u|^2_{H^1(\Omega)} \\ &\leq C H_0^2 \, |u|^2_{H^1(\Omega)} .\end{aligned}$$

To finish the proof, we turn out to 3.2.4.b in $\hat{\Omega}$. We have

$$\begin{aligned}|Q_0 u|^2_{H^1(\hat{\Omega})} &\leq \sum_{i=1}^{n_0} |Q_0 u|^2_{H^1(\Theta_i \cap \hat{\Omega})} \\ &= \sum_{i=1}^{n_0} |Q\,(\bar{u}_i)|^2_{H^1(\Theta_i \cap \hat{\Omega})} \\ &\leq \sum_{i=1}^{n_0} \left| \sum_{j \in \mathcal{N}_i} \alpha_j(\bar{u}_i) \Phi_j \right|^2_{H^1(\Omega)} \\ &\leq \sum_{i=1}^{n_0} \left(\sum_{j \in \mathcal{N}_i} |\alpha_j(\bar{u}_i)| \cdot |\Phi_j|_{H^1(\Omega)} \right)^2 \\ &\leq C \frac{H_0^{d-1}}{\delta_0} H_0^{-d} \sum_{i=1}^{n_0} \sum_{j \in \mathcal{N}_i} \|\bar{u}_i\|^2_{L^2(\Theta_i)} \qquad \text{[by prop. 3.2.3.a and (3.33)]} \\ &\leq C \frac{H_0}{\delta_0} \sum_{i=1}^{n_0} |\bar{u}_i|^2_{H^1(\Xi_i)} \qquad \text{[by (3.37)]} \\ &\leq C \frac{H_0}{\delta_0} \, |u|^2_{H^1(\Omega)} .\end{aligned}$$

□

To prove theorem 3.2.4 we need other two lemmas. The first one is the following.

Lemma 3.2.5. *Let $\Omega_i \subset \mathbb{R}^d, d = 2, 3$, be a rectangle of diameter H, and let Γ_{δ_i} be a strip along its boundary of width $\delta > 0$. Then, for any function $u \in H^1(\Omega_i)$,*

$$\|u\|^2_{L^2(\Gamma_{\delta_i})} \leq C\delta^2 \left[\left(1 + \frac{H}{\delta} \right) |u|^2_{H^1(\Omega_i)} + \frac{1}{H\delta} \|u\|^2_{L^2(\Omega_i)} \right]. \tag{3.39}$$

Proof. The proof that we report here is taken from [DW94], and is only bounded to a square region $(0, H) \times (0, H)$. Since

$$u(x,0) = u(x,y) - \int_0^y \frac{\partial u(x,\tau)}{\partial y} d\tau,$$

we find, by elementary arguments, that

$$H \int_0^H |u(x,0)|^2 dx \leq 2 \int_0^H \int_0^H |u(x,y)|^2 dxdy + H^2 \int_0^H \int_0^H \left| \frac{\partial u}{\partial y} \right| dxdy.$$

Therefore,

$$H \int_0^H |u(x,0)|^2 dx \leq 2\|u\|_{L^2(\Omega_i)} + H^2 |u|^2_{H^1(\Omega_i)}.$$

Now consider the integral over a narrow layer of Ω_i next to one of the sides of the squares. Using similar arguments, we obtain

$$\int_0^H \int_0^\delta |u(x,y)|^2 dxdy \leq \delta^2 |u|_{H^1(\Omega_i)} + 2\delta \int_0^H |u(x,0)|^2 dx.$$

Combining this and the previous inequality yields

$$\int_0^H \int_0^\delta |u(x,y)|^2 dxdy \leq \delta^2 |u|_{H^1(\Omega_i)} + 2\delta \left(\frac{2}{H} \|u\|_{L^2(\Omega_i)} + H |u|^2_{H^1(\Omega_i)} \right).$$

□

We can now prove the following result, which extends other results presented in the literature.

Lemma 3.2.6. *Under definition 3.2.1 and for every finite element function $u \in V_h$, there exists a decomposition $\{u_i\}, i = 0, \ldots, M, u_i \in V_i$, such that*

$$u = \sum_{i=0}^{M} u_i,$$

and

$$\sum_{i=0}^{M} |u_i|^2_{H^1(\Omega)} \leq C \left(1 + \frac{H}{\delta}\right) \left(1 + \frac{H_0}{\delta_0}\right) |u|^2_{H^1(\Omega)},$$

with $H_0 \leq H$ and $\delta_0 \geq \delta$.

Proof. Define I_h to be the fine mesh operator $I_h : V \to V_h$ such that

$$I_h(u) = \sum_{i=1}^{n} u(\mathbf{x}_i) \varphi_i$$

where φ_i is a generic finite element basis on the fine mesh, and $\mathbf{x}_i$'s are the fine mesh nodal points. Let η_i be a partition of unity such that $\eta_i \in C_0^\infty(\Omega_i)$ and $0 \le \eta_i \le 1$. We then define

$$u_0 = Q_0 u, \quad w = u - u_0, \quad u_i = I_h(\eta_i w).$$

Clearly, by construction,

$$\sum_{i=0}^{M} u_i = u.$$

Because of the definitions on the overlap, we can ensure that the gradients of η_i are well behaved. That is, we can construct η_i so that $|\nabla \eta_i|^2_{L^\infty(\Omega)} \le C/\delta^2$. If K is a single element in the triangulation this can be expressed as

$$\|\eta_i - \bar{\eta}_i\|^2_{L^\infty(K)} \le C(h/\delta)^2, \tag{3.40}$$

$\bar{\eta}_i$ being the average of η_i on element K.

Over a single element K and using the inverse inequality, we get

$$\begin{aligned}
|u_i|^2_{H^1(K)} &= |I_h\left[\bar{\eta}_i w + (\eta_i - \bar{\eta}_i)w\right]|^2_{H^1(K)} \\
&\le 2\,|\bar{\eta}_i w|^2_{H^1(K)} + 2\,|I_h(\eta_i - \bar{\eta}_i)w|^2_{H^1(K)} \\
&\le 2\,|\bar{\eta}_i w|^2_{H^1(K)} + Ch^{-2}\,\|I_h(\eta_i - \bar{\eta}_i)w\|^2_{L^2(K)} \\
&\le 2\,|\bar{\eta}_i w|^2_{H^1(K)} + Ch^{-2}\,\|\eta_i - \bar{\eta}_i\|^2_{L^\infty(K)}\,\|I_h w\|^2_{L^2(K)}\,.
\end{aligned}$$

Now, we sum up over all the elements. The last term is identically zero for all elements K in the interior of Ω_i. Therefore, when we take the sum over all the elements, the last term only includes those elements in the overlapping region. Moreover, since a finite number, bounded independently of h, δ, H and H_0, of u_i is non-zero for any element K, we obtain summing over i and using (3.40)

$$\sum_{i=1}^{M} |u_i|^2_{H^1(\Omega)} \le C \sum_{i=1}^{M} |u_i|^2_{H^1(\Omega)} + C\delta^{-2} \sum_{i=1}^{M} \|u_i\|^2_{L^2(\Gamma_{\delta_i})}\,.$$

Using lemma 3.2.5 to bound the last term, we get

$$\begin{aligned}
\sum_{i=1}^{M} |u_i|^2_{H^1(\Omega)} &\le C\,|w|^2_{H^1(\Omega)} + C \sum_{i=1}^{M} \left[\left(1 + \frac{H}{\delta}\right)|u_i|^2_{H^1(\Omega)} + \frac{1}{H\delta}\,\|u_i\|^2_{L^2(\Omega)}\right] \\
&\le C\,|w|^2_{H^1(\Omega)} + C\left(1 + \frac{H}{\delta}\right)|w|^2_{H^1(\Omega)} + C\frac{1}{H\delta}\,\|w\|^2_{L^2(\Omega)} \\
\text{[by lemma 3.2.4.a]} \qquad &\le C\left(1 + \frac{H}{\delta}\right)|u - Q_0 u|^2_{H^1(\Omega)} + C\frac{H_0^2}{H\delta}\,|u|^2_{H^1(\Omega)} \\
\text{[by 3.2.4.b, } H_0 \le H\text{]} \qquad &\le C\left(1 + \frac{H}{\delta}\right)\left(1 + \frac{H_0}{\delta_0}\right)|u|^2_{H^1(\Omega)} + C\frac{H_0}{\delta}\,|u|^2_{H^1(\Omega)} \\
[\delta_0 \ge \delta] \qquad &\le C\left(1 + \frac{H}{\delta}\right)\left(1 + \frac{H_0}{\delta_0}\right)|u|^2_{H^1(\Omega)}\,.
\end{aligned}$$

Note that we have used the fact that $\delta_0 \geq \delta$ and $H_0 \leq H$. This latter inequality states that the coarse space must be sufficiently rich with respect to the number of subdomains.

□

The following theorem holds.

Theorem 3.2.4 (aggregation coarse space). *Let properties 3.2.2 and 3.2.3 hold. Then, for the additive two-level overlapping Schwarz method, when the overlap is uniform of width $\mathcal{O}(\delta)$ and $V_0 = \text{span}\{\Phi_i, i = 1, \dots, n_0\}$, there exists $C > 0$ such that*

$$\kappa(P_{S,C,add}^{-1} A) \leq C \left(1 + \frac{H_0}{\delta_0}\right) \left(1 + \frac{H}{\delta}\right). \tag{3.41}$$

Proof. The proof of theorem 3.2.4 is easily obtained using lemma 3.2.3. As verified by lemma 3.2.6, the first parameter C_0^2 of definition 3.2.1 is bounded by

$$C_0^2 \leq \left(1 + \frac{H}{\delta}\right) \left(1 + \frac{H_0}{\delta_0}\right).$$

The second parameter can be estimated by using a colouring argument. We know that we can colour the subdomains with N_c colours, independently of h and H. Taking the coarse space into account we have therefore that we can group the T_i into $N_c + 1$ classes. Hence, $\rho(\epsilon) \leq N_c + 1$. $\omega = 1$ since we suppose to use exact solvers on the subdomains. From lemma 3.2.3 the thesis follows. □

Remark 3.2.9. *If $\delta_0 = \delta = \eta h$, $\eta \geq 1$, and the diameter of the aggregates Θ_i bounded by H/χ, where $\chi \geq 1$, that is, $\chi H_0 = H$, we have that*

$$\begin{aligned} \kappa\left(P_{S,C,add}^{-1} A\right) &\leq C \left(1 + \frac{H}{\eta h}\right) \left(1 + \frac{H}{\chi \eta h}\right) \\ &\leq \frac{C}{\chi \eta^2} \left(\frac{H}{h}\right)^2. \end{aligned} \tag{3.42}$$

Equation (3.42) states that we can reduce the condition number by increasing the number of aggregates and the overlap. The overlap has a stronger influence on the reduction of the condition number; however, it is usually computationally expensive to use wider overlap, whereas larger values of χ have influence only on the coarse problem, whose size is remarkably smaller than that of the global problem. Hence, using a given overlap the effect of the preconditioner increases with the number of aggregates.

Theorem 3.2.4 considers only the final aggregate space V_0. It is possible to separate the effect of the smoother $\tilde{S}_0$ and of the non-smoothed aggregates. To that aim, we need the following two lemmas.

Lemma 3.2.7. *Consider non-smoothed functions $\tilde{\Phi}_i, i = 1, \ldots, n_0$, such that $\nabla\tilde{\Phi}_i$ is zero outside the overlapping part $\Gamma_{\tilde{\Theta}_i}$ of $\tilde{\Theta}_i$, and bounded by $1/\tilde{\delta}_0$ in $\Gamma_{\tilde{\Theta}_i}$. Then, $\tilde{\Phi}_i$ satisfy property 3.2.3 with $H_0 = \tilde{H}_0$ and $\delta = \tilde{\delta}_0$.*

Proof. Let $\tilde{\Phi}_i, i = 1, \ldots, n_0$ be a generic coarse function. We have that

$$|\tilde{\Phi}_i|^2_{H^1(\Omega)} \leq \left\|\nabla\tilde{\Phi}_i\right\|^2_{L^\infty(\Gamma_{\tilde{\Theta}_i})} |\Gamma_{\tilde{\Theta}_i}| \leq C\frac{1}{\tilde{\delta}_0^2}\tilde{\delta}_0\tilde{H}_0^{d-1}.$$

The second inequality follows from the fact that $\Gamma_{\tilde{\Theta}_i}$, where the gradient of $\tilde{\Phi}_i$ is non-zero, has a Lebesgue measure of order $\mathcal{O}(\tilde{\delta}_0\tilde{H}_0^{d-1})$ and the mesh is quasi-uniform. This proves 3.2.3.a. Now, 3.2.3.b is proved noting that

$$||\tilde{\Phi}_i||^2_{L^2(\Omega)} \leq C|\tilde{\Theta}_i| \leq C\tilde{H}_0^d.$$

where $\tilde{\Theta}_i$ is the support of $\tilde{\Phi}_i$. □

Lemma 3.2.8. *Let $\tilde{S}_0$ be defined as in equation (3.16), and let $\tilde{\Phi}_i, i = 1, \ldots, n_0$ satisfy properties 3.2.2 and 3.2.3 with H_0 replaced by $\tilde{H}_0$ and δ_0 by $\tilde{\delta}_0$. Then, for a fixed degree k of the smoother, the coarse functions $\Phi_i = \tilde{S}_0(k)\tilde{\Phi}_i$ satisfy properties 3.2.2 and 3.2.3 with $H_0 = \tilde{H}_0 + kh$ and $\delta_0 = \tilde{\delta}_0 + kh$.*

Proof. The effect of the smoother is to increase the diameter of the subdomains and the overlap. The application of the smoother results in a diameter of the aggregates of $\mathcal{O}(\tilde{H}_0+kh)$ and an overlap of order $\mathcal{O}(\tilde{\delta}_0 + kh)$. Since the triangulation is quasi-uniform, we have that

$$\text{diam}(\Theta_i) = C(\tilde{H}_0 + kh) \leq CH_0.$$

This verifies property 3.2.2.a. Using similar arguments, property 3.2.2.b is verified.

Now, let us prove that property 3.2.3.a holds. We have

$$\begin{aligned} |\Phi_i|^2_{H^1(\Omega)} = \left|\tilde{S}_0\tilde{\Phi}_i\right|^2_{H^1(\Omega)} &\leq \rho(\tilde{S}_0)^2 \left|\tilde{\Phi}_i\right|^2_{H^1(\Omega)} \\ &\leq C\rho(\tilde{S}_0)^2\frac{\tilde{H}_0^{d-1}}{\tilde{\delta}_0} \\ &\leq C\rho(\tilde{S}_0)^2\frac{H_0^{d-1}}{\delta_0}, \end{aligned} \tag{3.43}$$

since $\tilde{H}_0 \leq H_0$ and $\delta_0 = \tilde{\delta}_0 + kh = ih + kh \geq C\tilde{\delta}_0$ for fixed values of i and k.

As regards property 3.2.3.b, we have:

$$\|\Phi_i\|^2_{L^2(\Omega)} = \left\|\tilde{S}_0\tilde{\Phi}_i\right\|^2_{L^2(\Omega)} \leq \rho(\tilde{S}_0)^2 \left\|\tilde{\Phi}_i\right\|^2_{L^2(\Omega)}. \tag{3.44}$$

To prove property 3.2.3.c, we define the function

$$u(x) = \sum_{i=1}^{n_0} \tilde{\Phi}_i(x),$$

is equal to one $\forall \mathbf{x} \in \hat{\Omega}$, and consequently at every $x \in \Omega$ outside a strip of width $\mathcal{O}(\tilde{\delta}_0)$ around $\partial\Omega$. Thus, we obtain

$$\sum_{i=1}^{n_0} \Phi_i(x) = \left(\tilde{S}_0 \sum_{i=1}^{n_0} \tilde{\Phi}_i \right)(x) = (\tilde{S}_0 u)(x) = 1$$

at every $x \in \Omega$ outside a strip of width $\mathcal{O}(\tilde{\delta}_0) + kh = \mathcal{O}(\delta_0)$ around $\partial\Omega$. □

The following theorem holds.

Theorem 3.2.5 (smoothed coarse space). *Let properties 3.2.2 and 3.2.3 hold, with H_0 replaced by $\tilde{H}_0$ and δ_0 by $\tilde{\delta}_0$, and let $\tilde{S}_0$ be a smoother, such that $R_0 = \tilde{S}_0 \tilde{R}_0$, and $\rho(\tilde{S}_0)$ be the spectral radius of $\tilde{S}_0$. Then, there exists a $C > 0$ such that*

$$\kappa\left(P_{S,C,add}^{-1} A\right) \leq C \left(1 + (1 + \rho(\tilde{S}_0)^2) \frac{\tilde{H}_0}{\tilde{\delta}_0}\right) \left(1 + \frac{H}{\delta}\right).$$

Proof. The proof can be carried out by modifying the final part of the proof of theorem 3.2.4. Let $Q_0 : V_h \rightarrow V_0$ be the restriction operator corresponding to matrix R_0 and $\tilde{Q}_0 : V_h \rightarrow V_0$ the restriction operator defined by a tentative matrix $\tilde{R}_0$. Then, we have

$$\begin{aligned} |Q_0 u|^2_{H^1(\hat{\Omega})} &\leq \mathbf{u}^T R_0^T A R_0 \mathbf{u} \\ &= \mathbf{u}^T \tilde{R}_0^T \tilde{S}_0 A \tilde{S}_0 \tilde{R}_0 \mathbf{u} \\ &\leq \rho(\tilde{S}_0^2) \mathbf{u}^T \tilde{R}_0^T A \tilde{R}_0 \mathbf{u} \\ &= \rho(\tilde{S}_0^2) |\tilde{Q}_0 u|^2_{H^1(\Omega)}. \end{aligned}$$

We finally obtain

$$|Q_0 u|^2_{H^1(\Omega)} \leq C(1 + \rho(\tilde{S}_0)^2) \frac{\tilde{H}_0}{\tilde{\delta}_0}.$$

□

In the applications, often minimal-overlap Schwarz preconditioners are used. This means that $\delta = h$. In this case, it is convenient to use $\delta_0 = \delta = h$. This correspond to consider $\tilde{R}_0$, as defined by equation (3.14). Then, this tentative coarse matrix can be smoothed out using the simple Richardson smoother of equation (3.16). The required estimation of the spectral radius of A can be obtained by computing the spectral radius (or an approximation of it) of $\tilde{A}_0$, whose dimension is remarkably smaller than that of A. This matrix, of moderate dimension, can be used to estimate the largest eigenvalue of A. The resulting smoother reads

$$\tilde{S}_0(k) = \left(I - \frac{1.5}{\rho(\tilde{A}_0)} A\right)^k, \tag{3.45}$$

where 1.5 is a chosen correction factor to correct possible under-estimation of $\rho(A)$.

For each subdomain, the extended support of the smoothed coarse function can be used to increase the overlap, see remark 3.2.6.

The application of the minimal overlap smoothed aggregation preconditioner gives rise to the following algorithm 3.2.1.

Algorithm 3.2.1 (minimal-overlap smoothed aggregation preconditioner).

a. Define a minimal-overlap partition into subdomains;
b. Define a minimal-overlap partition into aggregates;
c. Form the non-smoothed restriction operator $\tilde{R}_0$, and compute the non-smoothed coarse matrix $\tilde{A}_0$;
d. Estimate the condition number of $\tilde{A}_0$;
e. Compute R_0 by smoothing out k times $\tilde{R}_0$, using the smoother (3.45);
f. Apply $P_{S,C,add}$ or $P_{S,C,hybrid}$ as defined by equations (3.9) and (3.10).

An Improved Convergence Bound

A possible improvement of the estimation given by theorem 3.2.5 can be obtained by assuming that each subdomain defines an aggregate. The proof, already presented in [LT01], is here reported for the sake of completeness. Using the notation of that paper, in this case one has $H_0 = H$, $\delta_0 = \delta$, and $n_0 = M$. We also need an additional function $\Phi_0 \in V_h$ associated to the boundary of $\partial\Omega$, so that the augmented set of functions forms a partition of the unity on the entire $\bar{\Omega}$. This additional function is only needed for the proof and not implemented in practice.

Property 3.2.4. *We assume that the basis functions* $\{\Phi_i\}, i = 0, \ldots, M$ *of the coarse space* V_0 *satisfy*

a. $\|\Phi_i\|^2_{L^\infty(\Omega)} \leq C$;

b. $\|\nabla\Phi_i\|^2_{L^\infty(\Omega)} \leq C/\delta^2$;

c. $\sum_{i=0}^{M} \Phi_i(x) = 1, \quad \forall x \in \bar{\Omega}$;

d. $\mathrm{supp}\,(\Phi_i) \subseteq \bar{\Omega}_i$.

The following theorem holds [LT01, Lemma 7].

Theorem 3.2.6. *Let the properties 3.2.2 (with H_0 replaced by H and δ_0 by δ) and 3.2.4 hold. Then, there exists a constant $C > 0$ such that*

$$\kappa(P_{S,C,add}^{-1}A) \leq C\left(1 + \frac{H}{\delta}\right).$$

Proof. The proof is similar to that of theorem 3.2.5. The difference is that now we consider the coarse basis functions Φ_i to decompose $u \in V_h$, instead of the partition of the unity as we did before. This gives

$$u_0 = Q_0 u, \quad u_i = I_h(\Phi_i(u - u_{0,i})), \quad i = 0, \ldots, M$$

where $u_0 = \sum_{i=0}^{n_0} \Phi_i u_{0,i}$ and, by construction, $n_0 = M + 1$ (the M aggregates plus the additional function Φ_0. This is a proper decomposition, since

$$\begin{aligned}
\sum_{i=0}^{M} u_i &= \sum_{i=0}^{M} u_{0,i} + \sum_{i=0}^{M} I_h(\Phi_i(u - u_{0,i})) \\
&= \sum_{i=0}^{M} I_h(\Phi_i u_{0,i} + \Phi_i u - \Phi_i u_{0,i}) \\
&= \sum_{i} \Phi_i u = u
\end{aligned}$$

since $\sum_i \Phi_i(x) = 1, \forall x \in \Omega$, owing to assumption 3.2.4. Let $\bar{\Phi}_{K,i}$ indicate the average of Φ_i over the element K. Using inverse inequality, we get

$$\begin{aligned}
|u_i|^2_{H^1(K)} &= \left|I_h\left[\bar{\Phi}_{K,i}u + (\Phi_{K,i} - \bar{\Phi}_{K,i})u\right]\right|^2_{H^1(K)} \\
&\leq 2\left|\bar{\Phi}_{K,i}u\right|^2_{H^1(K)} + 2\left|I_h(\Phi_{K,i} - \bar{\Phi}_{K,i})u\right|^2_{H^1(K)} \\
&\leq 2\left|\bar{\Phi}_{K,i}u\right|^2_{H^1(K)} + Ch^{-2}\left\|I_h(\Phi_i - \bar{\Phi}_{K,i})u\right\|^2_{L^2(K)} \\
&\leq 2\left|\bar{\Phi}_{K,i}u\right|^2_{H^1(K)} + Ch^{-2}\left\|\Phi_i - \bar{\Phi}_{K,i}\right\|^2_{L^\infty(K)}\left\|I_h u\right\|^2_{L^2(K)}.
\end{aligned}$$

Now, we sum up over all the elements. The last term has a non-zero contribution only near the boundaries of Ω_i, giving

$$\begin{aligned}
|u_i|^2_{H^1(\Omega_i)} &\leq C\left|u_i\right|^2_{H^1(\Omega_i)} + C\delta^{-2}\left\|u_i\right\|^2_{L^2(\Gamma_{\delta_i})} \\
&= C\left|I_h(\Phi(u - u_{0,i}))\right|^2_{H^1(\Omega_i)} + C\delta^{-2}\left\|I_h(\Phi(u - u_{0,i}))\right\|^2_{L^2(\Gamma_{\delta_i})} \\
&= C\left|I_h(\Phi(u - u_{0,i}))\right|^2_{H^1(\Omega_i)} + C\left(1 + \frac{H}{\delta}\right)\left|I_h(\Phi(u - u_{0,i}))\right|^2_{H^1(\Omega_i)} \\
&\leq C\left(1 + \frac{H}{\delta}\right)|u|^2_{H^1(\Omega)},
\end{aligned}$$

where we have used lemma 3.2.5. Finally, from lemma 3.2.4.b with $H_0 = H$ and $\delta_0 = \delta$, we obtain

$$\begin{aligned}\sum_{i=0}^{M} |u_i|^2_{H^1(\Omega)} &\leq |u_0|^2_{H^1(\Omega)} + \sum_{i=0}^{M} |u_i|^2_{H^1(\Omega)} \\ &\leq C\frac{H}{\delta}|u|^2_{H^1(\Omega)} + \left(1+\frac{H}{\delta}\right)|u|^2_{H^1(\Omega)} \\ &\leq C\left(1+\frac{H}{\delta}\right)|u|^2_{H^1(\Omega)}\,.\end{aligned}$$

□

3.2.7 Numerical Results

We report in this section some numerical results for the problem

$$\begin{cases} -\Delta u &= 1 \quad \text{in } \Omega = (0,1)\times(0,1) \\ u &= 0 \quad \text{on } \partial\Omega, \end{cases} \tag{3.46}$$

which is a special case of our model problem (3.1), with $\alpha_{r,s} = 1$, $f = 1$, and $\Omega = (0,1)\times(0,1)$, $\Gamma_D = \partial\Omega$. The mesh is built by dividing Ω into n^2 equal squares and subdividing them into two triangles. Thus, we obtain a triangulation with $h = \frac{1}{n}$. As regards the domain decomposition, we consider overlapping squares Ω_i of area H^2. We use linear finite-elements, and solve the linear system (2.1) by the conjugate gradient method. The tables report the estimated condition number for the preconditioned system; see for instance [GvL83]. This is obtained by computing the condition number of a suitable Lanczos matrix, which approximates the preconditioned operator. The stopping criteria is ϵ^2, where ϵ is the machine precision.

Table 3.1 gives the condition number for the one-level Schwarz preconditioner. The minimal overlap version ($\delta = h$) is used here after. The condition number grows as $\mathcal{O}(1/hH)$ as theory predicts [QV99]. The following tables 3.2 and 3.3, which are about the two-level Schwarz preconditioner, confirm the bound provided by theorem 3.2.2. Hybrid preconditioners behave only slightly better than additive ones, therefore their use for these problems seems inconvenient. Tables 3.4 and 3.5 report the influence of a non-smoothed aggregation procedure, with minimal overlap among the aggregates ($\delta_0 = h$). The effect of the parameter χ of equation (3.42) can be clearly appreciated. $P^{-1}_{S,C,hybrid}$ shows the same convergence rate of the additive version, although the estimated condition number is sensibly smaller. Note that two matrix-vector products are needed by $P^{-1}_{S,C,hybrid}$, thus making its application computationally more expensive than that of $P^{-1}_{S,C,add}$. However, differently from "classical" coarse spaces, hybrid preconditioners perform much better than their additive versions, with significant reduction in the condition number.

Finally, the tables 3.6, 3.7, 3.8, 3.9, and 3.10 show the influence of the smoother (3.45) for different values of k. Note that, for all the smoothers, the hybrid version performs significantly better than the additive version.

P_S	$H = 1/2$	$H = 1/4$	$H = 1/8$	$H = 1/16$
$h = 1/16$	15.95	27.09	52.08	-
$h = 1/32$	31.69	54.52	104.85	207.67
$h = 1/64$	63.98	109.22	210.07	416.09
$h = 1/128$	127.99	218.48	420.04	832.57

Table 3.1: Estimated condition number using the one-level Schwarz preconditioner P_S with minimal overlap ($\delta = h$).

$P_{S,C,add}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/32$	7.03	4.94	-	-
$h = 1/64$	12.73	7.59	4.98	-
$h = 1/128$	23.62	13.17	7.66	4.99
$h = 1/256$	45.33	24.34	13.28	-

Table 3.2: Estimated condition number for $P_{S,C,add}^{-1}A$, with a coarse space built using a coarse grid (standard coarse space).

$P_{S,C,hybrid}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/32$	6.11	3.56	-	-
$h = 1/64$	11.47	6.24	3.58	-
$h = 1/128$	22.26	11.71	6.27	3.58
$h = 1/256$	43.86	22.71	11.77	-

Table 3.3: Estimated condition number for $P_{S,C,hybrid}^{-1}A$, with a coarse built using a coarse grid (standard coarse space).

$P_{S,C,add}$	χ	$H = 1/4$	$H = 1/8$	$H = 1/16$
$h = 1/16$	1	13.37	8.87	-
$h = 1/32$	1	26.93	17.71	9.82
$h = 1/64$	1	54.33	35.21	19.70
$h = 1/128$	1	109.39	70.22	39.07
$h = 1/32$	2	13.13	7.78	-
$h = 1/64$	2	27.18	15.28	9.96
$h = 1/32$	3	7.61	-	-
$h = 1/64$	3	17.13	8.39	-

Table 3.4: Estimated condition number for $P_{S,C,add}^{-1}A$, with a coarse space built using non-smoothed aggregation.

$P_{S,C,hybrid}$	χ	$H = 1/4$	$H = 1/8$	$H = 1/16$
$h = 1/16$	1	5.24	2.89	-
$h = 1/32$	1	10.64	5.66	2.97
$h = 1/64$	1	21.60	11.34	5.79
$h = 1/128$	1	43.65	22.77	11.55
$h = 1/32$	2	5.5	2.7	-

Table 3.5: Estimated condition number for $P_{S,C,hybrid}^{-1}A$, with a coarse space built using non-smoothed aggregation.

$P_{S,C,add}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/16$	11.91	6.02	-	-
$h = 1/32$	25.59	14.95	6.28	-
$h = 1/64$	50.03	32.64	16.23	6.36
$h = 1/128$	108.13	67.75	35.81	16.631
$h = 1/256$	218.57	137.91	74.55	-

Table 3.6: Estimated condition number for $P_{S,C,add}^{-1}A$, with a coarse space built using aggregation and smoother (3.45) with $k = 1$ and $\chi = 1$.

$P_{S,C,hybrid}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/16$	5.09	2.86	-	-
$h = 1/32$	10.49	5.63	2.96	-
$h = 1/64$	21.46	11.31	5.77	2.99
$h = 1/128$	43.51	22.75	11.54	5.82

Table 3.7: Estimated condition number for $P_{S,C,hybrid}^{-1}A$, with a coarse space built using aggregation and smoother (3.45) with $k = 1$ and $\chi = 1$.

$P_{S,C,add}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/16$	10.71	5.70	-	-
$h = 1/32$	24.28	12.78	5.92	-
$h = 1/64$	51.77	30.35	19.70	10.12
$h = 1/128$	106.89	65.42	32.97	13.82

Table 3.8: Estimated condition number for $P_{S,C,add}^{-1}A$, with a coarse space built using aggregation and smoother (3.45) with $k = 2$ and $\chi = 1$.

$P_{S,C,hybrid}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/16$	5.00	2.82	-	-
$h = 1/32$	10.39	5.611	.	-
$h = 1/64$	21.36	11.29	5.78	2.98

Table 3.9: Estimated condition number for $P^{-1}_{S,C,hybrid}A$, with a coarse space built using aggregation and smoother (3.45) with $k = 2$ and $\chi = 1$.

$P_{S,C,add}$	$H = 1/4$	$H = 1/8$	$H = 1/16$	$H = 1/32$
$h = 1/16$	9.77	5.59	-	-
$h = 1/32$	23.12	11.08	5.88	-
$h = 1/64$	50.56	28.31	11.55	-

Table 3.10: Estimated condition number for $P^{-1}_{S,C,add}A$, with a coarse space build using aggregation, using smoother (3.45) with $k = 3$ and $\chi = 1$.

3.2.8 Concluding Remarks about Schwarz Preconditioners

In this section we have considered both one-level and two-level Schwarz preconditioners. For the latter, a coarse space V_0 based on the concept of aggregation has been introduced. The basis functions of V_0 are built as a linear combination of the basis functions of the fine space, and the use of a coarse grid is not required.

A theoretical analysis is reported for an elliptic model problem, discretized on a quasi-uniform mesh with finite-elements. The theory extends results presented in literature; see [JKMK00, LT01]. There, the authors assume equal overlap among the subdomains δ and the aggregates δ_0, and equal size of the subdomains H and the aggregates H_0. Instead, in the bound of theorem 3.2.5, the influence of the subdomain decomposition, the non-smoothed aggregates, and the smoother, are kept separate. The bound involves geometrical quantities on the subdomains (H and δ) and on the aggregates (H_0 and δ_0), and the spectral radius of the smoother $\rho(\tilde{S}_0)$.

Although the presented condition number estimate is less favourable than other estimates proposed in literature based on a coarser triangulation, the aggregation procedure here outlined has many advantages. First of all, it does not require the generation of a coarse grid, as other two-level methods do. This opens the possibility to use the aggregation procedures for more general cases with complex geometries without loosing the power of two-level methods. Secondly, it can be 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. Moreover, the computational complexity of the aggregation procedure is smaller since the method is simpler to implement. This simplicity has its origin in the way the restriction and interpolation operators are defined.

For the special case $H = H_0$ and $\delta = \delta_0$, theorem 3.2.6 furnishes an improved convergence bound, which depends linearly on H/δ. Although, the basis functions of the coarse space must satisfy different properties: property 3.2.3 for theorem 3.2.5 and property 3.2.4 for theorem 3.2.6. It can be proved that the smoother presented in section 3.2.3 satisfies property 3.2.3, while, to our knowledge, there are no proofs that they satisfy property 3.2.4. Theorem 3.2.5 is more general and allows aggregates and subdomains of different shapes and overlap, but numerical results (at least for our model problem) seem to satisfy theorem 3.2.6.

3.3 Part II: Schur Complement Based Preconditioners

In this section we present non-overlapping DD methods based on the solution of the Schur complement system.

We will introduce two different decompositions, named element-oriented (EO) and vertex-oriented (VO). The focus is on the latter. First, we reinterpret the VO decomposition as a partition of the computational domain Ω into M non-overlapping subdomains $\Omega_i, i = 1, \ldots, M$, plus a further "subdomain" Ω_Γ, so that $\overline{\Omega_i} \cap \overline{\Omega_j} = \emptyset$ and $\overline{\Omega_i} \cap \overline{\Omega_\Gamma} \neq \emptyset, i = 1, \ldots, M$. Hence, one subdomain is connected to all the others, that are in fact disconnected components. The resulting decomposition may be seen as a 2-subdomain formulation where one of the two subdomains plays a special role and allows a rapid transfer of information. Several preconditioners for the Schur complement matrix arising from the VO decomposition are presented and analysed for a model elliptic problem. Numerical comparison with the Neumann/Neumann (NN) and the balancing Neumann/Neumann (BNN) preconditioners for EO decompositions are given. Finally, an algebraic preconditioner derived from approximate solutions of the internal problems is presented.

This part is organised as follows. After some general results about SC in section 3.3.1, in section 3.3.2 we outline the main differences between EO and VO decompositions. The solution of the SC system and the set up of a also preconditioner is also addressed. Here, an unusual precondition, the so-called "swiss carpet preconditioner", is presented for the VO decomposition. This preconditioner will be extended in section 3.3.4. Section 3.3.5 introduces the approximate SC methods. Numerical results are presented in section 3.3.6. In section 3.3.7 we draw some concluding remarks.

3.3.1 General Results about Schur Complements

In this section we report some general results about Schur complements, taken from [Axe94, Chapter 3] and [LM00].

Let I and B be two disjoint sets of indices, whose cardinality is n_I and n_B, respectively, such that a generic vector $\mathbf{x} \in \mathbb{R}^n$ can be written as $\mathbf{x} = [\mathbf{x}_I, \mathbf{x}_B]$, and a generic matrix

$A \in \mathbb{R}^{n\times n}$ can be reordered consistently with I and B,

$$A = \begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix}, \tag{3.47}$$

where A_{II} and A_{BB} are square matrices.

Definition 3.3.1. *If A_{II} is non-singular, we define*

$$S \equiv A/A_{II} = A_{BB} - A_{BI}A_{II}^{-1}A_{IB}. \tag{3.48}$$

If A_{II} is singular, we define

$$S \equiv A/A_{II} = A_{BB} - A_{BI}A_{II}^{+}A_{IB} \tag{3.49}$$

where A_{II}^{+} is the Moore-Penrose inverse of matrix A_{II}. S is called the Schur complement of A with respect to A_{II}.

Note that S is the matrix that results when we eliminate the block A_{BI} using block Gaussian elimination with A_{II} as pivot block. In fact, the block matrix triangular factorisation of A is readily found to be

$$A = \begin{pmatrix} I_I & 0 \\ A_{BI}A_{II}^{-1} & I_B \end{pmatrix} \begin{pmatrix} A_{II} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I_I & A_{II}^{-1}A_{IB} \\ 0 & I_B \end{pmatrix}, \tag{3.50}$$

I_I and I_B being the identity matrices whose dimension is n_I and n_B, respectively. From (3.50), if A is definite,

$$A^{-1} = \begin{pmatrix} I & -A_{II}^{-1}A_{IB} \\ 0 & I \end{pmatrix} \begin{pmatrix} A_{II}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -A_{IB}A_{II}^{-1} & I \end{pmatrix} \tag{3.51}$$

or

$$A^{-1} = \begin{pmatrix} A_{II}^{-1} + A_{II}^{-1}A_{IB}S^{-1}A_{BI}A_{II}^{-1} & -A_{II}^{-1}A_{IB}S^{-1} \\ -S^{-1}A_{BI}A_{II}^{-1} & S^{-1} \end{pmatrix}, \tag{3.52}$$

An alternative expression for A^{-1} is

$$A^{-1} = \begin{pmatrix} {S_I}^{-1} & -A_{II}^{-1}A_{IB}S^{-1} \\ -S^{-1}A_{BI}A_{II}^{-1} & S^{-1} \end{pmatrix}, \tag{3.53}$$

where $S_I \equiv A/A_{BB} = A_{II} - A_{IB}A_{BB}^{-1}A_{BI}$.

If A (and henceforth S) is indefinite but A_{II}^{-1} exists, we find

$$A^{+} = \begin{pmatrix} A_{II}^{-1} + A_{II}^{-1}A_{IB}S^{+}A_{BI}A_{II}^{-1} & -A_{II}^{-1}A_{IB}S^{+} \\ -S^{+}A_{BI}A_{II}^{-1} & S^{+} \end{pmatrix}. \tag{3.54}$$

Note that from formula (3.54) we can obtain A^+ as follows:

$$S^+ = \begin{pmatrix} 0 & I_B \end{pmatrix} A^+ \begin{pmatrix} 0 \\ I_B \end{pmatrix}. \tag{3.55}$$

Thus

$$S^+ \mathbf{v}_B = \begin{pmatrix} 0 & I_B \end{pmatrix} A^+ \begin{pmatrix} 0 \\ I_B \end{pmatrix} \mathbf{v}_B,$$

for a given vector $\mathbf{v}_B \in \mathbb{R}^{n_B}$.

Whether A is invertible, equation (3.55) can be rewritten as

$$S^{-1} = \begin{pmatrix} 0 & I_B \end{pmatrix} A^{-1} \begin{pmatrix} 0 \\ I_B \end{pmatrix}. \tag{3.56}$$

The following theorem shows that if A is symmetric and positive definite, then S is symmetric and positive definite too.

Theorem 3.3.1. *Let A be a symmetric positive definite, A_{II} be regular, and $\mathbf{x} = [\mathbf{x}_I, \mathbf{x}_B]$ be a partitioning of $\mathbf{x}$ consistent with the partitioning of A (3.47). Then*

a. A_{II} and A_{BB} are symmetric and positive definite;

b. $\langle A\mathbf{x}, \mathbf{x}\rangle \geq \langle S\mathbf{x}_B, \mathbf{x}_B\rangle$;

c. For any $\mathbf{x}_B$, $\min_{\mathbf{x}_I} \langle A\mathbf{x}, \mathbf{x}\rangle = \langle S\mathbf{x}_B, \mathbf{x}_B\rangle$;

d. $\min_{\mathbf{x}_I, \mathbf{x}_B \neq 0} \langle A\mathbf{x}, \mathbf{x}\rangle = \min_{\mathbf{x}_B \neq 0} \langle S\mathbf{x}_B, \mathbf{x}_B\rangle$.

Proof. To prove 3.3.1.a, note that $\langle A\mathbf{x}, \mathbf{x}\rangle = \langle A_{II}\mathbf{x}_I, \mathbf{x}_I\rangle$ if $\mathbf{x}_B = \mathbf{0}$ while $\langle A\mathbf{x}, \mathbf{x}\rangle = \langle A_{BB}\mathbf{x}_B, \mathbf{x}_B\rangle$ if $\mathbf{x}_I = \mathbf{0}$, provided that $\mathbf{x}$ is partitioned consistently with the partition of A. Next, from (3.50) and the fact that $A_{BI}^T = A_{IB}$ it follows

$$\langle A\mathbf{x}, \mathbf{x}\rangle = \langle A_{II}\mathbf{x}_I + A_{II}^{-1}A_{IB}\mathbf{x}_B, \mathbf{x}_I + A_{II}^{-1}A_{IB}\mathbf{x}_B\rangle + \langle S\mathbf{x}_B, \mathbf{x}_B\rangle \geq \langle S\mathbf{x}_B, \mathbf{x}_B\rangle,$$

which shows 3.3.1.b. Properties 3.3.1.c and 3.3.1.d follow from the above inequality by choosing $\mathbf{x}_I = -A_{II}^{-1}A_{IB}\mathbf{x}_B$. □

The following relation between $\kappa(A)$ and $\kappa(S)$ holds.

Lemma 3.3.1. *Let A be symmetric positive definite and partitioned into the form (3.47), where A_{II} is regular, and let S be its the Schur complement (3.48). Then*

(a) $\lambda_{min}(A) \leq \lambda_{min}(A_{II}) \leq \lambda_{max}(A_{II}) \leq \lambda_{max}(A)$ and, hence, $\kappa(A_{II}) \leq \kappa(A)$;

(b) $\lambda_{min}(A) \leq \lambda_{min}(S) \leq \lambda_{max}(S) \leq \lambda_{max}(A)$ and, hence, $\kappa(S) \leq \kappa(A)$.

Proof. Being A (and henceforth S) symmetric positive definite, their extreme eigenvalues are given by

$$\lambda_{min}(A) = \min_{\mathbf{x}\neq\mathbf{b}} \frac{\langle A\mathbf{x},\mathbf{x}\rangle}{\langle \mathbf{x},\mathbf{x}\rangle}, \quad \lambda_{max}(A) = \max_{\mathbf{x}\neq\mathbf{b}} \frac{\langle A\mathbf{x},\mathbf{x}\rangle}{\langle \mathbf{x},\mathbf{x}\rangle}.$$

We have

$$\frac{\langle A\mathbf{x},\mathbf{x}\rangle}{\langle \mathbf{x},\mathbf{x}\rangle} = \frac{\langle A_{II}\mathbf{x}_I,\mathbf{x}_I\rangle}{\langle \mathbf{x}_I,\mathbf{x}_I\rangle},$$

for any $\mathbf{x} = [\mathbf{x}_I, \mathbf{0}]$. In particular,

$$\lambda_{min}(A) \leq \frac{\langle A_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle}{\langle \boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle} = \lambda_{min}(A_{II}), \tag{3.57}$$

where $\boldsymbol{\eta}_I$ is the eigenvector of A_{II} for $\lambda_{min}(A_{II})$. Similarly,

$$\lambda_{max}(A) \geq \lambda_{max}(A_{II}),$$

which together with (3.57) shows 3.3.1.a. Similarly, we have

$$\lambda_{min}(A) \leq \frac{\langle A\mathbf{x},\mathbf{x}\rangle}{\langle \mathbf{x},\mathbf{x}\rangle} = \frac{\langle S\mathbf{x}_B,\mathbf{x}_B\rangle}{\langle \mathbf{x}_I,\mathbf{x}_I\rangle + \langle \mathbf{x}_B,\mathbf{x}_B\rangle} \leq \frac{\langle S\mathbf{x}_B,\mathbf{x}_B\rangle}{\langle \mathbf{x}_B,\mathbf{x}_B\rangle}$$

for any $\mathbf{x} = [\mathbf{x}_I, \mathbf{x}_B]$, where $\mathbf{x}_I = -A_{II}^{-1}A_{IB}\mathbf{x}_B$ and $\mathbf{x}_B$ is arbitrary. Hence

$$\lambda_{min}(A) \leq \lambda_{min}(S).$$

From (3.53) and (3.57) we have that $\lambda_{min}(A^{-1}) \leq \lambda_{min}(S^{-1})$, that is,

$$\lambda_{max}(A) \geq \lambda_{max}(S)$$

which shows part 3.3.1.b. □

In the case when A is symmetric semi-positive definite, the following result holds.

Theorem 3.3.2. *Let A be a symmetric semi-positive definite matrix, and let ε be a positive number. Then,*

$$\lim_{\varepsilon\to 0} S_\varepsilon = S, \tag{3.58}$$

where S_ε is the Schur complement matrix of $A + \varepsilon I$ with respect to the set of nodes B.

