4

Domain Decomposition Techniques

Luca Formaggia[1], Marzio Sala[2], and Fausto Saleri[1]

[1] MOX, Dipartimento di Matematica "F. Brioschi", Politecnico di Milano, Italy
[luca.formaggia,fausto.saleri]@polimi.it
[2] Sandia National Laboratories, Albuquerque, USA
msala@sandia.gov

Summary. We introduce some parallel domain decomposition preconditioners for iterative solution of sparse linear systems like those arising from the approximation of partial differential equations by finite elements or finite volumes. We first give an overview of algebraic domain decomposition techniques. We then introduce a preconditioner based on a multilevel approximate Schur complement system. Then we present a Schwarz-based preconditioner augmented by an algebraic coarse correction operator. Being the definition of a coarse grid a difficult task on unstructured meshes, we propose a general framework to build a coarse operator by using an agglomeration procedure that operates directly on the matrix entries. Numerical results are presented aimed at assessing and comparing the effectiveness of the two methodologies. The main application will concern computational fluid dynamics (CFD), and in particular the simulation of compressible flow around aeronautical configurations.

4.1 Introduction

Modern supercomputers are often organized as a distributed environment and an efficient solver for partial differential equations (PDEs) should exploit this architectural framework. Domain decomposition (DD) techniques are a natural setting to implement existing single processor algorithm in a parallel context.

The basic idea, as the name goes, is to decompose the original computational domain Ω into subdomains $\Omega_i, i = 1, \ldots .M$, which may or may not overlap, and then rewrite the global problem as a "sum" of contributions coming from each subdomain, which may be computed in parallel. Parallel computing is achieved by distributing the subdomain to the available processors; often, the number of subdomains equals the number of processors, even if this is not, in general, a requirement. Clearly one cannot achieve a perfect (ideal) parallelism, since interface conditions between subdomains are necessary to recover the original problem, which introduce the need of inter-processor communications. The concept of parallel efficiency is clearly stated for the case of homogeneous systems (see [13]). An important concept is that of scalability: an algorithm is scalable if its performance is proportional to the number of processor employed. For the definition to make sense we should keep the processor

workload approximately constant, so the problem size has to grow proportionally to the number of processor.

This definition is only qualitative and indeed there is not a quantitative definition of scalability which is universally accepted, and a number of scalability models proposed in the last years [14]. They are typically based on the selection of a measure which is used to characterize the performance of the algorithm, see for instance [22, 23, 11]. We may consider the algorithm *scalable* if the ratio between the performance measure and the number of processors is sub-linear. In fact, the ideal value of this ratio would be 1. Yet, since this ideal value cannot be reached in practice, a certain degradation should be tolerated.

Typical quantities that have been proposed to measure system performance include CPU time, latency time, memory, etc. From the user point of view, global execution time is probably the most relevant measure. A possible definition is the following. If $E(s, N)$ indicates the execution time for a problem of size s when using N processor on a given algorithm, then the scalability from N to $M > N$ processors is given by

$$S_{M,N} = \frac{E(M\gamma, M)}{E(N\gamma, N)},$$

where γ is the size of the problem on a single processor.

A few factors may determine the loss of scalability of a parallel code: the cost of inter-processor communication; the portion of code that has to be performed in a scalar fashion, may be replicated on each processor (for instance i/o if your hardware does not support parallel i/o). A third factor is related to a possible degradation of the parallel algorithm as the number of subdomain increases. In this work we will address exclusively the latter aspect, the analysis of the other two being highly dependent on the hardware. In particular, since we will be concerned with the solution of linear systems by parallel iterative schemes, the condition number of the parallel solver [17] is the most important measure related to the algorithm scalability properties. In this context, the algorithm is scalable if the condition number remains (approximately) constant as the ratio between problem size and number of subdomains is kept constant. In a domain decomposition method applied to the solution of PDE's (by finite volumes or finite elements) in $\mathbb{R}^d$ this ratio is proportional to $(H/h)^d$, being H and h the subdomain and mesh linear dimension, respectively. We are assuming a partition with subdomains of (approximately) the same size and a quasi-uniform mesh.

Domain decomposition methods may be classified into two main groups [17, 21]. The first includes methods that operate on the differential problem, we will call them *differential domain decomposition methods*. Here, a differential problem equivalent to the single domain one is written on the decomposed domain. Conditions at the interface between subdomains are recast as boundary conditions for local differential problems on each Ω_i. Then, the discretisation process is carried out on each subdomain independently (even by using different discretisation methods, if this is considered appropriate).

The second group includes the DD techniques that operates at the algebraic level. In this case the discretisation is performed (at least formally) on the original, single

domain, problem and the decomposition process is applied on the resulting algebraic system. The latter technique has the advantage of being "problem independent" and may often be interpreted as a preconditioner of the global solver. The former, however, may better exploit the characteristics of the differential problem to hand and allows to treat problems of heterogeneous type more naturally. We refer to the relevant chapter in [17] for more insights on heterogeneous DD techniques.

In this chapter, we deal with DD schemes of algebraic type. In particular, we address methods suited to general problems, capable of operating on unstructured mesh based discretisations. For a more general overview of the DD method the reader may refer to the already cited literature, the review paper [5] and the recent monograph [25].

We will focus on domain decomposition techniques that can be applied to the numerical solution of PDEs on complex, possibly three dimensional, domains. We will also consider discretisations by finite element or finite volume techniques, on unstructured meshes. The final result of the discretisation procedure is eventually a large, sparse linear system of the type

$$A\mathbf{u} = \mathbf{f}\,, \tag{4.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a sparse and often non-symmetric and ill conditioned real matrix. Indeed, also non-linear problems are usually treated by an iterative procedure (e.g. a Newton iteration) that leads to the solution of a linear system at each iteration. This is the case, for instance, of implicit time-advancing schemes for computational fluid dynamics (CFD) problems.

The decomposition of the domain will induce a corresponding block decomposition of the matrix A and of the vector $\mathbf{f}$. This decomposition may be exploited to derive special parallel solution procedures, or parallel preconditioners for iterative schemes for the solution of system (4.1). Perhaps the simplest preconditioner is obtained using a block-Jacobi procedure, where each block is allocated to a processor and is possibly approximated by an incomplete factorization [18] (since usually an exact factorization is too expensive). This approach may work well for simple problems, yet its performance degrades rapidly as the size of the matrix increases, leading to poor scalability properties. Other popular techniques are the Schwarz methods with a coarse grid correction [2] and the preconditioners based on the Schur complement system, like the balancing Neumann/Neumann [24], the FETI [10, 25] and the wire-basket method [1, 21]. In this work we will address preconditioners based either on an approximate Schur complement (SC) system or on Schwarz techniques, because of their generality and relatively simple implementation.

Schwarz iterations is surely one of the DD based parallel preconditioner with the simplest structure. In its basic form, it is equivalent to a block-Jacobi preconditioner, where each block is identified by the set of unknowns contained in each subdomain. In order to improve the performance of Schwarz iterations, the partitions of the original domain are extended, so that they overlap and the overlapping region acts as means of communication among the subdomains. In practice, the domain subdivision is commonly carried out at discrete level, that is by partitioning

the computational mesh. In the minimal overlap version of the Schwarz method the overlap between subdomains is reduced to a single layer of elements. Although a bigger overlap may improve convergence, a minimal overlap allows to use the same data structure normally used for the parallel matrix-vector multiplication, thus saving memory. However, the scalability is scarce and a possible cure consists in augmenting the preconditioner by a coarse operator, either in an additive or in a multiplicative fashion [21] . The coarse operator may be formed by discretising the problem to hand on a much coarser mesh. The conditions by which a coarse mesh is admissible and is able to provide an appropriate coarse operator have been investigated in [3]. Another possible way to construct the coarse operator is to form a reduced matrix by resorting to a purely algebraic procedure [28, 20]. We will here describe a rather general setting for the construction of the coarse operator.

The set up of the Schur complement system in a parallel setting is only slightly more involved. The major issue here is the need of preconditioning the Schur complement system in order to avoid the degradation of the condition number as the number of subdomain increases. We will here present a technique to build preconditioners for the Schur complement system starting from a preconditioner of the original problem.

The chapter is organized as follows. Schur complement methods are introduced in Sections 4.2 and 4.3. Schwarz methods are detailed in Section 4.4. Numerical results for a model problem and for the solution of the compressible Euler equations are presented in Section 4.5. Section 4.6 gives some further remarks on the techniques that have been presented.

4.2 The Schur Complement System

Let us consider again (4.1) which we suppose has been derived by a finite element discretisation of a differential problem posed on a domain $\Omega \subset \mathbb{R}^d, d = 2, 3$. Indeed, the considerations of this Section may be extended to other type of discretisations as well, for instance finite volumes, yet we will here focus on a finite element setting. More precisely, we can think of (4.1) as being the algebraic counterpart of a variational boundary value problem which reads: find $u_h \in V_h$ such that

$$a(u_h, v_h) = (f, v_h) \qquad \forall v_h \in V_h\,, \tag{4.2}$$

where V_h is a finite element space, a the bilinear form associated to the differential problem to hand and $(f, v) = \int_\Omega f v d\Omega$ is the standard L^2 product.

We consider a decomposition of the domain Ω made in the following way. We first triangulate Ω and indicate by $\mathcal{T}_h^{(\Omega)}$ the corresponding mesh. For the sake of simplicity we assume that the boundary of Ω coincides with the boundary of the triangulation and we consider the case where the degrees of freedom of the discrete problem are located at mesh vertices, like in linear finite elements. In particular, a partition into two subdomains is carried out by splitting $\mathcal{T}_h^{(\Omega)}$ into 3 parts, namely $\mathcal{T}_h^{(1)}$, $\mathcal{T}_h^{(2)}$ and $\Gamma^{(1,2)}$ such that $\mathcal{T}_h^{(1)} \cup \mathcal{T}_h^{(2)} \cup \Gamma^{(1,2)} = \mathcal{T}_h^{(\Omega)}$. We may associate to $\mathcal{T}_h^{(1)}$ and $\mathcal{T}_h^{(2)}$ the two disjoint subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$ formed by the interior

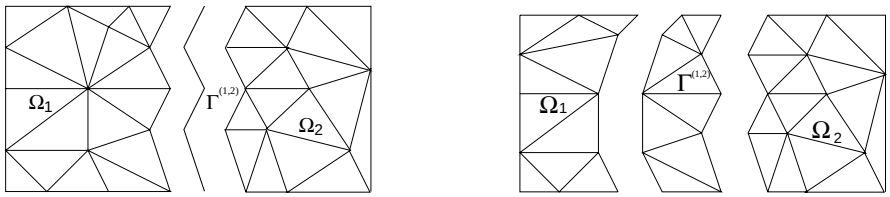

Fig. 4.1. Example of element-oriented (left) and vertex-oriented (right) decomposition.

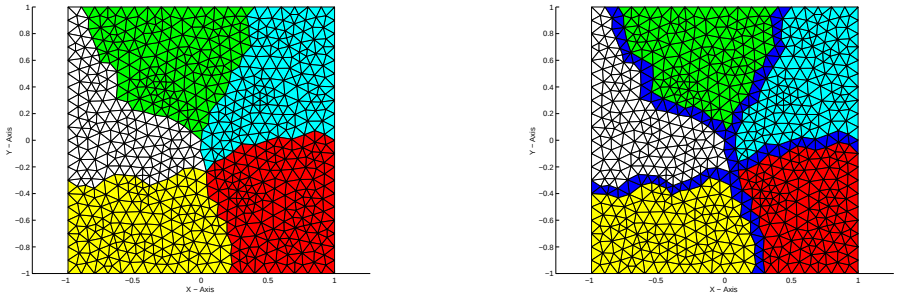

Fig. 4.2. Example of element-oriented (left) and vertex-oriented (right) decomposition in the case of a partition of Ω into several subdomains.

of the union of the elements of $\mathcal{T}_h^{(1)}$ and $\mathcal{T}_h^{(2)}$ respectively, while $\Gamma^{(1,2)} \equiv \Gamma^{(2,1)}$ is clearly equal to $\Omega \setminus (\Omega^{(1)} \cup \Omega^{(2)})$.

Two notable cases are normally faced, namely

- $\Gamma^{(1,2)}$ reduces to a finite number of disjoint measurable $d-1$ manifolds. An example of this situation is illustrated in the left drawing of Figure 4.1, where $\Gamma^{(1,2)} = \overline{\Omega}^{(1)} \cap \overline{\Omega}^{(2)}$, i.e. $\Gamma^{(1,2)}$ is the common part of the boundary of $\Omega^{(1)}$ and $\Omega^{(2)}$. This type of decomposition is called *element oriented* (EO) decomposition, because each element of $\mathcal{T}_h$ belongs exclusively to one of the subdomains $\overline{\Omega}^{(i)}$, while the vertices laying on $\Gamma^{(1,2)}$ are shared between the subdomains triangulations.
- $\Gamma^{(1,2)} \subset \mathbb{R}^d, d = 2, 3$ is formed by one layer of elements of the original mesh laying between $\Omega^{(1)}$ and $\Omega^{(2)}$. In Figure 4.1, right, we show an example of such a decomposition, which is called *vertex oriented* (VO), because each vertex of the original mesh belongs to just one of the two subdomains $\overline{\Omega}^{(i)}$. We may also recognize two extended, overlapping, sub-domains: $\widetilde{\Omega}^{(1)} = \Omega^{(1)} \cup \Gamma^{(1,2)}$ and $\widetilde{\Omega}^{(2)} = \Omega^{(2)} \cup \Gamma^{(1,2)}$. We have here the minimal overlap possible. Thicker overlaps may be obtained by adding more layers of elements to $\Gamma^{(1,2)}$.

Both decompositions may be readily extended to any number of subdomains, as shown in Figure 4.2.

The choice of a VO or EO decomposition largely affects the data structures used by the parallel code. Sometimes, the VO approach is preferred since the transition region $\Gamma^{(1,2)}$ may be replicated on the processors which holds $\Omega^{(1)}$ and $\Omega^{(2)}$ respectively and provides a natural means of data communication among processor which also allow to implement a parallel matrix-vector product. Furthermore, the local matrices may be derived directly from the global matrix A, with no work needed at the level of the (problem dependent) assembly process. For this reason the VO technique is also the matter of choice of many parallel linear algebra packages. We just note that for the sake of simplicity, in this work we are assuming that the graph of the matrix A coincides with the computational mesh (which is the case of a linear finite element approximation of a scalar problem). This means that if the element a_{ij} of the matrix is different from zero, then the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ of the computational mesh are connected by an edge. However, the techniques here proposed may be generalized quite easily to more general situations.

We now introduce some additional notations. The nodes at the intersection between subdomains and $\Gamma^{(1,2)}$ are called *border* nodes. More precisely, those in $\mathcal{T}_h^{(i)} \cap \Gamma^{(1,2)}$ are the border nodes of the domain $\Omega^{(i)}$. A node of $\mathcal{T}_h^{(i)}$ which is not a border node is said to be *internal* to $\Omega^{(i)}, i = 1, 2$. We will consistently use the subscripts I and B to indicate internal and border nodes, respectively, while the superscript (i) will denote the subdomain we are referring to. Thus, $\mathbf{u}_I^{(i)}$ will indicate the vector of unknowns associated to nodes internal to $\Omega^{(i)}$, while $\mathbf{u}_B^{(i)}$ is associated to the border nodes. For ease of notation, in the following we will often avoid to make a distinction between a domain Ω and its triangulation $\mathcal{T}_h$ whenever it does not introduce any ambiguity.

4.2.1 Schur complement system for EO domain decompositions

Let us consider again the left picture of Figure 4.1. For the sake of simplicity, we assume that the vector $\mathbf{u}$ is such that the unknowns associated to the points $\mathbf{u}_I^{(1)}$ internal to $\Omega^{(1)}$ are numbered first, followed by those internal to $\Omega^{(2)}$ ($\mathbf{u}_I^{(2)}$), and finally by those on $\Gamma^{(1,2)}$ ($\mathbf{u}_B = \mathbf{u}_B^{(1)} = \mathbf{u}_B^{(2)}$) (obviously this situation can always be obtained by an appropriate numbering of the nodes). Consequently, equation (4.1) can be written in the following block form

$$\begin{pmatrix} A_{II}^{(1)} & 0 & A_{IB}^{(1)} \\ 0 & A_{II}^{(2)} & A_{IB}^{(2)} \\ A_{BI}^{(1)} & A_{BI}^{(2)} & A_{BB}^{(1)} + A_{BB}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_I^{(1)} \\ \mathbf{u}_I^{(2)} \\ \mathbf{u}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_I^{(1)} \\ \mathbf{f}_I^{(2)} \\ \mathbf{f}_B \end{pmatrix} . \tag{4.3}$$

Here, $A_{II}^{(i)}$ contains the elements of A involving the nodes internal to subdomain $\Omega^{(i)}$, while the elements in $A_{IB}^{(i)}$ are formed by the contribution of the boundary nodes to the rows associated to the internal ones. Conversely, in $A_{BI}^{(i)}$ we have the terms that link border nodes with internal ones (for a symmetric matrix $A_{BI}^{(i)} = A_{IB}^{(i)^T}$). Finally, $A_{BB} = A_{BB}^{(1)} + A_{BB}^{(2)}$ is the block that involves only border nodes,

which can be split into two parts, each built from the contribution coming from the corresponding subdomain. For instance, in a finite element procedure we will have

$$[A_{BB}^{(i)}]_{kj} = a(\phi_k|_{\Omega^{(i)}}, \phi_j|_{\Omega^{(i)}}) \,, \tag{4.4}$$

where ϕ_k, ϕ_j are the finite element shape functions associated to border nodes k and j, respectively, restricted to the subdomain $\Omega^{(i)}$ and a is the bilinear form associated to the differential problem under consideration. An analogous splitting involves the right hand side $\mathbf{f}_B = \mathbf{f}_B^{(1)} + \mathbf{f}_B^{(2)}$.

A formal LU factorization of (4.3) leads to

$$\begin{pmatrix} A_{II}^{(1)} & 0 & 0 \\ 0 & A_{II}^{(2)} & 0 \\ A_{BI}^{(1)} & A_{BI}^{(2)} & I \end{pmatrix} \begin{pmatrix} I & 0 & A_{II}^{(1)^{-1}} A_{IB}^{(1)} \\ 0 & I & A_{II}^{(2)^{-1}} A_{IB}^{(2)} \\ 0 & 0 & S_h \end{pmatrix} \begin{pmatrix} \mathbf{u}_I^{(1)} \\ \mathbf{u}_I^{(2)} \\ \mathbf{u}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_I^{(1)} \\ \mathbf{f}_I^{(2)} \\ \mathbf{f}_B \end{pmatrix} \,,$$

where S_h is the **Schur complement** (SC) matrix, given by $S_h = S_h^{(1)} + S_h^{(2)}$ where

$$S_h^{(i)} = A_{BB}^{(i)} - A_{BI}^{(i)} A_{II}^{(i)^{-1}} A_{IB}^{(i)} \tag{4.5}$$

is the contributions associated to the subdomain $\Omega^{(i)}$, for $i = 1, 2$. We may note that we can solve the system (at least formally) using the following procedure. We first compute the border values $\mathbf{u}_B$ by solving

$$S_h \mathbf{u}_B = \mathbf{g}, \tag{4.6}$$

where $\mathbf{g} = \mathbf{g}^{(1)} + \mathbf{g}^{(2)}$, with

$$\mathbf{g}^{(i)} = \mathbf{f}_B^{(i)} - A_{BI}^{(i)} A_{II}^{(i)^{-1}} \mathbf{f}_I^{(i)} \quad , \; i = 1, 2 \,.$$

Then, we build the internal solutions $\mathbf{u}_I^{(i)}$, for $i = 1, 2$, by solving the two completely independent linear systems

$$A_{II}^{(i)} \mathbf{u}_I^{(i)} = \mathbf{f}_I^{(i)} - A_{IB}^{(i)} \mathbf{u}_B \quad , \; i = 1, 2 \,. \tag{4.7}$$

The second step is perfectly parallel. Furthermore, thanks to the splitting of S_h and $\mathbf{g}$, a parallel iterative scheme for the solution of (4.6) can also be devised. However, some communications among subdomains is here required. The construction of the matrices $A_{BB}^{(i)}$ in (4.4) 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. Therefore, this technique is less suited for "black box" parallel linear algebra packages. More details on the parallel implementation of the Schur complement system are given in Section 4.2.3.