Proof. Let U be a unitary matrix such that

$$A = U^T \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} U,$$

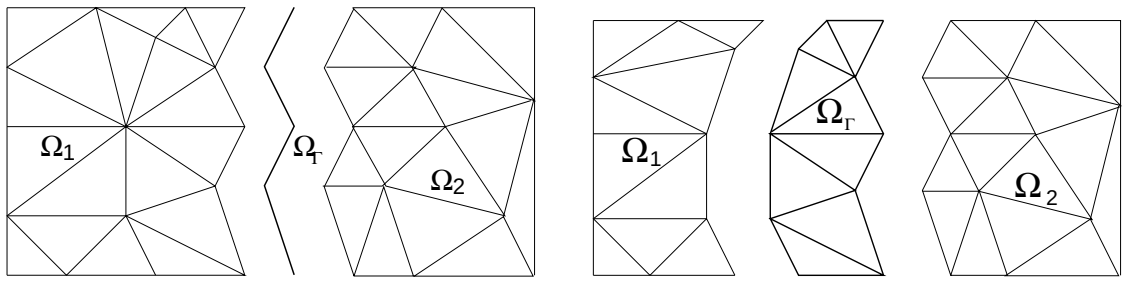


Figure 3.8: Example of element-oriented (left) and vertex-oriented (right) decomposition.

where $D > 0$ is a diagonal matrix of eigenvalues. Then for any $\varepsilon > 0$, by computation and using the fact that $A_{IB} = A_{II}R$ for a suitable matrix R (see [LM00]),

$$\begin{aligned}
S_\varepsilon &= (A_{BB} + \varepsilon I) - A_{BI}(A_{II} + \varepsilon I)^{-1} A_{IB} \\
&= (A_{BB} + \varepsilon I) - R^T A_{II}(A_{II} + \varepsilon I)^{-1} A_{II} R \\
&= (A_{BB} + \varepsilon I) - R^T U^T \begin{pmatrix} D(D+\varepsilon I)^{-1} D & 0 \\ 0 & 0 \end{pmatrix} UR \\
&\overset{\varepsilon \to 0}{\to} A_{BB} - R^T A_{II} R \\
&= A_{BB} - R^T A_{II} A_{II}^{+} A_{II} R \\
&= A_{BB} - A_{BI} A_{II}^{+} A_{IB} \\
&= S.
\end{aligned}$$

□

Remark 3.3.1. *According to [Axe94], the term Schur complement seems to have first appeared in Haynsworth [Hay68]. Nowadays, the amount of literature about SC is enormous. The reader is referred to [WZZ99, LM00] for recent overviews on the subject.*

3.3.2 Element-oriented and Vertex-oriented Decomposition

Let $\Omega \subset \mathbb{R}^d, d = 2, 3$ be a generic shape-regular domain. The construction of the partition of Ω will be made in the following way. We first triangulate Ω and indicate by $\mathcal{T}_h$ the corresponding mesh. For the sake of simplicity we assume that the boundary of Ω coincides with that of the triangulation.

We partition $\mathcal{T}_h$ into $M + 1$ non-overlapping parts, namely $\mathcal{T}_{h,1}$, $\mathcal{T}_{h,2}, \ldots, \mathcal{T}_{h,M}$ and $\mathcal{T}_{h,\Gamma}$. For each $i = 1, \ldots, M$ we may associate to $\mathcal{T}_{h,i}$ a subdomain Ω_i, formed by the interior of the union of the elements of $\mathcal{T}_{h,i}$, while Ω_Γ is formed by union of the elements contained on $\mathcal{T}_{h,\Gamma}$. For $\Omega_\Gamma = \Omega \setminus \cup_{i=1}^{M} \Omega_i$ we may face the two following cases:

- Ω_Γ reduces to a finite number of disjoint measurable $d-1$ manifolds; see the picture on the left of figure 3.8 for an example of such decomposition in the case $M = 2$. This

situation represents the common case where $\cap_{i=1}^{M} \overline{\Omega_i} = \Omega_\Gamma$, i.e. Ω_Γ is the discretisation of the common part Ω_Γ of the boundary of Ω_i. This type of decomposition will be called *element oriented* (EO) decomposition, because each element of $\mathcal{T}_h$ belongs exclusively to one of the M subdomains $\overline{\Omega_i}$;

- Ω_Γ is a subset of $\mathbb{R}^d$ formed by only one layer of elements; see the picture on the right of figure 3.8 for an example of such a decomposition. The portion of space Ω_Γ of which $\mathcal{T}_{h,\Gamma}$ is a triangulation, is now formed by the union of a finite number of "strips" laying between the Ω_i. It will be called *vertex oriented* (VO) decomposition, because each vertex belongs exclusively to one of the M subdomains $\overline{\Omega_i}$. The crucial aspect of this partitioning is that $\dim(\Omega_\Gamma) = \dim(\Omega_i)$.

The nodes on Ω_Γ will be called *border* nodes, and in particular those on $\mathcal{T}_{h,i} \cap \Omega_\Gamma$ are the border nodes of the domain Ω_i. A node of $\mathcal{T}_{h,i}$ which is not a border node is said to be *internal* to $\Omega_i, i = 1, \ldots, M$.

We will consistently use the subscripts I and B to indicate internal and border nodes, respectively, while the subscript i will denote the domain we are referring to. Thus, $\mathbf{u}_{I,i}$ will indicate the vector of unknowns associated to nodes internal to Ω_i, while $\mathbf{u}_{B,i}$ is associated to the border nodes. These two sets will be indicated by $\sigma_{I,i}$ and $\sigma_{B,i}$, and their cardinality by $n_{I,i}$ and $n_{B,i}$, respectively. For ease of notation, in the following we will often avoid to make a distinction between a domain Ω and its triangulation $\mathcal{T}_h$, when no ambiguity is possible.

Note that by construction, in an EO decomposition all the vertices on Ω_Γ are shared among two or more subdomains Ω_i, while in a VO decomposition those vertices belong to exactly two subdomains: Ω_Γ and one of the Ω_i. EO decompositions lead to well-known SC based schemes, while VO decomposition allows a variational reinterpretation of some algebraic schemes and gives rise to new schemes.

The choice of a VO or EO decomposition largely affects both the numerical algorithm and the data structures used in a parallel code. Often, a VO approach is preferred since the transition region Ω_Γ may be replicated on the processors which holds Ω_i and provides a means of data communication. In this way if the discrete operator associated to matrix A has a compact stencil (as it is often the case with finite elements or finite volume techniques) the matrix-vector product may be easily made in parallel. The VO technique is also the matter of choice of many parallel linear algebra packages. Indeed, with a VO approach one may derive the local operators directly from the assembled global matrix A, while adopting an EO decomposition would require to work at the level of the (problem dependent) assembly process. On the other side, some quasi-optimality results have been proved for EO SC matrices.

The difference between the two approaches can be easily seen for the case $M = 2$.

We will first consider the EO approach.

We renumber the vector $\mathbf{u}$ of the unknowns by putting first the unknowns associated to nodes internal to Ω_1, followed by the ones internal to Ω_2, and finally those on Ω_Γ. Equation

(2.1) can be written in block form as

$$\begin{pmatrix} A_{II,1} & 0 & A_{IB,1} \\ 0 & A_{II,2} & A_{IB,2} \\ A_{BI,1} & A_{BI,2} & A_{BB,1}+A_{BB,2} \end{pmatrix} \begin{pmatrix} \mathbf{u}_{I,1} \\ \mathbf{u}_{I,2} \\ \mathbf{u}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{I,1} \\ \mathbf{f}_{I,2} \\ \mathbf{f}_B \end{pmatrix} . \tag{3.59}$$

Here $A_{II,i}$ contains the elements of A involving internal nodes of subdomain Ω_i, while $A_{IB,i}$ are blocks formed by the contribution of boundary nodes to the linear equations associated to the internal nodes. In $A_{BI,i}$ we have the terms that link the border nodes with the internal ones. Since A is symmetric matrix, $A_{BI,i} = A_{IB,i}^T$. Finally, $A_{BB} = \sum_{i=1}^{2} A_{BB,i}$ is the block that involves only border nodes and it is split into two parts, each built from the contribution coming from the corresponding subdomain,

$$[A_{BB,i}]_{k,j} = a(\varphi_j|_{\Omega_i}, \varphi_k|_{\Omega_i}), \tag{3.60}$$

where φ_k, φ_j are the finite element shape functions associated to border nodes k and j, respectively, restricted to the subdomain Ω_i. An analogous splitting involves the right hand side $\mathbf{f}_B = \mathbf{f}_{B,1} + \mathbf{f}_{B,2}$. The two blocks of zeros arise since the discrete operator has a compact stencil.

The elimination of internal nodes leads to the equation

$$S_{EO}\mathbf{u}_B = \mathbf{g} \tag{3.61}$$

where $S = S_1 + S_2$ and $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2$, with

$$S_i = A_{BB,i} - A_{BI,i}A_{II,i}^{-1}A_{IB,i} \tag{3.62}$$

$$\mathbf{g}_i = \mathbf{f}_{B,i} - A_{BI,i}A_{II,i}^{-1}\mathbf{f}_{I,i}. \tag{3.63}$$

The generalisation to an arbitrary number of subdomains is straightforward. We introduce the restriction matrix $\bar{R}_i \in \mathbb{R}^{n_{B,i}\times n_B}$ which extracts from a global vector $\mathbf{u} \in \mathbb{R}^{n_B}$ the entries corresponding to the border nodes of Ω_i, i.e. $\mathbf{u}_{B,i} = \bar{R}_i\mathbf{u}_B$. Its transpose $\bar{R}_i^T$ is the prolongation matrix relative to subdomain Ω_i which extends by zero a vector of border values.

In the multi-domain SC system, we have

$$S_{EO} = \left(\sum_{i=1}^{M} \bar{R}_i^T S_i \bar{R}_i\right), \quad \mathbf{g} = \mathbf{f}_B - \sum_{i=1}^{M} \bar{R}_i^T A_{BI,i}A_{II,i}^{-1}\,\mathbf{f}_{I,i}.$$

Remark 3.3.2. *The construction of the matrices $A_{BB,i}$ in (3.60) requires to operate at the level of the matrix assembly by the finite element code. In general, there is no way to recover them from the assembled matrix A.*

We turn now to the VO approach. If $M = 2$,

$$A\mathbf{u} = \begin{pmatrix} A_{II,1} & A_{IB,1} & 0 & 0 \\ A_{BI,1} & A_{BB,1}+A_{BB,\Gamma}^{(1,1)} & 0 & A_{BB,\Gamma}^{(1,2)} \\ 0 & 0 & A_{II,2} & A_{IB,2} \\ 0 & A_{BB,\Gamma}^{(2,1)} & A_{BI,2} & A_{BB,2}+A_{BB,\Gamma}^{(2,2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_{I,1} \\ \mathbf{u}_{B,1} \\ \mathbf{u}_{I,2} \\ \mathbf{u}_{B,2} \end{pmatrix} \begin{pmatrix} \mathbf{f}_{I,1} \\ \mathbf{f}_{B,1} \\ \mathbf{f}_{I,2} \\ \mathbf{f}_{B,2} \end{pmatrix} . \tag{3.64}$$

Here,

$$[A_{BB,l}]_{i,j} = a(\varphi_j|_{\Omega_l}, \varphi_i|_{\Omega_l})$$

for two generic nodes $i, j \in \sigma_{B,l}, l = 1, 2$, and

$$[A_{BB}^{(l,m)}]_{i,j} = a(\varphi_j|_{\Omega_\Gamma}, \varphi_i|_{\Omega_\Gamma})$$

for two generic nodes $i \in \sigma_{B,l}$ and $j \in \sigma_{B,m}, l, m = 1, 2$.

Formally proceeding as for the EO approach, we obtain

$$S_{VO}\mathbf{u}_B = \begin{pmatrix} S_1 + A_{BB,\Gamma}^{(1,1)} & A_{BB,\Gamma}^{(1,2)} \\ A_{BB,\Gamma}^{(2,1)} & S_2 + A_{BB,\Gamma}^{(2,2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_{B,1} \\ \mathbf{u}_{B,2} \end{pmatrix} = \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{pmatrix}, \tag{3.65}$$

with S_i and $\mathbf{g}_i$ defined by (3.62) and (3.63).

If M subdomains are considered, we have:

$$S_{VO}\mathbf{u}_B = \begin{pmatrix} S_1 + A_{BB,\Gamma}^{(1,1)} & A_{BE}^{(1,2)} & \dots & A_{BE}^{(1,M)} \\ A_{BE}^{(2,1)} & S_2 + A_{BB,\Gamma}^{(2,2)} & \dots & A_{BE}^{(2,M)} \\ \vdots & \vdots & \ddots & \vdots \\ A_{BE}^{(M,1)} & A_{BE}^{(M,2)} & \dots & S_M + A_{BB,\Gamma}^{(M,M)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_{B,1} \\ \mathbf{u}_{B,2} \\ \vdots \\ \mathbf{u}_{B,M} \end{pmatrix} = \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_M \end{pmatrix}, \tag{3.66}$$

Beware that some of the $A_{BB,\Gamma}^{(i,j)}$ blocks may be in fact empty. This happens when no element in the triangulation contains nodes belonging to both $\overline{\Omega_i}$ and $\overline{\Omega_j}$.

Note that the matrix S_{VO} can be formed operating algebraically on A, using the block form (3.48) or (3.49).

3.3.3 Solution of the Schur Complement System

Both linear systems (3.61) and (3.66) are usually solved by an iterative method. This is because S_i can be applied to a vector without its explicit formation, using three matrix-vector products and solving one linear system in $A_{II,i}$. This last step corresponds to the solution of an internal Dirichlet problem in Ω_i.

Once we have computed the border values $\mathbf{u}_B$, we may build the internal solutions $\mathbf{u}_{I,i}$, by solving the completely independent linear systems

$$A_{II,i}\mathbf{u}_{I,i} = \mathbf{f}_{I,i} - A_{IB,i}\mathbf{u}_B. \tag{3.67}$$

Remark 3.3.3. *Note that thanks to the splitting of S and $\mathbf{g}$, an iterative scheme for the solution of (3.66) can be made parallel. However, some communications among subdomains will be required, whereas (3.67) may be solved in a perfect parallel fashion.*

As regards EO decompositions, it can be shown that $\kappa(S_{EO}) \leq Ch^{-1}$ for a suitable constant C independent of h; see [QV99, pag. 51]. To obtain a bound for $\kappa(S_{VO})$, we rewrite equation (3.66) using a two-subdomain formulation. The subdomains are Ω_Γ and $\Omega_\star = \cup_{i=1}^{M}\Omega_i$. We have that

$$\begin{aligned} S_{VO} &= \begin{pmatrix} S_1 & 0 & \dots & 0 \\ 0 & S_2 & 0 & \vdots \\ \vdots & & \ddots & \\ 0 & 0 & \dots & S_M \end{pmatrix} + \begin{pmatrix} A_{BB,\Gamma}^{(1,1)} & A_{BB,\Gamma}^{(1,2)} & \dots & A_{BB,\Gamma}^{(1,M)} \\ A_{BB,\Gamma}^{(2,1)} & A_{BB,\Gamma}^{(2,2)} & \dots & A_{BB,\Gamma}^{(2,M)} \\ \vdots & & \ddots & \vdots \\ A_{BB,\Gamma}^{(M,1)} & & \dots & A_{BB,\Gamma}^{(M,M)} \end{pmatrix} \\ &= S_\star + A_{BB,\Gamma} \\ &= S_\star + S_\Gamma. \end{aligned} \tag{3.68}$$

Here, $S_\star = \text{blockdiag}\{S_i, i = 1, \dots, M\}$, and

$$[A_{BB,\Gamma}]_{k,j} = a(\varphi_j|_{\Omega_\Gamma}, \varphi_k|_{\Omega_\Gamma}),$$

for two generic nodes k and l of $\overline{\Omega_\Gamma} \setminus \partial\Omega$. In the two-subdomain formulation $\overline{\Omega} = \overline{\Omega_\star} \cup \overline{\Omega_\Gamma}$, $S_\star$ is the Schur complement obtained by eliminating the internal nodes of $\Omega_\star$, while the Schur complement of domain Ω_Γ, indicated by S_Γ, coincides with $A_{BB,\Gamma}$, since no internal nodes have been included in Ω_Γ.

Note that $S_\star$ has a (dense) diagonal block form, while $A_{BB,\Gamma}$ is sparse and, in general, with many zero-blocks. Since S_{VO} is symmetric, eigenvalues are given by the Rayleigh quotient

$$\lambda_{S_{VO}} = \frac{\langle S_{VO}\boldsymbol{\eta}, \boldsymbol{\eta}\rangle}{\langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle}, \quad \boldsymbol{\eta} \in \mathbb{R}^{n_B} \setminus \{\mathbf{0}\}. \tag{3.69}$$

Classical theory of finite elements gives the following bounds for $S_i, i = 1, \dots, M$ and $A_{BB,\Gamma}$ [QV94, QV99].

Proposition 3.3.1. *For each $i = 1, \dots, M$, there exist two constants $\tilde{C}_i$ and $\hat{C}_i$, independent of h, such that*

$$\tilde{C}_i\langle \boldsymbol{\eta}_i, \boldsymbol{\eta}_i\rangle \leq \langle S_i\boldsymbol{\eta}_i, \boldsymbol{\eta}_i\rangle \leq \hat{C}_i\langle \boldsymbol{\eta}_i, \boldsymbol{\eta}_i\rangle h^{-1} \quad \forall \boldsymbol{\eta}_i \in V_i,$$

being V_i the finite-dimensional space associated to the set of nodes $\sigma_{B,i}$.

Proposition 3.3.2. *There exist two constants $\tilde{C}_\Gamma$ and $\hat{C}_\Gamma$, independent of h, such that*

$$\tilde{C}_\Gamma\langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle \leq \langle A_{BB,\Gamma}\boldsymbol{\eta}, \boldsymbol{\eta}\rangle \leq \hat{C}_\Gamma\langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle h^{-2} \quad \forall \boldsymbol{\eta} \in V_B.$$

Lemma 3.3.2. *Consider a VO decomposition with $M+1$ subdomains, where the width of Ω_Γ is of 1 element. Then the following bound holds*

$$\kappa(S_{VO}) \leq h^{-1}\left(C_1 + C_2h^{-1}\right), \tag{3.70}$$

where the two constants C_1 and C_2 are both independent of h and M.

h	$\kappa(S)$	$\kappa(A)$	$\kappa(A_{BB}^{-1}S)$	$\kappa(A_{BB,\Gamma}^{-1}S)$
1/10	20.34	58.47	4.42	9.35
1/20	44.79	235.28	9.84	17.15
1/30	69.31	529.97	15.61	25.57
1/40	93.85	942.52	21.27	33.15
1/50	118.41	1472.39	26.95	41.83

Table 3.11: VO decomposition with a strip of width $\delta = 1$, and $M = 2$.

Proof. A lower bound for (3.69) can be obtained using propositions 3.3.1 and 3.3.2 as follows

$$\begin{aligned}
\langle (S_\star + A_{BB,\Gamma})\boldsymbol{\eta}, \boldsymbol{\eta}\rangle &= \langle S_\star \boldsymbol{\eta}, \boldsymbol{\eta}\rangle + \langle A_{BB,\Gamma}\boldsymbol{\eta}, \boldsymbol{\eta}\rangle \\
&= \sum_{i=1}^{M} \langle S_i \boldsymbol{\eta}_i, \boldsymbol{\eta}_i\rangle + \langle A_{BB,\Gamma}\boldsymbol{\eta}, \boldsymbol{\eta}\rangle \\
&\geq (\min_i \tilde{C}_i) \sum_{i=1}^{M} \langle \boldsymbol{\eta}_i, \boldsymbol{\eta}_i\rangle + \gamma \tilde{C}_\Gamma \langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle \\
&= \left(\min_i \tilde{C}_i + \tilde{C}_\Gamma\right) \langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle
\end{aligned}$$

for all $\boldsymbol{\eta} \in \mathbb{R}_{n_B} \setminus \mathbf{0}$.

For the upper bound, still using propositions 3.3.1 and 3.3.2, we obtain

$$\begin{aligned}
\langle (S_\star + A_{BB,\Gamma})\boldsymbol{\eta}, \boldsymbol{\eta}\rangle &\leq (\max_i \hat{C}_i) h^{-1} \langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle + \hat{C}_\Gamma h^{-2} \langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle \\
&= \left(\max_i \hat{C}_i h^{-1} + \hat{C}_\Gamma h^{-2}\right) \langle \boldsymbol{\eta}, \boldsymbol{\eta}\rangle.
\end{aligned}$$

The inequality (3.70) follows now from the definition of the spectral condition number. □

Although the condition number of A and S_{VO} share the same kind of upper bound with respect to h, it is found numerically that S is better conditioned than A; see table 3.11.

Neumann-Neumann Preconditioners

The original Neumann-Neumann (NN) preconditioner operator P_{NN}^{-1} was proposed by De Roeck and le Tallec [RT91] and is based on the assembly of the global SC from the local subdomain SC matrices. The idea is to precondition the sum of the local matrices with the sum of the inverses of these local matrices. The preconditioner reads

$$P_{NN}^{-1} = \sum_{i=1}^{M} \bar{R}_i D_i {S_i}^{-1} D_i \bar{R}_i. \tag{3.71}$$

The weight matrices D_i are an important component of the method allowing to preserve good conditioning even when the subdomains differ in certain aspects, for instance because of the different values of the coefficients of the differential operator.

Note that if all subdomains are similar (that is, all the S_i are spectrally equivalent), this yields an optimal preconditioner, since

$$\kappa\left(\sum_{i=1}^{M} \bar{R}_i^T S_i \bar{R}_i \sum_{i=1}^{M} \bar{R}_i {S_i}^{-1} \bar{R}_i\right) \approx \mathcal{O}(1).$$

At the differential level, the application of S_i^{-1} corresponds to solving local Neumann problems on the subdomains. This can be done without forming explicitly S_i, as stated by equation (3.56). If A_i is non-definite, exact application of formula (3.56), using for instance a singular value decomposition, may be computationally expensive. More favourable solutions are available: one can compute the LU decomposition of A_i and replace the zero-pivots by large numbers, or else to compute the inverse of $A_i + \varepsilon I$ for $\varepsilon > 0$ small enough. In the latter case, from Theorem 3.3.2 we know that the SC matrix of $A_i + \varepsilon I$ is "close enough" to the SC matrix of A.

The following estimation holds.

Theorem 3.3.3. *For the Neumann-Neumann preconditioner applied to the model problem 3.1, we have*

$$\kappa\left(P_{NN}^{-1} S\right) \leq \frac{1}{H^2}\left(1 + \log(H/h)\right)^2. \tag{3.72}$$

Proof. See [RT91]. □

The term $1/H^2$ cannot be avoided with this formulation since the NN algorithm fails to provide any means of global distribution of information beyond that of local exchange and, as expected, suffers from steep deterioration of convergence as the number of subdomains increases.

The convergence of the Neumann-Neumann algorithm for a small number of subdomains is quite good. As noted in [Tal94], the NN preconditioner becomes impractical when applied to problems with a number of subdomains larger than about 16. The cause of this deterioration is the lack of a global distribution of the information. Therefore, in order to overcome this drawback, Mandel [Man93] proposed the balancing Neumann-Neumann preconditioner (BNN) by adding to the NN preconditioner a simple coarse correction constructed by using piecewise constant coarse grid space. The corresponding coarse operator involves one unknown per subdomain in the scalar case. It can be proved that the resulting preconditioner satisfies

$$\kappa(P_{BNN}^{-1} A) \leq C\left(1 + \log\frac{H}{h}\right)^2. \tag{3.73}$$

We will not detail the algorithm, referring the reader to [Man93, DSW94] or [QV99, Section 3.2.2].

"Swiss Carpet" Preconditioners for VO Decompositions

Now we address the choice of the preconditioner for the VO decomposition. The results here presented are taken from [QSV03].

Some preconditioners may be derived from equation (3.68):

- $P_\star = S_\star$;
- $P_\Gamma = S_\Gamma$;
- $P_{NN} = (S_\star^{-1} + S_\Gamma^{-1})^{-1}$.

$P_\star$ and P_Γ are formally equivalent to Dirichlet-Neumann (DN) preconditioners [QV99, Section 3.2]. In the former, M Neumann problems are solved (one in every Ω_i). In the latter, one Neumann problem is solved on Ω_Γ; this problem is somehow unusual since all the nodes of Ω_Γ stand on the border of the domain Ω_Γ. P_{NN} is the 2-subdomain NN preconditioner. Note that other two preconditioners may be derived as well: $P_1 = A_{BB,\star}$ and $P_2 = A_{BB} = A_{BB,\star} + A_{BB,\Gamma}$.

Remark 3.3.4. *The DN and NN preconditioners are optimal for the two-subdomain case. However, the analysis requires that the subdomains have a given measure which is independent of h. This is not verified by preconditioners $P_\star$, P_Γ, and P_{NN}, since the width of Ω_Γ is h. Clearly, if the computational domain is refined so that δ is fixed (and therefore elements are added inside Ω_Γ), then $P_\star$, P_Γ, and P_{NN} yield optimal preconditioners.*

We now derive a convergence estimate for the swiss carpet preconditioner. For the sake of simplicity, the estimate of the condition number of the preconditioned Schur complement $S_\Gamma^{-1}S$ will be presented only in a case of simple geometry and the Laplace operator with homogeneous boundary conditions. However, by following similar arguments, the interested reader could easily extend the result to many more general situations (for instance, the result is true for any second order elliptic operator having no zero order terms and being associated to a symmetric and coercive bilinear form, or for the two-dimensional elasticity system; but also other boundary conditions and domains can be taken into account).

Let us consider the square $\Omega = (0,1) \times (0,1)$, and a family of uniform triangulations $\mathcal{T}_h$ of it. We assume that Ω is subdivided into the union of the disjoint squares Ω_j, $j = 1, \ldots, \sqrt{M}$, with side-length $H = \sqrt{M}^{-1}(1+h) - h$, and of the domain Ω_Γ, which is connected and consists of the union of $(\sqrt{M}-1)^2$ non-disjoint strips of length 1 and width h. Let us also set

$$\Omega_\star = \cup_{j=1}^{M} \Omega_j \quad , \quad \Omega_\Gamma = \Omega \setminus \overline{\Omega_\star} \quad , \quad \Gamma = \overline{\Omega_\star} \cap \overline{\Omega_\Gamma} \; . \tag{3.74}$$

This subdivision of the domain Ω will be called *the swiss carpet*.

$\eta_h \in \Lambda_h$ denote the trace space on Γ of the piecewise linear functions belonging to V_h. Then, $\forall \eta_h \in \Gamma_h$, $E_{h,\star}\eta_h$ the discrete harmonic extension of η_h in $\Omega_\star$, and $E_{h,\Gamma}\eta_h$ the discrete

harmonic extension of η_h in Ω_Γ. Since in Ω_Γ there are no internal nodes (the width of Ω_Γ is equal to h), the function $E_{h,\Gamma}\eta_h$ is simply the interpolant of η_h in $\overline{\Omega_\Gamma}$.

As it is well-known, the action of the Schur complements S and S_Γ can be expressed through the local bilinear forms in the following way (here $\boldsymbol{\eta}$ denotes the vector of the nodal values of η_h) (see [QV99, pag. 22]):

$$\langle S\boldsymbol{\eta},\boldsymbol{\eta}\rangle = \int_{\Omega_\star} |\nabla E_{h,\star}\eta_h|^2 + \int_{\Omega_\Gamma} |\nabla E_{h,\Gamma}\eta_h|^2 \quad , \quad \langle S_\Gamma\boldsymbol{\eta},\boldsymbol{\eta}\rangle = \int_{\Omega_\Gamma} |\nabla E_{h,\Gamma}\eta_h|^2 \ .$$

We can prove the following proposition.

Proposition 3.3.3. *The following estimate holds:*

$$\langle S_\Gamma\boldsymbol{\eta},\boldsymbol{\eta}\rangle \le \langle S\boldsymbol{\eta},\boldsymbol{\eta}\rangle \le C_1 \frac{H}{h} \langle S_\Gamma\boldsymbol{\eta},\boldsymbol{\eta}\rangle \quad \forall \boldsymbol{\eta} \in \mathbb{R}^{n_B},$$

where the constant C_1 does not depend on h and H.

Proof. The first inequality is evident. To prove the second one, we need to bound $\int_{\Omega_\star} |\nabla E_{h,\star}\eta_h|^2$, and this can be done by adding the contributions to the integral from each square Ω_j.

By means of a translation, instead of Ω_j we can consider the square Q_H centred in $(0,0)$ and with sides of length H. Then the change of variable $(\widehat{x},\widehat{y}) = T(x,y) = (x/H, y/H)$, we map Q_H over the unit square $\widehat{Q}$. The triangulation over Q_H is transformed into a triangulation over $\widehat{Q}$; the correspondent mesh size is given by h/H. Introducing, for each function g defined in Q_H, the function $\widehat{g} = g \circ T^{-1}$ defined in $\widehat{Q}$, and denoting by $\widehat{E}_{h/H}\widehat{\eta_h}$ the discrete harmonic extension of $\widehat{\eta_h}$ with respect to the triangulation in $\widehat{Q}$, it is easily verified that $\widehat{E}_{h/H}\widehat{\eta_h} = \widehat{E_{h,\star}\eta_h}$ and that

$$\int_{Q_H} |\nabla E_{h,\star}\eta_h|^2 = \int_{\widehat{Q}} |\widehat{\nabla}\widehat{E}_{h/H}\widehat{\eta_h}|^2. \tag{3.75}$$

The well-known uniform extension theorem (see, e.g., [XZ98], Sect. 4.2.2) gives that

$$\int_{\widehat{Q}} |\widehat{\nabla}\widehat{E}_{h/H}\widehat{\eta_h}|^2 \le C_0 |\widehat{\eta_h}|^2_{1/2,\partial\widehat{Q}} \ , \tag{3.76}$$

where the constant C_0 does not depend on h and H and $|\cdot|_{1/2,\partial\hat{Q}}$ denotes the seminorm of $H^{1/2}(\partial\widehat{Q})$.

Let us now observe that the set Ω_Γ is the union of $F_{j,\star} \cap \Omega$, where the "frames" $F_{j,\star}$ are the sets of width h around $\Omega_{j,\star}$(note that these sets are not disjointed, but any possible intersection concerns at most four of them). Moreover, by means of a translation, we can think that each set $F_{j,\star}$ is the "frame" $F_{H,h}$ of the square Q_H.

Let us consider now the set $\widehat{F}_{1,1/2}$, that is, the "frame" of width $1/2$ around $\widehat{Q}$. We want to construct a function in $\widehat{F}_{1,1/2}$ such that its value on the internal boundary of $\widehat{F}_{1,1/2}$ (that is, on $\partial\widehat{Q}$) is equal to $\widehat{\eta}_h$. The natural idea is to start from $E_{h,\Gamma}\eta_h$, that is, from the interpolant of η_h in $\overline{\Omega_\Gamma}$. For simplicity, we will write $\psi_h = E_{h,\Gamma}\eta_h$, and we recall that $\psi_{h|\Gamma} = \eta_h$.

Now we have to pass from the domain $F_{j,\star}$ (or, equivalently, from $F_{H,h}$) onto $\widehat{F}_{1,1/2}$. First of all, note that the transformation T maps $F_{H,h}$ over the set $\widehat{F}_{1,h/H}$. Moreover, having defined

$$t(\xi) := \begin{cases} \xi & \text{if } |\xi| \leq 1/2 \\ 1/2[1 + h^{-1}H(|\xi| - 1/2)]\xi/|\xi| & \text{if } 1/2 \leq |\xi| \leq 1/2 + h/H \end{cases} ,$$

we introduce the transformation

$$K(\widehat{x}, \widehat{y}) := (t(\widehat{x}), t(\widehat{y})) ,$$

that maps $\widehat{F}_{1,h/H}$ onto $\widehat{F}_{1,1/2}$, leaving fixed the internal boundary $\partial\widehat{Q}$. We are now in a position to set

$$\psi^\sharp := \psi_{h|F_{H,h}} \circ T^{-1} \circ K^{-1} , \ \psi^\sharp : \widehat{F}_{1,1/2} \to \mathbb{R} ,$$

and we easily verify that $\psi^\sharp_{|\partial\widehat{Q}} = \widehat{\eta}_h$.

We can thus use the trace theorem and obtain

$$|\widehat{\eta}_h|^2_{1/2,\partial\widehat{Q}} \leq C_1 ||\nabla\psi^\sharp||^2_{0,\widehat{F}_{1,1/2}} \tag{3.77}$$

(see [XZ98, Theorem 4.1]). We can estimate of the right hand side, taking into consideration the explicit expression of the transformations T and K:

$$||\nabla\psi^\sharp||^2_{0,\widehat{F}_{1,1/2}} \leq C_2 h^{-1} H \int\int_{F_{H,h}} |\nabla\psi_h|^2 . \tag{3.78}$$

Coming back to the domain Ω_j, from (3.75), (3.76), (3.77) and (3.78) we have thus obtained

$$\begin{aligned} \int_{\Omega_j} |\nabla E_{h,\star}\eta_h|^2 &\leq C_3 h^{-1} H \int\int_{F_{j,\star}} |\nabla E_{h,\Gamma}\eta_h|^2 \\ &= C_3 h^{-1} H \int_{F_{j,\star}\cap\Omega} |\nabla E_{h,\Gamma}\eta_h|^2 , \end{aligned}$$

having extended η_h by 0 on the boundary of $F_{j,\star}$ outside Ω (this happens for the squares Ω_j that have at least an edge on $\partial\Omega$).

Keeping in mind the geometry of Ω_Γ we easily obtain the estimate

$$4\int_{\Omega_\Gamma} |\nabla E_{h,\Gamma}\eta_h|^2 \geq \sum_j \int_{F_{j,\star}\cap\Omega} |\nabla E_{h,\Gamma}\eta_h|^2 .$$

The thesis now follows by summing up the contribution of all the domains $\Omega_{j,\star}$ and the frames $F_{j,\star}$. □

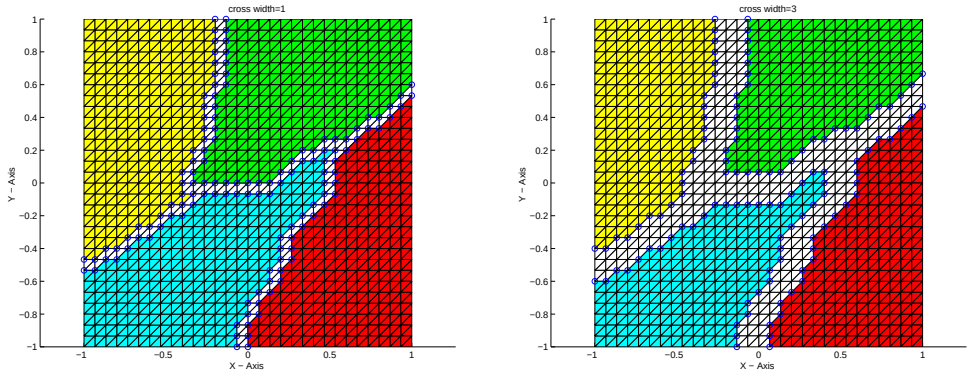


Figure 3.9: Variation of the cross width δ using an unstructured decomposition, with $M = 4$ and $\delta = 1$ (left) and $\delta = 3$ (right).

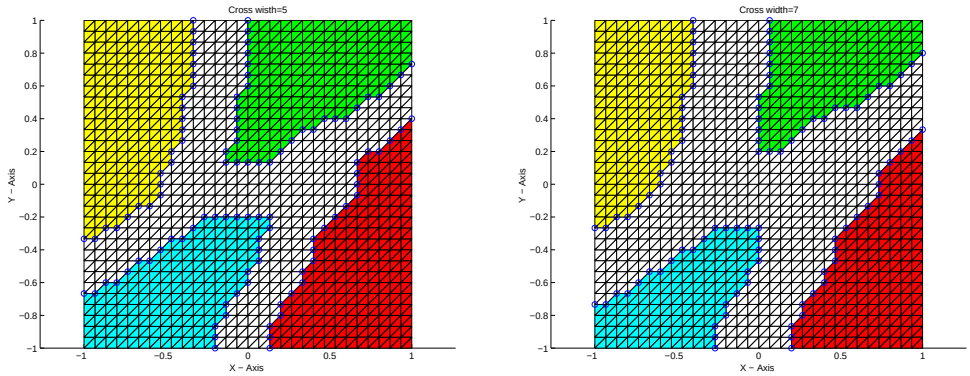


Figure 3.10: Variation of the cross width δ using an unstructured decomposition, with $M = 4$ and $\delta = 5$ (left) and $\delta = 7$ (right).

3.3.4 Generalised VO Decompositions

The VO decomposition presented in section 3.3.2 can be generalised as follows. Let us indicate with Δ_{ij} the minimum number of edges that one has to cross when moving from an arbitrary node belonging to Ω_i to a node of Ω_j, $i, j = 1, \ldots, M, i \neq j$. Now, we will indicate with

$$\delta = \min_{i,j=1,\ldots,M} \Delta_{ij}$$

the *width* of Ω_Γ. Clearly, in the VO decomposition described so far, we have $\delta = 1$. However, this value can be easily increased using the following algorithm.

Algorithm 3.3.1 (VO decomposition with $\delta > 1$).

a. For $k = 1, \ldots, N$ Do
b. For every element j of $\mathcal{T}_{h,i}$, $i = 1, \ldots, M$ Do:
c. If k is a border node, assign the element j to $\mathcal{T}_{h,\Gamma}$
d. End For
e. Update the set of internal and border nodes
f. End For

Algorithm 3.3.1 allows the construction of a cross whose width is $\delta = 1 + 2N$. Example of VO decomposition for $M = 4$ and $\delta = 1, 3, 5, 7$ are given in figures 3.9 and 3.10. As δ increases, Ω_Γ covers a wider part of Ω. However it is worthwhile to note that, as the numerical results reported in the following section show, we aim at a value of δ as close as possible to 1. Indeed, although higher values of δ have positive effect on the preconditioner we are about to propose, as we will see in section 3.3.6, the computational complexity of the associated problem on Ω_Γ can become of the same order as the solution of A.

3.3.5 Approximate SC Methods

The computational bottleneck of SC-based methods is due to the solution of the internal Dirichlet problems. Whether this step is replaced by an approximate solver, a preconditioner based on the SC can be derived.

Let $\hat{A}_{II}$ and $\hat{S}$ be two approximations of A_{II} and S, respectively; an approximate SC (ASC) preconditioner can be derived by setting

$$P_{ASC} = \begin{pmatrix} I_I & 0 \\ A_{BI}\hat{A}_{II}^{-1} & I_B \end{pmatrix} \begin{pmatrix} \hat{A}_{II} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I_I & \hat{A}_{II}^{-1}A_{BI} \\ 0 & I_B \end{pmatrix} \tag{3.79}$$

The following simple lemma holds.

Lemma 3.3.3. *Let C_1, C_2, C_3 and C_4 be four constants such that*

$$C_1\langle\hat{A}_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle \leq \langle A_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle \leq C_2\langle\hat{A}_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle \quad \forall\boldsymbol{\eta}_I\in\mathbb{R}^{n_I} \tag{3.80}$$

$$C_3\langle\hat{S}\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle \leq \langle S\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle \leq C_4\langle\hat{S}\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle \quad \forall\boldsymbol{\eta}_B\in\mathbb{R}^{n_B}. \tag{3.81}$$

Then, the preconditioner defined by (3.79) satisfies

$$\min\{1,C_1,\frac{1}{C_2},C_3\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle \leq \langle A\boldsymbol{\eta},\boldsymbol{\eta}\rangle \leq \max\{1,C_2,\frac{1}{C_1},C_4\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle \tag{3.82}$$

Proof. P_{ASC} can be written as

$$P_{ASC}=\begin{pmatrix}\hat{A}_{II} & A_{IB}\\ A_{BI} & A_{BI}\hat{A}_{II}A_{IB}+\hat{S}\end{pmatrix}$$

We have, with $\boldsymbol{\eta}=(\boldsymbol{\eta}_I,\boldsymbol{\eta}_B)$,

$$\begin{aligned}\langle A\boldsymbol{\eta},\boldsymbol{\eta}\rangle &= \langle A_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle + 2\langle A_{IB}\boldsymbol{\eta}_B,\boldsymbol{\eta}_I\rangle + \langle A_{II}^{-1}A_{IB}\boldsymbol{\eta}_B,A_{IB}\boldsymbol{\eta}\rangle + \langle S\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle\\ &\leq C_2\langle\hat{A}_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle + 2\langle A_{IB}\boldsymbol{\eta}_B,\boldsymbol{\eta}_I\rangle + \frac{1}{C_1}\langle\hat{A}_{II}^{-1}A_{IB}\boldsymbol{\eta}_B,A_{IB}\boldsymbol{\eta}\rangle + C_4\langle\hat{S}\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle\\ &\leq \max\{1,C_2,\frac{1}{C_1},C_4\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle.\end{aligned}$$

In an analogous way,

$$\begin{aligned}\langle A\boldsymbol{\eta},\boldsymbol{\eta}\rangle &\geq C_1\langle\hat{A}_{II}\boldsymbol{\eta}_I,\boldsymbol{\eta}_I\rangle + 2\langle A_{IB}\boldsymbol{\eta}_B,\boldsymbol{\eta}_I\rangle + \frac{1}{C_2}\langle\hat{A}_{II}^{-1}A_{IB}\boldsymbol{\eta}_B,A_{IB}\boldsymbol{\eta}\rangle + C_3\langle\hat{S}\boldsymbol{\eta}_B,\boldsymbol{\eta}_B\rangle\\ &\geq \min\{1,C_1,\frac{1}{C_2},C_3\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle.\end{aligned}$$

□

Once $\hat{A}_{II}$ is chosen, $\hat{S}$ can be defined as

$$\hat{S}=A_{BB}-A_{BI}\hat{A}_{II}^{-1}A_{IB}. \tag{3.83}$$

The following corollary holds.

Corollary 3.3.1. *Let C_1 and C_2 be defined by (3.80), and $\hat{S}$ by (3.83). Then,*

$$\min\{1,C_1,\frac{1}{C_2}\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle \leq \langle A\boldsymbol{\eta},\boldsymbol{\eta}\rangle \leq \max\{1,C_2,\frac{1}{C_1}\}\langle P_{ASC}\boldsymbol{\eta},\boldsymbol{\eta}\rangle. \tag{3.84}$$

Proof. The thesis follows immediately by noting that $C_3=\min\{1,\frac{1}{C_2}\}$ and $C_4=\max\{1,\frac{1}{C_1}\}$, and applying lemma 3.3.3. □

Equation (3.84) states that the spectral properties of the global preconditioner (3.79) depend on the quality of the local approximation (3.80). In particular, if $\tilde{A}_{II}$ is spectrally equivalent to A_{II}, then P_{ASC} provides an optimal preconditioner for A. Note however that the use of P_{ASC} requires the application of the exact inverse of P_S, i.e., a linear system with the matrix P_S has to be solved up to the machine precision.

Remark 3.3.5. *Each application of P_{ASC} requires two applications of $\tilde{A}_{II}$ and one of $\tilde{S}$. In principle, the same approximation $\hat{A}_{II}$ may be used also to build $\hat{S}$ as in (3.83). Note that P_{ASC} operates on the whole system while $\hat{S}$ on the interface variables only. Note however that P_{ASC} does not need to be explicitly built up.*

Remark 3.3.6. *ASC preconditioners have received a certain attention in literature. Axelsson and co-workers defined two nested grids, and then compute the SC matrix with respect to vertices of the coarse grid [AL97]. Other approaches are closer to the DD paradigm, see for example [Zha00].*

3.3.6 Numerical Results

In this section we present some numerical results to compare the "swiss carpet" preconditioner P_Γ with one of the state-of-art preconditioner for the EO SC system, the BNN preconditioner P_{BNN}, applied to problem (3.46) of section 3.2.7. The computational cartesian grid, whose elements have characteristic dimension h, has been partitioned using METIS. The SC system has been solved with the CG method up to a tolerance of 10^{-10}.

In table 3.12, `it_`Γ and `it_BNN` indicate the iterations to converge using P_Γ and P_{BNN}, respectively. `flops_`Γ and `flops_BNN` refer to the flops required to solve the problem, as indicated by the MATLAB command `flops`. To count the flops, we have not considered the operations required to explicitly form the SC matrix, and the coarse matrix of BNN. The Neumann problems have been solved using a LU decomposition. The operations needed to compute the LU factors have not been included in the flop count.

Few considerations are in order. BNN always outperforms P_Γ in terms of iterations to converge; however, it is worth to note that the preconditioning phase is much cheaper for P_Γ than for BNN, for what concerns both CPU-time and memory storage. In fact, to apply the BNN preconditioner we have to solve a Neumann problem on each subdomain. If a direct method is used, we have to compute the LU decomposition of the matrix A_i, while, in the case of an iterative method, we usually have to provide a (local) preconditioner. Instead, to apply P_Γ we have to solve just one Neumann problem in Ω_Γ. Therefore, at least for this test case, P_Γ reveals to be computationally cheaper than P_{BNN}.

Finally, in table 3.13 we report some results about the generalised VO decomposition of section 3.3.4. One can note that the gain in iteration number can be significant as δ gets large. When $1/h = 60$ and we make use of 16 connected components for $\Omega_\star$, we can observe a reduction of about 40% in the iterations to converge from $\delta = 1$ to $\delta = 4$.

$1/h$	M	$\kappa(S_{EO})$	it_BNN	flops_BNN	$\kappa(S_{VO})$	it_Γ	flops_Γ
1/10	4	19.1	10	1.15e+05	34.4	21	7.82e+04
1/10	8	27.5	11	1.86e+05	47.0	28	1.32e+05
1/10	16	38.8	14	4.05e+05	56.1	29	1.80e+05
1/20	4	34.1	9	4.55e+05	80.5	30	4.09e+05
1/20	8	52.7	13	9.20e+05	109.5	33	5.46e+05
1/20	16	91.28	18	1.80e+06	145.8	36	7.99e+05
1/20	32	111.8	16	2.94e+06	190.6	39	1.08e+06
1/30	4	59.6	12	1.91e+06	123.9	36	1.08e+06
1/30	8	85.82	15	2.62e+06	189.3	40	1.45e+06
1/30	16	123.5	18	4.24e+06	254.6	45	2.06e+06
1/30	32	185.9	19	7.11e+06	321.9	46	2.62e+06
1/40	4	71.9	10	3.65e+06	180.1	38	1.78e+06
1/40	8	128.2	16	5.90e+06	254.1	48	2.99e+06
1/40	16	168.2	19	8.31e+06	343.9	52	3.96e+06
1/40	32	230.1	20	1.33e+07	462.3	53	4.91e+06
1/40	64	343.8	22	2.66e+07	608.7	51	7.72e+06
1/40	128	463.8	21	5.72e+07	770.4	44	1.29e+07
1/60	4	125.1	12	1.65e+07	268.4	45	4.69e+06
1/60	8	187.7	16	1.82e+07	406.3	59	8.04e+06
1/60	16	255.2	20	2.31e+07	522.0	57	8.97e+06
1/60	32	357.8	22	3.37e+07	712.3	59	1.12e+07
1/60	64	539.4	22	5.61e+07	971.4	59	1.53e+07

Table 3.12: Comparison of P_{BNN} (EO decomposition) and P_Γ (VO decomposition). it_BNN and it_Γ indicate the iterations to converge of P_{BNN} and P_Γ, respectively, while flops_BNN and flops_Γ the MATLAB flops required by the solution of the problem using CG.