4.2.2 Schur complement system for VO domain decompositions

Let us consider again problem (4.1) where we now adopt a VO partition into two subdomains like the one on the right of Figure 4.1. The matrix A can be written again in a block form, where this time we have

$$A\mathbf{u} = \begin{pmatrix} A_{II}^{(1)} & A_{IB}^{(1)} & 0 & 0 \\ A_{BI}^{(1)} & A_{BB}^{(1)} & 0 & E^{(1,2)} \\ 0 & 0 & A_{II}^{(2)} & A_{IB}^{(2)} \\ 0 & E^{(2,1)} & A_{BI}^{(2)} & A_{BB}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_I^{(1)} \\ \mathbf{u}_B^{(1)} \\ \mathbf{u}_I^{(2)} \\ \mathbf{u}_B^{(2)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_I^{(1)} \\ \mathbf{f}_B^{(1)} \\ \mathbf{f}_I^{(2)} \\ \mathbf{f}_B^{(2)} \end{pmatrix} . \tag{4.8}$$

Here, the border nodes have been subdivided in two sets: the set $B^{(1)}$ of nodes of $\Gamma^{(1,2)}$ which lay on the boundary of $\Omega^{(1)}$ (the *border nodes* of $\Omega^{(1)}$) and the analogous set $B^{(2)}$ of the border nodes of $\Omega^{(2)}$. Correspondingly, we have the blocks $\mathbf{u}_B^{(1)}$ and $\mathbf{u}_B^{(2)}$ in the vector of unknowns and $\mathbf{f}_B^{(1)}$ and $\mathbf{f}_B^{(2)}$ in the right hand side. The entries in $E^{(i,j)}$ are the contribution to the equation associated to nodes in $B^{(i)}$ coming from the nodes in $B^{(j)}$. We call the nodes in $B^{(j)}$ contributing to $E^{(i,j)}$ *external nodes* of domain $\Omega^{(i)}$.

The nodes internal to $\Omega^{(i)}$ are the nodes of the triangulation $\mathcal{T}_h^{(i)}$ whose "neighbors" all belong to $\Omega^{(i)}$. In a matrix-vector product, values associated to internal nodes may be updated without communication with the adjacent subdomains. The update of the border nodes requires instead the knowledge of the values at the corresponding external nodes (which are in fact border nodes of neighboring subdomains). This duplication of information lends itself to efficient implementation of inter-processor communications.

Analogously to the previous section we can construct a Schur complement system operating on the border nodes, obtaining

$$S_h \mathbf{u}_B = \begin{pmatrix} S_h^{(1)} & E^{(1,2)} \\ E^{(2,1)} & S_h^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_B^{(1)} \\ \mathbf{u}_B^{(2)} \end{pmatrix} = \begin{pmatrix} \mathbf{g}^{(1)} \\ \mathbf{g}^{(2)} \end{pmatrix} ,$$

where $S_h^{(1)}$ and $S_h^{(2)}$ are defined as in (4.5). Note, however, that now the entries in $A_{BB}^{(i)}$, $i = 1, 2$, are equal to the corresponding entries in the original matrix A. Thus they can be built directly from A as soon as the topology of the domain decomposition is known.

Once we have computed the border values $\mathbf{u}_B$, the internal solutions $\mathbf{u}_I^{(i)}, i = 1, 2$, are obtained by solving the following independent linear systems,

$$A_{II}^{(i)} \mathbf{u}_I^{(i)} = \mathbf{f}_I^{(i)} - A_{IB}^{(i)} \mathbf{u}_B^{(i)}, \quad i = 1, 2 .$$

In Figure 4.3 we report the sparsity pattern of S_h in the case of a decomposition with 2 subdomains.

This procedure, like the previous one, can be generalized for an arbitrary number of subdomains. If we have M subdomains the decomposition of system (4.8) may be written in a compact way as

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} \mathbf{u}_I \\ \mathbf{u}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_I \\ \mathbf{f}_B \end{pmatrix} , \tag{4.9}$$

where

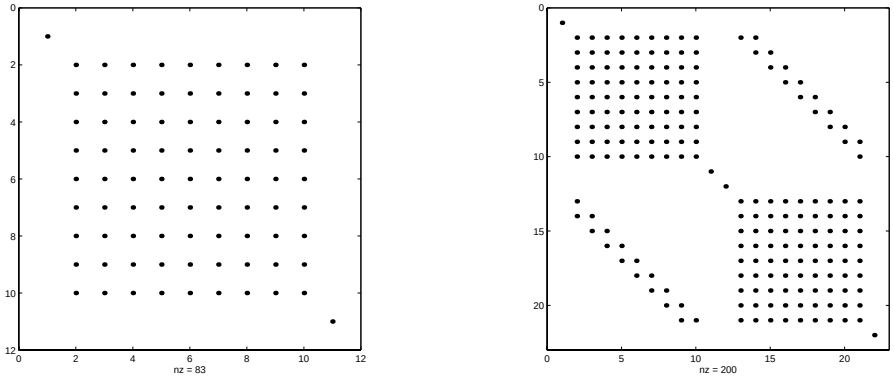

Fig. 4.3. Sparsity pattern for SC matrix derived from an EO decomposition (left) and a VO one (right).

.

$$A_{II} = \begin{pmatrix} A_{II}^{(1)} & & 0 \\ & \ddots & \\ 0 & & A_{II}^{(M)} \end{pmatrix}, \qquad A_{BB} = \begin{pmatrix} A_{BB}^{(1)} & E^{(1,2)} & \dots & E^{(1,M)} \\ E^{(2,1)} & & & \\ \vdots & & \ddots & \vdots \\ E^{(M,1)} & & & A_{BB}^{(M)} \end{pmatrix},$$

$$A_{IB} = \left(A_{IB}^{(1)} \; A_{IB}^{(2)} \; \dots \; A_{IB}^{(M)} \right), \qquad A_{BI} = \left(A_{BI}^{(1)} \; A_{BI}^{(2)} \; \dots \; A_{BI}^{(M)} \right)^T$$

and

$$\mathbf{u}_I = \left(\mathbf{u}_I^{(1)} \; \dots \; \mathbf{u}_I^{(M)} \right)^T, \qquad \mathbf{u}_B = \left(\mathbf{u}_B^{(1)} \; \dots \; \mathbf{u}_B^{(M)} \right)^T,$$
$$\mathbf{f}_I = \left(\mathbf{f}_I^{(1)} \; \dots \; \mathbf{f}_I^{(M)} \right)^T, \qquad \mathbf{f}_B = \left(\mathbf{f}_B^{(1)} \; \dots \; \mathbf{f}_B^{(M)} \right)^T.$$

For the sake of space, we have transposed some matrices. Note however that here the transpose operator acts on the block matrix/vector, not on the blocks themselves, i.e. $\left(\mathbf{a} \; \mathbf{b} \right)^T$ equals to $\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}$ and not $\begin{pmatrix} \mathbf{a}^T \\ \mathbf{b}^T \end{pmatrix}$.

The Schur complement system of problem (4.1) can now be written as $S_h \mathbf{u}_B = \mathbf{g}$, where

$$S_h = A_{BB} - A_{BI} A_{II}^{-1} A_{IB}, \quad \text{and } \mathbf{g} = \mathbf{f}_B - A_{BI} A_{II}^{-1} \mathbf{f}_I \, .$$

To conclude this Section, we wish to note that an EO arrangement is often the direct result of a domain decomposition carried out at a differential level. In this case, the Schur complement matrix may be identified as the discrete counterpart of a

particular differential operator acting on the interface Γ (the Steklov-Poincarè operator [17]). Instead, a VO decomposition is normally the result of a purely algebraic manipulation and in general is lacking an immediate interpretation at differential level. Finally, the VO arrangement produces a larger number of degrees of freedom in the resulting Schur complement system.

4.2.3 Parallel solution of the Schur complement system

Schur complement matrices are usually full and in large scale problems there is no convenience in building them in an explicit way. Thus, the SC system is normally solved by an iterative method, such as a Krylov acceleration method [26], which requires only the multiplication of the matrix with a vector. To compute $\mathbf{w}_B = S_h \mathbf{v}_B$, one may proceed as indicated in the following algorithm, where R_i is the restriction operator from the global border values $\mathbf{v}_B$ to those associated to subdomain $\Omega^{(i)}$.

ALGORITHM 1: COMPUTATION OF $\mathbf{w}_B = S_h \mathbf{v}_B$

1. Restrict $\mathbf{v}_B$ to each subdomain boundary,
$$\mathbf{v}_B^{(i)} = R_i \mathbf{v}_B, \quad i = 1, \ldots, M\,.$$
2. For every $\Omega^{(i)}, \quad i = 1, \ldots M$ solve
$$A_{II}^{(i)} \mathbf{u}_I^{(i)} = -A_{IB}^{(i)} \mathbf{v}_B^{(i)}\,,$$
then compute
$$\mathbf{w}_B^{(i)} = \sum_{j=1}^{M} E^{(ij)} v_B^j + A_{BB}^{(i)} \mathbf{v}^{(i)}{}_B - A_{BI}^{(i)} \mathbf{u}_I^{(i)}\,.$$
3. Apply the prolongation operators to get $\mathbf{w}_B = \sum_{i=1}^{M} R_i^T \mathbf{w}_B^{(i)}$.

In general, Steps 1 and 3 are just formal if we operate in a parallel environment since each processor already handles the restricted vectors $\mathbf{u}_B{}^{(i)}$ instead of the whole vector. Note that the linear system in Step 2 must be solved with high accuracy in order to make the Schur complement system equivalent to the original linear system (4.1).

Algorithm 1 requires four matrix-vector products and the solution of a linear system for each subdomain. Even if carried out in parallel, the latter operation can be rather expensive and is one of the drawbacks of a SC based method. The local problems have to be computed with high accuracy if we want to recover an accurate global solution.

Although (at least in the case of symmetric and positive-definite matrices) the condition number of the Schur matrix is no larger than that of the original matrix A [21, 17], nevertheless it increases when h decreases (for a fixed number of subdomains), but also when H decreases (for a fixed h, i.e. for a fixed problem size). This

Table 4.1. Convergence rate for different preconditioner of the Schur complement system with respect to the discretisation size h and the subdomain size H, for an elliptic problem. The constants C (which are different for each method) are independent from h and H, yet they may depend on the coefficients of the differential operator. The $\delta \in (0, 1]$ in the Vertex Space preconditioner is the overlap fraction, see the cited reference for details.

Preconditioner	*Estimation of the condition number of the preconditioned Schur complement operator*
P_h^J	$K((P_h^J)^{-1}S_h) \leq CH^{-2}(1+\log(H/h))^2$
P_h^{BPS}	$K((P_h^{BPS})^{-1}S_h) \leq C(1+\log(H/h))^2$
P_h^{VS}	$K((P_h^{VS})^{-1}S_h) \leq C(1+\log\delta^{-1})^2$
P_h^{WB}	$K((P_h^{WB})^{-1}S_h) \leq C(1+\log(H/h))^2$
$P_h^{NN,b}$	$K((P_h^{NN,b})^{-1}S_h) \leq C(1+\log(H/h))^2$

is a cause of loss of scalability. A larger number of subdomains imply a smaller value of H, the consequent increase of the condition number causes in turn a degradation of the convergence of the iterative linear solver. The problem may be alleviated by adopting an outer preconditioner for S_h, we will give a fuller description in the next Paragraph.

We want to note that if we solve also Step 2 with an iterative solver, a good preconditioner must be provided also for the local problems in order to achieve a good scalability (for more details, see for instance [21, 24]).

Preconditioners for the Schur complement system

Many preconditioners have been proposed in the literature for the Schur complement system with the aim to obtain good scalability properties. Among them, we briefly recall the Jacobi preconditioner P_h^J, the Dirichlet-Neumann P_h^{ND}, the balancing Neumann-Neumann $P_h^{NN,b}$ [24], the Bramble-Pasciak P_h^{BPS} [1], the Vertex-Space P_h^{VS} [8] and the wire-basket P_h^{WB} preconditioner. We refer [17, 25] and to the cited references for more details. Following [17], we summarize in Table 4.1 their preconditioning properties with respect to the geometric parameters h and H, for an elliptic problem (their extension to non-symmetric indefinite systems is, in general, not straightforward). We may note that with these preconditioners the dependence on H of the condition number becomes weaker, a part from the Jacobi preconditioner which is rather inefficient. The most effective preconditioners are also the ones more difficult to implement, particularly on arbitrary meshes.

Alternative and rather general ways to build a preconditioner for the SC system exploits the identity

$$S_h^{-1} = \begin{pmatrix} 0 & I \end{pmatrix} A^{-1} \begin{pmatrix} 0 \\ I \end{pmatrix}, \tag{4.10}$$

where I is the $n_B \times n_B$ identity matrix, being n_B the size of S_h. We can construct a preconditioner P_{Schur} for S_h from *any* preconditioner P_A^{-1} of the original matrix A by writing

$$P_{Schur} = \begin{pmatrix} 0 & I \end{pmatrix} P_A^{-1} \begin{pmatrix} 0 \\ I \end{pmatrix}.$$

If we indicate with $\mathbf{v}_B$ a vector of size n_B and with R_B the restriction operator on the interface variables, we can compute the operation $P_{Schur}^{-1}\mathbf{v}_B$ (which is indeed the one requested by an iterative solver) by an application of P_A^{-1}, as follows

$$P_{Schur}^{-1}\mathbf{v}_B = R_B P_A^{-1} \begin{pmatrix} 0 \\ \mathbf{v}_B \end{pmatrix} = R_B P_A^{-1} R_B^T \mathbf{v}_B \,. \tag{4.11}$$

In a parallel setting, we will of course opt for a parallel P_A^{-1}, like a Schwarz-based preconditioner of the type outlined in Section 4.4. This is indeed the choice we have adopted to precondition the SC matrix in many examples shown in this work.

4.3 The Schur Complement System Used as a Preconditioner

Although the Schur complement matrix is better conditioned than A, its multiplication with a vector is in general expensive. Indeed Step 2 of Algorithm 1 requires the solution of M linear systems, which should be carried out to machine precision, otherwise the iterative scheme converges slowly or may even diverge.

An alternative is to adopt a standard (parallel) iterative scheme for the global system (4.1) and use the SC system as a preconditioner. This will permit us to operate some modifications on the SC system in order to make it more computationally efficient. Precisely, we may replace S_h with a suitable approximation $\widetilde{S}$ that is cheaper to compute. The preconditioning matrix can then be derived as follows. We consider again the block decomposition (4.9) and we write A as a product of two block-triangular matrices,

$$A = \begin{pmatrix} A_{II} & 0 \\ A_{BI} & I \end{pmatrix} \begin{pmatrix} I & A_{II}^{-1}A_{IB} \\ 0 & S_h \end{pmatrix}.$$

Let us assume that we have good, yet cheaper, approximations of A_{II} and S_h, which we indicate as $\widetilde{A}_{II}$ and $\widetilde{S}$, respectively, a possible preconditioner for A is then

$$P_{\mathrm{ASC}} = \begin{pmatrix} \tilde{A}_{II} & 0 \\ A_{BI} & I \end{pmatrix} \begin{pmatrix} I & \tilde{A}_{II}^{-1}A_{IB} \\ 0 & \tilde{S} \end{pmatrix},$$

where ASC stands for *Approximate Schur Complement*. Indeed the approximation $\widetilde{A}_{II}$ may be used also to build $\widetilde{S}$ by posing $\widetilde{S} = A_{BB} - A_{BI}\tilde{A}_{II}^{-1}A_{IB}$. Note that P_{ASC} operates on the whole system while $\widetilde{S}$ on the interface variables only, and that P_{ASC} does not need to be explicitly built, as we will show later on. A possible approximation for A_{II} is an incomplete LU decomposition [18], i.e. $\widetilde{A}_{II} = \tilde{L}\tilde{U}$, where $\tilde{L}$ and $\tilde{U}$ are obtained from an incomplete factorization of A_{II}. Another possibility is to approximate the action of the inverse of A_{II} by carrying out a few cycles of an iterative solver or carrying out a multigrid cycle [12].

The solution of the preconditioned problem $P_{\rm ASC} = \mathbf{zr}$, where $\mathbf{r} = (\mathbf{r}_I, \mathbf{r}_B)^T$ and $\mathbf{z} = (\mathbf{z}_I, \mathbf{z}_B)$ may be effectively carried out by the following Algorithm.

ALGORITHM 2: APPLICATION OF THE ASC PRECONDITIONER

1. Apply the lower triangular part of $P_{\rm ASC}$. That is solve $\widetilde{A}_{II}\mathbf{y}_I = \mathbf{r}_I$ and compute $\mathbf{y}_B = \mathbf{r}_B - A_{BI}\mathbf{y}_I$.
2. Apply the upper triangular part of $P_{\rm ASC}$. That is solve

$$\widetilde{S}\mathbf{z}_B = \mathbf{y}_B, \tag{4.12}$$

with $\widetilde{S} = A_{BB} - A_{BI}\widetilde{A}_{II}^{-1}A_{IB}$, and compute $\mathbf{z}_I = \mathbf{y}_I - \tilde{A}_{II}^{-1}A_{IB}\mathbf{z}_B$. The solution of (4.12) may be accomplished by an iterative scheme exploiting Algorithm 1.

Notice that the Step 1 and the computation of $\mathbf{z}_I$ in Step 2 are perfectly parallel. On the contrary (4.12) is a *global* operation, which, however, may be split into several parallel steps preceded and followed by scatter and gather operations involving communication among subdomains. Note that the matrix $\widetilde{S}$ may itself be preconditioned by using the technique outlined in Section 4.2.3, in particular by using a Schwarz-type preconditioner, as illustrated in Section 4.3.

As already said, $\widetilde{A}_{II}$ may be chosen as the ILU(f) incomplete factorization of A_{II}, where f is the fill-in factor. Furthermore, (4.12) may be solved inexactly, using a fixed number L of iterations of a Krylov solver. We will denote such preconditioner as **ASC-L-iluf**. Alternatively, one may avoid to factorize A_{II} and build its approximation implicitly by performing a fixed number of iteration when computing the local problems in Step (2) of Algorithm 1.

In both cases the action the ASC preconditioner corresponds to that of a matrix which changes at each iteration of the outer iterative solver. Consequently, one needs to make a suitable choice of the Krylov subspace accelerator for the the solution of (4.1) like, for instance, GMRESR [27] or FGMRES [18]. The former is the one we have used for the numerical results shown in this work.

We mention that the ASC preconditioner lends itself to a multilevel implementation, where a family of increasingly coarser approximations of the Schur complement matrix is used to build the preconditioner. The idea is that the coarsest approximation should be small enough to be solved directly. The drawback is the need of assembling and storing a number of matrices equal to the number of levels.

4.4 The Schwarz Preconditioner

The Schwarz iteration is a rather well known parallel technique based on an overlapping domain decomposition strategy. In a VO framework, it is normally built as follows (we refer again to Figure 4.1).