$N_x \times N_y$	M	P_{BNN}	$P_{\Gamma,\delta=1}$	$P_{\Gamma,\delta=2}$	$P_{\Gamma,\delta=3}$
20×20	4	9	30	20	17
30×30	4	12	36	23	20
60×60	4	11	45	-	-
20×20	8	-	33	23	-
30×30	8	13	40	26	23
60×60	8	16	59	35	31
30×30	16	18	45	28	-
60×60	16	21	57	39	33

Table 3.13: Iterations to converge using P_{BNN} and P_Γ for different values of δ.

3.3.7 Concluding Remarks about Schur Complement Preconditioners

We have introduced a new decomposition and a corresponding preconditioner. Instead of defining a coarse space, the main idea is to decompose the original domain in order to have one subdomain which is connected to all the others. This subdomain Ω_Γ is reduced to the minimum width of 1 element, and it allows the definition of global preconditioners. The preconditioner requires the solution of a Neumann problem in Ω_Γ. Consequently, the preconditioning step requires the solution of only one linear system on a subdomain of a particular shape. Numerical tests confirm the good properties of such a decomposition, and comparison with the balancing Neumann/Neumann preconditioner shows that, from the point of view of computational cost, this approach may be of interest.

The exact SC matrix is dense and expensive to compute. In some cases, even if good preconditioners are available for this matrix, the resulting method can be computationally expensive because of the time required to form (or to apply) the SC matrix. Therefore, we have also presented approximate SC (ASC) preconditioners for the global unreduced system.

4

Application to the Compressible Euler Equations

This chapter is devoted to the applications of preconditioners based on the DD approach to a classic aerodynamic problem, namely, the steady compressible Euler equations (CEE).

The space discretisation is based on multidimensional upwind residual distribution (MURD) schemes. These schemes are targeted to the solution of advection-dominated systems on unstructured grids. The MURD formalism is introduced in section 4.1 for a scalar equation, and in section 4.2 for a system of equations. The extension to the CEE is carried out in section 4.3, where the elliptic/hyperbolic splitting of the 3D CEE is presented.

This space discretisation results in a system of non-linear equations, solved using the Newton method. Consequently, a linear system with the Jacobian matrix has to be solved at each Newton step. This is done with a Krylov method with DD-based preconditioners, as described in section 4.4. A-posteriori mesh adaptation schemes are covered in section 4.5. Results about space discretisation and mesh adaptation are reported in section 4.6. Finally, concluding remarks are given in section 4.7.

4.1 Fluctuation Splitting Formalism

In this section we consider the hyperbolic multidimensional scalar conservation law

$$\nabla \cdot \mathbf{f}(u) = 0, \tag{4.1}$$

together with its quasi-linear form

$$\boldsymbol{\lambda} \cdot \nabla u = 0. \tag{4.2}$$

Here, $d = 2, 3$ is the dimension of the problem, $\nabla = \sum_{i=1}^{d} \partial/\partial x_i$, $u = u(x_1, \dots, x_d)$ the conserved scalar variable, $\mathbf{f}(u) = [f_1(u), \dots, f_d(u)]$ the flux function, that can be linear or non-linear depending on the considered equations, and $\boldsymbol{\lambda} = d\mathbf{f}/du$. Suitable boundary conditions need to be specified in order to close the differential problems (4.1) and (4.2).

We now focus on the quasi-linear form (4.2). First, we introduce a conforming triangulation $\mathcal{T}_h(\Omega)$ of the computational domain, composed by simplices (triangles in 2D and tetrahedra in 3D). Then, we look for the numerical solution u_h in a piecewise-linear FE space V_h, which will be chosen in order to fulfill the specified boundary conditions. If $n = \dim(V_h)$, we have that

$$u_h(\mathbf{x}) = \sum_{i=1}^{n} u_{h,i} \varphi_i(\mathbf{x}), \tag{4.3}$$

where $u_{h,i}$ is the value of u_h at node i and the trial (or shape) function $\varphi_i(\mathbf{x})$ is the Lagrange basis function associated to node i.

Let $\psi_i(\mathbf{x})$ be the test function (to be properly chosen) associated to node i. We will choose it as a function of $\boldsymbol{\lambda}$, as will explain in section 4.1.1. In particular, in the non-linear case when $\boldsymbol{\lambda} = \boldsymbol{\lambda}(u)$, ψ_i will depend on $\boldsymbol{\lambda}(u_h)$. The Petrov-Galerkin discretisation of (4.2) reads

$$\sum_{T} \int_{T} \psi_i \boldsymbol{\lambda} \cdot \nabla u_h d\Omega = 0, \quad i = 1, \dots, n. \tag{4.4}$$

Assume that $\boldsymbol{\lambda}$ is constant over a generic element T. As a consequence, the term $\boldsymbol{\lambda} \cdot \nabla u_h$ is constant over T. Now, let us denote

$$\phi^T(u_h) = \int_{T} \boldsymbol{\lambda} \cdot \nabla u_h d\Omega \tag{4.5}$$

the *residual* or *fluctuation* over T, and by

$$\beta_i^T = \frac{1}{|T|} \int_{T} \psi_i d\Omega, \tag{4.6}$$

the *distribution coefficient* of element T associated to node i. In (4.6), $|T|$ is the area (or volume) of T. Note that β_i^T depends on u_h through $\boldsymbol{\lambda}(u_h)$ in the non-linear case.

Using equation (4.5), we can rewrite (4.4) as

$$\begin{aligned} \sum_{T|i\in T} \int_{T} \psi_i \boldsymbol{\lambda} \cdot \nabla u_h d\Omega &= \sum_{T|i\in T} \int_{T} \psi_i d\Omega \frac{\phi^T(u_h)}{|T|} \\ \text{[by (4.6)]} \qquad &= \sum_{T|i\in T} \beta_i^T \phi^T(u_h). \end{aligned}$$

Therefore,

$$R_i(u_h) = 0, \quad i = 1, \dots, n, \tag{4.7}$$

where $R_i(u_h) = \sum_{T|i\in T} \beta_i^T \phi^T(u_h)$ is the called the *nodal residual*. Equation (4.7) expresses that the nodal residual, which is the sum of all the contributions from neighbouring elements of node i, vanishes.

Finally, let us indicate

$$\phi_i^T(u_h) = \beta_i^T \phi^T(u_h) \tag{4.8}$$

the part of the elemental residual distributed to node i. For consistency reasons, we require that

$$\sum_{i\in T} \beta_i^T = 1.$$

This yields

$$\phi^T(u_h) = \phi_1^T(u_h) + \ldots + \phi_{d+1}^T(u_h).$$

The distribution coefficients can also be defined as

$$\beta_i^T = \phi_i^T / \phi^T$$

if $\phi^T \neq 0$.

Note that the test functions ψ_i appears in (4.7) only through their averages $\{\beta_i^T\}$ over the elements T. Therefore, what counts is the choice of the $\{\beta_i^T\}$, which will be addressed in section 4.1.1. Once the $\{\beta_i^T\}$ have been specified, the solution of (4.7), to be completed with the appropriate boundary conditions, gives the numerical solution of (4.2).

Equation (4.6) is not sufficient to uniquely associate a test function ψ_i to a residual distribution coefficient β_i^T unless the functional form of ψ_i is prescribed and depends on a single parameter. However, we could look for a piecewise-linear test function over T, which can be represented as

$$\psi_i|_T = \varphi_i|_T + \alpha_i^T \gamma^T, \tag{4.9}$$

where α_i^T is the upwind bias coefficient and γ^T is a piece-wise constant function equal to 1 on element T and 0 elsewhere. Note that ψ_i turns out to be a piecewise-linear discountinuous function on $\overline{\Omega}$. The equation (4.6) yields

$$\beta_i^T = \frac{1}{|T|} \int_T (\varphi_i + \alpha_i^T \gamma^T) d\Omega,$$

giving the following relation between the coefficients $\{\alpha_i^T\}$ and $\{\beta_i^T\}$

$$\alpha_i^T = \beta_i^T - \frac{1}{d+1},$$

where $d+1$ is the number of vertices of the simplex. Hence

$$\psi_i|_T = \varphi_i|_T + \left(\beta_i^T - \frac{1}{d+1}\right)\gamma^T.$$

Clearly, should the residual be equidistributed over all vertices, $\beta_i^T = \frac{1}{d+1}$, one has $\psi_i = \varphi_i$ (i.e., test and trial functions do coincide), thus the residual distribution scheme reduces to a pure Galerkin FE method.

The Petrov-Galerkin interpretation of MURD schemes allows a straightforward extension of the discretisation to the advection-diffusion equation

$$\boldsymbol{\lambda} \cdot \nabla u + \nabla \cdot (\nu \nabla u) = 0, \quad \nu > 0, \tag{4.10}$$

completed by appropriate boundary conditions. Multiplying formally the diffusion term of equation (4.10) by the test function ψ_i, integrating by parts and using the Gauss theorem, we obtain

$$\sum_T \int_T \sum_{j=1}^{d} \frac{\partial \psi_i}{\partial x_j} \nu \frac{\partial u_h}{\partial x_j} d\Omega - \oint_{\partial\Omega} \psi_i \nu \frac{\partial u_h}{\partial \mathbf{n}_{ext}} d\Gamma.$$

In the case the test functions satisfy (4.9),

$$\nabla \psi_i|_T = \nabla \varphi_i|_T,$$

and therefore this reduces to a standard Galerkin discretisation of the diffusion term.

The main computational kernel of MURD schemes is the computation of $\phi^T(u_h)$. This can be done in an efficient way as follows. For each node ℓ (in local element numbering) of T, we can define the inflow parameter

$$k_\ell = \frac{1}{d}\boldsymbol{\lambda} \cdot \mathbf{n}_\ell, \tag{4.11}$$

$\mathbf{n}_\ell$ being the vector normal to the face (or the edge if $d = 2$) opposite to node ℓ, scaled by its areas (or length if $d = 2$) opposed to node ℓ; see figure 4.1. In 2D, these inward scaled normals are given by

$$\begin{aligned} \mathbf{n}_1 &= [x_{2,2} - x_{3,2}, -x_{2,1} + x_{3,1}] \\ \mathbf{n}_2 &= [x_{3,2} - x_{1,2}, -x_{3,1} + x_{1,1}] \\ \mathbf{n}_3 &= [x_{1,2} - x_{2,2}, -x_{1,1} + x_{2,1}], \end{aligned} \tag{4.12}$$

while in 3D one has

$$\begin{aligned} \mathbf{n}_1 &= \tfrac{1}{2}(\mathbf{x}_4 - \mathbf{x}_2) \times (\mathbf{x}_3 - \mathbf{x}_2) \\ \mathbf{n}_2 &= \tfrac{1}{2}(\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_1) \\ \mathbf{n}_3 &= \tfrac{1}{2}(\mathbf{x}_4 - \mathbf{x}_1) \times (\mathbf{x}_2 - \mathbf{x}_1) \\ \mathbf{n}_4 &= \tfrac{1}{2}(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1) \end{aligned} \tag{4.13}$$

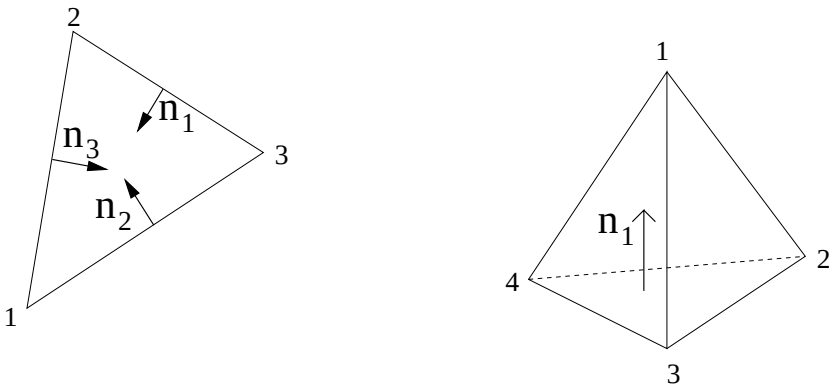


Figure 4.1: Tetrahedron with scaled inward vector normal $\mathbf{n}_i$.

where $\mathbf{a}\times\mathbf{b}$ represents the vector product between $\mathbf{a}$ and $\mathbf{b}$, and $\mathbf{x}_\ell = [x_{\ell,1}, x_{\ell,2}]$ is the position of node ℓ.

Since for any T

$$\oint_{\partial T} \mathbf{n} d\Gamma = 0,$$

where $\mathbf{n}$ is the non-scaled normal unit vector, the scaled normals $\mathbf{n}_\ell$ of a given element T satisfy the following identity:

$$\sum_{\ell=1}^{d+1} \mathbf{n}_\ell = \mathbf{0},$$

and, as a consequence,

$$\sum_{\ell=1}^{d+1} k_\ell = 0. \tag{4.14}$$

The normals $\mathbf{n}_\ell$ can be used to compute the (constant) gradient of the linear shape functions φ_ℓ over T as

$$\nabla\varphi_\ell|_T = \frac{1}{d}\frac{1}{|T|}\mathbf{n}_\ell. \tag{4.15}$$

Equations (4.3), restricted to T, and (4.15) give

$$\nabla u_h|_T = \sum_{\ell=1}^{d+1} u_{h,\ell}\nabla\varphi_\ell|_T = \frac{1}{|T|}\sum_{\ell=1}^{d+1} u_{h,\ell}\mathbf{n}_\ell. \tag{4.16}$$

Using equation (4.16) and the property that $\boldsymbol{\lambda}$ is constant over T, we obtain

$$\begin{aligned}\phi^T(u_h) &= \int_T \boldsymbol{\lambda}\cdot\nabla u_h|_T d\Omega \\ &= |T|\,\boldsymbol{\lambda}\cdot\nabla u_h|_T \\ &= |T|\,\boldsymbol{\lambda}\cdot\sum_{\ell=1}^{d+1} u_{h,\ell}\mathbf{n}_\ell \\ &= \sum_{\ell=1}^{d+1} k_\ell u_{h,\ell}.\end{aligned}$$

Therefore, the computation of the local residual $\phi^T(u_h)$ reduces to $d+1$ multiplications and additions, plus the operations required to compute the scaled normals $\mathbf{n}_\ell$ and the coefficients k_ℓ.

4.1.1 Properties of Distribution Schemes

The only remaining ingredient to be specified for a precise definition of the MURD scheme in equation (4.7) is a formula to calculate the coefficients β_i^T. Different schemes, corresponding to different ways of computing the distribution coefficients, have been designed in order to satisfy several kind of properties. The first, important property to be satisfied is the following one.

Property 4.1.1 (multidimensional upwind). *Let us define downwind vertex a vertex opposite to an inflow face, so that the corresponding coefficient given by (4.11) is positive. Conversely, an upwind vertex is a vertex for which $k_i \leq 0$. A scheme is called as multidimensional upwind scheme if*

$$(\mathcal{MU}) \quad \beta_i^T = 0 \text{ if } k_i \leq 0, \tag{4.17}$$

that is no residual is distributed to upwind nodes. This property is called upwinding because it is the exact extension of the upwinding notion for the resolution of the one dimensional convection equation.

A graphical representation of upwind and downwind nodes for a 2D case is reported in figure 4.2. The definition of β_i^T for the one-inflow case, depicted on the left of figure 4.2, is straightforward: the whole fluctuation is distributed to the downstream node, and the distribution is said to be *one target*. The two-inflow case, depicted on the right, on the other hand is more difficult to treat, since the fluctuation must be split between the two downstream nodes. In this case, the distribution is called *two-target*. Analogous definitions hold for the 3D case, represented in figure 4.3.

Property 4.1.2 (positivity). *A fluctuation splitting scheme is said to be positive if the scheme does not create local extrema or, if we write the contribution of element T to node i*

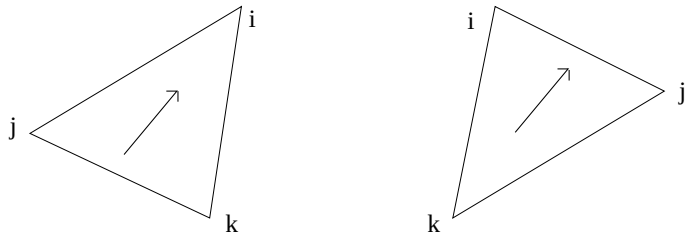


Figure 4.2: Upwind distribution of the fluctuation in 2D. On the left, $\beta_j^T = \beta_k^T = 0$. On the right, $\beta_k^T = 0$.

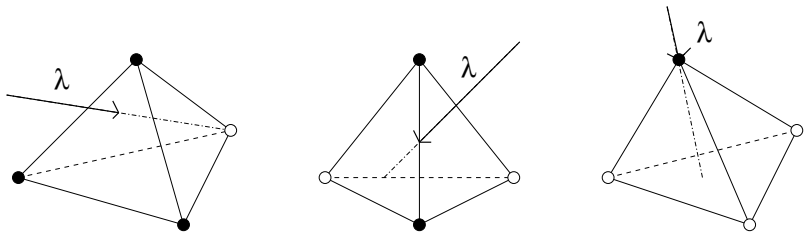


Figure 4.3: Upwind distribution of the fluctuation in 3D. From left to right: one-inflow face, two-inflow faces, three-inflow faces. '•' indicates downwind nodes, while '○' indicates upwind nodes.

as $\phi_i^T = \sum_j c_{i,j} u_{h,j}$, *we impose*

$$c_{i,j} \leq 0, \quad \forall i \neq j.$$

Note that $\sum_j c_{i,j} = 1$ *because of (4.14).*

A more restrictive property, called *local positivity*, is obtained by considering the contribution from each element, taken separately, and requiring the scheme to be positive for this contribution. Note that if a scheme is locally positive, i.e. considering only the contribution of a given element, it will also be positive for the global scheme, obtained by taking into account all the elements T surrounding the node i. Usually, local positivity is simpler to impose. If a scheme is locally positive, then a maximum principle will hold for the discrete steady-state solution; see [PD97, Section 2.2.2].

Property 4.1.3 (continuity). *The distribution coefficients* β_i^T *must be continuous (*$\mathcal{C}$*) functions of both the advection and the solution-gradient directions.*

When framed in a time-dependent setting, if the numerical scheme originates a solution with large jumps, the convergence toward steady-state may not be correct. These schemes can cause the solution to be struck into a limit circle. This is an instationary behaviour where the solution oscillates around the steady-state solution. Of course, this is an undesirable property

of a scheme and should be avoided. The continuity of the scheme is required to obtain a smooth convergence to the steady-state solution.

Property 4.1.4 (linearity preservation). *A scheme is said to have the linearity preservation property if*

$$(\mathcal{LP}) \quad \beta_i^T \boldsymbol{\phi}^T \to 0 \text{ when } \boldsymbol{\phi}^T \to 0.$$

Therefore the distribution functions β_i have to remain bounded.

($\mathcal{LP}$) expresses the ability of the numerical scheme to reproduce linear solutions of equation (4.2) exactly. Since we are using piece-wise linear FE functions, a linear solution can be exactly calculated. When this property is satisfied the scheme is said to be linearity preserving.

It is well-known that in one dimension, linear monotone schemes are at most first-order. This theorem, known as Godunov's theorem, has an equivalent formulation for multidimensional schemes on unstructured meshes: *linear schemes cannot be both positive and linearity preserving*; see [Str94]. As a consequence of this theorem, to combine both positivity and linearity preservation, it is necessary to introduce some non-linearity into the scheme.

Remark 4.1.1 (second order accuracy). *A more general way of getting a second order accuracy is to require that the residual $\phi_i^T(u_h)$ satisfies the property $\phi_i^T(u_h) = \mathcal{O}(h^3)$ for an exact solution assumed to be regular and u_h its piecewise linear interpolation on the mesh [Abg01]. Indeed, because of the accuracy property of the piece-wise linear interpolation on a regular grid, it can be proved that the total residual evaluate for u_h satisfies $\phi^T(u_h) = \mathcal{O}(h^3)$, so that the boundedness of $\beta_i^T = \frac{\phi_i^T}{\phi^T}$ implies $\phi_i^T(u_h) = \mathcal{O}(h^3)$.*

We will not cover the different scalar distribution schemes in details. The expression of some of them can be found in table 4.1. The N-scheme is a linear monotone first-order upwind scheme, and it is an optimal positive linear scheme in the sense that it has the lowest cross-diffusion of its class. The LDA is a second-order linear upwind scheme, and the PSI scheme is a non-linear second-order and monotone upwind scheme, obtained by applying a limiter to the distribution coefficients of the N-scheme. It is possible to show that the PSI scheme is related to the MinMod limiter, which allows to write the scheme in its simple form [PD97, Section 4.2]. Note that the limiting is applied on the same compact stencil as the original linear positive scheme, i.e., (almost) second-order accuracy is achieved without enlarging the discretisation stencil. This turns out to have very favourable consequences for the development of implicit time-step schemes.

Remark 4.1.2. *The first-order N-scheme cannot be obtained from a Petrov-Galerkin approach since the distribution coefficients and hence the corresponding test functions are unbounded for vanishing element residual, due to the fact that the N-scheme is not ($\mathcal{LP}$).*

The reader is referred to the specialised literature for more details on these schemes. See, for instance, [DSA00] and the references therein, or the book [DK96].

Scheme	β_i^T	type	$\mathcal{U}$	$\mathcal{P}$	$\mathcal{LP}$
N	$k_i^+(\phi^T\sum_j k_j^-)^{-1}\sum_j k_j^-(u_{h,i}-u_{h,j})$	linear	$\surd$	$\surd$	
LDA	$\sum_j(k_j^+)^{-1}k_i^+$	linear	$\surd$		$\surd$
LW	$\frac{1}{d+1}+\frac{k_i}{\sum_j\lvert k_j\rvert}$	linear			$\surd$
SUPG	$\frac{1}{d+1}+\tau\frac{k_i}{\lvert T\rvert},\quad \tau=\frac{1}{2}\frac{h}{\lVert\lambda\rVert}$	linear			$\surd$
PSI	$\max(0,\beta_i^N)/\sum_i\max(0,\beta_i^N)$	non-linear	$\surd$	$\surd$	$\surd$

Table 4.1: MURD schemes for a scalar equation. N is the Narrow-scheme, LDA the Low Diffusion A scheme, LW the Lax-Wendroff scheme, SUPG the Streamline Upwind Petrov-Galerkin scheme. $k_i^+=\max(0,k_i)$, $k_i^-=\min(0,k_i)$. β_i^N refers to the distribution coefficient of the N-scheme. Formulae valid both in 2D and 3D.

Remark 4.1.3. *MURD schemes have originally been developed for steady-state conservation laws. More recently, extensions to unsteady problems have been proposed, based on a space-time approach, see [CRDP01]. The idea is to consider the space-time operator* $\Box=[\frac{\partial}{\partial t},\nabla]$, *and then apply the MURD scheme to solve the "unsteady" equation*

$$\frac{\partial u}{\partial \tau}+\Box\cdot[u,f(u)]=0$$

on triangular element in space-time, whereby no distinction is made between the temporal and spatial components of the flux at the level of the scheme. Due to the upwind property of MURD schemes, a natural decomposition of the temporal slabs can be achieved for a well-defined construction of the space-time mesh, so that effectively a time-marching scheme is obtained; see [HDR01].

Remark 4.1.4. *MURD schemes are a multidimensional generalisation of Roe's schemes, and they have therefore in common some failures with Roe's schemes. Because shocks can be captured over a single element (or cell), the numerical scheme can produce unphysical expansion shocks. Also, on very regular or sufficiently refined grids, carbuncle phenomena may occur when trying to compute strong bow shocks. Therefore, entropy fix and carbuncle cure have been developed, see [SD02].*

In the case of advection vectors $\boldsymbol{\lambda}$ independent on u, equation (4.7) reduces to a linear system of type (2.1), where the element (i,j) of the matrix can be computed as

$$A_{i,j}=\sum_{T|(i,j)\in T}\beta_i^T k_j.$$

Table 4.2 reports the expression of $A_{i,j}$ for the linear schemes of table 4.1. Note that appropriate boundary conditions still have to be imposed in order to obtain a solution of the resulting linear system.

scheme	$A_{i,j}^{(T)}$
N	$\delta_{i,j} k_i^+ - \frac{k_i^+ k_j^-}{\sum_{m=1}^{d+1} k_m^-}$
LDA	$\frac{k_i^+ k_j}{\sum_{m=1}^{d+1} k_m^+}$
LW	$\left(\frac{1}{d+1} + \frac{k_i}{\sum_{m=1}^{d+1} \lvert k_m \rvert}\right) k_j$
SUPG	$\left(\frac{1}{d+1} + \frac{\tau k_i}{\lvert T \rvert}\right) k_j$

Table 4.2: Computation of the element (i, j) of the linear system matrix corresponding to the contribution of element T associated to equation (4.7). $\delta_{i,j}$ is the Kronecker symbol.

On the other hand, if non-linear MURD schemes are in order, or $\boldsymbol{\lambda}$ is a function of u, (4.7) represents a systems of non-linear equations. In this case, the numerical solution is usually found by considering the steady-state solution of the time-dependent equation

$$\int_\Omega \frac{\partial u_h}{\partial t} d\Omega = -\int_\Omega \boldsymbol{\lambda} \cdot \nabla u_h d\Omega. \tag{4.18}$$

Because time accuracy is not an issue, the time derivative is discretized using a first-order approximation in time:

$$\sum_T \int_T \frac{\partial u_h}{\partial t} d\Omega \approx \sum_T \frac{|T|}{d+1} \left[\sum_{i \in T} \frac{u_{h,i}^k - u_{h,i}^{k-1}}{\delta_k}\right] \approx S \frac{\mathbf{u}_h^k - \mathbf{u}_h^{k-1}}{\delta_k}. \tag{4.19}$$

Here, $S = \text{diag}\{S_i, i = 1, \ldots, n\}$, with

$$S_i = \sum_{T | i \in T} \frac{|T|}{d+1},$$

which is no more than the volume of the dual cell of node i (see figure 4.4), δ_k is the time-step parameter at time step k, and $\mathbf{u}_h^k$ and $\mathbf{u}_h^{k-1}$ are vectors containing the nodal values of the FE solution u_h at time step k and $k-1$, respectively.

Equation (4.19) reads, at node i,

$$\frac{S_i}{\delta_k}\left(u_{h,i}^k - u_{h,i}^{k-1}\right) = -R_i(u_h^{k-1})$$

if an explicit time-marching scheme is used, or

$$\frac{S_i}{\delta_k}\left(u_{h,i}^k - u_{h,i}^{k-1}\right) = -R_i(u_h^k) \tag{4.20}$$

if an implicit time-marching scheme is employed. Equation (4.20) is still non-linear; although, the Newton method applied to (4.20) generally results in a more robust scheme, with respect to the direct application to (4.7).

The process of adding a (pseudo-)time derivative to equation (4.2) is usually called pseudo-transient continuation. It will be covered in details in section 4.4.

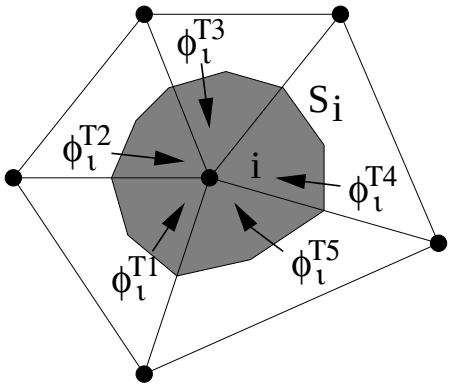


Figure 4.4: Node i and dual cell S_i, with surrounding elements and residuals ϕ_i^T.

Remark 4.1.5. *Note that by restricting the distribution schemes to the nodes of the element itself, the stencil of the scheme remains* compact*, at most a subset of the support of the linear shape functions φ_i^T, as indicated in figure 4.4 for a 2D case. As will be described in section 4.1.1, the coefficients β_i^T can be specified to satisfy certain properties of monotonicity and accuracy in the solution, while remaining the same compact stencil. This is one the advantages of the method compared to high-order FV schemes on unstructured meshes, for which the stencil typically includes more rows of cells around each node.*

As an example, we consider the solution of equation

$$\frac{\partial u}{\partial x_1} + 2\frac{\partial u}{\partial x_2} = 0, \tag{4.21}$$

in $\Omega = (0,1) \times (0,1)$ with the boundary conditions

$$u(y,0) = 3, \quad u(x,0) = 5, \quad u(0,0) = 4.$$

This equation represents a discontinuity in the origin of the coordinate system. The outlet boundaries are not submitted to any boundary conditions because the information is leaving the domain through these boundaries. Results using some of the schemes reported in table 4.1 are reported in figure 4.5. Note that the N-scheme is more diffusive than other schemes, which, for this test case, perform quite similarly. Finally, figure 4.6 reports the eigenvalues distribution of the linear system matrix for the LDA, LW and N-scheme. The SUPG scheme is also reported.

4.1.2 Extension to Non-linear Systems

MURD schemes presented so far requires that $\boldsymbol{\lambda}$ is constant over T. The general case of $\boldsymbol{\lambda}$ varying as a function of u can be taken into account by using a *conservative linearisation.*

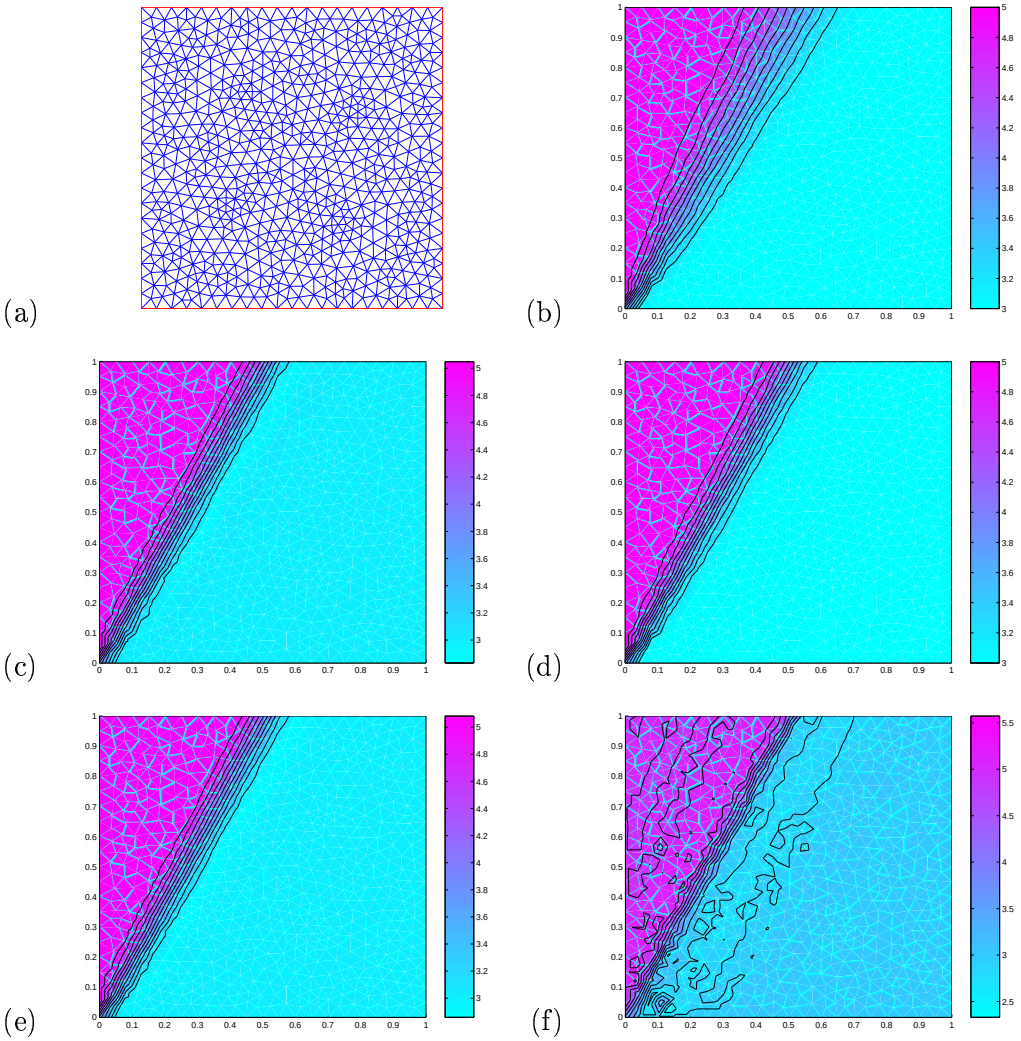


Figure 4.5: (a) Grid used for the solution of the linear problem (4.21). The grid has 724 nodes and 1366 elements. Schemes used: (b) N-scheme ($\max(u_h) = 5, \min(u_h) = 3$), (c) LDA-scheme ($\max(u_h) = 5.04, \min(u_h) = 2.83$), (d) PSI-scheme ($\max(u_h) = 5, \min(u_h) = 3$), (e) LW-scheme ($\max(u_h) = 5.08, \min(u_h) = 2.85$), (f) SUPG-scheme ($\max(u_h) = 5.56, \min(u_h) = 2.35$), using $\tau = \frac{1}{2}\frac{h}{\|\lambda\|}$ and $h = \max_\ell(\mathbf{n}_{\ell,1}, \mathbf{n}_{\ell,2})$.

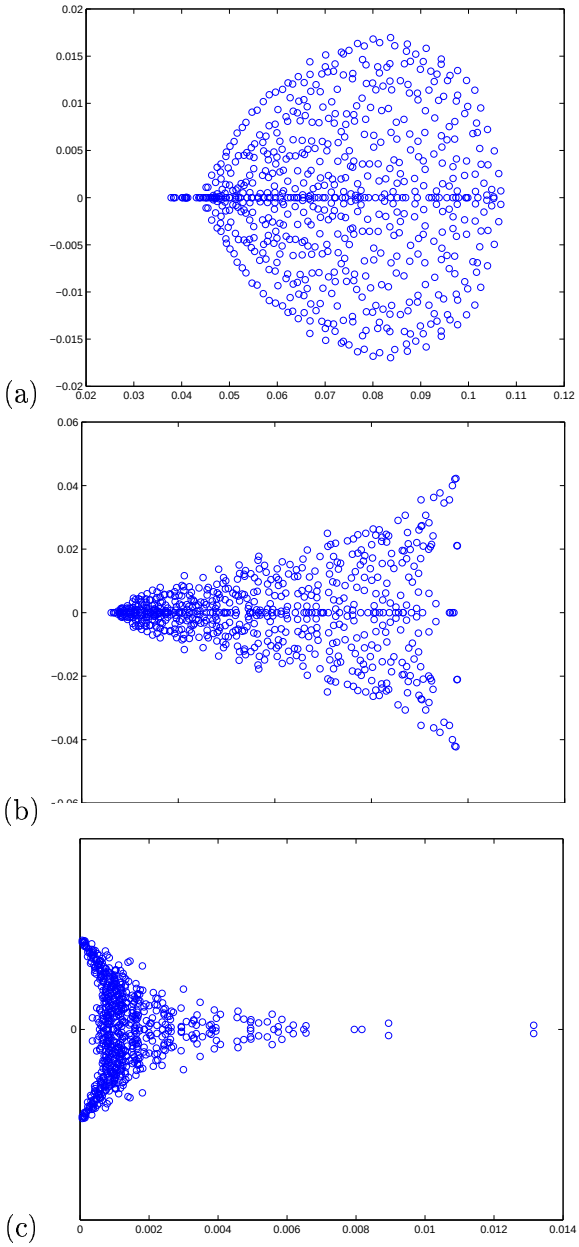


Figure 4.6: Distribution in the complex plane of the eigenvalues of the linear system matrix obtained from equation (4.21), using: (a) LDA-scheme, (b) LW-scheme, (c) SUPG-scheme.

This consists in finding on each T an average state $\bar{\boldsymbol{\lambda}}^T$ such that, for the integral form of the residual, one has

$$\int_T \nabla \cdot \mathbf{f} d\Omega = \oint_{\partial T} \mathbf{f} \cdot \mathbf{n}_{ext} d\Gamma = |T| \bar{\boldsymbol{\lambda}}^T \cdot \nabla u_h = \phi_i^T(u_h), \tag{4.22}$$

with ∇u_h constant over T because of (4.3). In this way, the non-linear system of conservation laws $\nabla \cdot \mathbf{f} = 0$ is locally approximated by the linear system $|T| \bar{\boldsymbol{\lambda}}^T \cdot \nabla u_h = 0$. Let us show that, with such a linearisation, conservation is indeed satisfied. Summing up over the elements of Ω and using (4.22), we obtain

$$\int_\Omega \nabla \cdot \mathbf{f} d\Omega = \sum_T \phi^T(u_h) = \sum_T |T| \bar{\boldsymbol{\lambda}}^T \cdot u_h = \sum_T \oint \mathbf{f} \cdot \mathbf{n}_{ext} d\Gamma = \oint_\Omega \mathbf{f} \cdot \mathbf{n}_{ext} d\Gamma.$$

In general, it is quite cumbersome to find an average state $\bar{\boldsymbol{\lambda}}^T$ that obeys (4.22). However, if $\mathbf{f}$ is a quadratic function of u_h a very simple analytical expression can be obtained. This is a severe restriction, but it turns out that the CEE belong to this class of equations having quadratic flux-functions.

In this case, the element residual $\phi^T(u_h)$ can be rewritten as

$$\begin{aligned} \phi^T(u_h) = \int_T \nabla \cdot \mathbf{f} d\Omega &= \int_T \sum_{i=1}^{d} \lambda_i \frac{\partial u_h}{\partial x_i} d\Omega \\ &= \sum_i \left(\int_T \bar{\lambda}_i d\Omega \right) \frac{\partial u_h}{\partial x_i} \\ &= \frac{|T|}{d+1} \sum_{i=1}^{d} \left(\sum_{\ell=1}^{d+1} \lambda_i(\mathbf{x}_\ell) \right) \frac{\partial u_h}{\partial x_i}. \end{aligned}$$

Here, use is made of

- u_h varies linearly with $\mathbf{x}$;
- λ_i is a linear function of u_h;
- the integral of a linear function over a simplex is the arithmetic average of that function times the area or the volume.

As an example, we examine the particular case of the 2D variant of the inviscid Burgers equation

$$\frac{1}{2} \frac{\partial u^2}{\partial x_1} + \frac{\partial u}{\partial x_2} = 0 \tag{4.23}$$

whose advection vector is $\boldsymbol{\lambda} = [u, 1]$. The problem is set up in $\Omega = (0,1) \times (0,1)$, and the boundary conditions are

$$u(0, y) = 1.5, \quad u(x, 0) = 1.5 - 2x, \quad x(1, y) = -0.5.$$

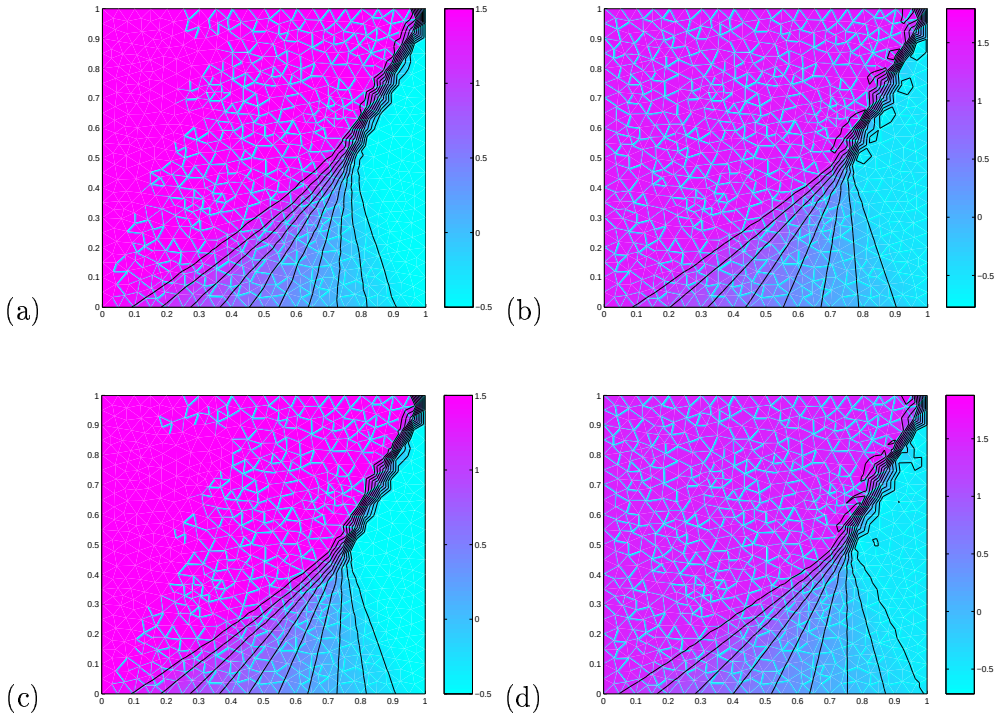


Figure 4.7: Solution of the inviscid Burgers equation (4.23) using: (a) N-scheme ($\max(u_h) = 1.5, \min(u_h) = -0.5$), (b) LDA-scheme ($\max(u_h) = 1.70, \min(u_h) = -0.79$), (c) PSI-scheme ($\max(u_h) = 1.5, \min(u_h) = -0.5$), (d) LW-scheme ($\max(u_h) = 1.87, \min(u_h) = -0.71$).

Here, the average state $\bar{\boldsymbol{\lambda}}^T = [\bar{\lambda}_1^T, \bar{\lambda}_2^T]$ can be calculated as

$$\bar{\lambda}_1^T = \frac{\sum_{\ell=1}^{3} u_\ell}{3}, \quad \bar{\lambda}_2^T = 1.$$

Results obtained using four of the schemes of table 4.1 are reported in figure 4.7. The grid is the same used for problem 4.21). The compression fan is resolved by all of the schemes in about the same way. LDA and LW, not satisfying property ($\mathcal{P}$), are non-monotone, and produce some oscillations around the discontinuity line. The N-scheme is monotone, but diffusive. The nonlinear PSI-scheme captures the shock without oscillations.

4.2 Hyperbolic Systems

Now, we turn our attention to the linear hyperbolic system

$$\sum_{j=1}^{d} A_j \frac{\partial \mathbf{U}}{\partial x_j} = \mathbf{0}, \tag{4.24}$$

whose numerical solution is searched in a piece-wise linear FE space defined on a triangulation $\mathcal{T}_h$ of the domain Ω made up of simplices. As in the scalar case, we suppose that the matrices A_j are constant over each element T. Appropriate boundary conditions need to be specified in order to close system (4.24).

In the remaining part of this chapter, we will indicate by $\mathbf{U} : \mathbb{R}^m \to \mathbb{R}^d$ both the solution of (4.24) and its numerical solution, where no ambiguity is possible. Furthermore, $\mathbf{U}_i$ will indicate the value of the numerical solution of (4.24) at node i.

Equation (4.7) is formally extended to systems as (4.24) as

$$\mathbf{R}_i(\mathbf{U}) = \mathbf{0},$$

where

$$\mathbf{R}_i(\mathbf{U}) = \sum_{T|i\in T} \boldsymbol{\phi}_i(\mathbf{U}) = \sum_T \beta_i^T \boldsymbol{\phi}_i^T$$

is a vector of dimension m which expresses the residual at node i of the mesh, $\beta_i^T \in \mathbb{R}^{m\times m}$ the distribution matrix, and $\boldsymbol{\phi}_i(\mathbf{U})$ the fluctuation, defined as

$$\boldsymbol{\phi}^T = \sum_i K_i \mathbf{U}_i,$$

where the inflow matrix $K_i \in \mathbb{R}^{m\times m}$ are written as

$$K_i = \sum_{j=1}^{d} A_j \mathbf{n}_i(j). \tag{4.25}$$

Here, $\mathbf{n}_i(j)$ represents the $j-$th component of the scaled normal vector $\mathbf{n}_i$, defined by equation (4.12) if $d = 2$ or (4.13) if $d = 3$.

Because of the hyperbolic nature of (4.24), the inflow matrices have a complete set of real eigenvalues and eigenvectors. Consequently, K_i can be written as

$$K_i = L_i^{-1} \Lambda_i L_i,$$

where the columns of L_i contain the left eigenvectors, Λ_i is the diagonal matrix of the eigenvalues.

To introduce upwinding in the numerical schemes, use is made of matrices K_i^- and K_i^+, which generalise the scalar coefficients k_i^- and k_i^+ of table 4.1. They are defined as

$$K_i^- = R_i \Lambda_i^- L_i, \quad K_i^+ = R_i \Lambda_i^+ L_i,$$

with

$$\Lambda_i^\pm = \frac{1}{2}\left(\Lambda_i \pm |\Lambda_i|\right).$$

The design criteria for distribution matrices are the same as for distribution coefficients for the scalar advection equation, plus the additional requirement of invariance under a similarity transformation.

Property 4.2.1 (invariance for similarity transformations). *An important design property of the system distribution schemes is that the residual sent to the nodes be independent of the choice of variables for which the actual distribution is performed. This property is called invariance (I). Under the similarity transformation $\partial \mathbf{U} = T \partial \mathbf{Q}$, the hyperbolic system (4.24) transforms into*

$$\sum_j A_{Q,j} \frac{\partial \mathbf{Q}}{\partial x_j} = 0,$$

where $A_{Q,j} = T^{-1} A_j T$. The residual of the transformed equation being

$$\boldsymbol{\phi}_i^T = \sum_{i \in T} \beta_{Q,i}^T \boldsymbol{\phi}_{Q,i}^T,$$

we require that

$$\beta_{Q,i}^T = T^{-1} \beta_i^T T.$$

Property 4.1.1 can be extended to systems by requiring that $\beta_i^T = 0$ when all the eigenvalues of K_i (which are real due to the hyperbolic nature of the system) are negative. As regards property 4.1.2, we have that, with $\boldsymbol{\phi}_i^T = \beta_i^T \boldsymbol{\phi}^T = \sum_j C_{i,j} \mathbf{U}_j$, the positivity condition becomes $C_{i,j} \leq 0, \forall j \neq i$. Property 4.1.4 now implies that the distribution matrices β_i^T remain bounded as $\boldsymbol{\phi}^T \to 0$, or, for a regular mesh and a regular solution of (4.24), $\boldsymbol{\phi}_i^T = \mathcal{O}(h^3)$. Property 4.1.3 is easily extended.