Each subdomain $\Omega^{(i)}$ is extended to $\widetilde{\Omega}^{(i)}$ by adding the strip $\Gamma^{(1,2)}$, i.e. $\widetilde{\Omega}^{(i)} = \Omega^{(i)} \cup \Gamma^{(1,2)},\ i = 1, 2$. A parallel solution of the original system is then obtained by

an iterative procedure involving local problems in each $\widetilde{\Omega}^{(i)}$, where on $\partial\widetilde{\Omega}^i \cap \Omega^{(j)}$ we apply Dirichlet conditions by getting the data from the neighboring subdomains.

What we have described here is the implementation with *minimum overlap*. A larger overlap may be obtained by adding further layers of elements. The procedure may be readily extended to an arbitrary number of subdomains. More details on the algorithm with an analysis of its main properties may be found, for instance, in [17].

A *multiplicative* version of the procedure is obtained by ordering the subdomains and solving the local problems sequentially using the latest available interface values. This is indeed the original Schwarz method and is sequential. Parallelism can be obtained by using the *additive* variant where all subdomain are advanced together, by taking the interface values at the previous iteration.

From an algebraic point of view, multiplicative methods can be reformulated as a block Gauss-Seidel procedure, while additive methods as block Jacobi procedure [18].

If used as a stand-alone solver, the Schwarz iteration algorithm is usually rather inefficient in terms of iterations necessary to converge. Besides, a damping parameter has to be added, see [17], in order to ensure that the algorithm converges. Instead, the method is a quite popular parallel preconditioner for Krylov accelerators. In particular its minimum overlap variant, which may exploit the same data structure normally used for the parallel implementation of the matrix-vector product, allowing a saving in memory requirement.

Let $B^{(i)}$ be the local matrix associated to the discretisation on the extended subdomain $\widetilde{\Omega}^{(i)}$, $R^{(i)}$ a restriction operator from the nodes in Ω to those in $\widetilde{\Omega}^{(i)}$, and $P^{(i)}$ a prolongation operator (usually, $P^{(i)} = (R^{(i)})^T$). Using this notation, the Schwarz preconditioner can be written as

$$P_{AS}^{-1} = \sum_{i=1}^{M} P^{(i)} B^{(i)} R^{(i)}, \tag{4.13}$$

being M the number of subdomains.

The matrices $B^{(i)}$ can be extracted directly from the set of rows of the global matrix A corresponding to the local nodes, discarding all coefficients whose indexes are associated to nodes exterior to the subdomain. The application of $R^{(i)}$ is trivial, since it returns the locally hosted components a the vector; the prolongation operator $P^{(i)}$ just does the opposite operation.

Although simple to implement, the scalability of the Schwarz preconditioner is hindered by the weak coupling between far away subdomains. We may recover a good scalability by the addition of a coarse operator [9, 21]. If the linear system arises from the discretisation of a PDE system a possible technique to build the coarse operator matrix A_H consists in discretising the original problem on a (very) coarse mesh, see for instance [4]. However the construction of a coarse grid and of the associated restriction and prolongation operators may become a rather complicated task when dealing with three dimensional problems and complicated geometries. An alternative is to resort to algebraic procedures, such as the aggregation or agglomeration technique [21, 1, 4, 2, 28, 20], which are akin to procedures developed in the context of

algebraic multigrid methods, see also [29]. Here, we will focus on the latter, and in particular we will propose an agglomeration technique.

4.4.1 The agglomeration coarse operator

To fix the ideas let us consider a finite element formulation (4.2). Thanks to a VO partitioning we can split the finite element function space V_h as

$$V_h = \bigcup_{i=1}^{M} V_h^{(i)},$$

where M is the number of subdomains, and $V_h^{(i)}$ is set of finite element functions associated to the triangulation of $\widetilde{\Omega}^{(i)}$ with zero trace on $\partial\widetilde{\Omega}^{(i)} \setminus \partial\Omega$. We suppose to operate in the case of minimal overlap among subdomains as described in the previous section and we indicate with $n^{(i)}$, the dimension of the space $V_h^{(i)}$. By construction, $n = \sum_{i=1}^{M} n^{(i)}$.

We can build a *coarse space* considering for each $\Omega^{(i)}$ a set of vectors $\{\boldsymbol{\beta}_s^{(i)} \in \mathbb{R}^{n^{(i)}},\ s = 1, \ldots, l^{(i)}\}$ of nodal weights $\boldsymbol{\beta}_s^{(i)} = \left(\beta_{s,1}^{(i)}, \ldots, \beta_{s,n^{(i)}}^{(i)}\right)$ that are linearly independent, with $\beta_{s,n^{(i)}}^{(i)} \neq 0$. The value $l^{(i)}$, $i = 1, \ldots,$ will be the (local) dimension of the coarse operator on the corresponding subdomain. Clearly, we must have $l^{(i)} \leq n^{(i)}$ and, in general, $l^{(i)} << n^{(i)}$. We indicate with l the global dimension of the coarse space, i.e.

$$l = \sum_{i=1}^{M} l^{(i)}\,.$$

With the help of the vectors $\boldsymbol{\beta}_s^{(i)}$, we can define a set of local coarse space functions as linear combination of basis functions, i.e.

$$\mathcal{V}_H^{(i)} = \left\{ \Phi_s^{(i)} : \Omega \to \mathbb{R} \ \mid\ \Phi_s^{(i)} = \sum_{k=1}^{n^{(i)}} \beta_{s,k}^{(i)} \phi_k^{(i)},\ s = 1, \ldots, l^{(i)} \right\}.$$

It is easy to verify that the functions in $\mathcal{V}_H^{(i)}$ are linearly independent. Finally, the set $\mathcal{V}_H = \bigcup_{i=1}^{M} \mathcal{V}_H^{(i)}$ is the base of the global coarse grid space V_H, i.e. we take $V_H = \mathrm{span}\{\mathcal{V}_H\}$. By construction, $\dim(V_H) = \mathrm{card}(\mathcal{V}_H) = l$.

We note that $V_H \subset V_h$, as it is built by linear combinations of function in V_h, and any function $u_H \in V_H$ may be written as

$$u_H(\mathbf{x}) = \sum_{i=1}^{M} \sum_{s=1}^{l^{(i)}} U_s^{(i)} \Phi_s^{(i)}(\mathbf{x}), \tag{4.14}$$

where the $U_s^{(i)}$ are the "coarse" degrees of freedom. Finally, the coarse problem is built as:

Find $u_H \in V_H$ so

$$a(u_H, w_H) = f(w_H), \quad \forall w_H \in V_H\,.$$

From an algebraic point of view, we have

$$\sum_{i=1}^{M}\sum_{s=1}^{l^{(i)}} U_s^{(i)} \sum_{k=1}^{n^{(i)}}\sum_{t=1}^{n^{(m)}} \beta_{s,k}^{(i)}\beta_{q,t}^{(m)} a(\phi_k^{(i)}, \phi_t^{(m)}) = \sum_{t=1}^{n^{(m)}} \beta_{q,t}^{(m)}(f, \phi_t^{(m)})$$
$$q = 1, \ldots, l^{(m)},\ m = 1, \ldots, M\,.$$

To complete the procedure we need to define a restriction operator $R_H : V_h \to V_H$ which maps a generic finite element function to a coarse grid function. Since $u \in V_h$ may be written as

$$u_h = \sum_{i=1}^{M}\sum_{k=1}^{n^{(i)}} \phi_k^{(i)} u_k^{(i)},$$

where the $u_k^{(i)}$ are the degrees of freedom associated to the triangulation of $\Omega^{(i)}$, a restriction operator may be defined by computing $u_H = R_h u$ as

$$u_H = \sum_{i=1}^{M}\sum_{s=1}^{l^{(i)}} U_s^{(i)} \Phi_s^{(i)},$$

where

$$U_s^{(i)} = \sum_{k=1}^{n^{(i)}} \beta_{s,k}^{(i)} u_k^{(i)},\ s = 1, \ldots l^{(i)}, \quad i = 1 \ldots, M.$$

At the algebraic level, we build a global vector $\mathbf{U}_H$ by assembling the $U_s^{(i)}$ on a subdomain basis, i.e.

$$\mathbf{U}_H = \left(U_1^{(1)} \ldots U_{l^{(1)}}^{(1)} \ldots U_{l^{(M)}}^{(M)}\right)^T,$$

and we arrange similarly the vector $\mathbf{u}_h$ of the nodal values of u_h, i.e.

$$\mathbf{u}_h = \left(u_1^{(1)} \ldots u_{n^{(1)}}^{(1)} \ldots u_{n^{(M)}}^{(M)}\right)^T.$$

The prolongation matrix $R_H^T \in \mathbb{R}^{n \times l}$ will then have the following block structure,

$$R_H^T = \begin{pmatrix} \boldsymbol{\beta}_1^{(1)^T} & \boldsymbol{\beta}_2^{(1)^T} & \ldots & \boldsymbol{\beta}_{l^{(1)}}^{(1)^T} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \boldsymbol{\beta}_1^{(2)^T} & \boldsymbol{\beta}_2^{(2)^T} & \ldots & \boldsymbol{\beta}_{l^{(2)}}^{(2)^T} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \boldsymbol{\beta}_1^{(M)^T} & \boldsymbol{\beta}_2^{(M)^T} & \ldots & \boldsymbol{\beta}_{l^{(M)}}^{(M)^T} \end{pmatrix},$$

and the coarse matrix A_H and right-hand side of the coarse system can be written as

$$A_H = R_H A R_H^T, \quad \mathbf{f}_H = R_H \mathbf{f}\,,$$

respectively.

The conditions imposed on the $\boldsymbol{\beta}_s^{(i)}$ vectors guarantees that R_H has full rank. Moreover, if A is symmetric and positive definite, then A_H will share the same property. The application of the agglomeration coarse grid operator does not require to build R_H explicitly. Even the construction of the vectors $\boldsymbol{\beta}$ can be avoided if they have a simple structure. In fact the construction of A_H involves just a weighted sum of the element of A. Concerning the parallel implementation, the overhead of the coarse problem depends in general on the number of local coarse degrees of freedom. In general $l^{(i)} << n^{(i)}$ (in the limit we may even take $l^{(i)} = 1$ for all i!), and consequently the matrix A_H is rather small compared to A. This a major difference from the use of this technique in an algebraic multigrid setting, where many levels of coarse operator are considered.

The build up of the coarse linear system can be carried out as follows. Each processor computes the contribution to A_H and $\mathbf{f}_H$ corresponding to the associated subdomains, then it broadcasts the results to the other processors. Being the coarse system small, the cost of the broadcast operation is limited. Furthermore, it is usually carried out only once.

The domain decomposition technique may be used also for the set up of the $\boldsymbol{\beta}_s^{(i)}$ vectors. Indeed, after having set up the basic Schwarz preconditioner by assigning each extended subdomain $\widetilde{\Omega}_i$ to a different processor, at a second stage, each subdomain $\widetilde{\Omega}^{(i)}$ can be further partitioned into $l^{(i)}$ connected parts with minimum overlap. We will indicate this second level partition as $\omega_s^{(i)}$ with $s = 1, \ldots, l^{(i)}$. Then we may take

$$\beta_{s,k} = \begin{cases} 1 & \text{if node } k \text{ belongs to } \omega_s^{(i)} \backslash \partial\omega_s^{(i)}, \\ 0 & \text{otherwise.} \end{cases}$$

As already explained, the coarse grid operator is used to improve the scalability of a Schwarz-type parallel preconditioner P_S. We will indicate with P_{ACM} a preconditioner augmented by the application of the coarse operator (ACM stands for *agglomeration coarse matrix*) and we illustrate two possible strategies for its construction.

A one-step preconditioner, $P_{\mathrm{ACM},1}$, may be formally written as

$$P_{\mathrm{ACM},1}^{-1} = P_S^{-1} + R_H^T\, A_H^{-1}\, R_H, \tag{4.15}$$

and it corresponds to an additive application of the coarse operator.

An alternative formulation adopts the following two-steps Richardson method

$$\begin{aligned} \mathbf{u}^{n+1/2} &= \mathbf{u}^n + P_S^{-1}\mathbf{r}^n\,, \\ \mathbf{u}^{n+1} &= \mathbf{u}^{n+1/2} + R_H^T\, A_H^{-1}\, R_H \mathbf{r}^{n+1/2}\,, \end{aligned} \tag{4.16}$$

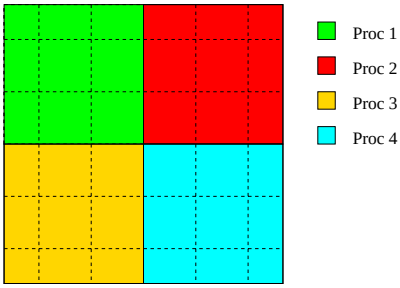

Fig. 4.4. Example of two-level decomposition. First level in continuous line, and second level in dashed line. Typically, each first-level decomposition subdomain is given to a different processor.

where $\mathbf{r}^n$ and $\mathbf{u}^n$ are respectively the residual and the approximate solution at the $n-th$ iteration of the outer iterative solver. The corresponding preconditioning matrix can be formally written as

$$P^{-1}_{\mathrm{ACM},2} = P_S^{-1} + R_H^T\, A^{-1}_{\mathrm{ACM}}\, R_H - P_S^{-1} A R_H^T\, A^{-1}_{\mathrm{ACM}}\, R_H\,. \tag{4.17}$$

The implementation of (4.15) and (4.17) requires the parallel solution of the coarse problem

$$A_H \mathbf{u}_H = \mathbf{r}_H, \tag{4.18}$$

where $\mathbf{r}_H = R_H \mathbf{r}$. If one has only a limited number of processors at disposal, perhaps the best approach is to gather the entire coarse matrix A_H on one processor (say, processor 0), and perform each solution phase of (4.18) using an appropriate sequential linear solver. Each processor computes its contribution to $\mathbf{r}_H$ and sends it to processor 0 (gathering phase). The solution computed by processor 0 will then be scattered to the other processors, for example by a broadcast operation and the final prolongation operation will be carried out independently on each processor. As the coarse problem is likely to be small, its sequential solution causes little overhead and the cost of broadcast communication is also likely to be negligible as well.

Instead, if hundreds or thousands of processors are used, the coarse problem is likely to have a non-negligible size, and furthermore, the cost of the gathering-scattering communication phases may be substantial. Therefore, in this case a parallel direct solver is to be preferred. Generic interfaces to serial and parallel direct solvers are available, see [19]. We also mention that the presented two-level algebraic preconditioner (see Figure 4.4) can be extended in a multilevel fashion; see for instance [16].

4.5 Applications

In this Section we show some applications of the techniques here illustrated. We will report some academic tests used mainly to assess the basic properties of the scheme

Table 4.2. 2D Poisson problem. Comparison of different preconditioners. Number of iterations needed to reduce the initial residual by a factor of 10^6, M is the number of subdomains. The starting mesh has 180×180 squares, each of them has been divided into 2 triangles.

	solver	$M = 4$	$M = 9$	$M = 16$	$M = 25$
P_S	GMRES	-	57	70	76
P_C	GMRES	-	42	40	39
$P_{\mathrm{ACM},1}$	GMRES	-	56	69	70
$P_{\mathrm{ACM},2}$	GMRES	-	51	49	46
ASP-2-ilu0	GMRESR	99	97	97	99
ASP-4-ilu0	GMRESR	82	78	75	71
ASP-2-ilu1	GMRESR	68	68	70	69
ASP-2-ilu2	GMRESR	52	53	56	52

and more realistic applications. The latter include the solution of compressible flow around aircrafts and free surface hydrodynamics problems. For the decomposition of the domain, we have adopted the software package Metis [15], which operates on the finite element mesh, and can produce both EO and VO decompositions. For the overlapping Schwarz preconditioner, wider levels of overlap are recursively created by adding internal nodes that are linked with a side to the current overlap region. If we use higher order finite elements, we may still make the basic partitioning using just the mesh information. Then the new nodes corresponding to the additional degrees of freedom are added and their nature (interior, exterior or border) can be immediately identified by the geometrical entity that is associated to the node. For instance, additional in a VO framework, additional nodes on a side linking to nodes internal to subdomain Ω_i are internal to Ω_i etc.

4.5.1 2D Poisson Problem

We have considered the following Poisson problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

where $\Omega = (0,1) \times (0,1)$. For the discretisation we have used P1 finite elements on a regular mesh. The linear system has been solved using GMRES(60), in the case of the Schwarz preconditioner, or GMRESR with the ASC preconditioner, up to a tolerance of 10^{-6}. The right-hand side is made up of random numbers between 0 and 1.

Being the mesh structured, we have divided the computational domain into M square subdomains, and we have used these subdomains to build a "classical" coarse correction for a 2-level Schwarz preconditioner, following [9].

In Table 4.2 we have compared the Schwarz preconditioner (P_S) without the coarse grid correction, the one augmented by the "classical" coarse operator, indicated by P_C, and the proposed preconditioners $P_{\mathrm{ACM},1}$ and $P_{\mathrm{ACM},2}$. Finally, we report the results obtained with the ASC preconditioner ASP-L-iluf.

Table 4.3. 2D Poisson problem. Comparison of different preconditioners. CPU-time in seconds to reduce the initial residual by a factor of 10^6.

	$M = 4$	$M = 9$	$M = 16$	$M = 25$
P_S	-	3.90	1.59	0.77
P_C	-	5.50	1.94	2.13
$P_{\mathrm{ACM},1}$	-	5.10	2.08	1.68
$P_{\mathrm{ACM},2}$	-	4.16	2.03	0.89
ASP-2-ilu0	12.64	3.94	3.34	2.12
ASP-4-ilu0	9.73	5.46	2.23	1.88
ASP-2-ilu1	8.04	3.87	1.80	2.11
ASP-2-ilu2	6.44	4.32	1.77	1.54

Table 4.4. 2D Poisson problem. $P_{\mathrm{ACM},2}$. The table reports the number of iterations needed to reduce the initial residual by a factor of 10^6.

Local dimension of coarse space	$M = 9$	$M = 16$	$M = 25$
1	51	49	46
2	57	54	50
4	55	49	46
8	50	45	41
32	50	34	32

Table 4.5. 2D Poisson problem. $P_{\mathrm{ACM},2}$. The table reports the CPU time (in seconds) needed to reduce the initial residual by a factor of 10^6.

Local dimension of coarse space	$M = 9$	$M = 16$	$M = 25$
1	4.16	2.03	0.89
2	4.28	1.60	0.98
4	4.22	1.55	0.96
8	4.16	1.54	0.88
32	4.17	1.52	0.94

The performance of the Schwarz method without any coarse correction degrades rapidly, demonstrating the poor scalability of the basic algorithm. Preconditioning with $P_{\mathrm{ACM},1}$ has a very poor influence (probably also because of the relative small number of subdomains), while the 2-level version behaves much better. The ASC-type preconditioners show a very good scalability, even if the CPU times, reported in Table 4.3, are less interesting. Another important parameter is the dimension of the agglomeration coarse space, and it is analyzed in Tables 4.4 and 4.5. Note that increasing the dimension of the coarse space has a positive effect on the convergence. Even if the numerical experiments show that the "classic" preconditioner is in general more effective unless a "rich" local coarse space dimension is used in the agglomeration procedure, we point out again the greater flexibility and generality of the latter.