From property 4.2.1, it results that if the system is diagonalisable, i.e., if the matrices A_j have common eigenvectors, then taking Q as the diagonalising transformation, the transformed distribution matrix $\beta_{Q,i}^T$ is the diagonal matrix of scalar distribution coefficients for each decoupled equation.

Instead, when system (4.24) is not diagonalisable, the schemes listed in table 4.1 can be formally extended to systems as follows.

- The *N-scheme* for systems is defined by

$$\boldsymbol{\phi}_i^{T,N} = K_i^+ \left(\sum_j K_j^- \right)^{-1} \sum_j K_j^- (\mathbf{U}_i - \mathbf{U}_j). \tag{4.26}$$

  The scheme is defined provided that $\sum_j K_j^-$ is invertible. It can be shown that for the CEE, this matrix is always invertible, and depends continuously of the data. This scheme is dissipative.

- The *LDA-scheme* for systems is defined by

$$\beta_i^{T,LDA} = K_i^+ \left(\sum_j K_j^+ \right)^{-1}.$$

For the CEE, it is possible to show that distribution matrix is always well-defined, whatever the flow conditions. No dissipation property of the scheme has yet been proved. In [Abg01] it is shown that, for symmetric systems, the LDA scheme is less dissipative than the N-scheme.

- The non-linear *PSI-scheme* for system is more difficult to define. Considering the scalar PSI-scheme as a limited N-scheme, its formal generalisation is

$$\beta_i^{T,PSI} = \left(\sum_j \beta_j^{T,N^+} \right)^{-1} \beta_i^{T,N^+}.$$

Note that the distribution matrix of the N-scheme $\beta_i^{T,N}$ is not explicitly defined by equation (4.26). For diagonalisable systems, the condition of invariance under similarity transformations suffices to define $\beta_i^{T,N}$ and hence $\beta_i^{T,PSI}$ uniquely. For general non-diagonalisable systems, the invariance condition is no longer sufficient. The PSI scheme for systems used in the numerical applications is based on a particular definition of $\beta_i^{T,N}$ which satisfies this condition. For further details, see for instance [PDvdW97].

- The *Lax-Wendroff scheme* for systems is obtained with a formal generalisation of the scalar scheme given in table 4.1. We have

$$\beta_i^{T,LW} = \frac{1}{d+1} I_d + \frac{\Delta t}{2|T|} K_i,$$

with I_d the identity matrix of dimension d. The Rudqyard extension of the time step Δt allows to obtain a matrix LW scheme which does not depend on Δt:

$$\beta_i^{T,LW} = \frac{1}{d+1} I_d + \frac{\mu_{cell}}{|T|} K_i \left(\sum_j K_j^+ \right),$$

where μ_{cell} is a constant taken to 1; see [BR99].

4.3 The Compressible Euler Equations

The stationary compressible Euler equations describing non-viscous fluid flows in a domain $\Omega \subset \mathbb{R}^3$ can be written as

$$\frac{\partial \mathbf{F}_1(\mathbf{W})}{\partial x_1} + \frac{\partial \mathbf{F}_2(\mathbf{W})}{\partial x_2} + \frac{\partial \mathbf{F}_3(\mathbf{W})}{\partial x_3} = 0, \tag{4.27}$$

where

$$\mathbf{W} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho e \end{pmatrix}$$

is the vector of conservative variables. More precisely, 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 $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{F}_3$ are homogeneous functions of $\mathbf{W}$, called fluxes. Their expression is

$$\mathbf{F}_1 = \begin{pmatrix} \rho u_1 \\ \rho u_1^2 + p \\ \rho u_1 u_2 \\ \rho u_1 u_3 \\ \rho u_1 h \end{pmatrix}, \quad \mathbf{F}_2 = \begin{pmatrix} \rho u_2 \\ \rho u_1 u_2 \\ \rho u_2^2 + p \\ \rho u_2 u_3 \\ \rho u_2 h \end{pmatrix}, \quad \mathbf{F}_3 = \begin{pmatrix} \rho u_3 \\ \rho u_3 u_1 \\ \rho u_3 u_2 \\ \rho u_3^2 + p \\ \rho u_3 h \end{pmatrix}.$$

Here p is the (static) pressure, and h the specific total enthalpy. Considering air as a perfect gas, p and h are defined as

$$p = \gamma \left(e - \frac{\mathbf{V}^2}{2} \right), \quad h = \frac{\gamma}{\gamma - 1} \frac{p}{\rho} + \frac{\mathbf{V}^2}{2},$$

γ being the heat coefficient ratio. We have neglected all the heat conduction terms, and we have considered the fluid as a perfect gas [Hir89, Chapter 2]. This means, $p = \rho RT$, where T is the absolute temperature and R is the gas constant. If the gas is calorically perfect, then the specific energy is directly proportional to the absolute temperature and is given by

$$e(T) = C_v T,$$

where C_v is the specific heat at constant volume for the gas, and the specific enthalpy by

$$h(T) = e + \frac{h}{\rho} = C_p T$$

where C_p is the specific heat at constant pressure, $R = C_p - C_v$, and $\gamma = \frac{C_p}{C_v}$. For air, $\gamma = 1.4$. Finally, one can define the speed of sound a as

$$a = \sqrt{\gamma RT} = \sqrt{\frac{\gamma P}{\rho}}.$$

The CEE are currently used in aeronautical contexts, e.g. for calculating aerodynamic loads, in flutter analysis, and for verification of the shock locations in transonic flows. For application of interest, Equations (4.27) are posed in a bounded domain $\Omega \subset \mathbb{R}^d, d \in \{2, 3\}$, and completed by appropriate boundary conditions; see [Hir90].

Remark 4.3.1. *One of the perceived advantage of schemes based on element vertices is the fact that, on the boundaries, the solutions values are actually stored at those vertices that lie on the boundaries. Therefore, it is not necessary to extrapolate to the boundary from the cell-centers that lie off of the boundary as in FV schemes. The counterpart is that in FV schemes it is easy to apply boundary conditions on the fluxes, whereas MURD schemes do not actually compute fluxes through control domains. For a more detailed discussion on the boundary conditions for the CEE, the reader is referred to [Iss97, section 3.7] or [vdW98, section 5.2].*

To obtain a numerical solution of (4.27), we make use of MURD schemes. To that aim, we consider the quasi-linear form of (4.27), which reads

$$A_1 \frac{\partial \mathbf{W}}{\partial x_1} + A_2 \frac{\partial \mathbf{W}}{\partial x_2} + A_3 \frac{\partial \mathbf{W}}{\partial x_3} = 0, \tag{4.28}$$

with

$$A_1 = \frac{\partial \mathbf{F}_1}{\partial \mathbf{W}} \ , \ A_2 = \frac{\partial \mathbf{F}_2}{\partial \mathbf{W}} \ , \ A_3 = \frac{\partial \mathbf{F}_3}{\partial \mathbf{W}}.$$

Equation (4.28) can be discretized using the methods outlined in section 4.2. However, since now the A_j are not constant over a generic element T, we need to introduce a conservative linearisation, as done in section 4.1 for a scalar conservation law. This conservative linearisation, originally proposed by Roe [Roe81], introduces a vector $\mathbf{Z} = (1, u_1, u_2, u_3, H)$ such that over each simplicial element is assumed. Since the conserved variables and the flux vectors (4.28) are both quadratic in $\mathbf{Z}$, such that the Jacobian matrices are linear in $\mathbf{Z}$.

Another important ingredient for the application of MURD schemes to (4.27) is the elliptic/hyperbolic splitting, outlined in section 4.3.1. The result of this discretisation is a system of non-linear equations, which will be solved using a Newton-type scheme, as outlined in section 4.4. The Newton method leads to the solution of a sparse, ill-conditioned linear system, for which the application of iterative methods of Krylov types is the matter of choice. Finally, preconditioners based on Domain Decomposition strategies are applied. The resulting framework is referred to as ΨNKS framework. A further extension is given in section 4.5, where *a-posteriori* mesh adaptation techniques are applied to the resulting scheme.

4.3.1 Hyperbolic/Elliptic Splitting using Preconditioning

In this section we recall the hyperbolic/elliptic decomposition of the Euler equations. This decomposition, inspired by the Deconinck-Hirsch-Peuteman decomposition [DHP86], is crucial for the schemes used to discretise the Euler equations.

Two changes of variables have to be made in order to obtain the hyperbolic/elliptic decomposition. In a first step, we use the symmetrising variables

$$\partial \mathbf{Q} = \begin{pmatrix} \frac{\partial p}{\rho a} \\ \partial u_1 \\ \partial u_2 \\ \partial u_3 \\ \partial p - a^2 \partial \rho \end{pmatrix} = \varpi \partial \mathbf{U},$$

with

$$\mathbf{U} = \begin{pmatrix} \rho \\ u_1 \\ u_2 \\ u_3 \\ p \end{pmatrix}, \quad \varpi = \begin{pmatrix} 0 & 0 & 0 & 0 & 1/\rho a \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -a^2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where a is the speed of sound.

With the variable Q, the Euler equations become

$$\tilde{A}_1 \frac{\partial \mathbf{Q}}{\partial x_1} + \tilde{A}_2 \frac{\partial \mathbf{Q}}{\partial x_2} + \tilde{A}_3 \frac{\partial \mathbf{Q}}{\partial x_3} = 0$$

with

$$\tilde{A}_1 = \begin{pmatrix} u_1 & a & 0 & 0 & 0 \\ a & u_1 & 0 & 0 & 0 \\ 0 & 0 & u_1 & 0 & 0 \\ 0 & 0 & 0 & u_1 & 0 \\ 0 & 0 & 0 & 0 & u_1 \end{pmatrix}, \quad \tilde{A}_2 = \begin{pmatrix} u_2 & 0 & a & 0 & 0 \\ 0 & u_2 & 0 & 0 & 0 \\ a & 0 & u_2 & 0 & 0 \\ 0 & 0 & 0 & u_2 & 0 \\ 0 & 0 & 0 & 0 & u_2 \end{pmatrix}, \quad \tilde{A}_3 = \begin{pmatrix} u_3 & 0 & 0 & a & 0 \\ 0 & u_3 & 0 & 0 & 0 \\ 0 & 0 & u_3 & 0 & 0 \\ a & 0 & 0 & u_3 & 0 \\ 0 & 0 & 0 & 0 & u_3 \end{pmatrix}.$$

After this symmetrisation of the Euler equations, we introduce the second change of variables

$$\partial \tilde{\mathbf{W}} = \begin{pmatrix} \frac{\beta}{\rho a} \partial p + M \mathbf{s} \cdot \partial \mathbf{u} \\ \frac{\beta}{\rho a} \partial p - M \mathbf{s} \cdot \partial \mathbf{u} \\ M \mathbf{t} \cdot \partial \mathbf{u} \\ \frac{\partial p}{\rho a} + M \mathbf{n} \cdot \partial \mathbf{u} \\ \partial p - a^2 \partial \rho \end{pmatrix} = L \partial \mathbf{Q}$$

with

$$L = \begin{pmatrix} \beta & M s_1 & M s_2 & M s_3 & 0 \\ \beta & -M s_1 & -M s_2 & -M s_3 & 0 \\ 0 & M t_1 & M t_2 & M t_3 & 0 \\ 1 & M n_1 & M n_2 & M n_3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where M is the free-stream Mach number and β is defined as

$$\beta = \sqrt{\max\left(\iota^2, |M^2 - 1|\right)},$$

with $\iota = 0.05$, like in [Pai95], is introduced to remove the singularity in the sonic point, and $\mathbf{n} = \frac{\mathbf{u}}{\|\mathbf{u}\|}$. The vectors $(\mathbf{n}, \mathbf{s}, \mathbf{t})$ are mutually orthogonal.

This change of variables leads to

$$\frac{\partial \mathbf{W}}{\partial \mathbf{Q}} \left[\tilde{A}_1 R \frac{\partial \tilde{\mathbf{W}}}{\partial x} + \tilde{A}_2 R \frac{\partial \tilde{\mathbf{W}}}{\partial y} + \tilde{A}_3 R \frac{\partial \tilde{\mathbf{W}}}{\partial z} \right] = 0 \tag{4.29}$$

with $R = L^{-1}$.

In order to obtain the hyperbolic/elliptic decomposition, one can use the preconditioning matrix which was introduced by Van Leer and co-workers in [SDR91] and then extended to

the 3D case by Bonfiglioli and co-workers.

$$P = \frac{1}{q}\begin{pmatrix} \frac{\chi M^2}{\beta^2} & \frac{-\chi M u_1}{\beta^2 q} & \frac{-\chi M u_2}{\beta^2 q} & \frac{-\chi M u_3}{\beta^2 q} & 0 \\ \frac{-\chi M u_1}{\beta^2 q} & \eta\frac{u_1^2}{q^2}+\chi & \eta\frac{u_1 u_2}{q^2} & \eta\frac{u_1 u_3}{q^2} & 0 \\ \frac{-\chi M u_2}{\beta^2 q} & \eta\frac{u_1 u_2}{q^2}+\chi & \eta\frac{u_2^2}{q^2} & \eta\frac{u_2 u_3}{q^2} & 0 \\ \frac{-\chi M u_3}{\beta^2 q} & \eta\frac{u_1 u_3}{q^2}+\chi & \eta\frac{u_2 u_3}{q^2} & \eta\frac{u_3^2}{q^2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

with $q = \|\mathbf{u}\|$ and χ, η defined as

$$\chi = \frac{\beta}{\max(M,1)}\ , \quad \eta = \frac{\chi+\beta^2}{\beta^2} - \chi.$$

With this matrix, we write (4.29) in an equivalent form

$$\frac{\partial \mathbf{W}}{\partial Q} P^{-1} R \left[LP\tilde{A}_1 R \frac{\partial \tilde{\mathbf{W}}}{\partial x} + LP\tilde{A}_2 R \frac{\partial \tilde{\mathbf{W}}}{\partial y} + LP\tilde{A}_3 R \frac{\partial \tilde{\mathbf{W}}}{\partial z} \right] = 0.$$

This new form leads to

$$\frac{\partial \mathbf{W}}{\partial Q} P^{-1} R\, \Pi(\tilde{\mathbf{W}}) = 0 \tag{4.30}$$

with

$$\Pi(\tilde{\mathbf{W}}) = \begin{pmatrix} \chi\mu^+\mathbf{n} + \frac{\chi}{\beta}\mathbf{s} & \chi\mu^-\mathbf{n} & \frac{\chi}{\beta}\mathbf{t} & 0 & 0 \\ \chi\mu^-\mathbf{n} & \chi\mu^+\mathbf{n} - \frac{\chi}{\beta}\mathbf{s} & \frac{\chi}{\beta}\mathbf{t} & 0 & 0 \\ \frac{\chi}{2\beta}\mathbf{t} & \frac{\chi}{2\beta}\mathbf{t} & \chi\mathbf{n} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{n} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{n} \end{pmatrix} \nabla\tilde{\mathbf{W}} \tag{4.31}$$

with

$$\mu^{\pm} = \frac{M^2 - 1 \pm \beta^2}{2\beta^2}.$$

Thus, Equation (4.30) is an equivalent form of the Euler equations, at least locally on each element T.

In this new form, two different parts can be identified. The first three equations of (4.31) constitute a subsystem which is independent of the last two equations and is called the acoustic subsystem. The last two ones are entirely decoupled scalar advection equations. Using this decomposition we can treat the two parts in different ways, using different schemes. Indeed, the divergence of the fluxes can be written as the sum of the residual of the acoustic subsystem and of the residual of the last two scalar advection equations:

$$\frac{\partial \mathbf{F}_1}{\partial x} + \frac{\partial \mathbf{F}_2}{\partial y} + \frac{\partial \mathbf{F}_3}{\partial z} = \underbrace{(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) \left(\tilde{a} \nabla \begin{pmatrix} \tilde{\mathbf{W}}_1 \\ \tilde{\mathbf{W}}_2 \\ \tilde{\mathbf{W}}_3 \end{pmatrix} \right)}_{\phi^T_{coupled}} + \underbrace{\boldsymbol{\lambda}_4 \cdot \tilde{W}_4 \mathbf{r}_4}_{\phi^T_{enthalpy}} + \underbrace{\boldsymbol{\lambda}_5 \cdot \tilde{W}_5 \mathbf{r}_5}_{\phi^T_{entropy}}$$

where $\mathbf{r}_i$ are the vectors of the matrix $\frac{\partial W}{\partial Q}P^{-1}R$, $\boldsymbol{\lambda}_l$ the advection vectors of the last two scalar equations, and

$$\tilde{a} = \begin{pmatrix} \chi\mu^+\mathbf{n} + \frac{\chi}{\beta}\mathbf{s} & \chi\mu^-\mathbf{n} & \frac{\chi}{\beta}\mathbf{t} \\ \chi\mu^-\mathbf{n} & \chi\mu^+\mathbf{n} - \frac{\chi}{\beta}\mathbf{s} & \frac{\chi}{\beta}\mathbf{t} \\ \frac{\chi}{2\beta}\mathbf{t} & \frac{\chi}{2\beta}\mathbf{t} & \chi\mathbf{n} \end{pmatrix} = (\tilde{a}_1, \tilde{a}_2, \tilde{a}_3).$$

At this point, the strategy of the discretisation scheme is to split the acoustic residual, $\boldsymbol{\phi}^T_{coupled}$, with a matrix fluctuation splitting, and the two scalar $\boldsymbol{\phi}^T_{enthalpy}$ and $\boldsymbol{\phi}^T_{entropy}$ with scalar fluctuation splitting scheme.

Remark 4.3.2. *In the LW-PSI scheme, the system LW scheme is used to split $\boldsymbol{\phi}^T_{coupled}$ and the scalar PSI scheme is used to split $\boldsymbol{\phi}^T_{enthalpy}$ and $\boldsymbol{\phi}^T_{entropy}$. This scheme is outlined in details in [BR99]. There, the authors claim that this scheme allows Euler computations over a wide set of Mach numbers, from* 0.01 *up to* 2.

Remark 4.3.3. *Numerical comparison between MURD schemes and more classical FV schemes are reported in [IDe01]. The authors show that, compared to standard FV methods on comparable grids, the MURD schemes show an increased accuracy, which becomes comparable with structured grid algorithms.*

4.4 Generalities on Non-linear Solvers

This section presents a theoretical framework to characterise non-linear iterative solvers for the solution of a system of non-linear equations of type

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{4.32}$$

arising from the space discretisation of a steady PDE problem. In the following, we suppose that the spatial discretisation has been carried out by using MURD schemes.

The Newton method defines a suite $\{\mathbf{U}_k\}$ that, under some conditions, converges to $\mathbf{U}$, solution of (4.32). The algorithm is as follows: given $\mathbf{U}_0$, for $k = 1, \ldots$ until convergence, solve

$$J_k(\mathbf{U}_{k-1})\,(\mathbf{U}_k - \mathbf{U}_{k-1}) = -\mathbf{R}(\mathbf{U}_{k-1}), \quad J_k(\mathbf{U}_{k-1}) = \left[\frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k-1})\right] \tag{4.33}$$

Newton method introduces a local full linearisation of the equation. Solving a system of linear equations at each Newton step can be very expensive if the number of unknowns is large, and may not be justified if the current iterate is far from the solution. Therefore, a departure from the Newton framework consists of considering *inexact* Newton methods, which solve system (4.33) only approximatively.

In fact, in practical implementation of the Newton method, one or more of the following approximations are used:

1. The Fréchet derivative J_k for the Newton step is not recomputed at every Newton step;
2. The equation for the Newton step (4.33) is solved only inexactly;
3. Defect-correction methods are employed, that is, J_k is numerically computed using low-order (in space) schemes, while the right-hand side is built up using high-order methods.

Strategy 1 is particularly interesting when linear system (4.33) is solved by a direct method. In this case, evaluation and factorisation of the Jacobian matrix is not done at every Newton step. Other techniques aim to update the Jacobian without recomputing it, and for problems where the Jacobians are slowly varying, this leads to significant performances gains. On the contrary, for rapidly varying and ill-conditioned Jacobians, this procedure can produce a premature termination of the Newton process.

Strategy 2 means that the equation for the Newton step is solved inexactly in the sense that $\Delta\mathbf{U}_k = \mathbf{U}_k - \mathbf{U}_{k-1}$ is meant to satisfy

$$\|J_k(\mathbf{U}_{k-1})\Delta\mathbf{U}_k + \mathbf{R}(\mathbf{U}_{k-1})\| \leq \zeta_k \|\mathbf{\Phi}(\mathbf{U}_k)\|$$

for some small ζ_k, which may change as the iteration progresses.

Strategy 3 means that J_k is computed (or applied to a vector) using a low-order space discretisation scheme, while the right-hand side is calculated using a high-order scheme.

The convergence of exact and inexact Newton methods can be difficult to achieve. Newton method requires initial iterates sufficiently near to the root, which are usually not available, especially when the solution has complex flow features, such as discontinuities (shocks, slip lines) that are not present in the initial iterate. Compressible flows induce, with respect to these possible discontinuities in their solution, a different framework from other problems, like for instance incompressible flow problems, where one can use Newton techniques alone.

For these reasons, system (4.32) is usually framed in a time-dependent setting. Its root $\mathbf{U}$ is found by seeking for a steady-state solution of the time-dependent problem

$$S\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{4.34}$$

with a convenient initial condition $\mathbf{U}(t=0) = \mathbf{U}_0$. In (4.34), S is a non-singular (lumped) mass-matrix.

Framing the steady-state equation into a time-dependent setting constitutes a sort of continuation process, with continuation parameter t, 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. The resulting procedure is referred to as pseudo-transient Newton (ΨN) methods.

General results about Newton method and its variants can be found on reference books: see for instance [QSS00] for a global presentation on the subject, the book by Ortega and Rheinboldt [OR70], or the book by Kelley [Kel95].

4.4.1 The ΨN Framework

It its simplest form, the pseudo-transient continuation Newton method numerically integrates the initial value problem (4.34) with initial condition $\mathbf{U}(t=0)=\mathbf{U}_0$, to steady-state using an evolutive time-step strategy which attempts to increase the time-step as $\mathbf{U}_k \to \mathbf{U}$. The time derivative is discretised using a backward Euler scheme. Starting from a given $\mathbf{U}_0$, the solution at (pseudo-)time step $k=1,2,\ldots$ is found by solving the non-linear system

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

or, equivalently,

$$S\delta_k^{-1}\mathbf{U}_k - S\delta_k^{-1}\mathbf{U}_{k-1}+\mathbf{R}(\mathbf{U}_k)=\mathbf{0}, \tag{4.35}$$

where δ_k the time increment at step k.

A Newton iteration at time level k for the solution of (4.35) would then read

$$\mathbf{U}_{k,l}=\mathbf{U}_{k,l-1}+\left[S\delta_k^{-1}+\frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k,l-1})\right]^{-1}(-S\delta_k^{-1}\mathbf{U}_{k,l-1}-\mathbf{R}(\mathbf{U}_{k,l-1})+S\delta_k^{-1}\mathbf{U}_{k-1}) \tag{4.36}$$

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 [KK98b] show that, for steady-state computations, quadratic convergence can be eventually achieved provided that a suitable time-step evolution strategy for δ_k is found. Therefore, in the following we suppose to take just one Newton iteration of process (4.36). This leads to the following linear problem:

$$\mathbf{U}_k=\mathbf{U}_{k-1}+\left[S\delta_k^{-1}+\frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k-1})\right]^{-1}(-\mathbf{R}(\mathbf{U}_{k-1})) \tag{4.37}$$

that can be written as

$$A_k\mathbf{u}_k=\mathbf{f}_k, \tag{4.38}$$

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

$$\begin{aligned} A_k &= \left[S\delta_k^{-1}+\frac{\partial \mathbf{R}}{\partial \mathbf{U}}(\mathbf{U}_{k-1})\right] \\ \mathbf{u}_k &= (\mathbf{U}_k-\mathbf{U}_{k-1}) \\ \mathbf{f}_k &= -\mathbf{R}(\mathbf{U}_{k-1}). \end{aligned}$$

A strategy for the definition of δ_k is needed. One strategy consists of keeping the time step small until all flow features are resolved, then large time steps may be taken near the solution to obtain super-linear or quadratic convergence of Newton's method.

To that aim, the so-called "switched evolution relation rule" (SER) can be used; see [DES82]. In its simplest form, SER increases the time step in inverse proportionality to the residual reduction,

$$\delta_k = \delta_0 \frac{\|\mathbf{R}(\mathbf{U}_0)\|}{\|\mathbf{R}(\mathbf{U}_k)\|},$$

which implies that, for $k \geq 1$,

$$\delta_k = \delta_{k-1} \frac{\|\mathbf{R}(\mathbf{U}_{k-1})\|}{\|\mathbf{R}(\mathbf{U}_k)\|}.$$

Sometimes, δ_k is computed using the formula

$$\delta_k = \min \left\{ \delta_{k-1} \frac{\|\mathbf{R}(\mathbf{U}_0)\|}{\|\mathbf{R}(\mathbf{U}_k)\|}, \delta_{max} \right\}, \tag{4.39}$$

with δ_{max} a real parameter which may be ∞. This is called "switchover to steady-state form".

Another way is to adopt the "exponential rule", i.e.

$$\delta_k = \min \left\{ \delta_0 \times (\sigma)^k, CFL_{max} \right\}, \tag{4.40}$$

where δ_0 is the initial time step and $\sigma > 1$ is a prescribed growing factor.

A third, more involved strategy, called "expert rule", can be found in [VCR00]. Based on our numerical results, the performances of SER and exponential rule are comparable. Therefore, in the numerical results presented in section 4.6, we have adopted the latter to define the sequence $\{\delta_k\}$.

As an example, we have applied the ΨN framework to problem (4.23). Figure 4.8 reports the convergence history using (4.35) and (4.40) with $\sigma = 2$ and $CFL_{max} = \infty$ to define the time increment. For this scalar problem, convergence is obtained for all the tested MURD schemes. Note, from figure 4.9, that the spectral condition number of J_k rapidly increasing in the first 10 iterations, then slightly decreases, then it remains almost constant when $1/\delta_t \approx 0$. In the latter phase, ΨN reduces to the Newton method for the solution of the stationary problem.

The exponential rule (using $\sigma = 2$ and $\sigma = 10$) and SER are compared in figure 4.10. Finally, figure 4.11 reports the convergence history of process (4.35) and of (4.37) using 2,3 or 4 Newton iterations at each time level. Note that no improvements are achieved in the initial phase (the first 10 iterates) if more than one Newton iteration is used. The convergence using $L > 1$ Newton iterations is more rapid, but note that each step costs L times more with respect to (4.35). Note also that $L = 4$ results in the most irregular convergence history among the presented cases.

Process (4.37) requires the construction of the Jacobian matrix (as regards Jacobian-free methods, see the following remark 4.4.1). First, note that a generic (i, j) element of J_k is a-priori non-zero only if i and j correspond to quantities defined on nodes which are mesh

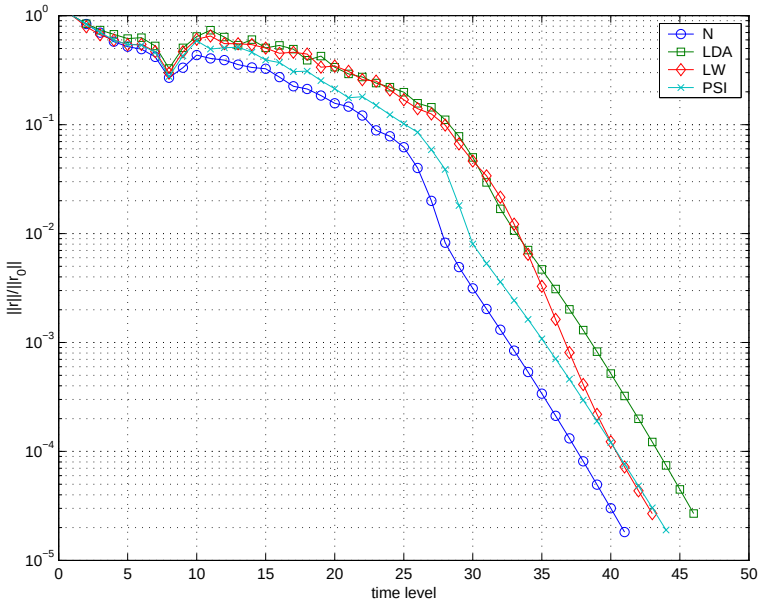


Figure 4.8: Convergence of the ΨN framework applied to problem (4.23) using different MURD schemes. The exponential rule has been chosen for δ_k with $\sigma = 2$, and one Newton iteration is used.

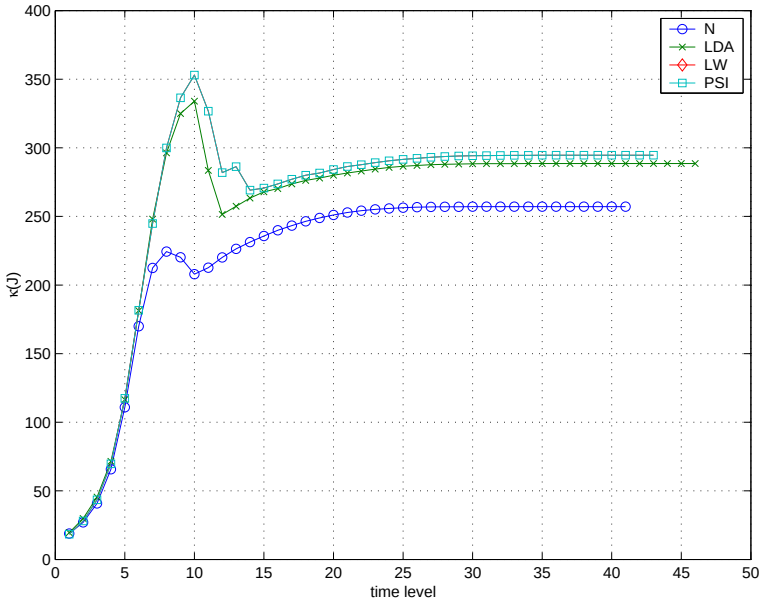


Figure 4.9: Spectral condition number of J_k at different time-levels.

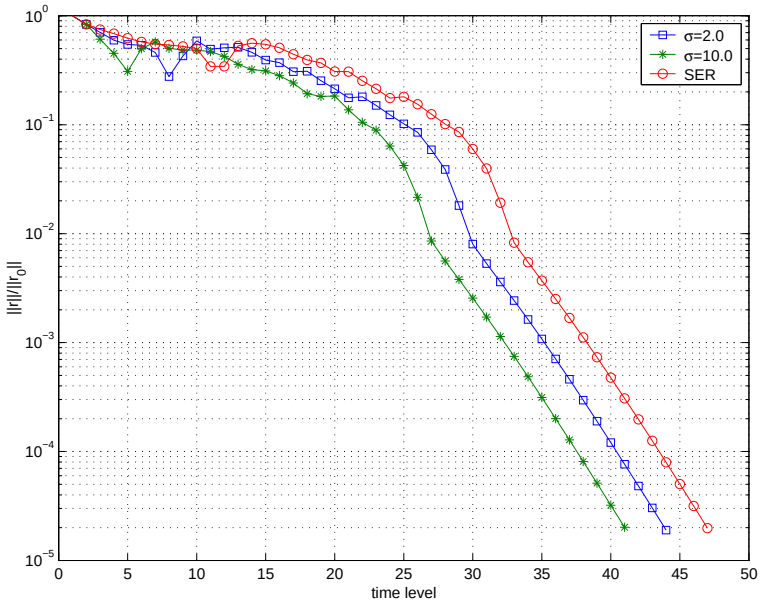


Figure 4.10: Convergence of the ΨN scheme applied to problem (4.23) using the PSI-scheme and one Newton iteration, using the exponential rule (with $\sigma = 2$ and $\sigma = 10$) or the SER.

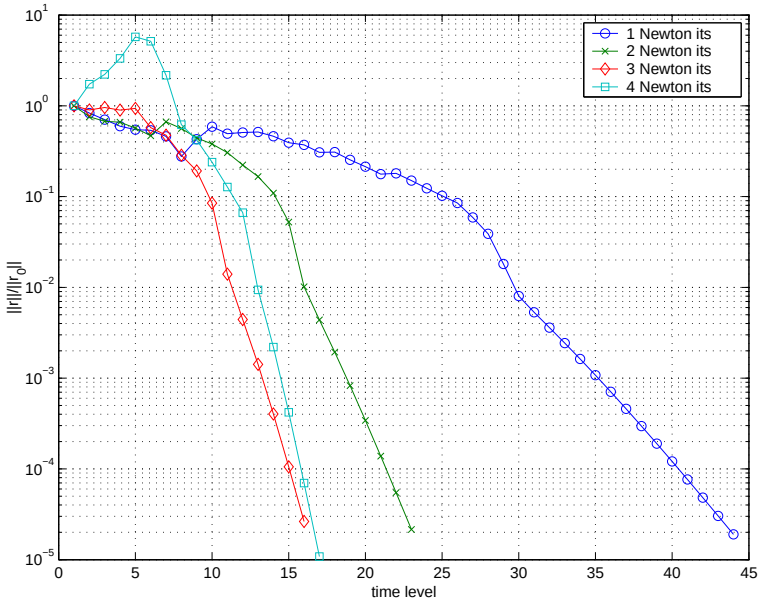


Figure 4.11: Convergence of the ΨN scheme applied to problem (4.23) using the PSI-scheme and 1,2,3 or 4 Newton iterations. The exponential rule has been chosen for δ_k with $\sigma = 2$.

connected. J_k is therefore a sparse matrix and the sparsity pattern for schemes on simplices is induced by the mesh structure itself. Moreover, for system of equations m unknowns are stored on each mesh node, so that the (i, j) Jacobian entry is a small dense matrix relating the influence of every variable at node i on every variable at node j. Suitable matrix storage formats can be used at this purpose.

To compute the sparse matrix J_k, different approaches are available. A first approach uses analytical formulae, that is, the element of J_k are evaluated using exact derivatives of the residual distributed to each node. The task of differentiating this quantity – and the boundary conditions – might be cumbersome and error-prone. This is because the nodal residual is computed according to a series of phases: the conservative linearisation, the evaluation of the residual, and its distribution to the nodes of the element. For this reason, analytical Jacobians are used only for some first-order schemes. Moreover, the exact derivatives are often replaced by approximate formulae, computationally cheaper to compute.

A second approach uses FD approximations of the derivative. Using a first-order forward formula, the generic (i, j) element is computed as

$$J_k(i,j)(\mathbf{U}) \approx \frac{\mathbf{R}_i(\mathbf{U} + \varepsilon \mathbf{e}_j) - \mathbf{R}_i(\mathbf{U})}{\varepsilon}$$

where $\mathbf{R}_i(\mathbf{U} + \varepsilon \mathbf{e}_j)$ represents the $i-$th component of $\mathbf{U}$ when the $j-$th component of $\mathbf{U}$ is perturbed by a small quantity ε. The advantage of this approach is that it requires only a function to evaluate the residual for a given $\mathbf{U}$, and therefore any spatial discretisation scheme can be numerically differentiated without insight of the discretisation chosen nor any intricated analytical formulae. One drawback is its cost: for the 3D laminar CEE, 21 residuals have to be computed on each element T. Another drawback is that the precision achieved depends on a proper choice of ε. For our problems, a value of $\varepsilon = 10^{-6}$ was experimentally found to provide the optimal result in terms of convergence of the overall scheme.

Other techniques that can be employed to compute the J_k: automatic differentiation (AD) tools, the Picard method, and the (sparse) Broyden update. In [Iss97], all the approaches mentioned in this section are numerically compared. It results that the computational cost of first-order FD schemes is comparable with that of analytical computations, and that the nonlinear convergence obtained by using first-order FD approximation is almost identical to that obtained using analytical Jacobians.

In the numerical results later reported, we have used both analytical and FD Jacobians for first-order MURD schemes, and FD Jacobians for second-order MURD schemes.

Remark 4.4.1. *In FV application, Jacobian free methods are often employed, meaning that the Jacobian matrix is not explicitly stored, but only applied to vectors (using Fréchet derivatives) within an iterative solver. However, a preconditioning matrix is to formed, usually resorting to low-order methods. For FV methods this approach is justified by the wider stencil of high-order methods. For MURD methods, instead, the Jacobian free approach is quite questionable, since high-order schemes have the same computational stencil of low-order ones*

(although they may be more expensive to compute). We have not considered Jacobian-free methods in this work

4.4.2 The ΨNKS Framework

Once the matrix and the right-hand-side of (4.38) have been formed, a linear system has to be solved at each pseudo-time step k. A Newton-Krylov method adopts for this purpose a Krylov acceleration procedure (such as GMRES). Although in principle the matrix of the system needs not to be explicitly formed (matrix-free method), it may be convenient to form it (being enough memory storage available), at least if one wishes to apply algebraic preconditioners. Here we will not consider here matrix-free methods. Indeed, our strategy is to use a Schwarz or Schur preconditioner (like those introduced in chapter 3) in the course of the solution of the Newton iterates by Krylov methods. The resulting algorithm is called ΨNKS (S stands for Schwarz or Schur); see, for instance, [CKK02, KKS99], [NAWK95, CKV97, GMTK00, GKKS01]). The parallel issues of the resulting scheme will be addressed in chapter 5.

4.5 The αΨNKS Framework

The ΨNKS algorithm allows the solution of problem (4.32), resulting from a space discretisation on a given mesh, say $\mathcal{T}^{(0)}$. To improve the solution quality and to optimise the computational resources, a-posteriori adaptation cycles are considered. This means that, after convergence on the initial grid, the mesh $\mathcal{T}^{(0)}$ is adapted one or several time, following adaptation criteria, as detailed in section 4.5.1. At adaptation cycle i, the solution $\mathbf{U}^{(i)}$ corresponding to the solution of the pseudo-transient problem on the mesh $\mathcal{T}^{(i-1)}$, is projected on the mesh $\mathcal{T}^{(i)}$, and then used as a starting solution for problem (4.32), discretized on $\mathcal{T}^{(i)}$.

Note that different space discretisation techniques can be used at different meshes adaptation cycles. In general, we start with first-order schemes on non-adapted meshes, then we turn to second-order schemes on adapted meshes. The final algorithm 4.5.1 will be called αΨNKS.

Algorithm 4.5.1 ($\alpha\Psi$NKS).

a. Define a mesh $\mathcal{T}^{(0)}$

b. Choose a starting solution $\mathbf{W}^{(0)}$

c. For $i = 0, \ldots$, Do

1. solve the non-linear ODE problem

$$S\frac{d\mathbf{U}^{(i)}}{dt} + \mathbf{R}(\mathbf{U}^{(i)}) = \mathbf{0} \quad \text{with } \mathbf{U}^{(i)}(t=0) = \mathbf{W}^{(i)}$$

resulting from the space discretisation on $\mathcal{T}^{(i)}$;

2. if the solution has the desired accuracy, Stop;

3. adapt $\mathcal{T}^{(i)}$ to obtain $\mathcal{T}^{(i+1)}$

4. set $\mathbf{W}^{(i+1)}$ as the projection of $\mathbf{U}^{(i)}$ on $\mathcal{T}^{(i+1)}$

d. EndFor

4.5.1 Generalities on Mesh Adaptation

One of the main advantages of unstructured grids is their capability to adapt dynamically the mesh by localised refinement/derefinement during the calculation to enhance the solution accuracy and to optimise the computational time. Mesh adaption is a very active research field; the reader is referred to [Ver96, AO00, BS01] for a complete presentation of the subject. This section outlines the main lines of the mesh adaptation algorithms we have used. The parallel issues related to parallel mesh adaptivity will be found in section 5.3.8.

The goal of mesh adaptation is to increase the accuracy of the solution process by reducing the mesh size when needed – thus increasing solution accuracy, while minimising the number of elements where the solution is already accurate enough – thus optimising the computational resources.

The first step in a mesh adaptation algorithm is to locally refine the mesh according to some criteria; these criteria used should be as close as possible to the error estimations of the underlying discretisation scheme. For the mesh adaptation procedures developed here, a strategy based on an *a-posteriori* error estimate where the computed residual of the solution $\mathbf{R}(\mathbf{U}_k)$ is used to define the error has proved to be robust and precise for inviscid flows and for tracking discontinuities. These solution based criteria evaluate the following: difference on the edge, gradient on the edge, flux through the edge (upwind and downwind), applied on the sensitive variables, such as the density, the pressure, the local Mach number or the entropy. Other criteria are based on geometrical quantities, for instance the absolute edge length, the relative edge length with respect to neighbouring elements, and the geometrical form of the elements (solid angle).

Using one or a combination of these criteria, mesh refinement and derefinement operations are performed. Then, an optimisation step follows, based on geometrical properties of the current mesh, to improve some mesh quality parameters without affecting the mesh connectivity. These phases are followed by repartition, reordering and renumbering.

Local Mesh Refinement

The local mesh refinement algorithm consists in testing the edges of an existing mesh and deciding whether they have to be split or not. First, the selected criterion function is evaluated on the entire mesh. Call this field variable $\mathbf{c}$. A low- or high-pass filter is applied on the solution field $\mathbf{c}$. Let $\hat{\mathbf{c}}$ denote the corresponding filtered field, and let $\hat{c}_i$ be the value of $\hat{\mathbf{c}}$ on the edge i.

For each edge, the function $\mathbf{f} = \mathcal{F}(\hat{\mathbf{c}})$ is computed, where $\mathcal{F}(\hat{\mathbf{c}})$ is one of the following:

- the difference of $\hat{\mathbf{c}}$ between the two vertices of the edge;
- the gradient of $\hat{\mathbf{c}}$ between the two vertices of the edge;
- the upwind flux of $\hat{\mathbf{c}}$ through the edge;
- the downwind flux of $\hat{\mathbf{c}}$ through the edge.

Finally, the refinement criterion is built with the mean value $\bar{f}$ of $\mathbf{f}$ over the grid. Edge i will be refined, that is, a vertex is added in the medium point of that edge, if $\hat{f}_i > \alpha \bar{f}$, or kept unchanged otherwise. Here, α is a real parameter. The factor $\mathcal{F}$ is used to set the criterion as a function of the mean value $\bar{f}$.

In 2D, this procedure may create 1, 2, or 3 new vertices in an adapted triangle. This subdivision strategy is called of the Red-Green type (figure 4.12). For geometrical considerations of maintaining non-acute angles, elements of type Green II are replaced by elements of type Red. The Red-Green concept is extended to the faces of the tetrahedral in 3D meshes. For each element, 1, 3 or 4 new vertices may be created (figure 4.13). Element refinement thus subdivides elements into 2, 4, 8 new elements in 3D (see figure 4.13).

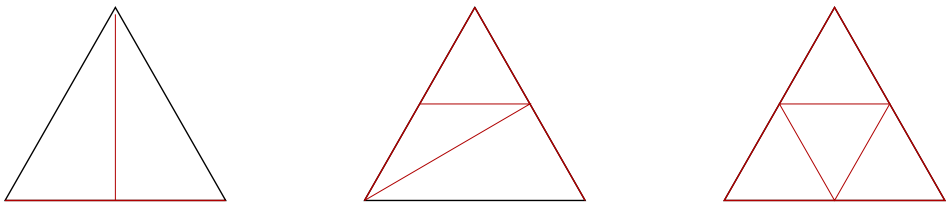
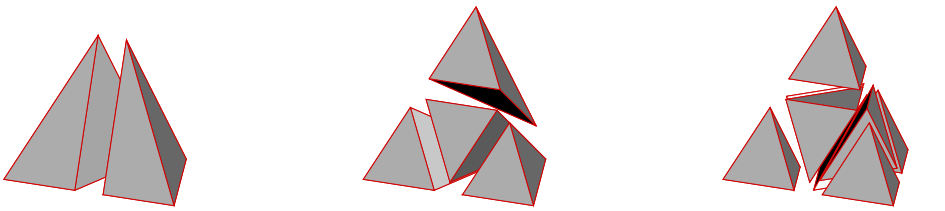


Figure 4.12: Two-dimensional element refinement types.



1 edge marked 3 edges marked 6 edges marked

Figure 4.13: Three-dimensional element refinement types.

Mesh Derefinement

In order to minimise the number of vertices and elements used in the computation, and to improve the geometrical quality of the mesh, a derefinement algorithm is employed. The procedure is as follows. First, a set of nodes not to be deleted is identified. This is done using the same criteria of the refinement. A particular care must be given to some nodes, like, for instance, the corners of the computational domain, the symmetry points, or the vertices belonging to edges to be refined. There is a number of possible fixed vertices allowing a maximal coarsening which come either from the boundary or from internal configurations. To find these, first the border of the domain is scanned so that each second vertex is marked as a fixed point. Then all elements containing no fixed points must have at least three fixed neighbouring vertices, as shown in figure 4.14. All the remaining vertices are marked, and the other ones are subject to be deleted.

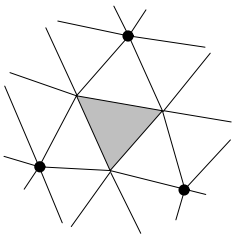


Figure 4.14: An element defined with 3 removable vertices should be surrounded by 3 fixed vertices. Fixed vertices are marked.

Vertices to be deleted are removed using an edge collapsing procedure, see figure 4.15. The shell built with the surrounding elements around a marked vertex is considered as a new element. The inner curvature angle of the shell is then computed at each non-marked vertex neighbour. The vertex is deleted by collapsing one of its neighbouring edges. The actual edge to be collapsed is chosen to be the one that joins the deleted vertex to the neighbouring vertex for which the shell curvature angle is maximal (figure 4.16). This procedure works in two and in three dimensions. If an element inversion happens, the incident is detected by controlling

the shell volume conservation and the vertex is not removed.

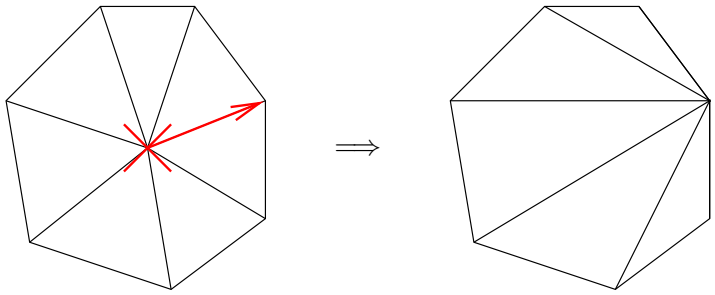


Figure 4.15: Vertex deletion using edge collapsing

Choosing the largest curvature angle tends to minimise the risk of element inversion, and therefore the number of failing deletion processes. This improves also the quality of the resulting mesh (figure 4.16).

Examples of 2D derefinement sweeps are given in figure 4.17. Note that exactly the same algorithm is applied on the shells in 3D.

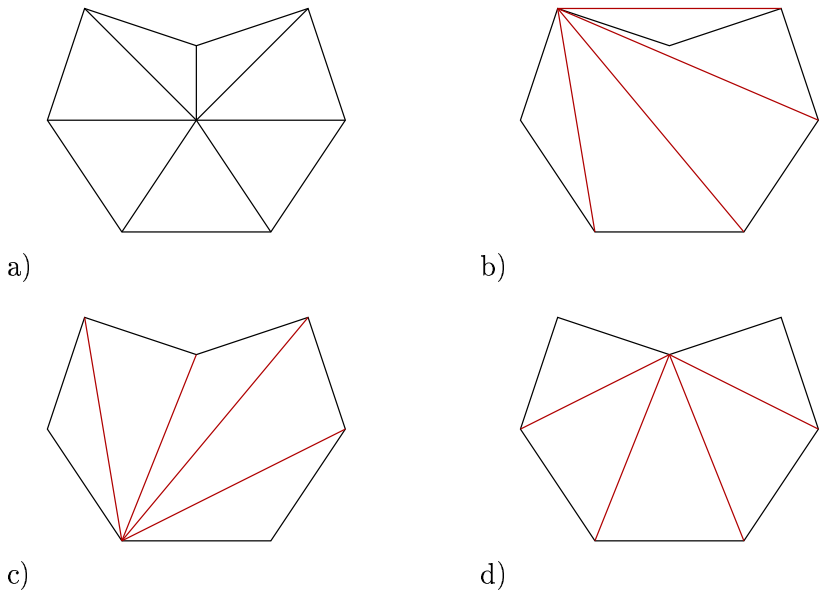
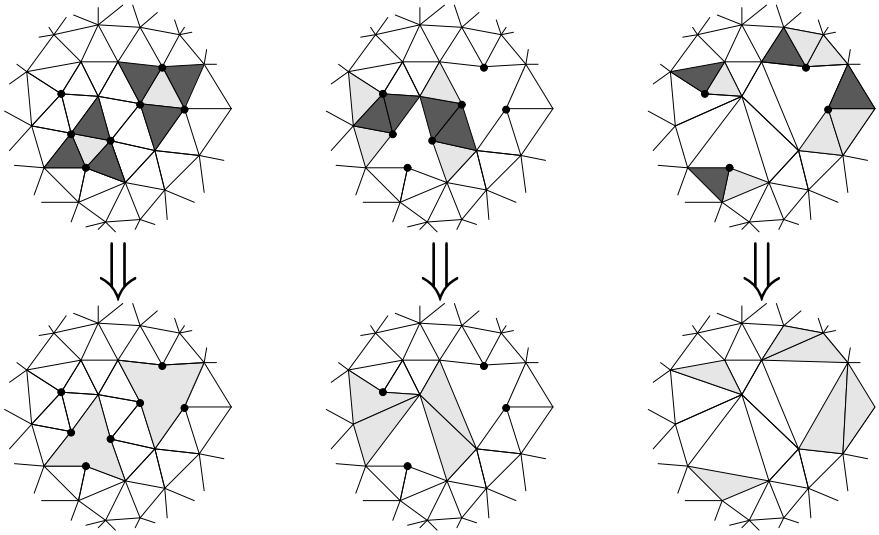


Figure 4.16: Coarsening by edge collapsing. a) Original shell. b) Wrong edge collapsing causing an element inversion. c) Wrong edge collapsing causing element distortions. d) Appropriate edge collapsing.



3 marked vertices 2 marked vertices 1 marked vertices

Figure 4.17: Derefinement sweeps in 2D.

Mesh Optimisation

The adaptive methods previously presented modify the local vertex density, and strong inhomogeneities may be created across the mesh, depending on the element size as well as the vertices angles. Moreover, the number of neighbours may vary in a large spectrum from one vertex to one other. To minimise such dependencies it is necessary to optimise the structural quality (element shape) and to globally smooth out the mesh.

First, it is necessary to determine the optimal number of neighbours for each element $\mathcal{N}_{\text{opt}}$. For 2D meshes, this is straightforward: the optimal vertex angle is $\pi/3$ in the Euclidean metric, and the complete area around the vertex is filled with six elements. The number of neighbouring vertices needed to form those elements around a vertex is then $\mathcal{N}_{\text{opt}} = 6$.

In 3D, the spherical angle of the tetrahedron at each vertex is the given by $\varepsilon = 3\alpha - \pi$. α is evaluated by considering the theorem of the cosine which leads to $\alpha = \arccos(1/3)$, see figure 4.18. The number of tetrahedral elements needed to fill the space around a vertex is then $T = 4\pi/\varepsilon = 23$. The Euler-Descartes relation may be used to express the relation between the numbers of elements, vertices, edges and faces within a given triangulation. Let consider the shell built with the elements surrounding the vertex as a triangulation. The Euler-Descartes relation $V - E + F = T + 1$ [Car97] links the number of vertices V, of edges E, of faces F of a grid with its element number T. On the shell boundary, the number of vertices is $\mathcal{N}_{\text{opt}}$, and the number of faces is T. It follows that $T = 2\mathcal{N}_{\text{opt}} - 4$, which leads to $13 < \mathcal{N}_{\text{opt}} < 14$.

Once defined the optimal number of neighbours for each element, structural optimisation operations are performed, so that each element has a number of neighbours which is not too far away from the optimal. These structural optimisation operations are *diagonal swapping*

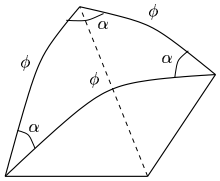
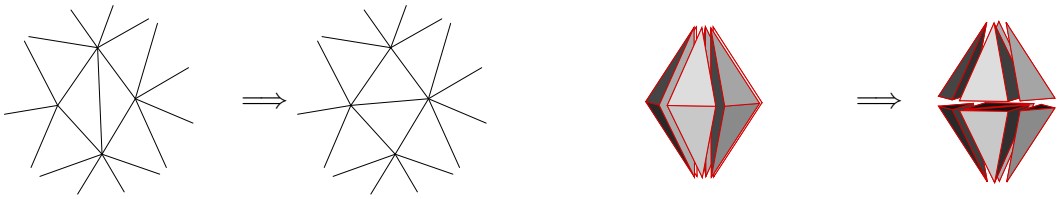


Figure 4.18: Spherical angle at an equilateral element vertex. The angle ϕ equals $\pi/3$, and $\alpha = \arccos(1/3)$.

and *collapsing* of edges in 2D and faces in 3D. Diagonal swapping checks for each edge if configuration is improved by a swap and reduces the number of neighbours $\mathcal{N}_i$ when this is too high. i.e swapping is accepted if:

$$\begin{cases} \mathcal{N}_3 + \mathcal{N}_4 & \leq & \mathcal{N}_1 + \mathcal{N}_2 \\ \max(\mathcal{N}_3, \mathcal{N}_4) & < & \max(\mathcal{N}_1, \mathcal{N}_2). \end{cases}$$

Graphically, we have:



Instead, the edge collapses $\mathcal{N}_i$ when it is too low. The procedure checks for each edge if a vertex has an insufficient number of neighbours – then it is removed by collapsing; see figure 4.19. A histogram of these structural optimisations is given in figure 4.20.

Once structural optimisation is completed, *geometrical* optimisation is performed by *smoothing* and *stretching algorithms*. The underlying idea of mesh smoothing is to optimise the mesh by modifying the vertex position, using the so-called spring analogy (see figure 4.22). Note that in this step the mesh connectivity – and the corresponding data structures – remains unchanged.

The spring analogy consists of replacing each mesh edge by an elastic spring and minimising the deforming energy of the corresponding elastic system. Let $\boldsymbol{x}_i$ be the spatial coordinated of vertex i. At each vertex, the resulting force is given by Hook's law:

$$\boldsymbol{F}_i = \sum_{j \in k_i} \alpha_{i,j} (\boldsymbol{x}_j - \boldsymbol{x}_i) = \boldsymbol{0}, \tag{4.41}$$

and therefore the equilibrium position is given by

$$\boldsymbol{x}_i = \frac{\sum_{j \in k_i} \alpha_{i,j} \, \boldsymbol{x}_j}{\sum_{j \in k_i} \alpha_{i,j}}. \tag{4.42}$$

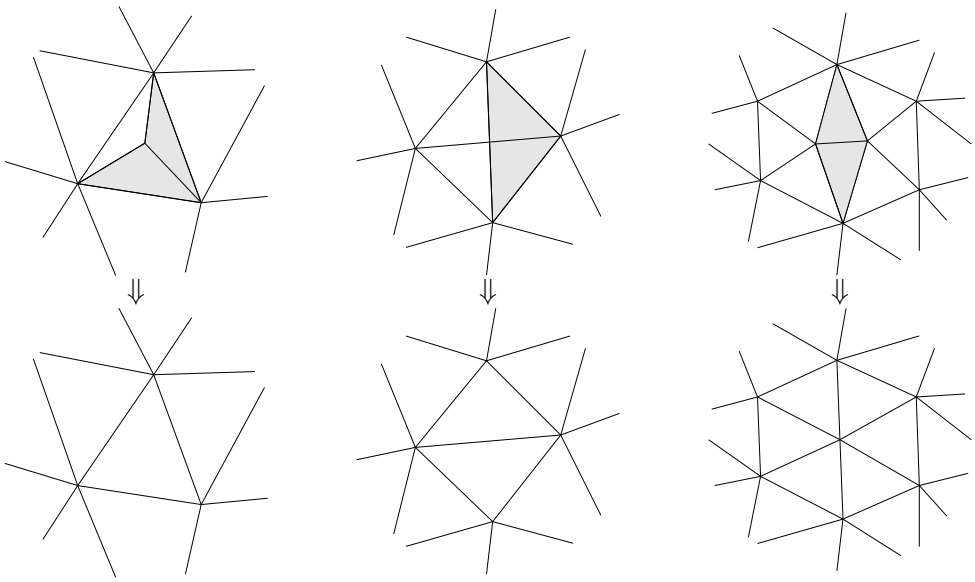


Figure 4.19: Collapsing of elements.

Here k_i indicates the set of neighbouring vertices of vertex i, $\alpha_{i,j}$ the spring stiffness of the edge joining the vertex i with the neighbour j.

Remark 4.5.1. *Equation (4.42) requires the resolution of a linear system, for example an iterative scheme like Jacobi can be used. This is particularly well suited for our parallel computations, since it mainly corresponds to parallel matrix-vector products (plus diagonal scaling). In general few steps of the Jacobi method are enough to significantly improve the mesh quality.*

To complete the algorithm a definition of the value $\alpha_{i,j}$ is needed. The simplest choice $\alpha_{i,j} = 1$ tends to produce equilateral elements. However, as shown in figure 4.23 (b), a vertex with $\mathcal{N}_i < \mathcal{N}_{\text{opt}}$ will attract its neighbours, while a vertex with $\mathcal{N}_i > \mathcal{N}_{\text{opt}}$ will move them away. Here $\mathcal{N}_{\text{opt}}$ is the optimal number of neighbours for each vertex (14 in 3D). x This effect can be reduced by using a weight function for $\alpha_{i,j}$, related to the number of neighbours for the vertex $j \in k_i$ $\mathcal{N}_j$:

$$\alpha_{i,j} = \alpha_j = \max\left[1, \mathcal{N}_{\text{opt}} + a\left(\mathcal{N}_i - \mathcal{N}_{\text{opt}}\right)\right], \tag{4.43}$$

where a is a real coefficient to be chosen. The influence of (4.43) appears in the figure 4.23 (c).

Further improvements of the spring analogy have been developed in [Blo98, SL00]. These techniques may be of particular interest near boundary regions. They correspond to taking different equilibrium lengths for Hook's law. Possible choices are:

- Lineal vertex spring with zero equilibrium length and stiffness $\alpha_{i,j} = \phi((x_i - x_j)^2 + (y_i - y_j)^2)^\psi$, where ϕ is a function of the boundary curvature and ψ is a parameter;

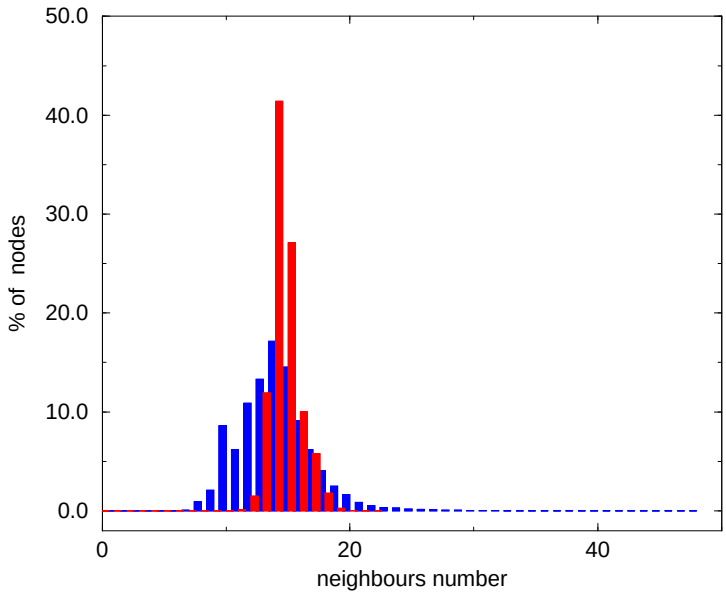


Figure 4.20: Structural optimisation for one of the test cases, the ONERA M6 wing. Red lines refers to before optimisation, blue lines after optimisation. Note the improvement of the neighbour number distribution.

- Edge spring taking the equilibrium length to be the edge length. Then, the Hook's law applied to vertex displacement $\boldsymbol{\delta}_j$ gives

$$\boldsymbol{F}_i = \sum_{j=1}^{k_i} \alpha_{i,j}(\boldsymbol{\delta}_j - \boldsymbol{\delta}_i)$$

  and the spring stiffness is taken proportional to the inverse of the edge length,

$$\alpha_{i,j} = \frac{1}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}.$$

Increasing ϕ near boundaries increases the stiffness and keeps the vertices close to the boundary. An increased rigidity near boundaries is also necessary to avoid distortion and inversion in these zones, and an additional term which is function of the local boundary curvature is also introduced; see figure 4.24. Further techniques include also torsional springs which enforce edge springs to work only in one direction to avoid inverted elements.

We conclude the section with the following remark.

Remark 4.5.2. *The extension of the presented adaptation techniques to the solution of the compressible Navier-Stokes equations requires a careful handling of the stretched elements used*

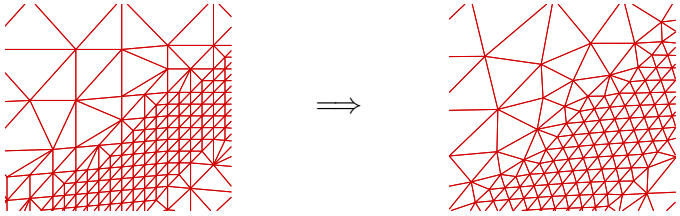


Figure 4.21: Example of the mesh improvement with an optimisation procedure.

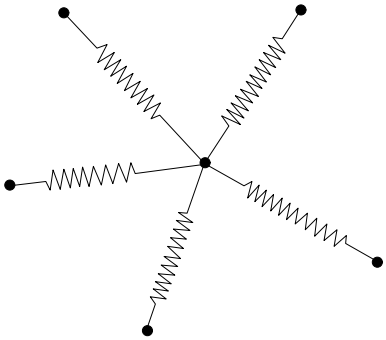


Figure 4.22: Physical interpretation of the spring analogy

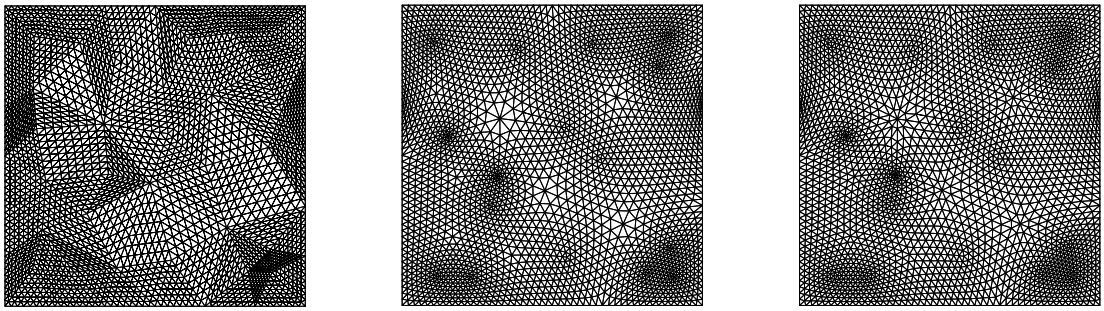
in the boundary layers. These elements have, by construction, a dimension which is much smaller that the others, and ad-hoc *adaptation techniques must be considered to exploit this peculiar aspect of the mesh.*

4.6 Validation of the Euler Code

A set of test cases has been run to validate the $\alpha\Psi$NKS. The test cases are summarised in table 4.4. In the table, `N_vertices` and `N_ele` refer, respectively, to the total number of vertices and elements of the computational grid, while `N_unks` is the number of unknowns of the linear system (before adaptation, if performed). All computations have been conducted on an SGI Origin 3800 computer, whose characteristics are outlined in table 4.3.

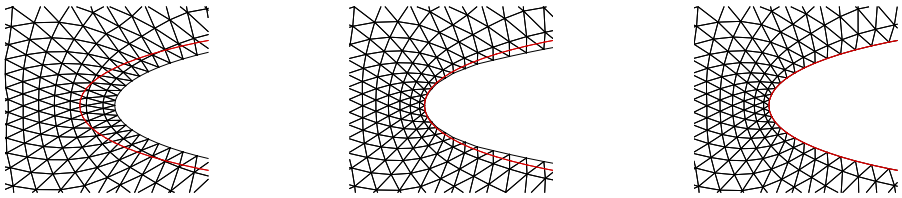
The CEE solver we have adopted is THOR. This parallel code for the solution of the compressible Euler (and Navier-Stokes) equations has been developed at the von Karman Institute (Belgium) by the group of Prof. H. Deconinck. THOR is written mainly by E. van der Weide and K. Sermeus. The code, common basis of the IDeMAS project, uses MURD schemes for the discretisation of the steady CEE, with the elliptic/hyperbolic preconditioning detailed in section 4.3.1. A continuation technique is used to find the solution of the steady equations, as explained in section 4.4.

THOR has been extended by the author of this thesis to include two-level DD precondi-



a) Initial mesh b) $\alpha_{i,j} = 1$ c) $\alpha_{i,j} = f(\mathcal{N}_j)$

Figure 4.23: Influence of $\alpha_{i,j}$ on the smoothed mesh.



no boundary spring filter too high correct

Figure 4.24: Effect of boundary spring.

tioners and SC-based solvers and preconditioners, whose parallel issues will be addressed in chapter 5 (see also remark 5.3.4). The adaptivity module of THOR described in section 4.5.1 was developed by P. Leyland and Y. Savoy at STI/LIN/EPFL during the IDeMAS project, and integrated in a common framework with the work on DD preconditioners by the author of this thesis together with P. Leyland and F. Moussaoui. The grid and the definition of the boundary conditions have been kindly furnished by the von Karman Institute (`NACA0012` and `M6`), Dassault Aviation (`FALCON`), and P. Leyland (`X29`).

A special comment is in order for the 3D visualisations. Once the numerical solution is computed, several interesting quantities (such as pressure distribution or Mach number contours) are firstly visualised on the border of the domain – i.e. the considered aeronautical body. Then, to have a better insight of the flow field, internal values are analysed. This task is not trivial for complex 3D configurations, and it must be driven by a physical knowledge of the problem, by experimental data, or by automatic algorithms capable to capture interesting flow features. We have mainly used streamlines and cutting planes, as reported in [QSS+03]. There, the authors have shown that appropriate visualisation tools for complex simulations can greatly help in both monitoring the solution and analysing it. Streamlines, iso-surfaces and particle tracking can provide a global understanding of the flow field that may be of paramount importance during the whole simulation process. To visualise the solution, we

Machine name	SGI-Origin 3800
Number of processors	128
Processors	MIPS R14000
MHz	500
RAM / processor	512 Mbytes
first level cache	32 Kbytes
second level cache	8 Mbytes
communication	MPI

Table 4.3: Characteristics of the SGI Origin 3800 used in our computations.

name	d	M_∞	α	N_vertices	N_ele	N_unks	adaptation
NACA0012	2	0.85	1.0	2355	9423	9420	yes
FALCON	3	0.45	1.0	45387	255944	226935	no
M6	3	0.84	3.06	94493	666569	472465	yes
X29	3	0.75	5.00	136767	683835	726713	yes

Table 4.4: Main characteristics of the test cases.

have used VISUAL3 by B. Haimes [Hai91] and TECPLOT (R).

For all the test cases, the starting solution is a constant solution. For first-order schemes, we have used the N-scheme for systems for the elliptic part and the scalar N-scheme for the hyperbolic part, while for second-order schemes we have used the system LW-scheme for the elliptic part and the PSI-scheme for the hyperbolic part. As regards the CFL number, we have used

$$CFL_k = \min\{CFL_0 \times \sigma^k, CFL_{max}\}, \tag{4.44}$$

where CFL_k is the CFL number at the $k-$time level.

NACA0012 is a transonic 2D test case. Figure 4.25 reports the complete $\alpha\Psi$NKS cycle, which is as follows. First, first-order N-schemes are used on the original, non-adapted grid, followed on the same grid by a second order PSI/LW scheme. Then, several adaptation cycles follows. The residual, based on second-order schemes, is reduced up to a factor of about 10^{-7}. From figure 4.26, it is possible to observe the improvements in solution quality induced by the use of second order schemes and adaptation techniques.

FALCON 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 rather coarse, and we have used this test case to validate the Krylov accelerators and the basic properties of the preconditioners. 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

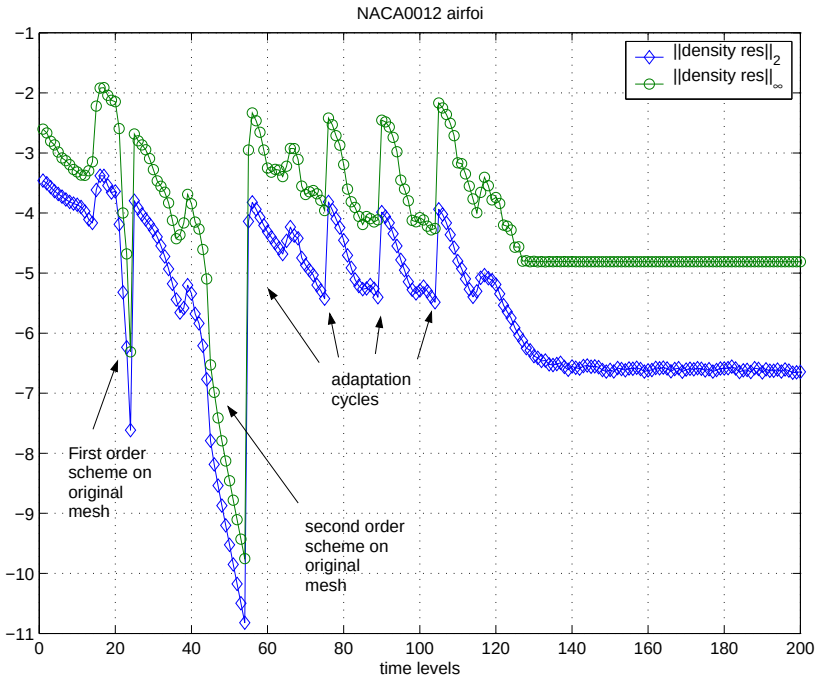


Figure 4.25: `NACA0012` test case. Convergence history for the complete $\alpha\Psi$NKS cycle. On the $x-$axis, we have reported the time levels, while on the $y-$axis the L^2 and L^∞ norm of the density residual.

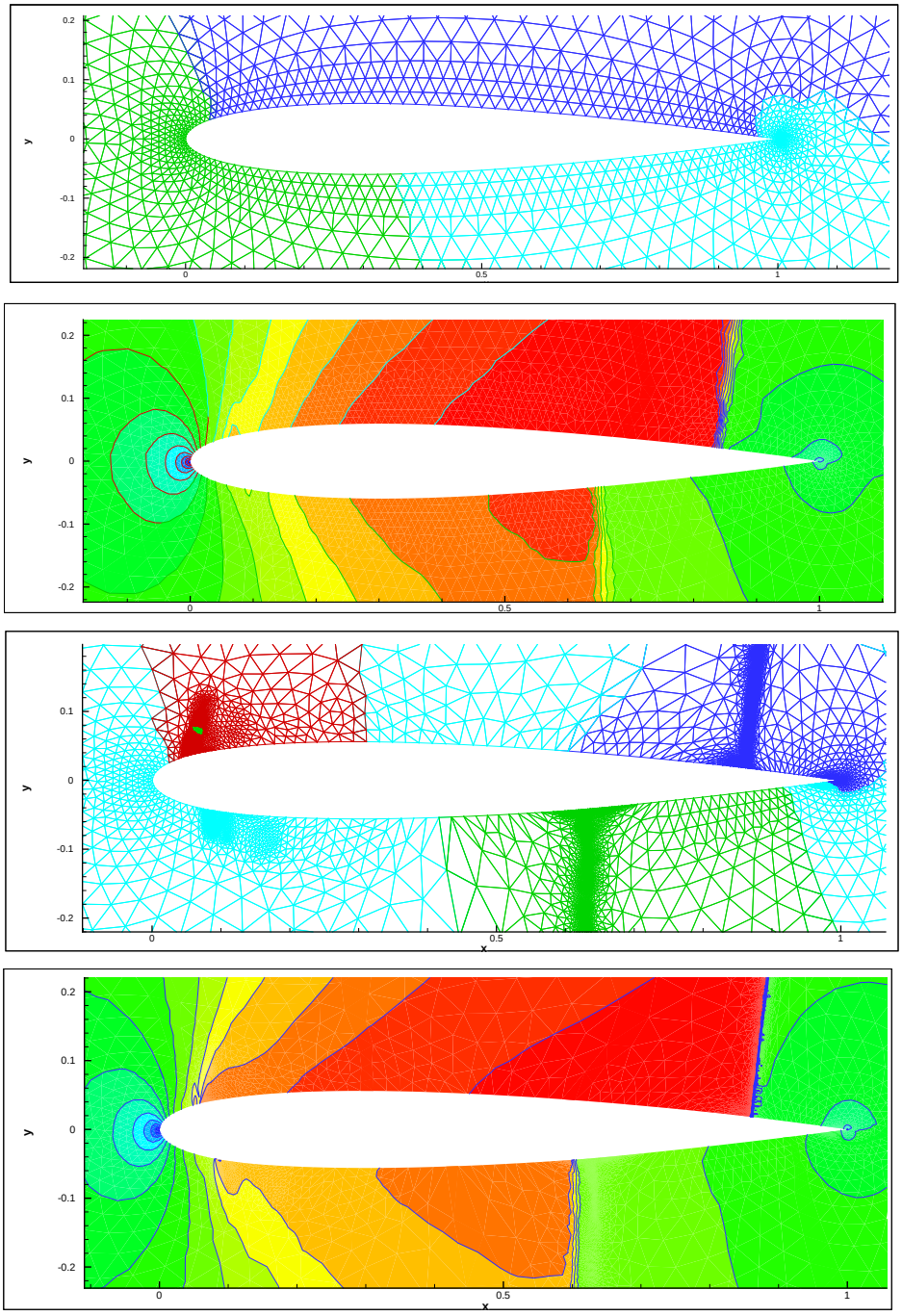


Figure 4.26: `NACA0012` test case. From top to bottom, the original grid, the first-order solution on the original grid (Mach number contours), the adapted grid, and the second-order solution on the adapted grid (Mach number contours). The different colours in the grid refer to the subdomain partitioning.

sphere, where we have imposed far-field and symmetry-plane boundary conditions. Referring to equation (4.44), we have used $CFL_0 = 1.0$, $\sigma = 2$, and $CFL_{max} = 10^6$. The numerical results using a first-order scheme are reported in figure 4.27.

`X29` is a very interesting transonic test case. 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^o, and with canards just behind the cockpit. Two airplanes were built to determine the effectiveness of forward swept wings, as well as advanced avionics and structural composites. A picture of the airplane is reported in figure 4.28, while a close-up of the computational grid is given in figure 4.29.

Equation (4.44) is used, with $CFL_0 = 0.2$, $\sigma = 2$, $CFL_{max} = 1000$. The complete $\alpha\Psi$NKS cycle has been applied. Starting from an initial grid $\mathcal{T}^{(0)}$, 2 stes of refinement and derefinement, and structural optimisation gave the mesh $\mathcal{T}^{(1)}$. Then, smoothing optimisation is performed on $\mathcal{T}^{(1)}$ to obtain $\mathcal{T}^{(2)}$. On $\mathcal{T}^{(2)}$ refinement and optimisation gave a final mesh $\mathcal{T}^{(3)}$ made up of 1.47 million nodes and 8.7 million elements.

The effect of smoothing optimisation can be seen by superposing $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$ in figures 4.30 and 4.31. Finally, figure 4.32 details the solutions on these first refinements, and further details of the refinement on the wing surface are shown in figures 4.33. Note that the shocks on the wing surface are only clearly visible when the refinement is carried out. Mach number contours on the surface and streamlines for the grid $\mathcal{T}^{(3)}$ are reported in figure 4.34.

`M6` is the last test case we have considered. It represents a ONERA M6 wing, a widely used test case for 3D flows from subsonic to transonic regimes. The selected transonic case if $M_\infty = 0.84$ 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. Contours of Mach number in critical areas of the flow field provide useful information on the flow characteristics. Iso-surfaces of Mach number over a wing are shown in figure 4.36. The red zone on the upper part of the wing represents a supersonic area. The sudden transition from red to green indicates the presence of a strong shock wave; moreover, the concentration of iso-surfaces behind it reveals that a further (weaker) shock wave develops.

The far-field is half a sphere with a radius of 12.5 root-chord length. Two steps of mesh adaptation are performed. Each adaptation phase corresponds to at least two steps of adaptation (refinement coarsening and optimisation). The final grid is composed by 585725 nodes and 3477090 elements.

As one can seen from figure 4.35, first-order N-scheme rapidly converges to steady-state solution in very rapidly. In this phase, $CFL_0 = 10, \sigma = 2.0, CFL_{max} = 10^6$, 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 $CFL_0 = 0.2, \sigma = 2.0, CFL_{max} = 1000$.

As already reported by [vdW98] and, more recently, by [Abg01], the iterative convergence

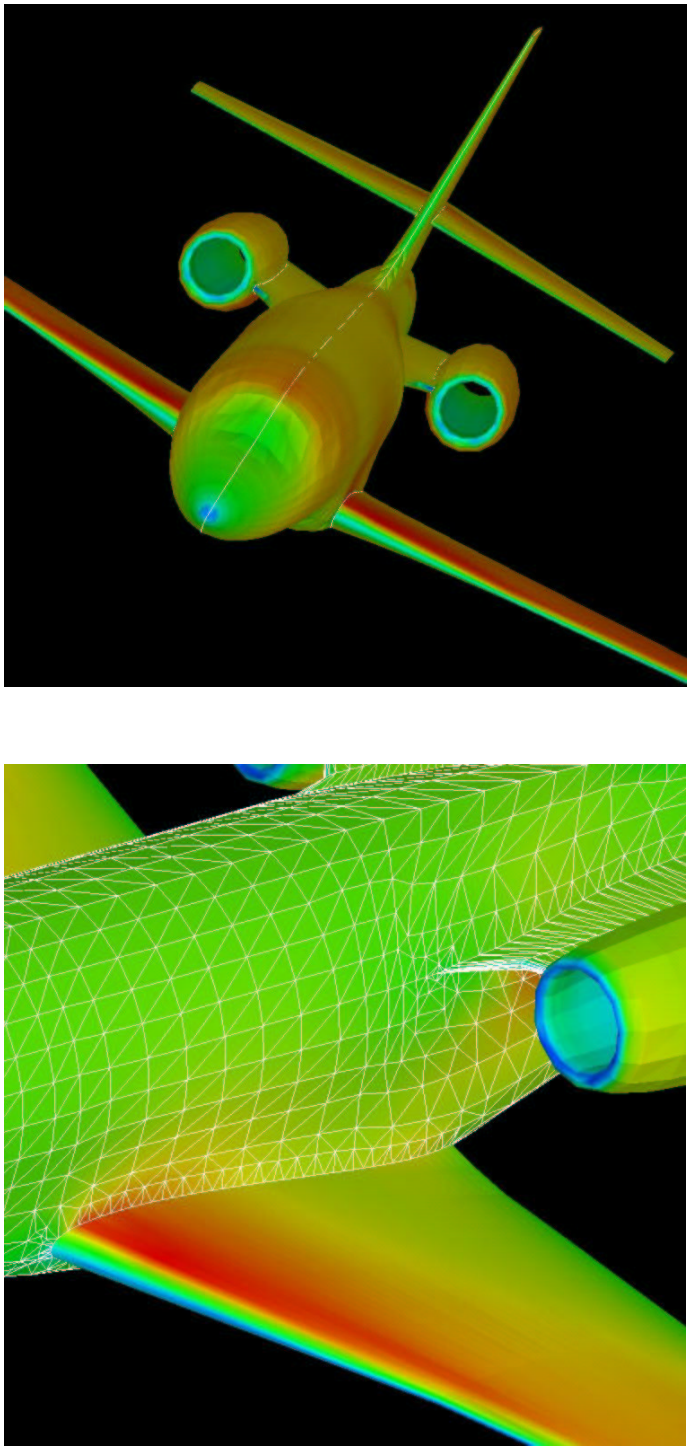


Figure 4.27: `FALCON` test case. Mach number contours on the airplane (top) and particular of the surface mesh (bottom).

Figure 4.28: `X29`. Picture of the X29 aircraft.

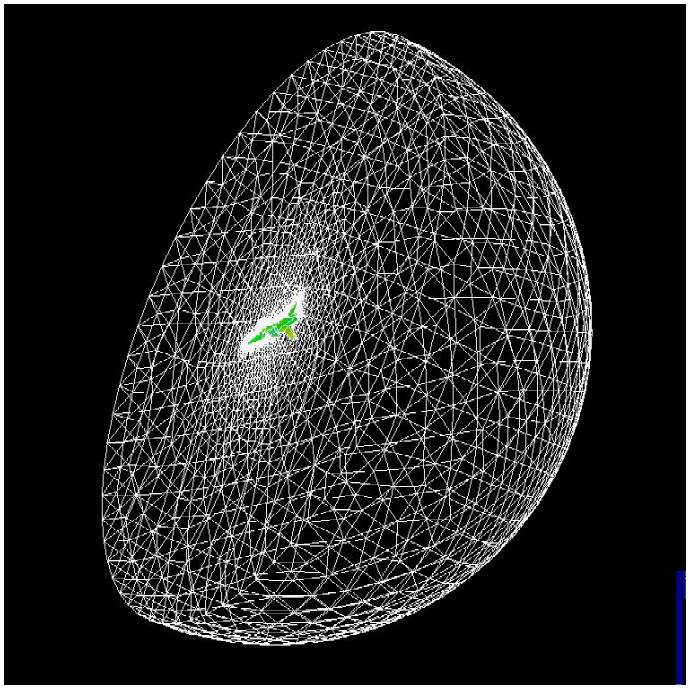


Figure 4.29: `X29`. Mesh used in the computations. The mesh is composed by half a sphere since the problem and the boundary conditions are symmetric.

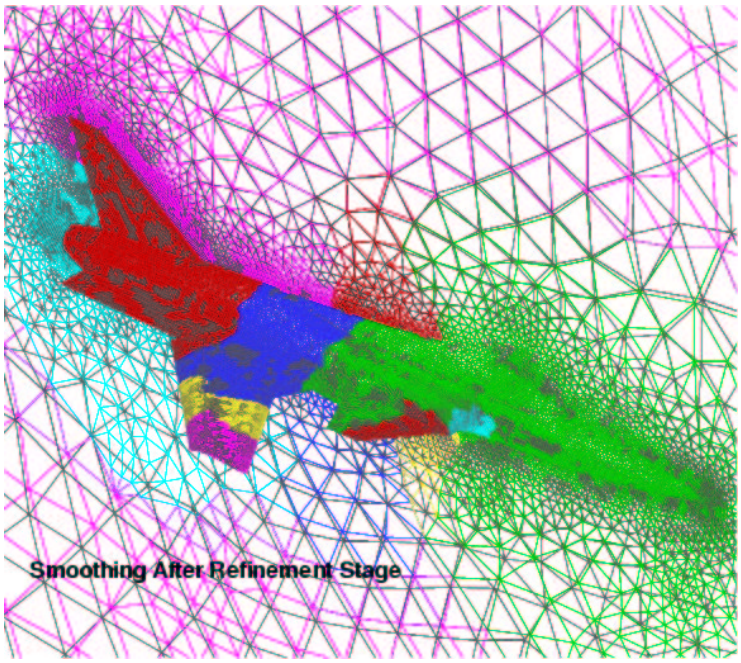


Figure 4.30: `X29`. Superposition of the symmetry plane meshes before (black lines) and after the first refinement (coloured by processors).

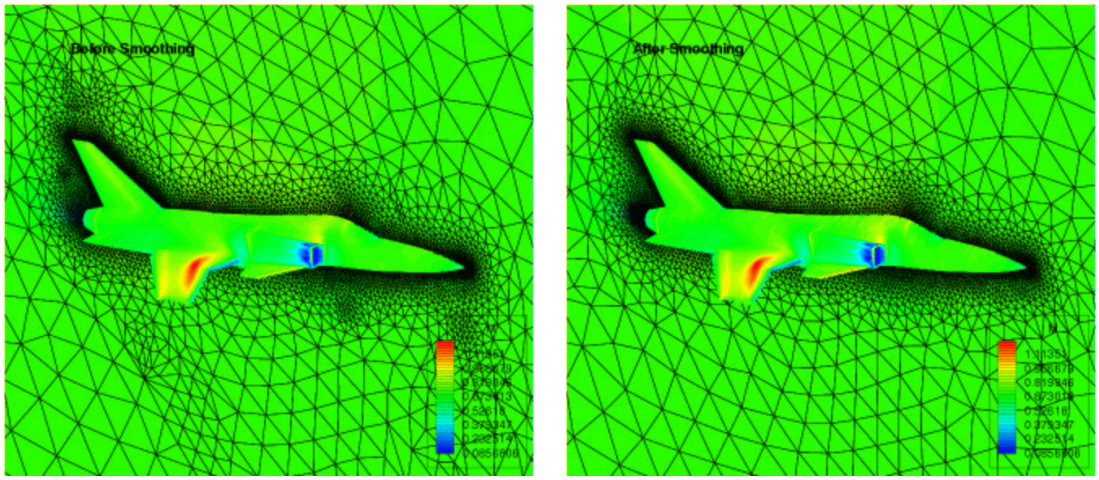


Figure 4.31: `X29`. Results obtained before and after mesh optimisation and smoothing. The non-smoothed and non-optimised mesh (on the left)shows, on the symmetry plane, elements with a number of neighbouring elements larger than the optimal one. This is no longer the case for the smoothed and optimised mesh (on the right).

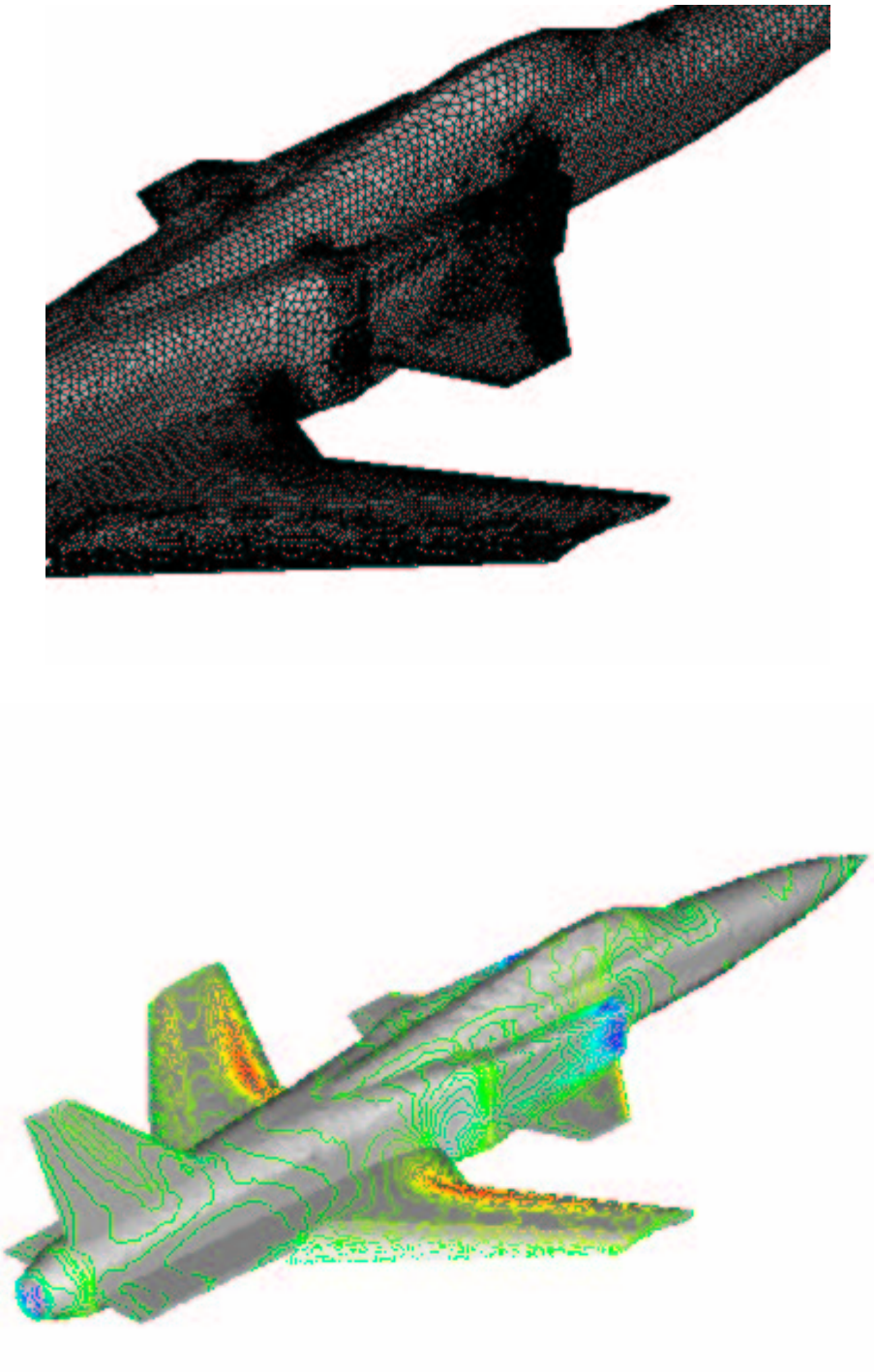


Figure 4.32: `X29`. Zoom over part of the refinement (top) and Mach number distribution on the body for the mesh $\mathcal{T}^{(1)}$ (bottom).

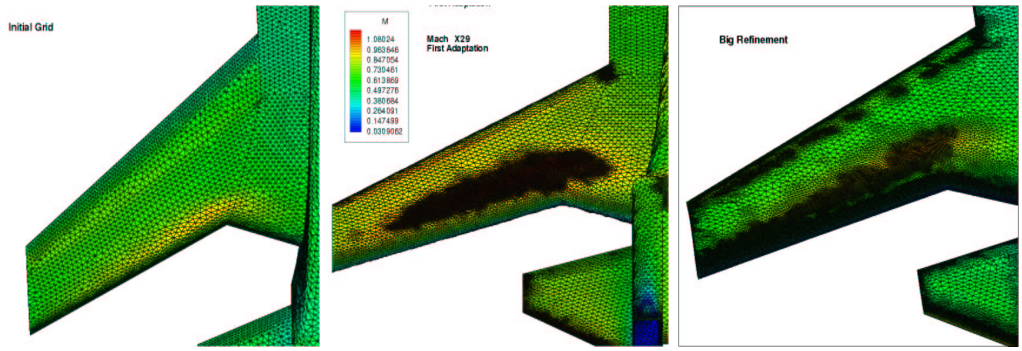


Figure 4.33: `X29`. Refinement criteria taken here are edge length and Mach gradients. The initial mesh $\mathcal{T}^{(0)}$ (on the left) steps by an intermediate refinement which detects the wing shocks (in the centre), and a final mesh $\mathcal{T}^{(3)}$ is obtained by leading and trailing edge refinement (on the right).