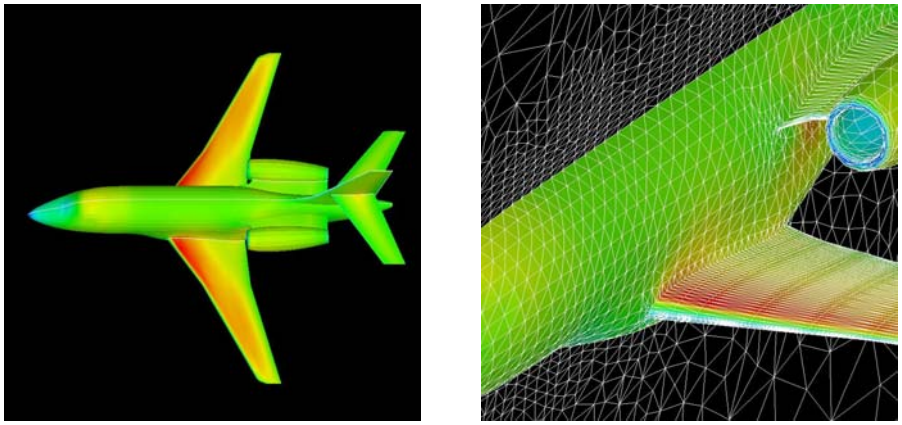

Fig. 4.5. Pressure coefficient contours for FALCON_45k.

4.5.2 The compressible Euler equations

Any standard spatial discretisation applied to the Euler equations leads eventually to a system of ODE in time, which may be written as

$$\frac{\mathrm{d}U}{\mathrm{d}t} = R\left(\mathbf{U}\right) , \tag{4.19}$$

where $\mathbf{U} = (U_1, U_2, \ldots, U_n)^T$ is the vector of unknowns with $U_i = U_i(t)$ and $R\left(U\right)$ the result of the spatial discretisation of the Euler fluxes. An implicit two-step scheme applied to (4.19), for instance a backward Euler method, yields

$$\mathbf{U}^{n+1} - \mathbf{U}^n = \Delta t R\left(\mathbf{U}^{n+1}\right) , \tag{4.20}$$

where Δt is a *diagonal matrix* of local time steps, i.e. $\Delta t = \mathrm{diag}(\Delta t_i, i = 1, \ldots, n)$. The non-linear problem (4.20) may be solved by employing a Newton iterative procedure, which computes successive approximations $\mathbf{U}_{(k+1)}$ of $\mathbf{U}^{n+1}$ by solving

$$\left[I + \Delta t \frac{\partial R}{\partial \mathbf{U}}\left(\mathbf{U}_{(k)}\right)\right]\left(\mathbf{U}_{(k+1)} - \mathbf{U}^n\right) = \mathbf{U}_{(k)} - \mathbf{U}^n - \Delta t R\left(\mathbf{U}_{(k)}\right) , \quad k = 0, \ldots ,$$

with $\mathbf{U}_{(0)} = \mathbf{U}^n$. We have reduced the original non-linear problem to the solution of a series of linear systems which will be finally tackled by the proposed parallel techniques.

Since we are considering steady state solutions, we have taken just a single iteration of the Newton procedure. Furthermore, we have taken an approximate Jacobian $\frac{\partial R}{\partial \mathbf{U}}$. More precisely, the Jacobian is the exact Jacobian of a first-order upwind spatial discretisation. This ease up the computation greatly. The resulting method is sometimes called pseudo-transient continuation [6].

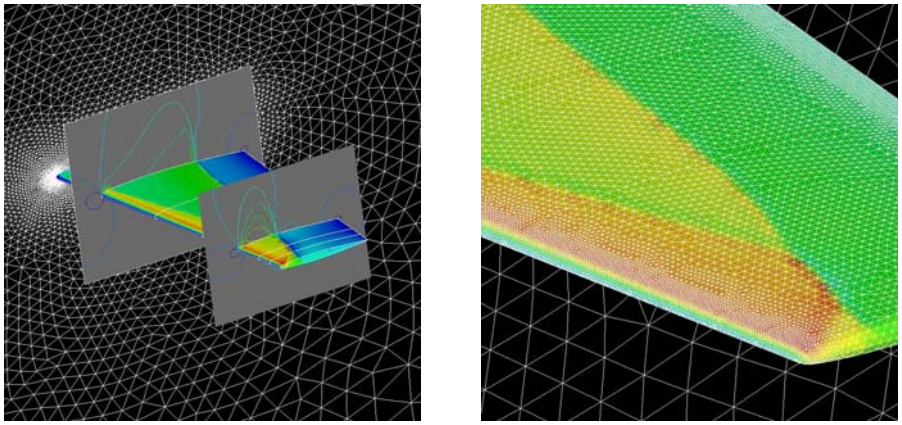

Fig. 4.6. Mach number contours for M6_316K.

For the numerical experiments at hand we have used the parallel version of the code THOR, developed at the Von Karman Institute. This code uses for the spatial discretisation the multidimensional upwind finite element scheme [7]. The solutions obtained on two of the presented test cases are illustrated in Figures 4.5 and 4.6.

The test cases are summarized in Table 4.6. For all of them, the starting solution is a constant vector, and the local CFL numbers are varied from 10 to 10^5, by multiply them at each time level by a factor of 2 until we reach CFL=10^5. The computations are stopped when the Euclidean norm of the density residual is less than 10^{-6}. As previously described, at each time level we have to solve a linear system. This is done using GMRES(60) in the case of Schwarz-type preconditioner or GMRESR if the ASC preconditioner is chosen, up to a tolerance on the relative residual $||r||/||r_0||$ of 10^{-6}. The starting solution is the zero vector. The Schwarz preconditioner P_S uses a minimal overlap and the local problems are solved inexactly using an ILU(0) decomposition. A first test case concerns the flow around a Falcon Aircraft. We have considered a free-stream Mach number of 0.45 and zero angles of yaw and attach. The mesh is formed by 45387 elements, corresponding to 226935 degrees of freedom.

Figures 4.7 and 4.8 report the iterations to converge at each time level for different values of L, using 4, 8, 16 and 32 SGI-Origin 3000 processors for FALCON_45k. The time step in Equation (4.19) is increased exponentially. This quite common practice, here adopted in order to minimize the effect of "bad" starting solutions on the convergence of the Newton method, makes the first linear systems relatively well conditioned. The iterations to converge increase at each time level, to decrease again when the system is approaching the steady-state solution. We may notice that using 16 and 32 processors the ASC preconditioner cannot guarantee good performance, at least for some combinations of the number of levels L and CFL number. This may suggest to adapt the value of L to the CFL number. The elapsed CPU times are reported in Table 4.7, where we have highlighted the best performance. We may notice

Table 4.6. Main characteristics of the test cases.

name	M_∞	α	N_nodes	N_cells
FALCON_45k	0.45	1.0	45387	255944
M6_23k	0.84	3.06	23008	125690
M6_42k	0.84	3.06	42305	232706
M6_94k	0.84	3.06	94493	666569
M6_316k	0.84	3.06	316275	1940182

Table 4.7. FALCON_45k. CPU-time (in seconds) for ASC preconditioner, using different values of L.

N_procs	ASC-1-ilu0	ASC-2-ilu0	ASC-4-ilu0	ASC-8-ilu0
4	2542.4	2401.7	**2393.2**	3319.7
8	925.5	**897.6**	1406.6	1423.2
16	863.7	753.7	**561.6**	707.2
32	443.8	332.1	**248.6**	398.6

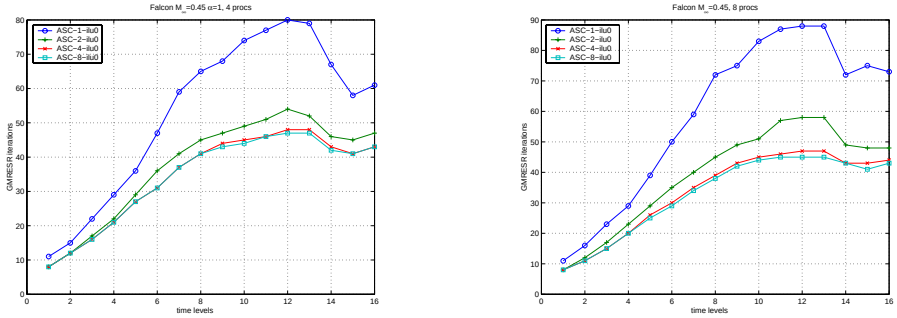

Fig. 4.7. FALCON_45k. Convergence history with 4 (left) and 8 processors (right) with the ASC preconditioner, for different values of L.

that one iteration of the nested solver is not enough to guarantee good convergence results, especially as the number of processors grows. At the same time, high values of L will decrease the performance. A value of about 4 seems a good compromise.

In Figure 4.9 we have reported the results obtained with the proposed Schwarz methodology and in particular the influence of the local dimension of the coarse space N_p for $P_{\mathrm{ACM},1}$ and $P_{\mathrm{ACM},2}$, using 16 MIPS 14000 processors. We recall again that N_p is the dimension of the coarse space on each processor. For low CFL numbers the value of N_p does not affect remarkably the convergence of GMRES, while as the CFL number increases the positive effect of an increasing coarse space becomes more evident, as we may notice in Figure 4.10. The two-level preconditioner seems a better choice from the point of view of both iterations to converge and CPU time, especially for low values of N_p. Figure 4.11 shows a comparison among the Schwarz preconditioner without coarse correction P_S, the ASC preconditioner and

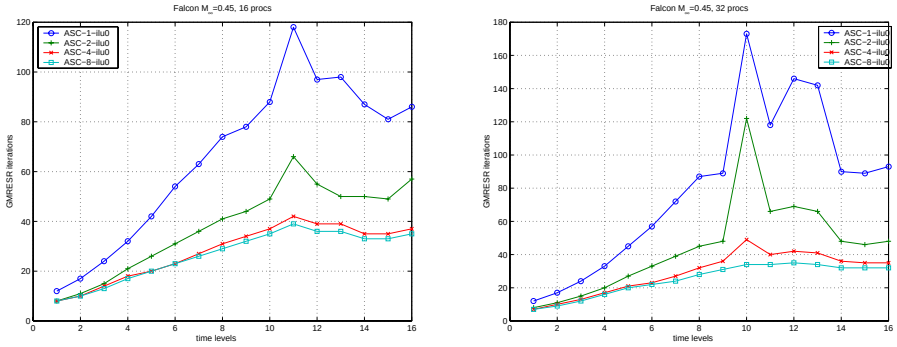

Fig. 4.8. FALCON_45k. Convergence history with 16 processors (left) and 32 processors (right) with the ASC preconditioner, for different values of L.