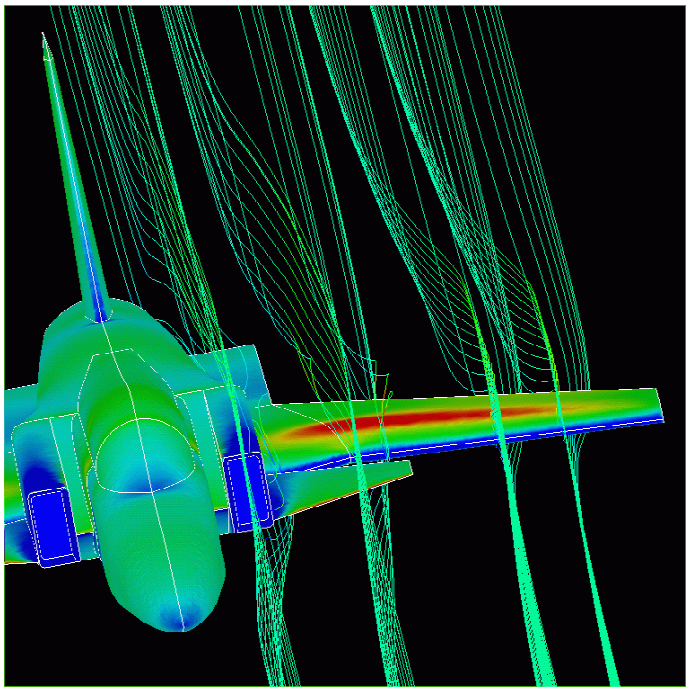


Figure 4.34: `X29`. Streamlines and Mach number contours on the airplane.

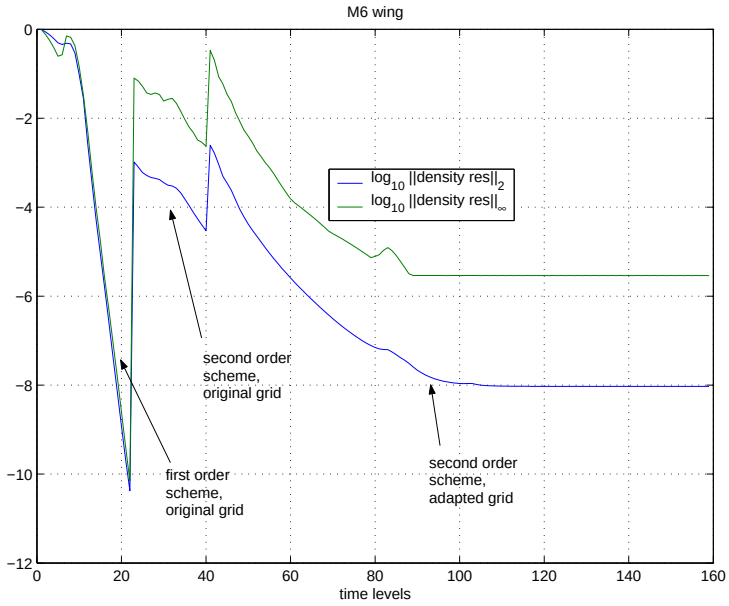


Figure 4.35: `M6`. Convergence history of the $\alpha\Psi$NKS cycke. On the $x-$axis, we have reported the time levels, while on the $y-$axis the L^2 and L^∞ norm of the density residual.

of second-order schemes is very poor; basically, for non-adapted meshes the L^2 density residual stagnate at about $10^{-2} \div 10^{-3}$, 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^{-8} with respect to the first iteration on the non-adapted mesh.

Mach number isolines over the wing and the symmetry plane are depicted in figure 4.37 for the first-order non-adapted grid, and in figure 4.38 for the second-order adapted grid.

Experimental surface pressure distributions at several spanwise cross-section are available [AGA79]. Figure 4.39 shows the comparison between the experiments and the computations of the CEE. Due to the absence of shocks,on the pressure side, the pressure distribution on this part of the wing is mainly determined by inviscid phenomena. Consequently, the results of the computations and experiments are almost identical. On the suction side, the prediction of the location of the first shock is rather good. The second shock, instead, in computed far downstream and therefore it is too strong. This is a consequence of neglecting the viscous term (the Reynolds number based on the mean aerodynamic chord is 11.74×10^6). Taking into account that the computation is inviscid, the agreement between computations and experiments is quite good.

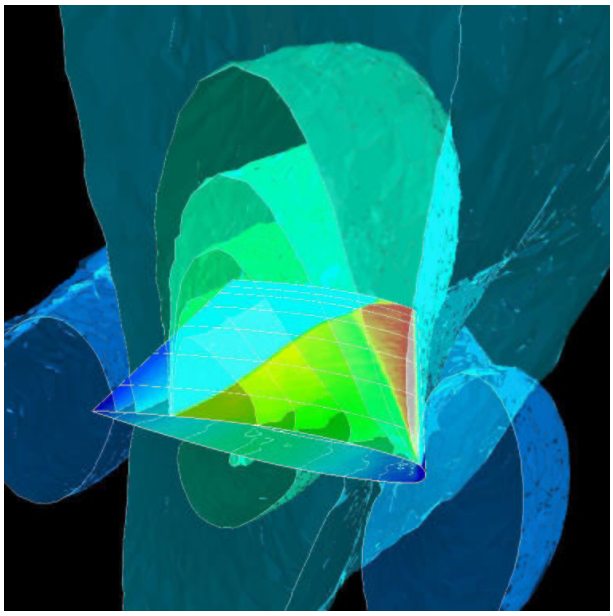


Figure 4.36: `M6`. Plot of Mach number contours on the wing and Mach number iso-surfaces. The asymptotic Mach number is 0.84 and the angle of attach 3.06.

4.7 Concluding Remarks

MURD schemes have reached a certain degree of maturity for the solution of the steady CEE on unstructured grids made up of simplices. In this chapter we have briefly recalled the main lines of MURD schemes, focusing with particular attention to their FE interpretation.

In their current stage of development, MURD schemes can be used for complex aerodynamic flows, both in the subsonic and transonic regime. However, it is clear that the full potential of this approach may be revealed only if these methods could effectively lead to scalable parallel codes. Moreover, mesh adaptation techniques are of paramount importance to capture the flow physics, especially when strong discontinuities (like shock waves) arise in the domain.

To that aim, we have presented a fully parallel algorithm, here called $\alpha\Psi$NKS, which couples a non-linear solver, a Krylov accelerator, DD preconditioners of both Schwarz and SC-based type, and mesh adaptation procedures. The numerical results we have reported show that this process can effectively improve the quality of the solution obtained using MURD schemes for problems of aeronautical interests.

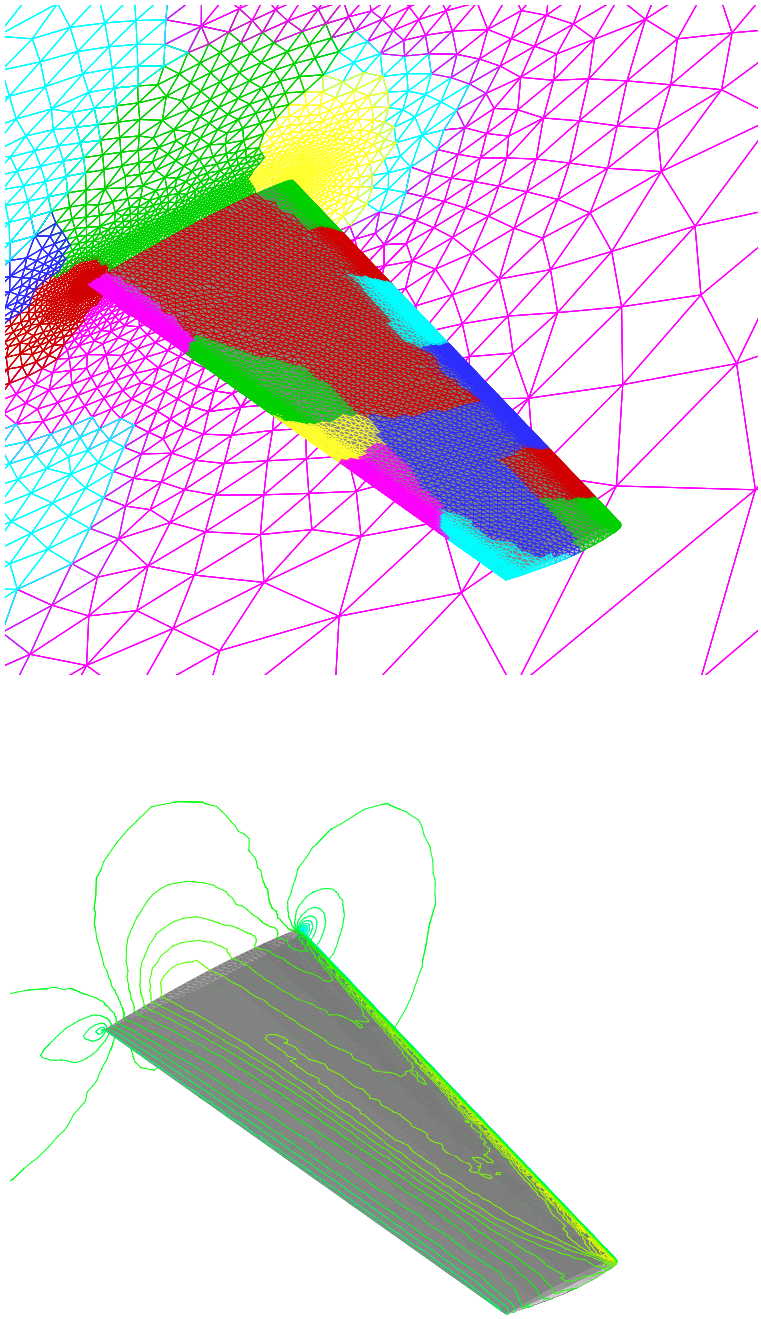


Figure 4.37: ONERA M6 wing. Partition and solution (iso Mach lines) with the initial mesh, composed by 555514 elements and 94493 nodes. Note that the wing shock is not captured since only first-order schemes are used.

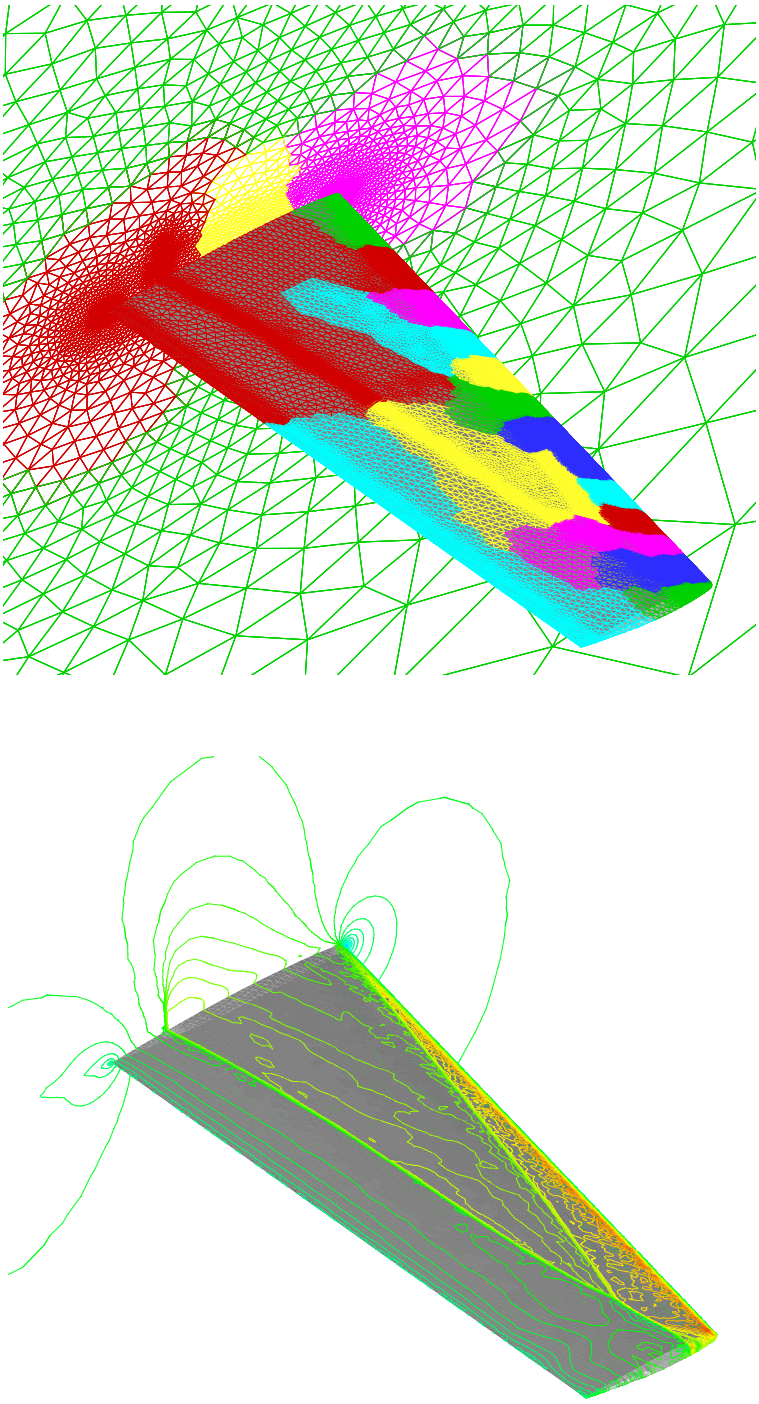


Figure 4.38: ONERA M6 wing. Partition and solution (iso Mach lines) with the final grid, composed by 3477090 elements and 585725 nodes. The PSI/LW scheme is used. Note that the lambda shock is very well captured.

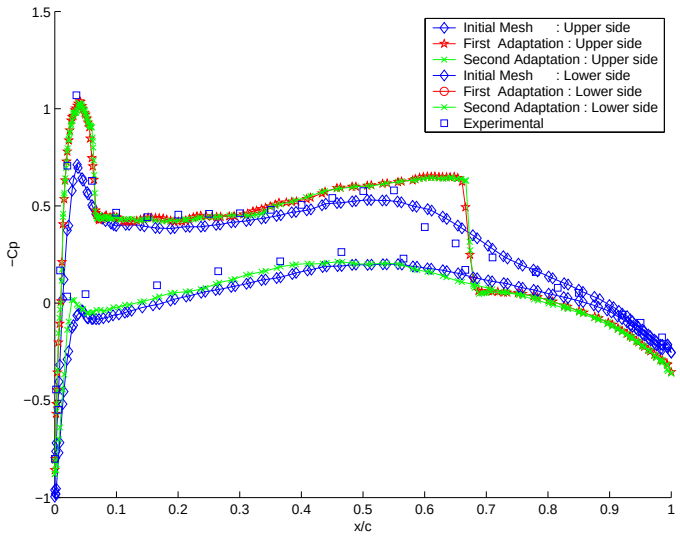


Figure 4.39: ONERA M6 wing. Pressure coefficient comparison for the different adaptations taken at a station at 20% of the wing span.

5 Parallel Aspects

THE DEVELOPMENT OF a scalable parallel $\alpha\Psi$NKS code requires the definition of an efficient distributed data structure. This data structure must efficiently handles 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.

The actual implementation claims for a in-depth analysis of many implementation issues, like memory organisation, communication protocols, and parallel architecture.

This chapter is organised as follows. In section 5.1 we present a brief introduction of the basic aspects of parallel computing. Section 5.2 covers the first step required by a parallel code: subdomains partitioning. The main computational kernels of a $\alpha\Psi$NKS solver are figured out in section 5.3, focusing in particular on the preconditioners presented in chapter 3. Parallel issues of mesh adaptation techniques are covered in section 5.3.8. Numerical results are reported in section 5.4 to validate most of the DD preconditioners presented in chapter 3. Finally, in section 5.5 we draw some concluding remarks.

5.1 Basic Aspects of Parallel Computing

The definition of an efficient parallel algorithm requires a model of the architecture one is actually using. This model should represent the main features that are supposed to influence the performances of the algorithm: mainly, the memory organisation and the ratio communication/computation. This analysis which is important to minimise communication, can be done resorting to the different taxonomies for parallel computing presented in literature, as done in section 5.1.1. Then, another model is required to understand how to optimise com-

munication that cannot be avoided. This is the subject of section 5.1.2. Finally, programming paradigms are addressed in section 5.1.3.

5.1.1 Taxonomies for Parallel Computers

A first taxonomy divides computers into three categories: *single instruction single data stream* (SISD), *single instruction multiple data stream* (SIMD), *multiple instruction multiple data stream* (MIMD). The SISD model takes a single sequence of instructions and operates on a single sequence of data. The speed of SISD is limited by the execution of a single sequence of instruction and the speed at which information is exchanged between memory and CPU. Information exchange can be increased by the implementation of cache memory and the execution rate can be improved by the execution pipelining techniques.

A SISD architecture is a sequential computer, while SIMD and MIMD architectures both belong to the parallel computer category. A SIMD architecture system has a single control unit furnishing instruction to each processing element in the system. The SIMD architecture is capable of applying exactly the same instruction stream to multiple streams of data simultaneously. For certain classes of problems this architecture is perfectly suited to achieve very high processing rates. This requires data to be split into many different independent pieces, so that multiple instruction units can operate on them at the same time. SIMD architectures proved to be too inflexible; consequenly, they are now used only for very specific applications.

A MIMD architecture system has each control unit in each processing element in such a way that each processor is capable of executing a different program independent of the others in the system. A MIMD architecture is capable of running in true "multiple-instruction" mode, with every processor doing something different, or every processor can be given the same code. The latter case is sometimes called Single Program Multiple Data Stream (SPMD). SPMD architecture is a generalisation of SIMD with much less strict synchronisation requirements.

A second taxonomy is based on *granularity*, i.e., the ratio of the time required for a basic communication operation to the time required for a basic computation. Parallel computers with small granularity are suitable for algorithms requiring frequent communication and the ones with large granularity are suitable for algorithms that do not require frequent communication. The granularity of a parallel computer may also be defined by the number or processors and the speed of each individual processor in the system. Computers with large number of less powerful processors will have small granularity and are called fine-grain computers or Massively Parallel Processors (MPP). In contrast computers with small number of very powerful processors have large granularity and are called coarse-grain computers or multicomputers.

In the middle of the 80's MPP computers were predicted to be the next big advance in high performance computing. Many companies were formed to build MPP systems, like nCUBE and Thinking Machine. In addition, several existing companies like INTEL, IBM, and Cray Research decided to build these systems. However, nowadays only few centres host and currently use MPP systems. Industry is more oriented to Symmetric Multi Processors

(SMP) systems. An SMP system is a coarse-grain parallel computer. To reduce price, they are usually composed of commodity processors. In addition to SMP systems, the so-called Commercial Off-the-Shelf (COTS) systems are increasingly adopted by many institutions, even if these systems are not really a single parallel computer but a collection of systems. Both SMP and COTS systems belong to the class of MIMD systems, with a moderate number of high-performance (and relative low price) processors.

More recently, research is focusing on NOW (Network Of Workstation) architectures, and its generalisation, COMPS (Cluster Of Multi-Processor Systems). Among the NOW systems, an example is represented by the Beowulf system. Is consists of a cluster of PCs or workstations dedicated to running high-performance computing tasks. Instead, a COMPS is a network of multiprocessor computers, and it differs from NOW in that it links multiprocessor workstations. The last frontier is now represented by *grid computing*. A *grid* is a collection of distributed computing resources available over a local or wide area network that appear to an end user or application as one large virtual computing system. For high-performance computing, the present feeling is that NOW, COMPS and grid computing are still not competitive with SMP systems.

A third taxonomy is based on memory organisation, and distinguishes parallel computers between *shared memory* systems and *distributed memory* systems. Shared memory systems were extensively used in the beginning of the 90's. A parallel code running on a shared memory system uses more processors (even if generally few), all sharing the same main memory. The CRAY X-MP, Y-MP, C-90, the IBM 3090-600, ALLIANT FX/80, the NEC SX/4 and SX/5, and the CONVEX C3800 belong to this class.

Using shared memory systems, the communication between processors is implicit and transparent. Processors access memory through the shared bus; see the picture on the left of figure 5.1. Processors do not explicitly communicate with each other so communication protocols are hidden within the system. This means communication can be close to the hardware (or built into the processor), with the shared bus system deciding on how most efficiently to manage communication. Therefore, a serial code can run on these systems with no modification (although maybe performances will be poor).

On the contrary, in distributed memory systems the processors must explicitly communicate with each other through messages. See the picture on the right of figure 5.1 for a graphical representation. The resulting parallel computer has a "simple" architecture overall, since it is composed by local components, connected by a network. The construction of the network (and the definition of a software library to access it) becomes the key to success in the definition of the parallel computer.

Because of cost reasons, industry has now turned the attention to distributed memory systems. The Connection Machine, the INTEL iPSC860 and Paragon, the IBM SP/2, the SGI Origin series, and the ASCI project supercomputers are example of this class of parallel machines. Here every processor has its own local memory and data is communicated among the processors via communication network. Due to the explicit interface to communication,

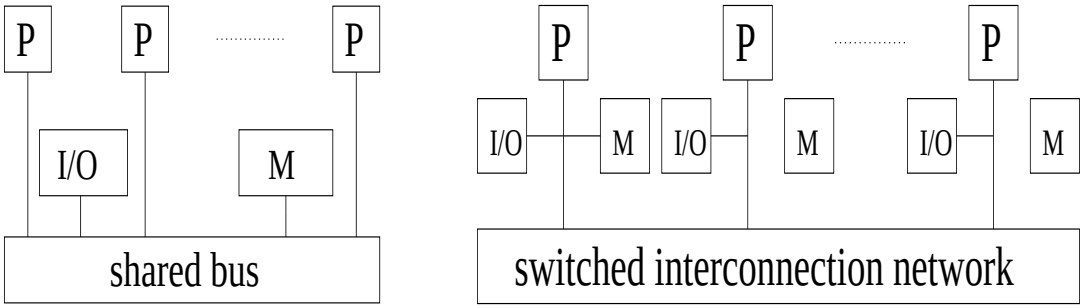


Figure 5.1: Shared memory systems (on the left) and distributed memory systems (on the right). **P** represents the CPU unit, **M** the memory, **I/O** a generic input/output systems.

distributed memory systems are more scalable in the sense that hundreds or even thousands of identical processors can work on one problem, without the need of an (expensive) global memory. Moreover, the distributed memory eliminates the cache coherency problems that are typical of the shared memory systems. The price to pay is a more aggressive redesign of the algorithm the code is actually running.

We conclude this section with few references on the subject. More details about the classification of parallel computers and a recent overview of the history of high-performance computing may be found, for instance, in [WDH+99, SDMS99]. The book by Hockney and Reship [HJ88] is a valid reference for vector and parallel computers till the end of the 1980's, while the book by Ortega [Ort88] is a good mix between implementation details and mathematical aspects of vector and parallel computers.

5.1.2 Cost of Communication Among Processors

We now discuss communication in more details, following [Ort88]. The typical communication problem is to send n words from the local memory of one processor, say P_1, to the memory of another processor, say P_2. On a distributed memory system this communication will be accomplished by a combination of hardware and software. First, the data are loaded into a buffer memory or collected into contiguous locations in memory. Then, a *send* command will be executed on P_1, while P_2 will execute a *receive* command and the data will be routed to their final destination in the memory of P_2. Although many technical details have been omitted in the previous description, the main points of communications are clear:

1. data are collected from the memory of the sending processor;

2. data are physically transfered over the network;

3. the received information are put in the correct memory location of the receiving processor.

Usually, the time t_n required for these 3 phases to send n words is approximately given by

$$t_n = s + \alpha\, n, \tag{5.1}$$

where s is the *startup time* and α is the *incremental time* necessary for each of the n words to be sent. In nowadays implementations, s is generally several order of magnitude greater than α.

This simple – although effective – model can furnish a guideline in the development of parallel algorithms: it is better to have an algorithm which groups communication, that is, sent many data few times, instead of one which requires many send/receive with a small amount of carried data. The number of processors that a given processor has to communicate with, is also to be minimised, since it contributes to additional communication time. This is because each time a processor has to communicate with a processor, a latency penalty for starting a new message is incurred.

Using equation (5.1), the parallel performances of simple algorithms can be studied to optimise the algorithms themselves. Instead, the parallel performances of complex algorithms are usually analysed looking to the global behaviour of a code implementing the algorithm, as the number of processors varies. This global evaluation considers the following parameters:

- *elapsed time* T_p, that is the CPU-time needed to carry out the algorithm using p processors;

- *speedup* s_p, defined as

$$s_p = \frac{\text{execution time using a single processor}}{\text{execution time using } p \text{ processors}} = \frac{T_1}{T_p}.$$

  One can also define the speedup of a parallel algorithm over the best serial algorithm:

$$s'_p = \frac{\text{execution time using a single processor for the fast serial algorithm}}{\text{execution time using } p \text{ processors}} = \frac{T'_1}{T_p};$$

- *relative speedup* $s_{p,q}$, defined as the ratio

$$s_{p,q} = \frac{T_q \times q}{T_p} \tag{5.2}$$

  where $T_q \approx T_1/q$ is a reference time using q processors. In fact, often s_p turns out to be hard to compute accurately. Problems that can run on a large number of processors may not fit the memory of a single processor, so that the time for one processor would have to include I/O time to secondary memory – typically, the one of another processor. In this case, q represents the minimum number of processors which can run the defined problem without requiring secondary memory.

- *efficiency* e_p, defined as

$$e_p = T_1/(pT_p) = s_p/p.$$

  It is also possible to define the efficiency of an algorithm with respect to the best serial algorithm as $e'_p = s'_p/p$.

One goal in the development of a parallel algorithm is to achieve as large a speedup as possible. For perfectly parallel algorithms we would expect an *ideal* speedup $s_p = s'_p = p$ (that is, using p processors we can reduce the elapsed time by a factor p). In practise, this value cannot be achieved unless the algorithm does not require communications, or the ratio between communication and computations is extremely low (the so-called *embarassing parallelism*). In general, an efficient parallel algorithm will present speedup values close enough to the ideal ones.

Remark 5.1.1. *Sometimes* superlinear *speedups may be observed, that is,* $s_p > p$ *or* $s'_p > p$. *This is due to non-linear effects, like, for instance, the cache reuse: as the dimension of the problem to be solved diminishes, the code can use in a more efficient way the values in the high-performance cache memory, therefore resulting in a consistently lower CPU time.*

The degradation of the speedup value is mainly due to the following factors:

1. lack of perfect degree of parallelism in the algorithm and/or of load balance. Load balance means the assignment of tasks to processors of the system so as to keep each processor doing useful work as much as possible. Load balance may be done either *statically* or *dynamically*. In static load balance tasks (and data) are assigned to processors at the beginning of computation. In dynamic load balance tasks (and data) are assigned to processors as computation proceeds. The implementation of a dynamic load balance algorithm may be difficult on distributed memory systems since data transfer between local memories is in general required as a part of a task assignment;

2. communication, contention, and synchronisation time. We have already discussed about communication. Data contention mainly arises in shared memory computers. Synchronisation is necessary when certain parts of computation must be completed before the overall computation can proceed. There are two aspects of synchronisation that contribute to the overhead. The first is time required to do the synchronisation. The second aspect is that some processors may become idle. Communication, contention, and synchronisation time are usually referred to as *data ready time*.

All these factors may be grouped in a model for the speedup, which reads:

$$s_p = \frac{T_1}{(\alpha_1 + \alpha_2/k + \alpha/p)T_1 + t_d}.$$

Here T_1 is the time for a single processor, α_1 is the fraction of operations done with one processor, α_2 the fraction of operations done with k processors ($k < p$), α if the fraction of operations done with degree of parallelism p, and t_d is data ready time. Clearly, $\alpha+\alpha_1+\alpha_2 = 1$.

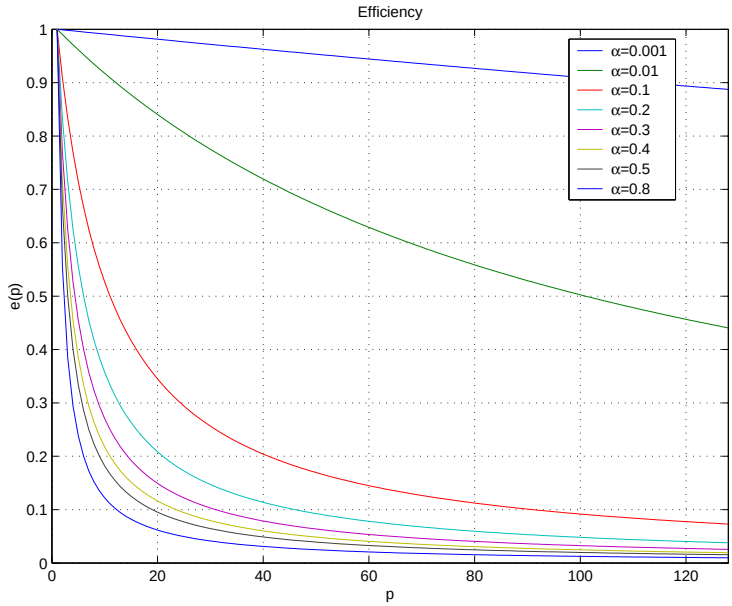


Figure 5.2: Efficiency e_p for various values of α.

A special (simple) case can be obtained considering $\alpha_2 = 0$ (that is, all the operations have the maximum degree of parallelism) and with no delays ($t_d = 0$). In this case, we obtain

$$s_p = \frac{T_1}{(1 - \alpha + \frac{\alpha}{p})T_1} = \frac{1}{1 - \alpha + \frac{\alpha}{p}} \tag{5.3}$$

and

$$e_p = \frac{\frac{1}{p}}{\frac{\alpha}{p} + (1 - \alpha)} = \frac{1}{\alpha + (1 - \alpha)p}. \tag{5.4}$$

Expression (5.3) is known as the *Amdahl's law* or *Ware's law*. Although very simple, (5.3) is instructive. Consider, for example, $\alpha = 1/2$. Then,

$$s_p = \frac{2}{1 + p} < 2,$$

that is, no matter how many processors there are, and ignoring all communication, delays, and synchronisation problems, the speedup is always less than 2. Hence, a "good" value of α is of fundamental importance to obtain interesting parallel properties. Figure 5.2 reports the behaviour of e_p with respect to p for several values of α.

5.1.3 Programming on MIMD Computers

Parallel programming is more challenging than serial programming for various reasons. First, parallel programs must include mechanisms for data exchange. Second, in an efficient parallel

program the work must be evenly divided among processors. This is an algorithmic problem that has no counterpart in the serial programming. And finally, the data structures must be divided among the processors to preserve data locality. This is obviously true for distributed memory systems in which data movement is costly. It is also true for shared memory systems since data locality reduces the cost of maintaining cache coherence, hence resulting in a faster implementation of the algorithm.

To help the programmer in developing parallel codes, several different parallel methodologies have been pursued. In the class of MIMD systems, there are different aspects for developing a parallel program, one more suited for shared memory and the other for distributed memory systems. In shared memory programming, programmers view their programs as a collection of processes accessing a central pool of shared variables. In the message passing programming, programmers view their programs as a collection of processes with private local variables and the ability to send and receive data between processes by passing messages. On one side, we have the *implicit parallelisation*, automatically performed by the machine (and the compiler); on the other side we find the *explicit parallelisation*, that requires the users to take control of all the parallel aspects.

For shared memory computers, one can exploit the parallel programming using threads. The parallel code is composed by many independent processors, thus manipulating data stocked in a common area. Each thread is given to a different processor. A more recent approach for the programming of SMP is the standard OpenMP. It consists in a set of procedures and directives that are added to the (sequential) source code to aid the compiler to build up the parallel code. In this case, the aim is to produce parallel code starting from the sequential one. Some compiler directives are added to help the compiler in detecting the parallel cycles and the optimal distribution of data among the processors. An example of this approach is High Performance FORTRAN. This approach is very practical for the programmer, and it can work well for problems with a regular structure. Instead, it is in general quite difficult to apply for unstructured data and for a large number of processors. This approach is somehow less common than others, and in general the automatic parallelisation is more effective on shared memory than on distributed memory architectures. Moreover, the best results are obtained for codes lying on cycles whose parallelism does not rely on the input data.

For distributed memory computer, instead, the interaction among the processor is made up using messages. The parallel program is organised as a collection of processes running on a certain number of processors, each of them owning its private local memory. Data exchange is performed only by means of explicit send and receive operations.

Message passing is often preferred to other approached because, although explicit message passing may be more complex to code with respect to other parallelisation techniques, it has a key advantage over other approaches: it focalizes the programmer's attention on the performance-critical issue of the parallel hardware – *data locality*. The programmer is forced to exploit the data structure of the problem, and to use (or develop) numerical methods which

minimise the use of non-local data in all the various phases of the code.

The two important standards for message passing are PVM and MPI [For95]. Although the first is somehow more flexible, the latter is the de facto standard for high-performance parallel computing.

PVM was mainly built around the notion of 'virtual machine' – a set of heterogeneous hosts connected by a network that appears logically to the user as a single large parallel computer. The exchange of data, or in other words, communication, is accomplished using simple message-passing constructs. The concept of portability was considered much more important than using the natural message passing constructs of the underlying hardware. This in fact has paved the way for the present large scale use of PVM.

MPI, in contrast to PVM, was designed not with portability as the main aim, but with the view of creating a standard message passing interface for high-performance parallel computing. The focus of MPI developers' team was more on performance, which they tried to improve with the natural message-constructs of the unrelying hardware. One of MPI's prime goals is to produce a system that would allow manufacturers of high performance massively parallel processing computers to provide highly optimised and efficient implementations, while PVM was designed primarily for network of workstations with the goal of portability gained at the sacrifice of optimal performance. MPI has advantages in the areas of point to point communication, collective communication, ability to specify communication topologies and ability to create derived data types. There is also a study going on for developing a system PVMPI [FD96], which uses the already proved and widely ported MPI message-passing system within PVM to enable interoperability with different implementations execution on heterogeneous distributed hardware and more recent projects to built interfaces portable to both MPI and PVM [AKK99].

5.2 Subdomain Partitioning

From the discussion of the previous section, it is clear that both load balance and data ready time are strongly affected by the way data is distributed among the processor. In a DD approach, this means that an algorithm to partition the computational domain is required, to have optimal load balance and as little communication between processors as possible. Ideally, one would like each processor to do exactly the same amount of work. That way, each processor is always working and not staying idle. The way to do this is to first determine what the basis computational task for the algorithm is. For our ΨNKS solver, the two main tasks are the matrix-vector product and the preconditioning step. Therefore, it makes sense to partition the grid such that each processor gets a (nearly) equal number of grid vertices. Moreover, the amount of communication among processors should be minimised, as well as the number of neighbouring processors.

Remark 5.2.1. *From the point of view of parallel computing, DD often means* data decomposition*, that is, the process of distributing data among the processors in a distributed memory*

computer. In fact, the cost function of memory access is composed by mildly varying plateaus separated by sharp discontinuities that correspond to changes in the memory hierarchy. Within a DD algorithm, these discontinuities can be explicitly respected by the user application by assigning data that are locally less used to the slowest memory, and the most used data to the fastest memory. As a result, the computational domain is partitioned into a set of subdomains and parallel computation is achieved by attributing each subdomain to a different processor.

During the last years, much work has been done is the area of unstructured grid partitioning. Here we review three algorithms of increasing complexity.

5.2.1 The Recursive Coordinate Bisection

The Recursive Coordinate Bisection (RCB) is the simplest and fastest partitioning strategy of the three presented here. It is based upon the ordering of the elements according to the spatial coordinates of their centroids.

Here is the principle. For a given grid, determine along which direction the bisection will operate (for example, the longest direction), say, the $x-$direction. Now, sort the elements according to the $x-$direction, and choose the median value of the $x-$components so that equal number of elements lies in each partition.

Although RCB is very simple, the partition it can produce may be disconnected, and this is not a desirable property since it will result in large communication times. Clearly, the poor parallel properties of RCB are caused by the fact that the algorithm ignores all the grid connectivity. The other two algorithms that we are about to present, instead, use the grid connectivity to obtain high-quality partitions.

5.2.2 The Recursive Graph Bisection

The basic ideas of the Recursive Graph Bisection (RGB) algorithm are as follows. The two vertices that are furthest apart graphically within the grid are chosen as the starting points for the method[1]. Actually, this is a difficult problem in graph theory, by there are fast ways to find two vertices that are approximately the furthest apart.

Now, start at one of these vertices; call it root. Then, find all the first-order vertices (see figure 5.3) to this root vertex. This set of vertices forms what is called the first level set. The next level set if found by determining all of the neighbours of the first level set that are not already part of a level set. This process continues until half of vertices are members of a level set. It turns out that RGB guarantees that one of the two subdomains produced will be connected. Note that the RGB algorithm is also called the Cuthill-McKee algorithm, and it is often used to reduce the bandwidth of sparse matrices.

[1] The graphical distance between two vertices is defined as the minimum number of edges one has to traverse to get from one vertex to the other

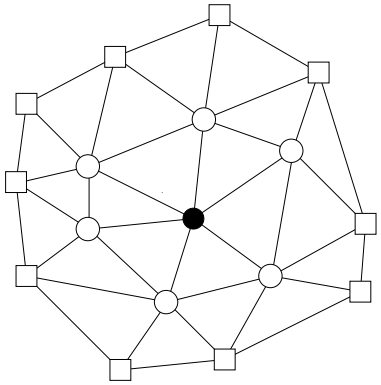


Figure 5.3: First-order nodes (indicated by '$\circ$') and second-order nodes (indicated by '$\Box$') of node '$\bullet$'.

5.2.3 The Recursive Spectral Bisection

The objective of Recursive Spectral Bisection (RSB) is to divide the graph into two parts having equal number of vertices such that the number of edges cut is approximately minimised. The key feature is its clever use of eigenstructure of what is called the Laplacian matrix of the graph.

Given a graph G with $|G| = n$ (where $|G|$ represents the number of nodes of G), the Laplacian of G is the matrix $L \in \mathbb{R}^{n \times n}$ defined as

$$L = -D + H,$$

where H is the adjacency matrix,

$$H_{i,j} = \begin{cases} 1 & \text{if edge}(v_i, v_j) \text{ belongs to } G \\ 0 & \text{otherwise,} \end{cases}$$

and D is a diagonal matrix with entries equal to the degree of each vertex, that is the number of its first-order neighbours. Clearly, the rows of L sum to zero, and therefore there is at least one zero eigenvalue. Disconnected graph may have multiple non-zero eigenvalues; however, for this discussion we assume that the graph is connected.

RSB is defined as in the following algorithm.

Algorithm 5.2.1 (2−**way recursive spectral bisection).**

a. Compute the Laplacian matrix L of the graph.
b. Determine its smallest (in magnitude) non-zero eigenvalue and the corresponding eigenvector. This is called the Fiedler vector.
c. Determine the median value of the entries in the Fiedler vector.
d. Those vertices whose Fiedler vector entry is greater than zero form one subgraph. The remaining vertices form the other subgraph.

It is interesting to see why and how the Fiedler vector plays in this procedure. The argument given below follows closely that given in [Bar94].

Define a partitioning vector $\mathbf{p}$ that assigns each vertex of the graph either a $+1$ or a -1, with $p_i = +1$ if vertex i belongs to the first partition. An equal partition of the graph, which will be assumed to have an even number of vertices, means that

$$\mathbf{s} \perp \mathbf{p},$$

where $\mathbf{s}$ is a vector of 1's.

The key observation is to notice that the number of cut edges E_c is precisely related to the L_1-norm of the Laplacian matrix multiplied by the partitioning vector,

$$4E_c = \|L\mathbf{p}\|_1.$$

Since the goal is to minimise the number of cut edges, one should find the vector $\mathbf{p} = \hat{\mathbf{p}}$ that minimises the norm $\|L\hat{\mathbf{p}}\|_1$, where $\|\hat{\mathbf{p}}\|_1 = n$, and $\mathbf{s} \perp \hat{\mathbf{p}}$. Because L is real symmetric negative semidefinite, it has a complete set of eigenvectors that can be orthogonalised with each others. Therefore, one can write $\hat{\mathbf{p}}$ as

$$\hat{\mathbf{p}} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i,$$

$\mathbf{v}_i$ being the eigenvectors of L. It is possible to show that $\|L\hat{\mathbf{p}}\|_1$ is minimised by choosing $\hat{\mathbf{p}} = n\mathbf{v}_2/\|\mathbf{v}_2\|_1$, and therefore the Fiedler vector defines a partition of the graph which minimises the number of cut edges. This is done using a Lanczos algorithm.

The numerical cost of RSB is higher than that of RCB or RGB because of the computation of the Fielder vector. However, RSB can be applied to partition general unstructured grids, and results in partition of good quality.

5.2.4 Multilevel Graph Partitioning

The methods proposed in sections 5.2.1, 5.2.2, and 5.2.3, are bisection methods, that is, the domain is decomposed in two subdomains. This is also called 2-way partition. The 2-way partition has a multilevel extension, which is briefly presented here. More detailed descriptions

can be found, for instance, in [KK98a, KK99]. A 2−way partition is commonly represented by a partition vector $\mathbf{p}$ of length n, such that for every vertex $v \in V, \mathbf{p}(v)$ is an integer between 1 and 2, indicating the partition at which vertex v belongs to.

A multilevel 2−way partitions algorithm for a graph $G = (V, E)$ can be written as in algorithm 5.2.2.

Algorithm 5.2.2 (2−way multilevel graph bisection algorithm).

a. Coarsening phase. The graph G is transformed into a sequence of smaller graphs $G_1, \ldots, G_m$ such that $|V| > |V_1| > |V_2| > \ldots > |V_m|$;

b. Partition phase. A 2-way partition P_m of the graph $G_m = (V_m, E_m)$ is computed that partitions V_m into two parts, each containing half the vertices of G, using algorithm 5.2.1;

c. Uncoarsening phase. The partition P_m of G_m is projected back to G by going through intermediate partitions $\mathbf{p}_{m-1}, \mathbf{p}_{m-2}, \ldots, \mathbf{p}_1, \mathbf{p}$.

We describe very briefly each of the 3 phases. As regards step 5.2.2.a, a set of vertices of G_i is combined to form a single vertex of the next level coarser graph. Let V_i^v be the set of vertices of G_i combined to form vertex v of G_{i+1}. v is usually referred to a multinode. To preserve the connectivity information in the coarser graph, the edges of v are the union of the edges of the vertices in V_i^v.

The step 5.2.2.b computes a high-quality bisection P_m of the coarse graph $G_m = (V_m, E_m)$ such that each part contains roughly half of the vertex of the original graph.

During the step 5.2.2.c the partition $\mathbf{p}_m$ of the coarser graph G_m is projected back to the original graph. Since each vertex of G_{i+1} contains a distinct subset of vertices of G_i, obtaining $\mathbf{p}_i$ from $\mathbf{p}_{i+1}$ is done by simply assigning the set of vertices V_i^v collapsed to $v \in G_{i+1}$ to the partition $\mathbf{p}_{i+1}(v)$. Since G_i has more degrees of freedom than G_{i+1}, it may be possible to improve the quality of the partition by local refinement heuristics, using a partition refinement algorithm.

The k−way partition problem is frequently solved by recursive bisection. That is, we first obtain a 2−way partition of V_m and then we further subdivide each part using 2−way partitions. After $\log k$ phases, the graph G is partitioned into k parts. Thus, the problem of performing a k-way partition can be solved by performing a sequence of 2−way parts or bisections. Even though this scheme does not necessarily lead to optimal partition, it is used extensively due to its simplicity.

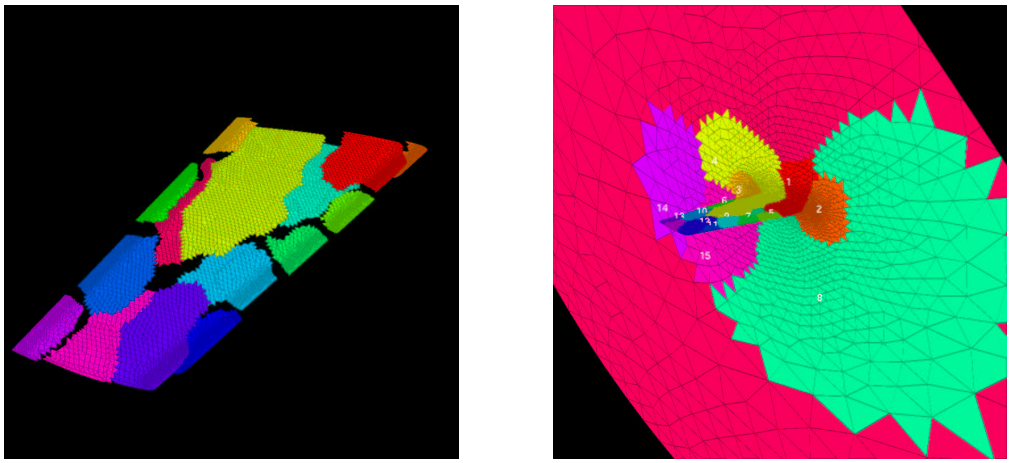


Figure 5.4: `M6`. Decomposition using 16 subdomains using RSB.

`X29`	p=16			p=32			p=64		
	$\min_i n_i$	$\bar{n}_i$	$\max_i n_i$	$\min_i n_i$	$\bar{n}_i$	$\max_i n_i$	$\min_i n_i$	$\bar{n}_i$	$\max_i n_i$
n_I	35790	37580	40115	16000	17506	19985	6750	7803	9670
n_B	2630	5165	6955	1385	3866	5375	1020	2882	3925
n_E	3070	5646	7640	1585	4387	6325	1195	3396	5115
`N_neighs`	3	6	14	3	9	18	2	9	21

Table 5.1: `X29`. Number of internal, border, and external nodes for different numbers of processors using RSB. In the table, $\bar{n}_i$ represents the average number of internal nodes $n_i, i = 1, \ldots, p$. `N_neighs` is the number of neighbouring processors.

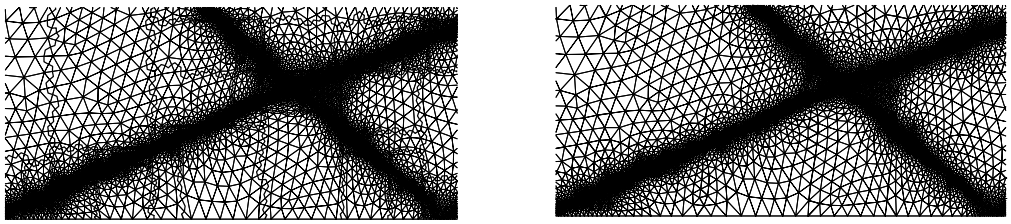


Figure 5.5: Example of vertex-oriented (left) and element-oriented domain decomposition (right).

5.3 Analysis of the Main Operations of the $\alpha\Psi$NKS Solver

The first step of a $\alpha\Psi$NKS solver is the generation of the starting grid, generally obtained from a CAD description of the computational domain. Today, grids of 2 or 3 millions of elements may be routinely obtained on workstations and PCs using Delauney generators, while for constructing larger grids one should use parallel algorithms. In this case, a possible approach is first to geometrically partition the computational domain, then to generate the grid in every subdomain independently, starting from the inter-domain boundary meshes, and finally post-processed the grid, mainly to equilibrate the workload. We will not consider parallel mesh generation in the following. On the contrary, we suppose that the starting mesh, of moderate size, say, up to some millions of elements, has been generated on one processor. This mesh is partitioned the p processors. The algorithm we have used to define the subdomains is the *k-way* graph partitioning algorithms. Then, parallel adaptation techniques are used to improve the solution quality, as described in section 4.5.1.

In the numerical results later reported, we have adopted a VO decomposition of the grid; see figure 5.5, left. The nodes assigned to processor i form its *update* set, whose size is n_i. Clearly, $\sum_{i=1}^{p} n_i = n$, n being the global dimension of the problem. The nodes belonging to the processor, but not in its update set, are called *external* nodes. The size of this set is $n_{E,i}$. The update set is further divided into two subsets: *internal* set and *border* set, whose sizes are $n_{I,i}$ and $n_{B,i}$, respectively. A component corresponding to an index in the internal set is updated using only information on the current processor, while a component whose index is contained in the border set uses informations stored in the external set (and henceforth assigned to other processors).

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 several 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 5.6 the shaded elements are stored on both processor P_1 and processor P_2.

Remark 5.3.1. *For unstructured grids, load balancing can be obtained using a one-to-one*

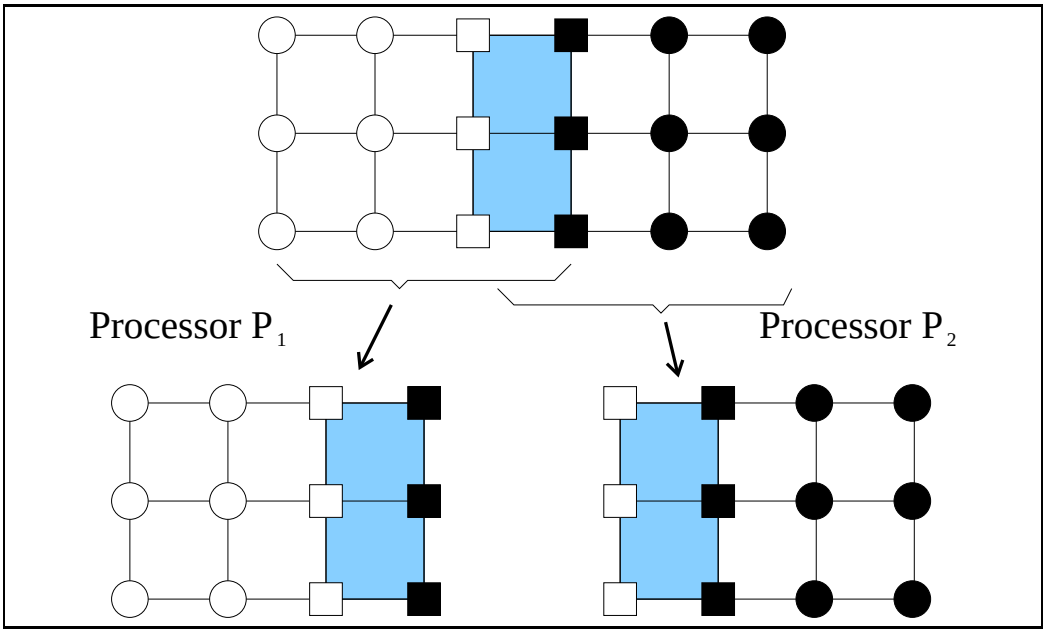


Figure 5.6: Vertex-oriented decomposition. Example of the overlapping stripe and respective mapping onto processors P_1 and P_2, update and external nodes. For processor p, '$\circ$' represents the internal nodes, '$\square$' the border nodes, and '$\blacksquare$' the external nodes. For processor q, '$\bullet$' represents the internal nodes, '$\blacksquare$' the border nodes, and '$\square$' the external nodes.

correspondence between subdomains and processors. This approach has proven to be optimal for the considered problems in the course of the IDeMAS project [IDe01]. Henceforth, it is assumed that the number of subdomains M is equal to the number of processors p.

The mesh generation step and the graph partitioning algorithm results in p input files; each input file contains a list of elements T (grid connectivity and spatial coordinates of each vertex), and a list of (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 mesh is represented in terms of element-based data structure.

At this point the parallel $\alpha\Psi$NKS solver is operated. Its most important computational kernels are:

1. construction of the distributed matrix;

2. distributed space matrix-vector products (DSMVP);

3. distributed AXPY[2] operations (DAXPY);

[2] An AXPY operation is defined as $\mathbf{u} \leftarrow \mathbf{u} + \alpha\mathbf{v}$, for two given vectors $\mathbf{u}$ and $\mathbf{v}$ and a real α

4. distributed vector-vector products (DVVP);

5. application of the parallel preconditioner;

6. parallel mesh adaptation procedure.

After convergence, each processor writes the solution corresponding to its update set as a part of the global data. Often, these partitioned data are 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 visualisation issues in the following, since we have used stand-alone software tools, as described in section 4.6.

We now analyse these operations in more details. Throughout our description, we make the following assumptions:

- the parallel computer is of multiple instruction multiple data (MIMD) architecture;
- 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;
- communications are achieved using a message passing interface, like MPI, whose implementations are currently available on almost every type of architectures.

We start with the DSMVP, since all the data structures are driven by this kernel.

5.3.1 Distributed Sparse Matrix-Vector Product

We recall that the main convention we have adopted is that when calculating elements of $\mathbf{y} = A\mathbf{x}$, a processor computes only the elements in its update set, that is $\mathbf{y}_i$. This means that the VO decomposition of section 3.3.2 is used to define the basic data structure.

Using the same notation of section 3.3.2, on processor i, the (local) matrix-vector product reads

$$\begin{pmatrix} \mathbf{y}_{I,i} \\ \mathbf{y}_{B,i} \end{pmatrix} = \underbrace{\begin{pmatrix} A_{II,i} & A_{IB,i} \\ A_{BI,i} & A_{BB,i} \end{pmatrix}}_{A_i} \begin{pmatrix} \mathbf{x}_{I,i} \\ \mathbf{x}_{B,i} \end{pmatrix} + \begin{pmatrix} 0 \\ A_{E,i}\mathbf{x}_{E,i} \end{pmatrix}, \tag{5.5}$$

where $A_{E,i} = \sum_{j \neq i} A^{(i,j)}_{BB,\Gamma}$, $\mathbf{x}_{I,i}$, $\mathbf{x}_{B,i}$, and $\mathbf{x}_{E,i}$ are the values of $\mathbf{x}$ corresponding to its internal, border, and external set, respectively.

Our DSMVP reads as in algorithm 5.3.1.

Algorithm 5.3.1 (DSMVP). On processor i and on all processors, do:

a. Exchange boundary data, that is, update the value of the external nodes $\mathbf{x}_{E,i}$;
b. Compute $\mathbf{y}_{I,i} = A_{II,i}\mathbf{x}_{I,i} + A_{IB,i}\mathbf{x}_{B,i}$;
c. Compute $\mathbf{y}_{B,i} = A_{BI,i}\mathbf{x}_{I,i} + A_{BB,i}\mathbf{x}_{B,i} + A_{BE,i}\mathbf{x}_{E,i}$.

Note that steps 5.3.1.a and 5.3.1.b can be overlapped, since the values of $\mathbf{x}_{E,i}$ are required only in step 5.3.1.c.

If matrices A_i and $A_{E,i}$ are explicitly formed, algorithm 5.3.2 can be used instead of 5.3.1.

Algorithm 5.3.2 (modified DSMVP). On processor i and on all processors, do:

a. Exchange boundary data, that is, update the value of the external nodes $\mathbf{x}_{E,i}$;
b. Compute the local matrix-vector product $\mathbf{y}_i = A_i\mathbf{u}_i$;
c. Compute the external matrix-vector product $\mathbf{y}_i \leftarrow \mathbf{y}_i + A_{E,i}\mathbf{u}_{E,i}$.

Also in this case steps 5.3.2.a and 5.3.2.b can be overlapped.

	$p=4$	$p=8$	$p=16$	$p=32$
$n = 226.935$				
Computation	1.98e-1	1.25e-1	3.77e-2	1.59e-2
Communication	1.15e-3	5.69e-3	6.96e-4	1.44e-3
s_p	4	6.3	21.0	49.2
$n = 472.465$				
Computation	-	1.63e-1	8.29e-2	4.29e-2
Communication	-	1.20e-2	1.17e-2	2.12e-2
s_p	-	8	14.8	31.1

Table 5.2: T_p (in seconds) for DSMVP.

Table 5.2 reports the communication and computational time required by the DSMVP for the two test cases. All the numerical results presented in this chapter refer to the computer of table 4.3. Note the almost linear speedup values. This means, if implemented carefully,

DSMVP does not constitute a severe bottleneck for the $\alpha\Psi$NKS solver. This optimal result requires some pre-processing and post-processing phases, as briefly described in the following.

We suppose that, on each processor i, the user provides the lines of A corresponding to the update set, stored in a structure `Amat`, and a vector `update` of size $n_{I,i}$. The elements of `Amat` are given in global numbering, while `update[i]` contains the global number of local node i.

First, a pre-processing phase is performed, to

1. determine the sets of internal and border nodes. The `update` vector is reordered so that all the internal nodes are inserted first, followed by border nodes. The new ordering is stored in a integer vector `update_index`. `update_index[i]` is the local numbering for `update[i]`;

2. determine the set of external nodes, using an integer vector `external`, of size $n_{E,i}$. Furthermore, the processor owner of each external node is found. `external` is ordered so that all the nodes belonging to the same processor are ordered contiguously;

3. the nodes contained in `external` are reordered. This local numbering is contained in an integer vector `extern_index`;

4. the local matrix A_i is reordered so that the lines corresponding to the internal set are inserted first. This local numbering is done processor-wise, and it is therefore a local operation;

5. additional information is computed, like the number of neighbouring processors, the number of internal, border, and external nodes, and stored in an auxiliary vector.

Note that for unstructured grids, step 2 this is not a trivial task. However, this is of paramount importance, since data to be sent to a processor have to be packed and sent using only one MPI call per processor to avoid latency problems occurring within the network, as suggested by equation (5.1). This step of packing data into buffers aims to create contiguous data from irregularly distributed structures.

Remark 5.3.2. *One of the advantage of MURD schemes is that the computational stencil of a generic vertex i is compact, i.e., it depends only on information from vertices that are one edge-distance away in the grid structure. Therefore, the Jacobian matrix will have non-zero elements only for the contributions corresponding to the first-order neighbouring nodes even for second order schemes, while the so-called second-order nodes do not contribute to the computation of the residual. This simplifies considerably the communication and allows the use of the same data structure for first-order and second-space discretisation schemes.*

5.3.2 The VO SC Matrix-vector Product

As pointed out in chapter 3, the SC matrix S is never formed. Therefore, for each matrix-vector multiplication that involves S, we have to solve a local Dirichlet problem in each

subdomain. This is by far the most time and resources consuming part of the application of S. We have used a direct solver, even if iterative solvers may be used as well. The resulting algorithm is as follows. As regards data exchange, it is possible to proceed as outlined in the previous section.

Algorithm 5.3.3 (VO SC matrix-vector product). On processor i and on all the processors, do:

a. Exchange boundary data, that is, update the value of the external nodes $\mathbf{u}_{E,i}$;
b. Compute $\mathbf{z}_{I,i} = -A_{IB,i}\mathbf{u}_{B,i}$;
c. Solve the linear system $A_{II,i}\mathbf{y}_{I,i} = \mathbf{z}_{I,i}$;
d. Compute $\mathbf{y}_{B,i} = A_{BB,i}\mathbf{u}_{B,i} + A_{BI,i}\mathbf{y}_{I,i}$;
e. Compute the external matrix-vector product $\mathbf{y}_{B,i} \leftarrow \mathbf{y}_{B,i} + A_{E,i}\mathbf{u}_{E,i}$.

Steps 5.3.3.b, 5.3.3.c, and 5.3.3.d are independent of step 5.3.3.a. Therefore, communications of step 5.3.3.a can be overlapped with the computations of the other three steps. Note that step 5.3.3.c can be fairly expensive if the size of $A_{II,i}$ is large. Before the first application of the SC matrix, a startup phase is needed to extract the submatrix $A_{II,i}$ and the submatrices $A_{BI,i}$, $A_{IB,i}$, $A_{BB,i}$ and $A_{E,i}$. Figure 5.7 graphically explains the process.

5.3.3 Construction of the Local Matrix

As typical in a finite-element code, the main operation in the construction of the matrix is a loop over the elements to compute the elemental matrix $A^{(T)}$ corresponding to each element T. Then, this contribution is added to the global matrix. To perform this operation in parallel using a VO decomposition, row contributions of $A^{(T)}$ not corresponding to nodes in the update set are simply discarded. The resulting matrix is sparse, as shown in figure 5.8.

5.3.4 Distributed AXPY Operation

An AXPY operation (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) is done as follows.

Algorithm 5.3.4 (DAXPY). On processor i and on all the processors, do:

a. Verify that all processors share the same value of α;
b. For all the elements of the local vector $\mathbf{u}_i$, do $\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha\mathbf{v}_i$.

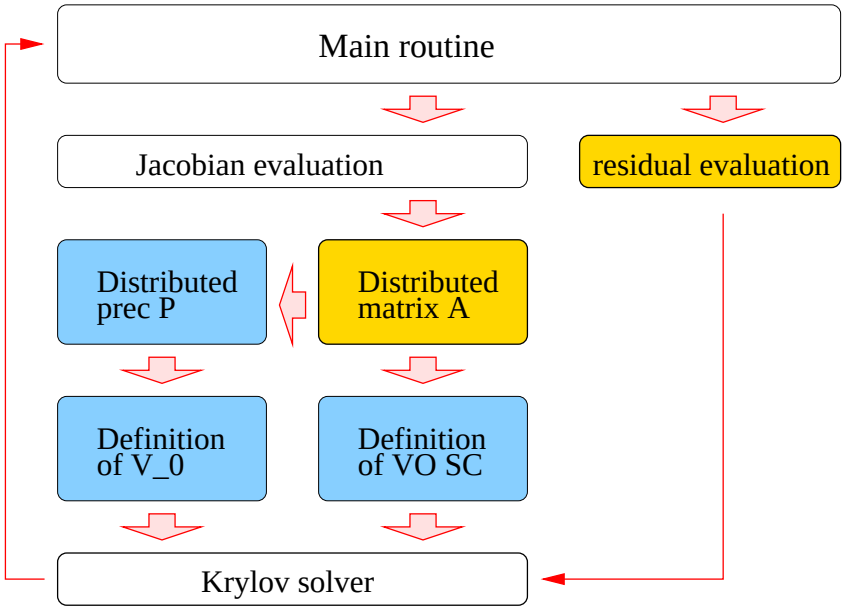


Figure 5.7: Structure for the construction and the solution of the linear system. Given a distributed matrix A and a right-hand side, both the SC matrix and its preconditioner are built by resorting to algebraic manipulations on A.

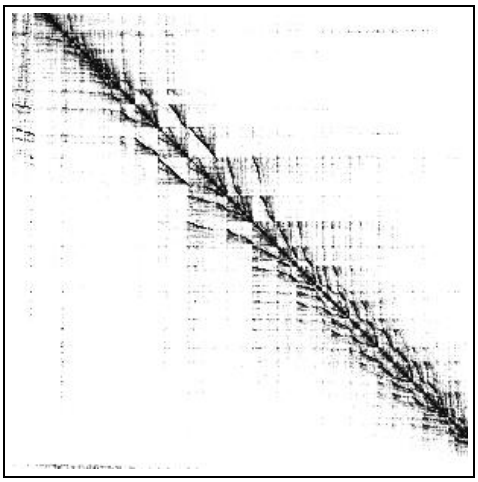


Figure 5.8: `FALCON`. Sparsity pattern of the Jacobian matrix.

Apart from the synchronisation of the value of α (this step is often avoided), no communication occurs in vector updates and DAXPY operations.

Table 5.3 reports the CPU time required by a typical DAXPY operation. Speedup values are almost equivalent to the ideal speedup. A super-linear value is found in one case, probably

due to the better cache reuse. Note the different performances for the basic vector operations carried out with or without using the BLAS libraries.

	$p=4$	$p=8$	$p=16$	$p=32$
$n = 226.935$				
Computation (without BLAS)	3.52e-3	1.67e-3	8.14e-4	4.28e-4
Computation (using BLAS)	6.08e-4	3.15e-4	1.65e-4	8.56e-5
s_p (with using BLAS)	4	8.4	17.3	32.9
s_p (using BLAS)	4	7.7	14.7	28.4
$n = 472.465$				
Computation (without BLAS)	-	3.48e-3	1.69e-3	9.32e-4
Computation (using BLAS)	-	6.36e-4	3.24e-4	1.48e-4
s_p (with using BLAS)	-	8	16.5	29.9
s_p (using BLAS)	-	8	15.7	34.4

Table 5.3: T_p (in seconds) for a DAXPY operation.

5.3.5 Distributed Vector-Vector Product

Using a VO decomposition, the parallel scalar product $\alpha = \langle \mathbf{u}, \mathbf{v} \rangle$ on processor i is carried out by the following algorithm.

Algorithm 5.3.5 (DVVP). On processor i and on all the processors, do:

a. Compute the local value $\alpha_i = \langle \mathbf{u}_i, \mathbf{v}_i \rangle$;

b. Sum up the local contributions of all the processors: $\alpha = \sum_{i=1}^{M} \alpha_i$.

Communications are required only in the second step. Before computing the final value of α, all processors must have finished the computations in the first step. Therefore, DVVP always act in a parallel environment as *synchronisation points* and require global communication. This is because all the processors have to send their contribution to complete this operation, and then wait to receive back the final result. Vector-vector operations area *blocking* operations, like other all-to-all reducing operations. The MPI library provides specific functions which can be used to this purpose.

Table 5.4 reports the time needed for the vector-vector product. We may note a difference of one order of magnitude between the non-BLAS and the BLAS routines.

	$p=4$	$p=8$	$p=16$	$p=32$
$n = 226.935$				
Computation (without BLAS)	2.11e-4	1.05e-3	4.04e-4	2.03e-4
Computation (using BLAS)	2.56e-5	4.85e-5	3.04e-5	3.68e-5
$n = 472.465$				
Computation (without BLAS)	-	8.48e-5	3.96e-4	5.27e-5
Computation (using BLAS)	-	3.04e-5	3.04e-5	4.08e-5

Table 5.4: T_p (in seconds) for DVVP.

5.3.6 Application of the Schwarz Preconditioner

Using a VO decomposition, it is particularly easy to implement a one-level Schwarz preconditioner P_S with minimal overlap, i.e., with an overlap among the subdomains of just one layer of element (see sections 3.2.1 and 3.2.2). The computation of $\mathbf{z} = P_S^{-1}\mathbf{r}$ reads, on processor i,

$$\begin{pmatrix} \mathbf{z}_{I,i} \\ \mathbf{z}_{B,i} \end{pmatrix} = \underbrace{\begin{pmatrix} A_{II,i} & A_{IB,i} \\ A_{BI,i} & A_{BB,i} \end{pmatrix}}_{A_i}^{-1} \begin{pmatrix} \mathbf{r}_{I,i} \\ \mathbf{r}_{B,i} \end{pmatrix}. \tag{5.6}$$

Matrix A_i is already stored and formed in the local memory of processor i. To use wider overlaps, the following algorithm 5.3.6 has to be used to create the necessary data structures.

Algorithm 5.3.6 (data exchange to increase by 1 the overlap). On processor i and on all the processors, do:

a. Group the external nodes such that the ones stored on the same processor are ordered contiguously in memory;
b. Allocate memory to store the local matrix corresponding to the wider overlap $\bar{A}_i$;
c. Request the lines of A corresponding to the external nodes, and store these lines in $\bar{A}_i$;
d. Update the data structure required for data communication in the preconditioning phase.

Step 5.3.6.a is required to reduce the number of calls to communication routines. These calls will be coded as asynchronous calls, meaning that the program starts sending data to all the neighbouring processors, then starts receiving, and finally checks if communication

is completed. Asynchronous calls can help to avoid dead lock phenomena and to overlap communication and computations. Note that intensive data exchanges occur in Step 5.3.6.c.

After one (or several) applications of algorithm 5.3.6, Equation (5.6) reads:

$$\begin{pmatrix} \mathbf{z}_{I,i} \\ \mathbf{z}_{B,i} \\ \mathbf{z}_{E,i} \end{pmatrix} = \underbrace{\begin{pmatrix} A_{II,i} & A_{IB,i} & 0 \\ A_{BI,i} & A_{BB,i} & A_{BE,i} \\ 0 & A_{EB,i} & A_{EE,i} \end{pmatrix}^{-1}}_{\bar{A}_i} \begin{pmatrix} \mathbf{r}_{I,i} \\ \mathbf{r}_{B,i} \\ \mathbf{r}_{E,i} \end{pmatrix} \tag{5.7}$$

where $\bar{A}_i$ represents the "extended" matrix, and $A_{EB,i}$ and $A_{EE,i}$ correspond to the lines exchanged by algorithm 5.3.6. Note that the values of $\mathbf{z}_{E,i}$ can be weighted or simply discarded (see Remark 3.2.1).

Remark 5.3.3. *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 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 [SF01]. In fact, compared with other variants, ILU(0) is computationally fast and memory efficient.*

Algorithm 5.3.7 (one-level Schwarz preconditioner). On processor i and on all the processors, do:

a. If the overlap involves more than 1 element, exchange the lines corresponding to the external nodes to increase the overlap;
b. Factorise the local matrix (using an incomplete factorisation) and store the $\tilde{L}$ and $\tilde{U}$ factors.

Table 5.5 reports the time in seconds for the setup and the application of P_S. Note that the time required by the setup phase is remarkably larger that the time needed for applying the preconditioner. The speedup for the setup phase is super-linear, mainly because of non-linearities in the factorisation algorithm. Moreover, the smaller the subdomain (and therefore the smaller the dimension of the local matrix), the more consistent the cache reuse.

	$p=4$	$p=8$	$p=16$	$p=32$
$n = 226.935$				
Setup	1.13e+2	4.21e+1	1.45e+1	5.17e+0
Speedup for setup	4	10.7	31.2	87.4
Computation	2.00e-1	1.26e-1	3.81e-2	1.61e-2
Communication	1.17e-3	5.76e-3	7.04e-3	9.45e-3
$n = 472.465$				
Setup	-	2.13e+2	6.65e+1	2.41e+1
Speedup for setup	-	8	25.6	70.7
Computation	-	1.73e-1	8.57e-2	4.44e-2
Communication	-	1.24e-2	1.42e-2	6.02e-1

Table 5.5: T_p (in seconds) for the one-level Schwarz preconditioner.

As regards two-level methods (section 3.2.3), we have used non-smoothed aggregation procedures. Consider again the algebraic interpretation given in section 3.2.4, and in particular the expression of R_0 given by (3.21). In our case (piece-wise linear finite-element functions), the graph of the local matrix A_i can furnish the information needed to produce the aggregates, and henceforth the grid data are not required by the algorithm. Note moreover that the vectors $\boldsymbol{\eta}_{s,i}$ are not explicitly required, as detailed in algorithm 5.3.8.

Algorithm 5.3.8 (decoupled aggregation). On processor i and on all the processors, do:

a. Build (locally) $n_{0,i}$ aggregates ϑ_j;
b. Assign to each aggregate a unique number, that is, the local aggregate j will be $j + \sum_{k=1}^{i-1} n_{0,k}$;
c. On each i, build up the lines of A_0 corresponding to the local aggregates;
d. Prepare to solve a linear system with A_0;
e. Prepare to apply R_0 and R_0^T.

Step 5.3.8.e can be implemented in this way. We constructs an integer vector $\mathbf{p}$, so that p_i represents the (local) degree of freedom associated to the aggregates $\vartheta_{s,i}$. The size of vector $\mathbf{p}$ is $n_{I,i}+n_{B,i}+n_{E,i}$. First, communication occurs 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 (3.13) no communication will occur to compute, on each subdomain, the rows of A_0 corresponding to the degree of freedom of V_0 contained in that subdomain.

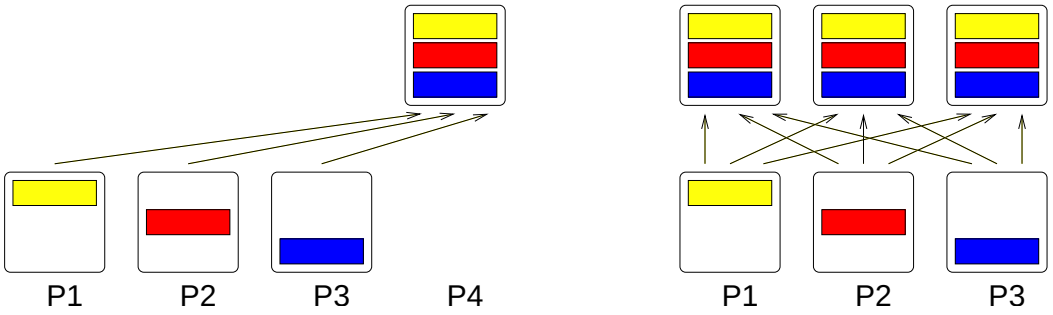


Figure 5.9: Construction of the coarse matrix. Each processor $P_i, i = 1, 2, 3$ builds up the lines of A_0 corresponding to the aggregates that it contains, then the complete matrix is formed on all processors (right) or on a (possibily dedicated) processor (left).

Solving the linear system with coarse grid on a parallel computer presents a potential performance problem, since the coarse grid problem is often relatively small. This is because solving small problems in parallel on distributed memory processors can often lead to slower solution than if only a single processor is used. In the case when A_0 is quite small, one is faced with two choices: redundantly solve the coarse grid problem on all processors, or solve it on one processor and distribute the result to the other processors. In the former, all coarse grid nodal values are scattered to all processors, each of them will independently form its portion of the result; see figure 5.9, left. In the latter, instead, a further phases of communication is required after the solution of the linear system; see figure 5.9, right. We have chosen to solve the coarse problem on each processor using a direct LU decomposition. For large values of n_0, instead, it may be worthy to keep A_0 distributed among the processors, then solve the linear system with A_0 in parallel, possibly by an iterative method.

5.3.7 The ASC Preconditioner

We now consider the approximate SC preconditioners introduced in section 3.3.5. A possible way to compute $\mathbf{z} = P_{ASC}^{-1}\mathbf{r}$ is described in the following algorithm 5.3.9.

Algorithm 5.3.9 (ASC preconditioner).

a. Compute $\mathbf{y}_I = \hat{A}_{II}^{-1}\mathbf{r}_I$;

b. Compute $\mathbf{y}_B = \mathbf{r}_B - A_{BI}\mathbf{y}_I$;

c. Solve the approximate SC system $\hat{S}\mathbf{z}_B = \mathbf{y}_B$;

d. Compute $\mathbf{z}_I = \mathbf{y}_I - \hat{A}_{II}^{-1}A_{IB}\mathbf{z}_B$.

Step 5.3.9 corresponds to the computation of the right-hand side for the (approximate) Schur complement system.

Steps 5.3.9.a and 5.3.9.b correspond to the appliocation of the lower triangular part of P_{ASC}, while steps 5.3.9.c and 5.3.9.d to the application of the upper triangular part of P_{ASC}.

Everyone of the computations 5.3.9.a, 5.3.9.b and 5.3.9.d can be accomplished in parallel, as they require no communications among the different subdomains. On the other hand, step 5.3.9.cq is a *global* operation, which, however, may be split in several parallel steps preceded and followed by scatter and gather operations involving communication among subdomains. Note that the matrix $\hat{S}$ may itself be preconditioned.

Remark 5.3.4. *The preconditioners described in sections 5.3.6 and 5.3.7 and the SC solver of section 5.3.2 have been implemented by the author in a common library, built on the top of the AZTEC library [THHS99]. In fact, the "open" structure of this library allows the user to write its own matrix-vector product and preconditioning functions. The library, written in C, implements some Krylov subspace methods and one-level Schwarz preconditioners, with either complete or various kind of incomplete factorisations.*

5.3.8 Parallel Mesh Adaptation

In order to perform the mesh adaptation procedures on a parallel computer, two approaches can be pursued. The first one is the *master-slave* approach, in which a processor is responsible for the management of the mesh data (and in general of I/O routines). In the master-slave approach, once a preliminary solution has been obtained, it is gathered from the slave processors to the master processor, where sequential mesh adaptation is started. Then, the mesh is partitioned using a graph partitioning algorithm, and redistributed among the processors. This approach works quite well for "small" meshes, while intensive calculations require a *no-master* approach, in which the completely parallel dynamic mesh adaptation algorithm takes place entirely on the network of processors. This is precisely the approach we have followed. A more detailed description of the no-master approach may be found in [LR00].

The paradigm we have adopted is based upon the concept of non-hierarchical mesh adaptation, i.e., the successive meshes do not remember their original affiliation, see [Ric93, RL95]. This allows high flexibility and quality for the different stages of adaptation as the mesh at a certain time does not rely on background macro mesh. 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"; see [Car97, SL00]. This non-hierarchical technique, associated with an optimisation, produces similar meshes to those obtained by a re-meshing strategy. The drawbacks reside in the complexity of programming and the coherent re-projection on the CAD definitions defining the boundary surfaces.

The parallelisation of the mesh adaptation schemes presented in section 4.5.1 requires careful renumbering and re-ordering techniques internally per processor (local) and globally.

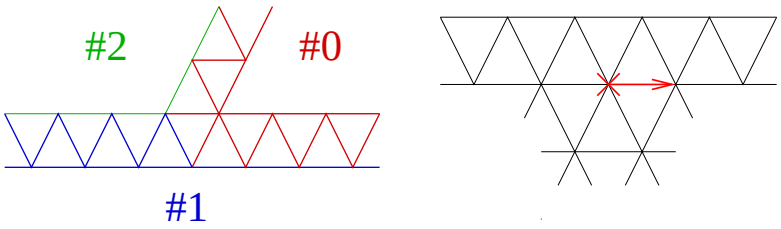


Figure 5.10: Layout of the segments during parallel adaptation.

Also the treatment of elements in the overlapping regions (the cut elements) is crucial. In particular, for *refinement*, nodes created on an updated segment become new processor interface border nodes, as reported on the left of figure 5.10. For *coarsening*, when attempting to delete a border node, a border segment must be chosen (figure 5.10, right). Also *diagonal swapping* and *edge collapsing* requires a careful implementation to avoid to recompute the sets of internal, border, and external nodes. The treatment of elements in the overlapping region is undoubtedly the most problematic aspect of parallel mesh adaptation using VO decompositions.

Repartitioning

The mesh 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 inefficient parallel performances. Effectively, the processor with the largest workload will delay the process. In fact, the starting domain decomposition was obtained to balance the workload for the initial mesh (that is, 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. Our implementation is based on the public library ParMETIS [KK97]. ParMETIS is called concurrently by each process and gives the new destination for each internal node. Then, data structure corresponding to moved nodes are sent using MPI calls.

Reordering and Renumbering

To complete the parallel adaptation procedure, fast efficient multiple renumbering techniques are necessary for the mesh 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); see [Wei92]. Consequently, all the vectors and matrices

n_0	$\dim V_0$ (see section 3.2.3)
p	number of processors
`ov`	overlap for VO decompositions (see section 3.3.2)
ILU(f)	incomplete LU factorisation with factor of fill-in f (see section 2.2.1)
CPU-time	elapsed CPU-time (in seconds)
total CPU-time	sum of elapsed CPU-times for the considered time levels
CFL	CFL number (see section 4.4.2)
P_S	one-level Schwarz preconditioner (see section 3.2.1)
$P_{S,C,add}$	two-level Schwarz preconditioner with aggregation, additive version (see section 3.2.3 and formula 3.9)
$P_{S,C,hybrid}$	two-level Schwarz preconditioner with aggregation, hybrid version (see section 3.2.3 and formula 3.10)
P_{ASC}	approximate SC preconditioner (see section 3.3.5)

Table 5.6: List of symbols used in this section.

used in the code may have to be re-allocated 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 height 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 unbalance 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 look-up is performed to balance again.

Look-up, 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 $\log(n) \leq S(n) \leq n$.

5.4 Numerical Results

We have carried out several numerical tests with the aim of analysing the parallel properties of some of the preconditioners outlined in section 5.3.6 and 5.3.7. The characteristics of the parallel computer we have used are reported in table 4.3.

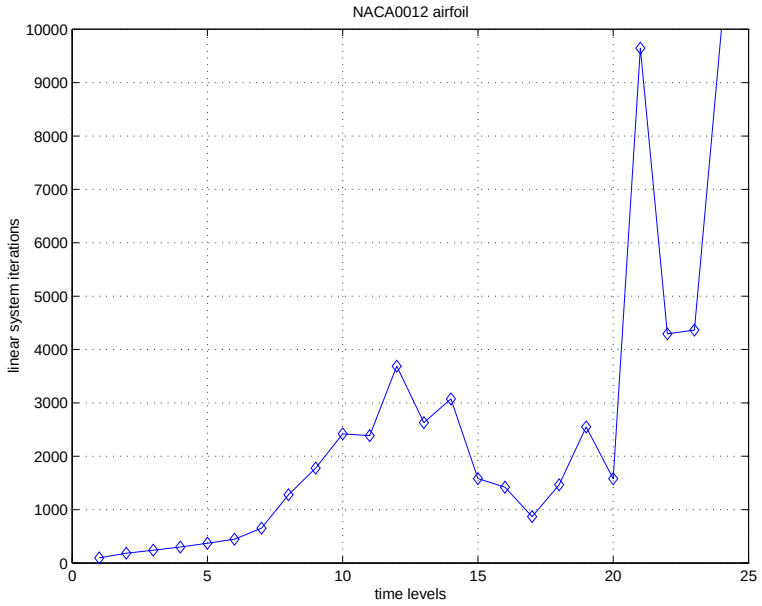


Figure 5.11: `NACA0012`. Number of iterations to converge at each time level using a non-preconditionned GMRES methods.

The choice of the iterative scheme is analised in figures 5.12 and 5.13. The former reports the iterations to converge at each time-level, while the latter the convergence history of the 10-th time step. Even for small 2D problems, like `NACA0012`, convergence cannot be achieved without preconditioning, unless small CFL numbers are used, as can be appreciated from figure 5.11. We recall that the CFL number is increased by a fixed factor at each time iteration. As CFL increases, the linear systems becomes more skew-symmetric and less diagonal-dominant. Therefore, more linear system iterations are required to converge. Therefore, we have always used preconditioned Krylov accelerators, Our reference preconditioner is the one-level Schwarz preconditioner P_S with ILU(0) and one-element of overlap among the subdomains.

The accelerators considered are: BI-CGSTAB, CGS, TFQMR, GMRES(25) and GMRES(60), and the test case is `FALCON`. It can be observed:

- for all the considered Krylov accelerators, convergence of the linear system solver is reached at any time step;
- in all cases, GMRES is characterised by a smooth monotonically decreasing convergence behaviour;
- 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.

- GMRES it is quite sensitive to the dimension of the Krylov space, especially as the number of processor used in the computation increases;
- CGS has the most erratic convergence history, followed by Bi-CGSTAB, which performs quite well but it is still erratic;
- TFQMR is characterised by an initial plateau, where no reduction in the residual norm is achieved, followed by rapid decreasing of the residual norm;
- the differences among the Krylov accelerators are particularly evident as the number of processors grows. This may reflect the fact the Schwarz preconditioners with no coarse correction worsen its performances as the number of subdomains increases.

From this analysis, it can be concluded that:

- GMRES is the most robust of the studied Krylov methods, but requires a certain amount of memory to store the Krylov vectors. Therefore, if enough memory is available, it should be preferred to the other schemes;
- if schemes not too memory-demanding are in order, TFQMR seems to be the best choice in terms of iterations to converge.

Most of the time required by the linear system solver is spent to set up the preconditioner, that is, to compute the extended matrix $\bar{A}_i$ if `ov`> 1 (see section 5.3.6), and to compute the incomplete LU factorisation.

Table 5.7 reports the CPU-time (in seconds) required to solve the linear system at the $10-$th time level, using different Krylov solvers. Once again, the preconditioners is P_S, and various fill-in factors and amount of the overlap are tested. It can be observed:

- as expected, for all the fill-in values and overlap, the number of iterations to converge increases as the number of subdomains increases;
- increasing the amount of the overlap always results in more CPU-time.

From this analysis, it can be concluded that

- using fill-in factors greater than 0, the reduction in the iterations to converge may be significant, although this gain is not enough to compensate the additional time required to compute the exact factorisation;
- using wider overlaps increases the CPU-time. For example, using 4 processors, GMRES(60) and ILU(0) factorisations on the subdomains, the total CPU-time needed to solve with minimal overlap is of 1243 seconds, with an overlap of 2 elements 1807 seconds, and of three elements 2319 seconds;
- the minimal overlap ILU(0) is the optimal compromise between robustness (that is, iterations to converge) and CPU-time.

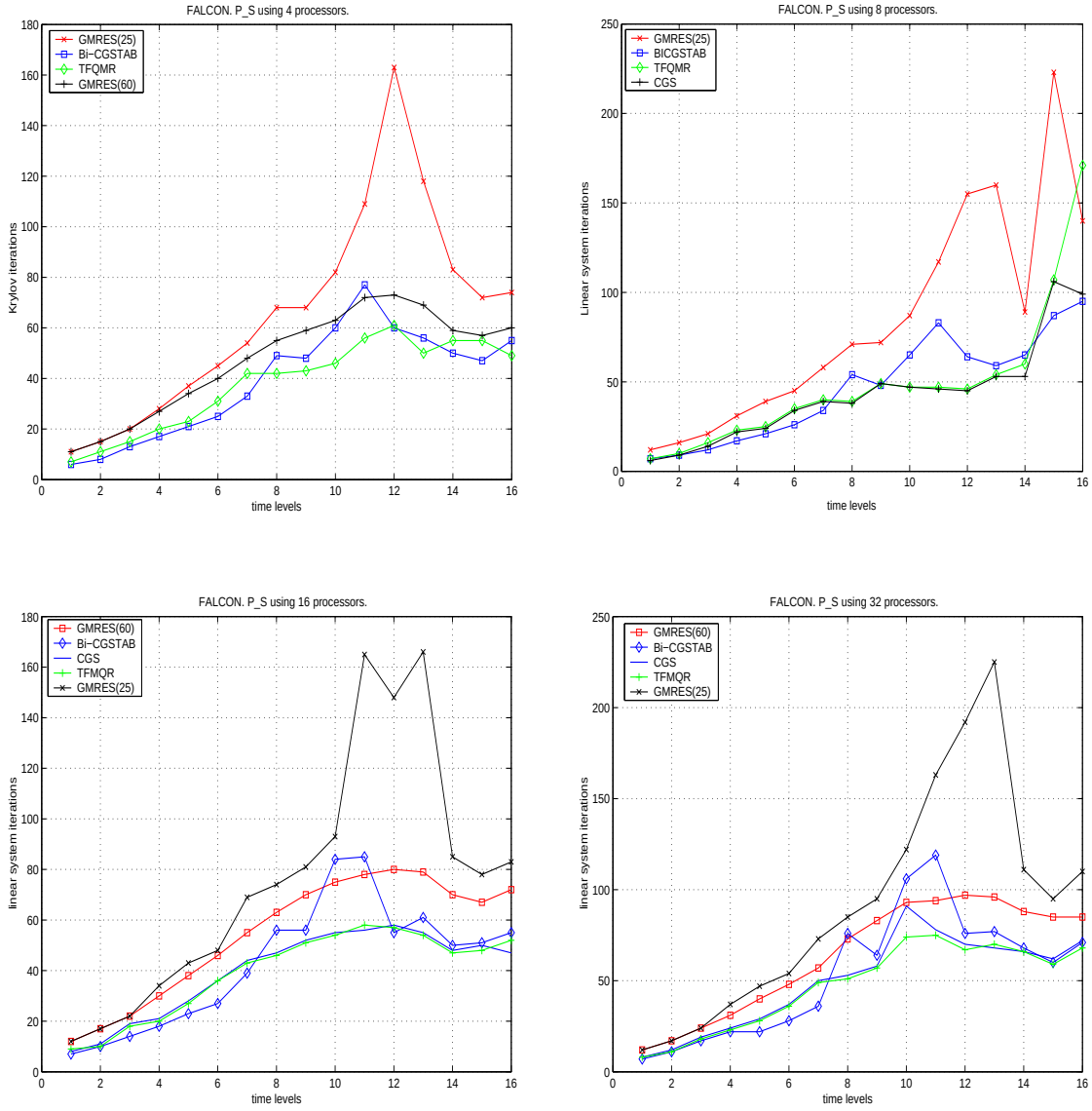


Figure 5.12: `FALCON`. Iterations to converge at each time level using P_S with minimal overlap and ILU(0). Processors used in the computations: 4 (top, left), 8 (top, right), 16 (bottom, left) and 32 (bottom, right).

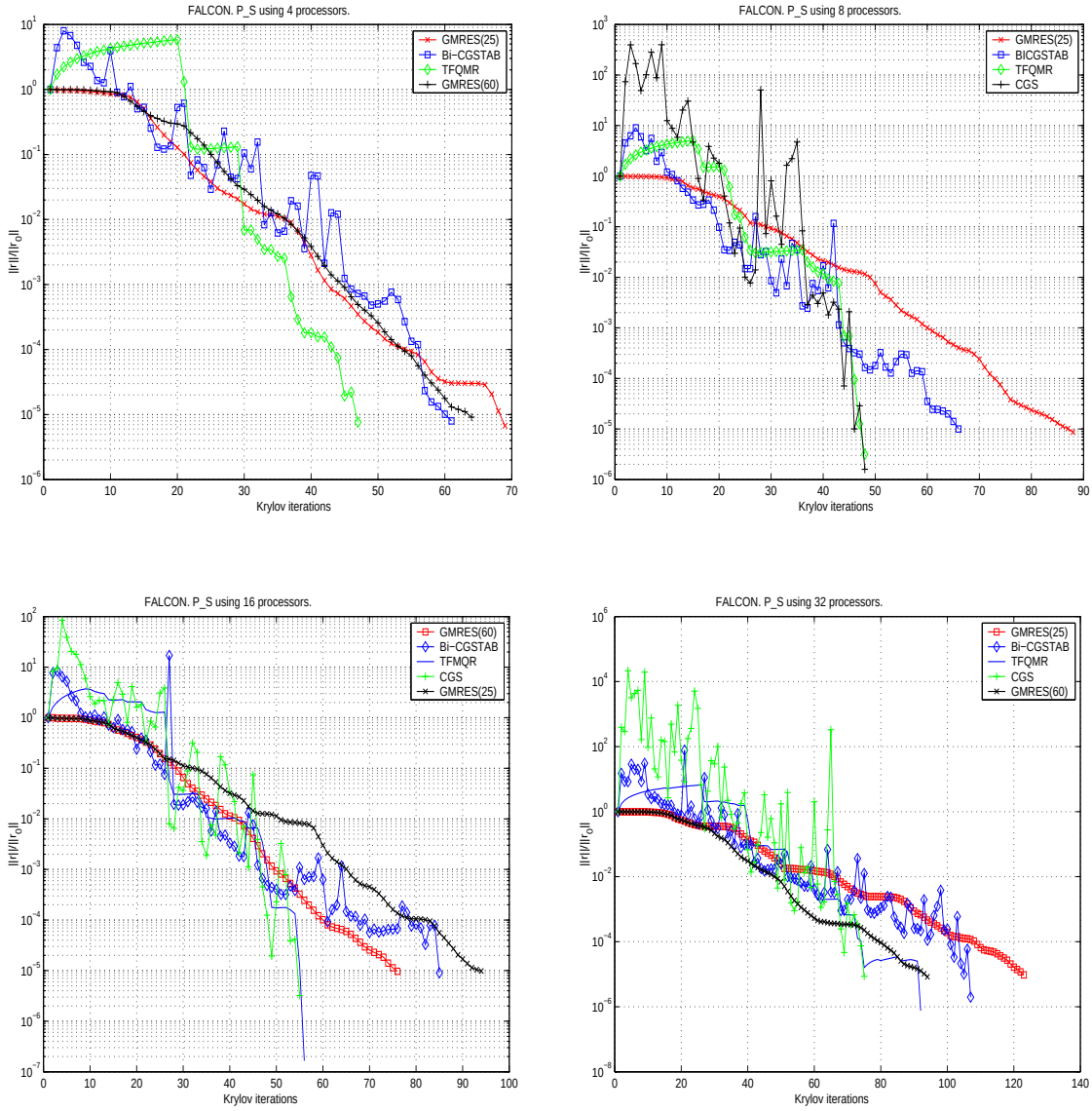


Figure 5.13: `FALCON`. Iterations to converge at the 10−th time level using P_S with minimal overlap and ILU(0). Processors used in the computations: 4 (top, left), 8 (top, right), 16 (bottom, left) and 32 (bottom, right).

As a result of this analysis, GMRES with one-level Schwarz preconditioner, using ILU(0) an minimal overlap, is found to be the optimal one-level Schwarz preconditioner.

Two level Schwarz methods are studied in tables 5.8 and 5.9 and figure 5.14. Table 5.8 is about the `X29` test case. Iterations to converge and CPU-time required by GMRES(10) at different time-levels are reported. Some results refer to P_S to compare one-level and two-level methods. Instead, table 5.9 reports the total CPU-time required by both the additive and the hybrid version of $P_{S,C}$ for `M6` to reduce the density residual by a factor of 10^{-6} using a first-order method. Figure 5.14 compares the iterations to converge required by the two-level additive and the hybrid method for `FALCON`.

We can note:

- the gain in number of iterations to converge is rather modest (if present) for low values of the CFL number;
- for large CFL numbers, the use of the aggregation coarse space reduces both the number of iterations to converge and the CPU-time;
- the optimal dimension of the coarse space seems to be of about 4 aggregates per subdomains. The use of just one aggregate per subdomains has a positive effect, although the reduction in the number of iterations to converge is modest, and only partially justified by the CPU-time;
- the construction of the coarse matrix requires a very modest time using a moderate number of aggregates per subdomains. In fact, in this test case and $n_0/p = 4$, this time was of 0.54 seconds for $p = 16$, of 0.37 seconds for $p = 32$, and of 0.91 seconds for $p = 64$;
- both the additive and the hybrid method outperform P_S in terms of iterations to converge;
- the hybrid version is much more performant with respect to the additive one for a fixed dimension of the coarse space. However, the gain in terms of CPU-time may be modest (or even the additive version may result quicker than the hybrid one).

We may conclude:

- P_S should be used instead of $P_{S,C}$ for low CFL numbers;
- $P_{S,C,hybrid}$ should be the matter of choice when high values of the CFL number are used;
- for the considered test cases, the number of aggregates on each subdomain should be of about 4;
- although no explicit rule could be found, the hybrid version should be preferred to the additive one.

Linear Solver	p	Prec	Iters	CPU time
GMRES(25)	4	P_S, ILU(0), ov=1	82	163.88
GMRES(25)	8	P_S, ILU(0), ov=1	87	87.38
GMRES(25)	16	P_S, ILU(0), ov=1	93	27.73
GMRES(25)	32	P_S, ILU(0), ov=1	122	21.67
GMRES(25)	16	P_S, ILU(0), ov=2	75	46.29
GMRES(25)	32	P_S, ILU(0), ov=2	98	36.64
GMRES(25)	8	P_S, ILU(0), ov=3	72	189.32
GMRES(25)	16	P_S, ILU(0), ov=3	68	89.14
GMRES(25)	16	P_S, ILU(1), ov=1	54	175.50
GMRES(25)	16	P_S, ILU(1), ov=2	43	239.56
TFQMR	4	P_S, ILU(0), ov=1	46	166.39
TFQMR	8	P_S, ILU(0), ov=1	47	83.56
TFQMR	16	P_S, ILU(0), ov=1	55	47.74
TFQMR	32	P_S, ILU(0), ov=1	88	24.25
TFQMR	32	P_S, ILU(0), ov=2	57	45.74
TFQMR	16	P_S, ILU(0), ov=2	49	69.43
TFQMR	8	P_S, ILU(0), ov=3	43	172.17
TFQMR	16	P_S, ILU(0), ov=3	41	111.31
TFQMR	16	P_S, ILU(1), ov=1	37	184.30
TFQMR	16	P_S, ILU(1), ov=2	28	244.84
Bi-CGSTAB	8	P_S, ILU(0), ov=1	68	104.47
Bi-CGSTAB	16	P_S, ILU(0), ov=1	72	55.09
Bi-CGSTAB	32	P_S, ILU(0), ov=1	113	28.82
Bi-CGSTAB	16	P_S, ILU(0), ov=2	61	75.81
Bi-CGSTAB	32	P_S, ILU(0), ov=2	92	59.62
Bi-CGSTAB	16	P_S, ILU(1), ov=1	49	183.17
CGS	16	P_S, ILU(0), ov=1	40	88.84
CGS	32	P_S, ILU(0), ov=1	88	29.30
CGS	32	P_S, ILU(0), ov=2	56	46.02

Table 5.7: FALCON. Iterations to converge and CPU-time at the 10−th time level (when the CFL number is 5120), using different Krylov accelerators. The preconditioner is P_S, with different combinations of the overlap and fill-in factor.

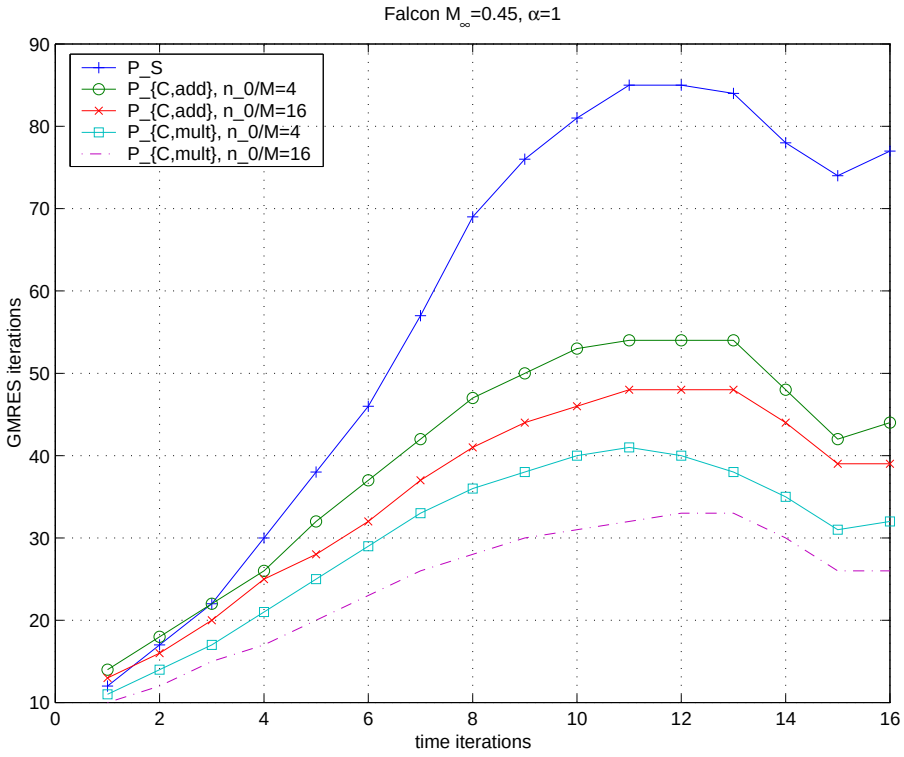


Figure 5.14: `FALCON`. Comparison among P_S, $P_{S,C,add}$ and $P_{S,C,hybrid}$. The coarse space is constructed using 4 and 16 aggregates per subdomain.

time level	CFL	p	Prec	n_0/p	Iters	CPU-time
2	0.4	32	P_S	-	2	27.04
4	1.6	32	P_S	-	2	27.06
10	102.4	32	P_S	-	29	33.26
14	1000.0	32	P_S	-	84	54.79
2	0.4	64	P_S	-	2	10.24
4	1.6	64	P_S	-	3	10.51
10	102.4	64	P_S	-	33	15.57
14	1000.0	64	P_S	-	90	25.28
2	0.4	16	$P_{S,C,hybrid}$	4	2	88.67
4	1.6	16	$P_{S,C,hybrid}$	4	3	89.77
10	102.4	16	$P_{S,C,hybrid}$	4	16	96.69
14	1000.0	16	$P_{S,C,hybrid}$	4	51	117.64
2	0.4	32	$P_{S,C,hybrid}$	4	2	27.45
4	1.6	32	$P_{S,C,hybrid}$	4	3	27.70
10	102.4	32	$P_{S,C,hybrid}$	4	16	31.38
14	1000.0	32	$P_{S,C,hybrid}$	4	39	37.93
2	0.4	64	$P_{S,C,hybrid}$	4	2	11.64
4	1.6	64	$P_{S,C,hybrid}$	4	3	11.28
10	102.4	64	$P_{S,C,hybrid}$	4	15	14.77
14	1000.0	64	$P_{S,C,hybrid}$	4	35	19.45
2	0.4	64	$P_{S,C,hybrid}$	1	2	10.42
4	1.6	64	$P_{S,C,hybrid}$	1	3	10.51
10	102.4	64	$P_{S,C,hybrid}$	1	20	13.53
14	1000.0	64	$P_{S,C,hybrid}$	1	79	23.46

Table 5.8: `X29`. Comparison between P_S and $P_{S,C,hybrid}$ at various time levels. ILU(0) factorisations and minimal overlap have been used to define P_S. The Krylov solver is GMRES(10).

p	Prec	n_0/p	Total CPU-time
8	$P_{S,C,add}$, ov=1, ILU(0)	4	1008.23
8	$P_{S,C,add}$, ov=1, ILU(0)	8	978.26
8	$P_{S,C,add}$, ov=1, ILU(0)	16	912.62
8	$P_{S,C,add}$, ov=1, ILU(0)	32	883.93
16	$P_{S,C,add}$, ov=1, ILU(0)	4	502.89
16	$P_{S,C,add}$, ov=1, ILU(0)	8	506.94
16	$P_{S,C,add}$, ov=1, ILU(0)	16	515.42
16	$P_{S,C,add}$, ov=1, ILU(0)	32	457.29
32	$P_{S,C,add}$, ov=1, ILU(0)	4	208.32
32	$P_{S,C,add}$, ov=1, ILU(0)	8	245.92
32	$P_{S,C,add}$, ov=1, ILU(0)	16	300.81
32	$P_{S,C,add}$, ov=1, ILU(0)	32	505.14
8	$P_{S,C,hybrid}$, ov=1, ILU(0)	4	934.25
8	$P_{S,C,hybrid}$, ov=1, ILU(0)	8	945.62
8	$P_{S,C,hybrid}$, ov=1, ILU(0)	16	909.40
8	$P_{S,C,hybrid}$, ov=1, ILU(0)	32	925.75
16	$P_{S,C,hybrid}$, ov=1, ILU(0)	4	458.87
16	$P_{S,C,hybrid}$, ov=1, ILU(0)	8	405.14
16	$P_{S,C,hybrid}$, ov=1, ILU(0)	16	413.97
16	$P_{S,C,hybrid}$, ov=1, ILU(0)	32	442.39
32	$P_{S,C,hybrid}$, ov=1, ILU(0)	4	164.29
32	$P_{S,C,hybrid}$, ov=1, ILU(0)	8	164.23
32	$P_{S,C,hybrid}$, ov=1, ILU(0)	16	181.82
32	$P_{S,C,hybrid}$, ov=1, ILU(0)	32	515.42

Table 5.9: M6. Total CPU-time required by GMRES(50) and $P_{S,C,add}$ and $P_{S,C,hybrid}$.

SC-based methods have been analysed using the FALCON test case. First, table 5.10 reports the total number of iterations to converge and the CPU-time in seconds to solve the original system $A\mathbf{u} = \mathbf{f}$ and the *exact* Schur complement system $S\mathbf{u}_B = \mathbf{g}$ using GMRES(60). For both system the preconditioner is P_S with ILU(0) and minimal overlap. It can be observed:

- the reduction in the iterations to converge is significant;
- the CPU-time is remarkably larger, except if many subdomains are used in the computation.

	$A\mathbf{u} = \mathbf{f}$		$S_{VO}\mathbf{u}_B = \mathbf{g}$	
p	iter	time	iter	time
1	440	11180.97	-	-
4	501	2751.95	174	10042.35
8	511	1342.63	199	3871.39
16	502	798.23	244	1818.70
32	559	466.72	288	719.96

Table 5.10: FALCON. Total iterations required to reach the convergence and CPU-time (in seconds) on a SGI-Origin 2000.

Instead, the effectiveness of the ASC preconditioner is analysed in figure 5.15. The outer Krylov solver is GMRESR, while the we have used L iterations of a nested GMRES solver for the solution of the linear system with $\hat{S}$. The approximation used to construct $\hat{A}_{II}$ is the ILU(0) decomposition of A_{II}. It can be obeserved that

- increasing the value of L results in a more robust method. Although, the CPU-time may increase significantly.
- the scalability of the P_{ASC} preconditioner is rather good;
- the CPU-time is larger with respect to the two-level Schwarz preconditioner.

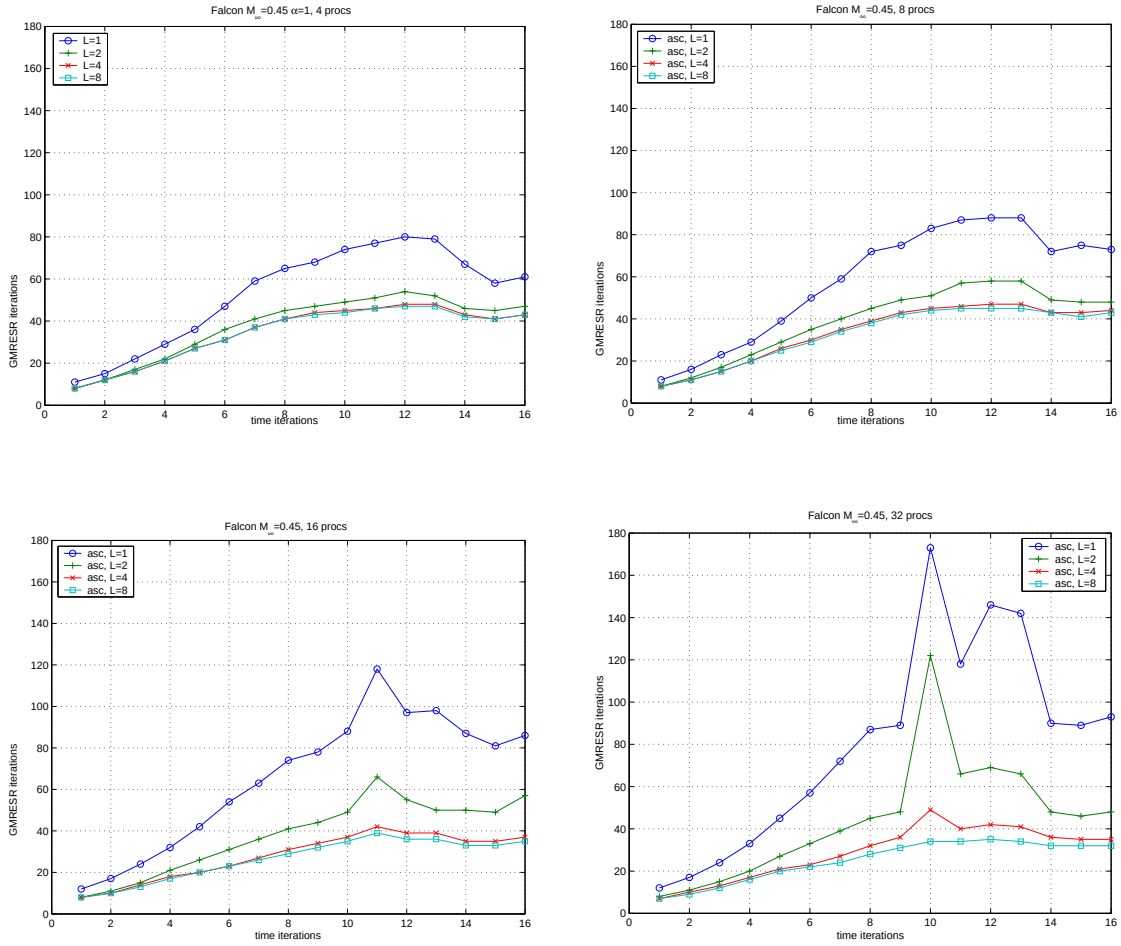


Figure 5.15: FALCON. Iterations to converge at each time level using P_{ASC} and GMRESR. Processors used in the computations: 4 (top,left), 8 (top,right), 16 (bottom,left) and 32 (bottom,right).

Linear Solver	p	Prec	Total CPU-time
GMRES(50)	8	$P_{S,Chybrid}$, ov=1, ILU(0), $n_0/p = 4$	909.40
GMRES(50)	16	$P_{S,C,hybrid}$, ov=1, ILU(0), $n_0/p = 4$	405.42
GMRES(50)	32	$P_{S,C,hybrid}$, ov=1, ILU(0), $n_0/p = 4$	164.09
GMRESR	8	P_{ASC}, $L = 2$	1538.82
GMRESR	8	P_{ASC}, $L = 4$	1600.52
GMRESR	16	P_{ASC}, $L = 2$	544.96
GMRESR	16	P_{ASC}, $L = 4$	569.15
GMRESR	32	P_{ASC}, $L = 2$	248.52
GMRESR	32	P_{ASC}, $L = 4$	286.63
GMRESR	32	P_{ASC}, $L = 8$	358.04

Table 5.11: M6. Comparison of the total CPU-time required by $P_{S,C,hybrid}$ and P_{ASC}.

Finally, table 5.11 and figure 5.11 compares $P_{S,C,hybrid}$ and P_{ASC} for the M6 test case. One can see that the both preconditioners have superlinear speedups, and that $P_{S,C,hybrid}$ outperforms P_{ASC}.

5.5 Concluding Remarks

Up to the author's knowledge, current parallel implementation of MU schemes are based on one-level Schwarz preconditioner and do not consider mesh adaptation procedure. Therefore, it was one of the aim of this thesis to present a complete framework which couples these two aspects with the characteristic aspects of MU schemes.

The main kernels of the $\alpha\Psi$NKS algorithm have been analysed. A data structure, based on a VO 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.

The parallel implementation of one-level and two-level Schwarz preconditionerss are presented. The coarse matrix is constructed automatically in an efficient way, 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 literature for multidimensional upwind residual distribution schemes.

Our numerical experiment show that the Schur Complement as a solver is not competitive in terms of CPU-time with methods which operate on the unreduced matrix. Clearly, the CPU-time results are largely affected by the type of the computer architecture and the char-

acteristics of the processors used in the computations. However, our feeling is that the Schur complement solver gives good results only when the ratio unknowns/number of processors is "low", or ratio CPU speed/communication speed is high. This is because of the cost of the solution of the internal problems. Instead, the approximate Schur complement preconditioners seems an interesting extension of the SC approach. Although the best results are obtained using two-level Schwarz method with agglomeration coarse space, the approximate SC preconditioner can be effectively used to solve the considered problem.

6 Conclusions

The use of domain decomposition methods for the solution of the compressible Euler equations was the main subject of this work. In this respect, the thesis has presented several preconditioning techniques, discussed their spectral properties on a model problem, their implementation on parallel computers with distributed memory, and finally compared their performance.

First, our analysis was focused on 2-level Schwarz preconditioners. We have defined a coarse correction operator that does not require the definition of a coarse grid. The basis functions of the coarse space are built as a linear combination of the basis function of the fine space. This procedure may be reinterpreted as an algebraic manipulation of the fine-grid matrix. The resulting method can be seen as a 2-level black-box method.

A theoretical estimation of the condition number was given for a model problem. This definition of a coarse space results in a less favourable condition number with respect to other definitions of coarse spaces presented in the literature. However, its use presents some advantages for problems defined on unstructured grids for domains with complex shape. The coarse matrix can be 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). The coarse triangulation is not necessary in the aggregation procedure as is the case for other 2-level methods. This opens the possibility of using the proposed method for more general cases with regions of complex geometry without losing the power of a 2-level method. The computational complexity of the proposed preconditioner is smaller since the method is simpler to implement. This simplicity has its origin mainly in the way the restriction and interpolation operators are defined. Moreover, even though the analysis assumes an elliptic functional discretized on

a quasi-uniform mesh of $\mathbb{P}^1$ or $\mathbb{Q}^1$ elements, numerical experiments confirm its applicability to general unstructured meshes and to more general problems.

We have also considered SC methods derived from either element-oriented or vertex-oriented non-overlapping decompositions. We have presented a new approach for the solution of the SC system based on the reinterpretation of the VO decomposition. Instead of defining a coarse space, the main idea is to decompose the original domain in order to have one subdomain which is connected to all the others. This subdomain Ω_Γ, which is reduced to the minimum width of 1 element, allows the definition of global preconditioners. In particular, among the preconditioner reported here, the best choice appears to be a DN preconditioner, for which a Neumann problem is solved in Ω_Γ. Consequently, the preconditioning step requires the solution of only one linear system on a subdomain of a special shape. Numerical tests confirm the good properties of such a decomposition, and comparison with the balancing Neumann/Neumann preconditioner shows that, from the point of view of computational cost, the new approach may be of interest.

We have also described preconditioners based on approximate SC solutions. The key idea is to replace the "exact" SC matrix with an approximate one, which is less expensive to compute and to store in memory. The resulting approximate SC system is solved itself using a few steps of a Krylov subspace accelerator.

The thesis then focused on the application of these preconditioners to the steady compressible Euler system for aerodynamical applications. The space discretisation was based on multidimensional upwind techniques on unstructured grids, which can be interpreted as Petrov-Galerkin finite-element schemes for the solution of CFD problems. We have presented a pseudo-continuation technique to obtain the solution of the nonlinear system of discrete equations. A pseudo time derivative is added to the stationary problem and discretised using the backward Euler method. At each time level, the nonlinear system is solved by one step of the Newton methods. This leads to a large, sparse, non-symmetric linear system, which is solved by resorting to Krylov techniques. The preconditioner arises from domain decomposition methods of both Schwarz and Schur complement type. The resulting framework, completed by mesh adaptation techniques, is called $\alpha\Psi$NKS. The numerical results show that mesh adaptation can effectively improve the quality of the solution obtained using multidimensional upwind discretisations for problems of aeronautical interest.

Finally, we have considered the parallel aspects related to an efficient implementation of the $\alpha\Psi$NKS algorithm on parallel computers with distributed memory. The main computational kernels of this algorithm have been analysed. A data structure, based on a vertex-oriented decomposition of the grid, has been described. The parallel implementation of most of the preconditioners presented in the first part of the thesis is outlined. A master-free mesh adaptation procedure is also presented. The various steps of mesh adaptation (refinement, derefinement, optimisation, reordering and renumbering) and the associated algorithms are described.

Numerical results, obtained on distributed memory parallel computers, are presented for

large scale computations. It is shown that the combination of the Krylov-Schwarz approach with a mesh adaptation procedure leads to an effective and scalable solution of the compressible Euler equations on unstructured grids.

Bibliography

[Abg01] R. Abgrall. Toward the ultimate conservative scheme: Following the quest. *Journal of Computational Physics*, 167:277–315, 2001.

[AGA79] AGARD. Experimental data base for computer program assessment. Technical Report 138, AGARD, 1979.

[AKK99] V. Annamalai, C.S. Krishnamoorthy, and V. Kamakoti. Adaptive finite element analysis on a parallel and distributed environment. *Parallel Computing*, 25:1413–1434, 1999.

[AL97] O. Axelsson and M. Larin. An algebraic multilevel iteration method for finite element matrices. *Journal of Computational and Applied Mathematics*, 89:135–153, 1997.

[AO00] M. Ainsworth and J.T. Oden. *A Posteriori Error Estimation in Finite Element Analysis*. Pure and Applied Mathematics. John Wiley & Sons, New York, 2000.

[AV91] O. Axelsson and P.S. Vassilevski. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioner. *SIAM Journal on Matrix Analysis and Applications*, 4(12):625–644, 1991.

[Axe94] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Cambridge, 1994.

[Bar94] T.J. Barth. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. In *VKI LS 1994-05, Computational Fluid Dynamics*, 1994.

[BBC+94] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[BCF+00] M. Brezina, A.J. Cleary, R.D. Falgout, V.E. Henson, J.E. Jones, T.A. Manteuffel, S.F. McCormick, and J.W. Ruge. Algebraic multigrid based on element interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2000.

[BGMR01] O. Broeker, M.J. Grote, C. Mayer, and A. Reusken. Robust parallel smoothing for multigrid via sparse approximate inverses. *SIAM Journal on Scientific Computing*, 23:1396–1417, 2001.

[Blo98] F. Blom. *Investigations on Computational Fluid-Structure Interaction.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1998.

[BMT96] M. Benzi, C.D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996.

[BR99] J. Bastin and G. Rogé. A multidimensional fluctuation splitting scheme for the three dimensional Euler equations. *Mathematical Modelling and Numerical Analysis*, 33(6):1241–1259, 1999.

[Bra86] A. Brandt. Algebraic multigrid: the symmetric case. *Appl. Math. Comp.*, 19:23–56, 1986.

[BS01] I. Babuška and T. Stroboulis. *The Finite Element Method and its Reliability.* Oxford University Press, Oxford, 2001.

[BSvD99] M. Benzi, D.B. Szyld, and A. van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM Journal on Scientific Computing*, 20(5):1652–1670, 1999.

[BT98] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19(3):968–994, 1998.

[BT99] M. Benzi and M. Tuma. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics: Transactions of IMACS*, 30(2–3):305–340, 1999.

[BV99] M. Brezina and P. Vaněk. A black–box iterative solver based on a two–level Schwarz method. *Computing*, 63:233–263, 1999.

[Car97] G.F. Carey. *Computational Grids: Generation, Adaptation, and Solution Strategies.* Talor & Francis, 1997.

[CFvH+00] A.J. Cleary, R.D. Falgout, v.E. Henson, J.E. Jones, T.A. Manteuffel, S.F. McCormick, G.N. Miranda, and J.W. Ruge. Robustness and scalability of algebraic multigrid. *SIAM Journal on Scientific Computing*, 22(5):1886–1908, 2000.

[CGZ99] T.F. Chan, S. Go, and J. Zou. Boundary treatments for multilevel methods on unstructured meshes. *SIAM Journal on Scientific Computing*, 21(1):46–66, 1999.

[Cho00] E. Chow. A priory sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, 21:1804–1822, 2000.

[CKK02] T.S. Coffey, C.T. Kelly, and D.E. Keyes. Pseudo-transient continuation and differential-algebraic equations. Technical report, ODU, 2002. Submitted to the special issue of the SIAM Journal on Scientific Computing dedicated to papers from the 2002 Copper Mountain Conference on Iterative Methods.

[CKV97] X.-C. Cai, D.E. Keyes, and V. Venkatakrishnan. Newton-Krylov-Schwarz: an implicit solver for CFD. In R. Glowinski et al., editor, *Proceedings of the Eight International Conference on Domain Decomposition Methods*, pages 387–400, 1997.

[CL00] G. Carré and S. Lanteri. Parallel linear multigrid by agglomeration for the acceleration of 3D compressible flow computations on unstructured meshes. *Numerical Algorithms*, 24:309–332, 2000.

[CM94] T.F. Chan and T.P. Mathew. Domain decomposition algorithms. In *Acta Numerica*, pages 61–143. Cambridge University Press, 1994.

[CRDP01] A. Csík, M. Ricchiuto, H. Deconinck, and S. Poedts. Space time residual distribution schemes for hyperbolic conservations laws. In *Proc. 15th AIAA Conference*, 2001.

[CS99] X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:239–247, 1999.

[CSZ96] T.F. Chan., B.F. Smith, and J. Zou. Overlapping Schwarz methods on unstructured meshes using non-matching coarse grids. *Numerische Mathematik*, 73(2):149–167, 1996.

[Cv97] T.F. Chan and H. van der Vorst. Approximate and incomplete factorizations. In D.E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, ICASE/LaRC Interdisciplinary Series in Science and Engineering, pages 167–202. Kluwer, Dordrecht, 1997.

[CZ96] T.F. Chan and J. Zou. A convergence theory of multilevel additive Schwarz methods on unstructured grids. *Numer. Algorithms*, 13:365–398, 1996.

[DES82] R.S. Dembo, S.C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19:400–408, 1982.

[DHP86] H. Deconinck, C. Hirsch, and J. Peuteman. Characteristic decomposition methods for the multidimensional Euler equations. In *Lecture Notes in Physics*, volume 264. Springer-Verlag, 1986.

[DK96] H. Deconinck and B. Koren, editors. *Notes on Numerical Fluid Mechanics*. Vieweg, 1996.

[DM80] U.S. Duff and G.A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1980.

[DSA00] H. Deconinck, K. Sermeus, and R. Abgrall. Status of multidimensional upwind residual distribution schemes and applications in aeronautics. *AIAA Paper*, 2000.

[DSW94] M. Dryja, B.F. Smith, and O.B. Widlund. Schwarz analysis of iterative substructuring algorithms for elliptic problems in three dimensions. *SIAM Jounrnal on Numerical Analysis*, 31(6):1662–1694, 1994.

[DW90] M. Dryja and O.B. Widlund. Towards a unified theory of domain decomposition algorithms for elliptic problems. In T.F. Chan, R. Glowinski, J. Périaux, and O. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–21. SIAM, Philadelphia, PA, 1990.

[DW94] M. Dryja and O.B. Widlund. Domain decomposition algorithms with small overlap. *SIAM Journal on Scientific Computing*, 15(3):604–620, 1994.

[FD96] G. Fagg and J. Dongarra. PVMPI: An integration of PVM and MPI systems. *Calculateurs Parallèles*, 8(2):151–166, 1996.

[Fle76] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Lecture Notes in Mathematics*, volume 506, pages 73–89, Berlin, 1976. Springer-Verlag.

[FN91] R.W. Freund and N.M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numeriske Mathematik*, 60:315–339, 1991.

[For95] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical report, 1995.

[Fre93] R.W. Freund. A transpose–free quasi–minimum residual algorithm for non–hermitian linear systems. *SIAM Journal on Scientific and Statistical Computing*, 14:470–482, 1993.

[FS01] A. Frommer and D.B. Szyld. An algebraic convergence theory for restricted additive Schwarz methods using weighted max norms. *SIAM Journal on Numerical Analysis*, 39:463–479, 2001.

[FSQ94] L. Formaggia, A. Scheinine, and A. Quarteroni. A numerical investigation of Schwarz domain decomposition techniques for elliptic problems on unstructured grids. *Mathematics and Computers in Simulations*, 44:313–330, 1994.

[GH97] M.J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[GKKS01] W.D. Gropp, D.K. Kaushik, D.E. Keyes, and B.F. Smith. High-performance parallel implicit CFD. *Parallel Computing*, 27(4):337–362, 2001.

[GMTK00] W.D. Gropp, L. McInnes, M. Tidriri, and D.E. Keyes. Globalized Newton–Krylov–Schwarz algorithms and software for parallel implicit CFD. *The International Journal of High Performance Computing Applications*, 14(2):102–136, 2000.

[GO89] G.H. Golub and D.P. O'Leary. Some history of the conjugate gradient algorithm and Lanczos algorithms. *SIAM Review*, 102:31–50, 1989.

[GS98] N. Gould and J. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM Journal on Scientific Computing*, 19(2):605–625, 1998.

[Gv97] G.H. Golub and H.A. van der Vorst. Closer to the solution: Iterative linear solvers. In I.S. Duff and G.A. Watson, editors, *The State of the Art in Numerical Analysis*, volume 63, pages 63–92, Oxford, 1997.

[GvL83] G.H. Golub and C.F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland, 1983.

[Hac94] W. Hackbusch. *Iterative Solution of Large Sparse Linear Systems of Equations.* Springer-Verlag, Berlin, 1994.

[Hai91] R. Haimes. Visual3: Interactive unsteady unstructured 3D visualisation. *AIAA Paper*, 91–0794, 1991.

[Hay68] E. Haynsworth. On the Schur complement. *Basel Math. Notes*, 20, 1968.

[HDR01] A. Csík H. Deconinck, K. Sermeus and M. Ricchiuto. Upwind residual distribution schemes for hyperbolic conservation laws. In *1er Congrès National de Mathématiques Appliquées et Industrielles, Pompadour, France*, 2001.

[Hir89] C. Hirsch. *Numerical Computation of Internal and External Flow, Volume 1, Fundamentals of Numerical Discretization.* John Wiley & Sons, New York, 1989.

[Hir90] C. Hirsch. *Numerical Computation of Internal and External Flow, Volume 2, Computational Methods for Inviscid and Viscous Flows.* John Wiley & Sons, New York, 1990.

[HJ88] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2.* Adam Hilger Ltd, Bristol, 1988.

[HJ98] B. Heise and M. Jung. Efficiency, scalability, and robustness of parallel multilevel methods for nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 20(2):553–567, 1998.

[HS52] M. Hestenes and E. Steifel. Method of conjugate gradients for solving linear systems. *Journal of Research*, 20:409–436, 1952.

[IDe01] IDeMAS. Industrial demonstration of accurate and efficient multidimensional upwind algorithms for aerodynamics simulations. Technical report, BRITE/EURAM BE 97-4162, 2001.

[Iss97] E. Issman. *Implicit Solution Strategies for Compressible Flow Equations on Unstructured Grids.* PhD thesis, Université Libre de Bruxelles, Belgium, 1997.

[JKMK00] L. Jenkins, T. Kelley, C.T. Miller, and C.E. Kees. An aggregation-based domain decomposition preconditioner for groundwater flow. Technical Report TR00–13, Department of Mathematics, North Carolina State University, 2000.

[Kel95] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, Philadelphia, 1995.

[KK97] G. Karypis and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Department of Computer Science, University of Minnesota, 1997.

[KK98a] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[KK98b] C.T. Kelley and D.E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35(2):508–523, 1998.

[KK99] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[KKS99] D.K. Kaushik, D.E. Keyes, and B.F. Smith. Newton-Krylov-Schwarz methods for aerodynamic problems: Compressible and incompressible flows on unstructured grids. In C.-H. Lai et al., editor, *Proceedings of the 11th International Conference on Domain Decomposition Methods.* Domain Decomposition Press, Bergen, 1999.

[KMJK01] C.E. Kees, C.T. Miller, E.W. Jenkins, and C.T. Kelley. Versatile multilevel Schwarz preconditioners for multiphase flow. Technical Report CRSC-TR01-32, Center for Research in Scientific Computation, North Carolina State University, 2001.

[Lan52] C. Lanczos. Chebyshev polynomials in the solution of large-scale linear problems. In *Proceedings of the ACM*, pages 123–133, 1952.

[Leo51] W. Leontief. *The structure of the American Economy.* Oxford University Press, New York, 1951.

[Lev02] R.J. Leveque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, 2002.

[LM00] C.-K. Li and R. Mathias. Extremal characterisations of the Schur complement and resulting inequalities. *SIAM Review*, 42(2):233–246, 2000.

[LR00] P. Leyland and R. Richter. Completely parallel compressible flow simulations using adaptive unstructured meshes. *Computer Methods in Applied Mechanics and Engineering*, 184:467–483, 2000.

[LT00] C. Lasser and A. Toselli. An overlapping domain decomposition preconditioner for a class of discontinuous Galerkin approximations of advection-diffusion problems. Technical Report 810, Dept. of Computer Science, Courant Institute, 2000. To appear in Mathematics of Computation.

[LT01] C. Lasser and A. Toselli. Convergence of some two-level overlapping domain decomposition preconditioners with smoothed aggregation coarse spaces. Technical Report TUM-M0109, Technische Universität München, 2001.

[Man93] J. Mandel. Balancing domain decomposition. *Comm. Appl. Numer. Methods*, 9:233–241, 1993.

[Mav96] D.J. Mavriplis. Agglomeration multigrid solver for the Reynolds-averaged-Navier-Stokes equations on unstructured meshes. *International Journal for Numerical Methods in Fluids*, 23:527–544, 1996.

[Mav98] D.J. Mavriplis. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. Technical Report TR-98-6, ICASE, 1998.

[Mav00] D.J. Mavriplis. Parallel perfomance investigation of an unstructured mesh Navier-Stokes solver. Technical Report TR-00-13, ICASE, 2000.

[Mav02] D.J. Mavriplis. An assessment of linear versus non-linear multigrid methods for unstructured mesh solvers. *Journal of Computational Physics*, 175:302–325, 2002.

[MS83] J. Mandel and B. Sekerka. A local convergence proof for the iterative aggregation method. *Linear Algebra and its Applications*, 51:163–172, 1983.

[MvdV77] J.A. Meijerink and H. A. var der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 21:148–162, 1977.

[NAWK95] 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.

[OR70] J.M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[Ort88] J. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.

[Pai95] H. Paillère. *Multidimensional Upwind Residual Distribution Schemes for the Euler and Navier-Stokes Equations on Unstructured Grids*. PhD thesis, Université Libre de Bruxelles, Belgium, 1995.

[PD97] H. Paillère and H. Deconinck. Compact cell vertex convection schemes on unstructured meshes. In *Notes on Numerical Fluid Mechanics*, volume 57, pages 1–49. Vieweg, 1997.

[PDvdW97] H. Paillère, H. Deconinck, and E. van der Weide. Upwind residual distribution methods for compressible flows: an alternative to finite volume and finite element methods. In *VKI LS 1997-02*. 1997.

[PSFQ97] L. Paglieri, A. Scheinine, L. Formaggia, and A. Quarteroni. Parallel conjugate gradient with Schwarz preconditioner applied to fluid dynamics problems. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics, Algorithms and Results using Advanced Computer, Proceedings of Parallel CFD'96*, pages 21–30, 1997.

[QSS00] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer-Verlag, New York, 2000.

[QSS+03] A. Quarteroni, M. Sala, M.L. Sawley, N. Parolini, and G. Cowles. Mathematical modelling and visualisation of complex three-dimentional flows. In *Mathematics and Visualization*. Springer-Verlag, 2003.

[QSV03] A. Quarteroni, M. Sala, and A. Valli. The swiss carpet domain decomposition preconditioner. 2003. In preparation.

[QV94] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer-Verlag, Berlin, 1994.

[QV99] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Oxford, 1999.

[Ric93] R. Richter. *Schémas de Capture de Discontinuités en Maillage Non-Structuré avec Adaptation Dynamique: Applications en Aérodynamique*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1993.

[RL95] 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*, 20:853–868, 1995.

[Roe81] P.L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journ. Comp. Physics*, 43(2), 1981.

[RS85] J.W. Rube and K. Stüben. Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG). In D. J. Paddon and H. Holstein, editors, *Multigrid Methods for Integral and Differential Equations, The Institute of Mathematics and its Application Conference Series*, 3, pages 169–212. Claredon Press, Oxford, 1985.

[RT91] Y.-H. De Roeck and P. Le Tallec. Analisys and test of a local domain decomposition. In R. Glowinsky, Y.-A. Kuznetsov, G.-A. Meurant, J. Périaux, and O.-B. Widlund, editors, *Fourth International symposium on Domain Decomposition Methods for Partial differential Equations*, pages 112–128. SIAM, Philadelphia, 1991.

[Saa93] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14:461–469, 1993.

[Saa96a] Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 4(17):830–847, 1996.

[Saa96b] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS, Boston, 1996.

[SBG96] B.F. Smith, P. Bjorstad, and W.D. Gropp. *Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambrige, 1996.

[Sch70] H.-A. Schwarz. Uber einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, 1870.

[SD02] K. Sermeus and H. Deconinck. An entropy fix for the multidimensional upwind residual distribution schemes. In *CFD2002: 10th Annual Conference of the CFD Society of Canada*, 2002.

[SDMS99] E. Strohmaier, J.J. Dongarra, H.W. Meuer, and H.D. Simon. The marketplace of high-performance computing. *Parallel Computing*, 25:1517–1544, 1999.

[SDR91] R. Struijs, H. Deconinck, and P. L. Roe. Fluctuation Splitting Schemes for the 2D Euler Equations. In *VKI LS 1991-01, Computational Fluid Dynamics*, 1991.

[SF01] M. Sala and L. Formaggia. Parallel Schur and Schwarz based preconditioners and agglomeration coarse corrections for CFD problems. Technical Report 15, DMA-EPFL, 2001.

[SF02] M. Sala and L. Formaggia. Algebraic coarse grid operators for domain decomposition based preconditioners. In P. Wilders, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, editors, *Parallel Computational Fluid Dynamics – Practice and Theory*, pages 119–126. Elsevier Science, The Netherlands, 2002.

[SL00] Y. Savoy and P. Leyland. Parallel mesh adaptation for unstructured grids within the IDeMas project. Technical report, IMHEF-DGM EPFL, 2000.

[SL02] M. Sala and P. Leyland. A parallel adaptive Newton-Krylov-Schwarz method for the 3D compressible Euler equations. Technical Report 9, EPFL-IMA, 2002.

[SLD92] H. Steve, M. Lallemand, and A. Dervieux. Unstructured multigridding by volume agglomeration: Current status. *Computers and Fluids*, 21(3):397–433, 1992.

[Son89] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10:36–52, 1989.

[SS86] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[SS02] Y. Saad and B. Suchomel. ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numer. Linear Algebra Appl.*, 5:359–378, 2002.

[Stü01] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2):281–309, 2001.

[Str94] R. Struijs. *A Multi-Dimensional Upwind Discretization Method for the Euler Equations on Unstructured Grids.* PhD thesis, The University of Delft, The Netherlands, 1994.

[Sv00] Y. Saad and H. van der Vorst. Iterative solution of linear systems in the 20-th century. *J. Comp. Appl. Math.*, pages 1–33, 2000.

[SZ99] Y. Saad and J. Zhang. BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 20:2103–2121, 1999.

[Tal94] P. Le Tallec. Domain decomposition methods in computational mechanics. In J.T. Oden, editor, *Computational Mechanics Advances*, volume 1, pages 121–220. North-Holland, 1994.

[THHS99] R. Tuminaro, M. Heroux, S. Hutchison, and J. Shadid. Official Aztec user's guide: Version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories, Albuquerque, NM, Dec 1999.

[TT00] R.S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In J. Donnelley, editor, *Supercomputing 2000 Proceedings*, 2000.

[Var00] R. Varga. *Matrix Iterative Analysis, Second Edition.* Springer-Verlag, Berlin, 2000.

[VBM01] P. Vanek, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, 2001.

[VBT99] P. Vanek, M. Brezina, and R. Tezaur. Two-grid method for linear elasticity on unstructured meshes. *SIAM Journal on Scientific Computing*, 21(3):900–923, 1999.

[VCR00] D. Vanderstraeten, Á. Csík, and D. Roose. An expert-system to control the CFL number of implicit upwind methods. Technical report, Katholieke Universiteit Leuven, Department of Computer Science, Belgium, 2000.

[vdV92] H. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[vdW98] E. van der Weide. *Compressible Flow Simulations on Unstructured Grids using Multi-Dimensional Upwind Schemes.* PhD thesis, Université Libre de Bruxelles, Belgium, 1998.

[Ver96] R. Verfürth. *A Review of a Posteriori Error Estimation and Adaptive Mesh Refinement Techniques.* Wiley-Teubner, 1996.

[VM94] V. Venkatakrishnan and D.J. Mavriplis. Agglomeration multigrid for the 3D Euler equations. *AIAA Paper*, 94-0069, 1994.

[VV94] H. Van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numer. Linear Algebra Appl.*, 1:369–386, 1994.

[WDH+99] D.E. Womble, S.S. Dosanjh, B. Hendrickson, M.A. Heroux, S.J. Plimpton, J.L. Tomkins, and D.S. Greenberg. Massively parallel computing: a Sandia perspective. *Parallel Computing*, 25:1853–1876, 1999.

[Wei92] M.A. Weiss. *Data Structure and Algorithm Analysis*. The Benjamin/Cummings Publishing Company, Inc., 1992.

[Wes92] P. Wesseling. *An Introduction To Multigrid Methods*. Wiley, New York, 1992.

[WZZ99] B.-Y. Wang, X. Zhang, and F. Zhang. Some inequalities on generalized Schur complements. *Linear Algebra and its Applications*, 302/303:163–172, 1999.

[XZ98] J. Xu and J. Zou. Some nonoverlapping domain decomposition methods. *SIAM Review*, 40:857–914, 1998.

[Zha98] J. Zhang. A sparse approximate inverse technique for parallel preconditioning of general sparse matrices. Technical Report 281-98, Department of Computer Science, University of Kentucky, Lexington, KY, 1998.

[Zha00] J. Zhang. Preconditioned Krylov subspace methods for solving nonsymmetric matrices from CFD applications. *Computer Methods in Applied Mechanics and Engineering*, 189(3):825–840, 2000.

Marzio Sala est né à Rho (MI) le 27 decembre 1974. Il a effectué son école secondaire au Lycée C. Cavalleri où il a obtenu en 1993 son Baccalauréat Scientifique. Ensuite, il a poursuivi ses études à l'Ecole Polytechnique de Milan pour obtenir en 1999 un diplôme d'ingénieur en aérospatial. Depuis septembre 1999, il a été engagé comme assistant dans la chaire de modélisation et calcul scientifique du Professeur Alfio Quarteroni. Il a participé à plusieurs conférences internationales et il est auteur ou co-auteur des publications suivantes:

- A. Quarteroni, M. Sala, M.-L. Sawley, N. Parolini, and G. Cowles, *Mathematical Modelling and Visualisation of Complex Three-dimensional Flows*. Accepted for publication in the book series *Mathematics and Visualisation*, Springer-Verlag, Berlin, 2003.

- M. Sala and P. Leyland. A Parallel Adaptive Newton-Krylov-Schwarz Method for the 3D Compressible Euler Equations, *EPFL-IMA Report* n. 7, Lausanne, 2002.

- M. Sala, An Algebraic 2-level Domain Decomposition Preconditioner with Applications to the Compressible Euler Equations. Accepted for publication in *International Journal for Numerical Methods in Fluids*.

- M. Sala and L. Formaggia, Algebraic Coarse Grid Operators for Domain Decomposition Based Preconditioners. Parallel Computational Fluid Dynamics - Practise and Theory, P. Wilders, A. Ecer, J. Periaux, N. Satofuka, and P. Fox editors, pages 119–126. Elsevier Science, The Netherlands, 2002.

- M. Sala and L. Formaggia, Parallel Schur and Schwarz Based Preconditioners and Agglomeration Coarse Corrections for CFD Problems. *EPFL-DMA Report* n. 15, Lausanne, 2001.

- M. Sala, Schur Complement Domain Decomposition Preconditioners for Compressible Flow Computations. *Domain Decomposition Methods in Science and Engineering*. N. Debit, M. Garbey, R. Hoppe, D. Keyes, Y. Kuznetsov, J. Périaux, editors, pages 504–511. Barcelona, 2002.

- IDeMAS Technical Reports 4.3 (with F. Maggio), 4.4 and 4.5 (with L. Formaggia), Validation Reports 6.1.3/6.2.3 (with L. Formaggia) and 7.3 (with V. Selmin, L. Formaggia, and F. Maggio), IDeMAS, 2000–2002.