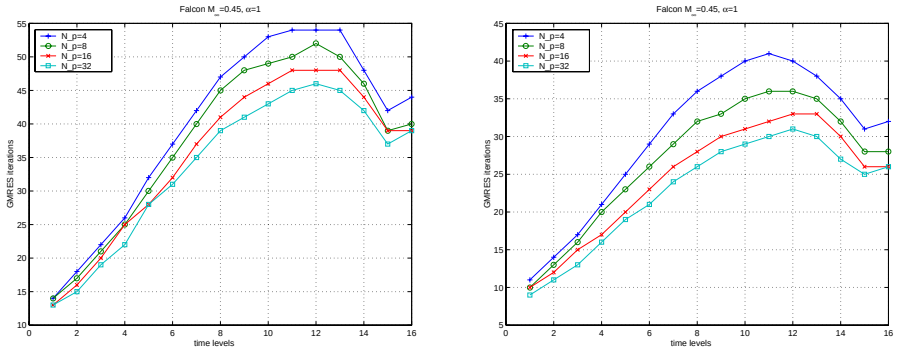

Fig. 4.9. FALCON_45k. Iterations to converge with different values of N_p for $P_{\mathrm{ACM},1}$ (left) and $P_{\mathrm{ACM},2}$ (right).

the ACM preconditioner. Although better than P_S in terms of converge rate, ASC-2-ilu0 doesn't seem to be a suitable choice, as one may observe from the left picture of Figure 4.11, where we have plotted the time residual versus the CPU-time. On the contrary, both $P_{\mathrm{ACM},1}$ and $P_{\mathrm{ACM},2}$ are substantially better than P_S.

We have obtained similar results for the test case M6_94k, as reported in Figures 4.12 and 4.13. The CPU times are reported in Tables 4.8, 4.9 and 4.10 for M6_94K, and in table 4.11 for M6_316k.

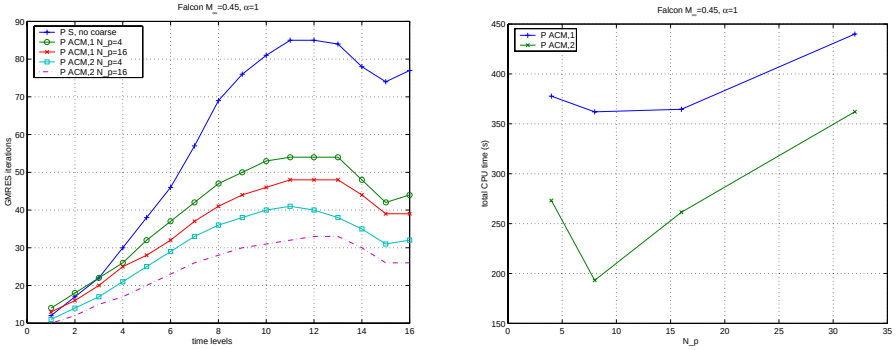

Fig. 4.10. FALCON_45k. Comparison among different preconditioners (left) and CPU time, in seconds (right). 16 SGI-Origin3000 processors.

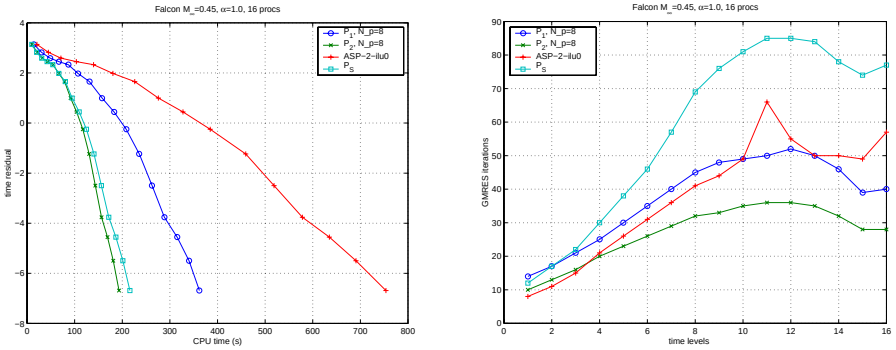

Fig. 4.11. FALCON_45k. Residual versus CPU-time (left) and iterations to converge at each time level (right), using 16 SGI-Origin3000 processors.

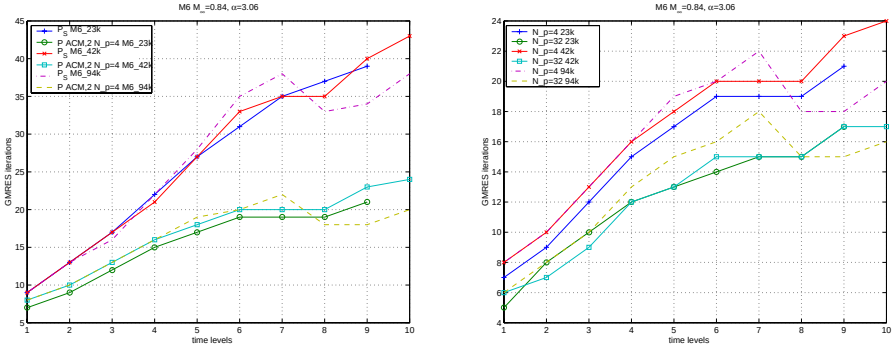

Fig. 4.12. M6_94k. Iterations to converge with P_S and $P_{\mathrm{ACM},2}$ (left), and iterations to converge with $P_{\mathrm{ACM},2}$ (right) using two different values of $N_p$and 16 processors.

Table 4.8. M6_94k. CPU-time (in seconds) for ASC preconditioner, using different values of L.

N_procs	ASC-2-ilu0	ASC-4-ilu0	ASC-8-ilu0
8	**1538.4**	1600.4	1859.9
16	544.8	**569.1**	1330.5
32	**248.5**	286.0	358.9

Table 4.9. M6_94k. CPU-time to converge, using $P_{\mathrm{ACM},1}$ and varying the number of processors. Best results are highlighted.

N_procs	N_p=4	N_p=8	N_p=16	N_p=32
8	1008.2	**978.4**	1251.3	883.4
16	**502.5**	506.9	515.0	457.3
32	**208.0**	245.3	300.5	505.0

Table 4.10. M6_94k. CPU-time to converge, using $P_{\mathrm{ACM},2}$, and varying the number of processors. Best results are highlighted.

N_procs	N_p=4	N_p=8	N_p=16	N_p=32
8	934.8	945.6	**909.3**	925.6
16	458.6	**405.2**	413.9	442.6
32	164.4	**164.7**	181.4	515.6

Table 4.11. M6_316k. CPU-time (seconds) required to reach the steady state solution, using 32 processors. Comparison between Schwarz preconditioner without coarse grid, the ACM preconditioner, multiplicative version, and an approximated Schur complement preconditioner.

N_procs	P_S	$P_{\mathrm{ACM},2}$	ASC-4-ilu0
32	1524.2	1370.6	2691.3

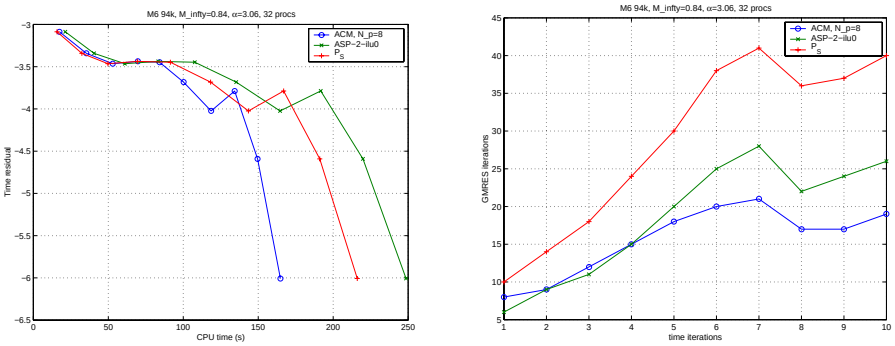

Fig. 4.13. M6_94k. Residual versus CPU-time (right) and iterations to converge at each time level (right), using 32 processors.

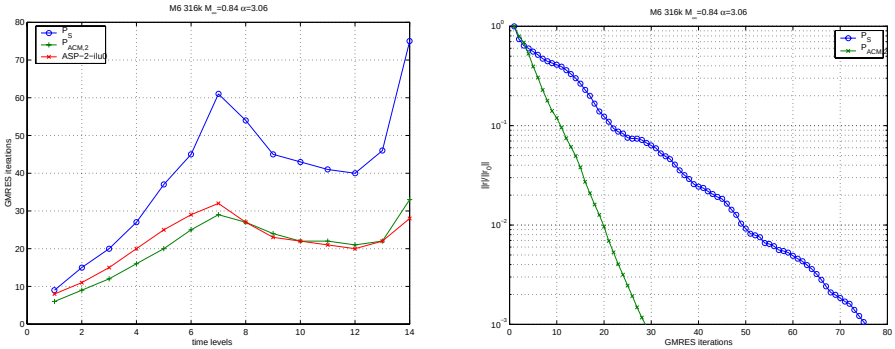

Fig. 4.14. M6_316k. Iterations at each time level (left) and converge history at the 14th time step (right).

4.6 Conclusions

In this chapter we have presented a class of preconditioners based on the DD approach that are well suited for parallel implementation. A great variety of methods are in fact available in literature and we have chosen to focus our attention on DD methods applied at the algebraic level, namely the Schur complement and the Schwarz algorithms. The reason being their generality and the fact that they are implemented in many available parallel linear algebra packages. We have illustrated mainly their use as preconditioners. Indeed, the first consideration we can make is that these methods are usually inefficient when used as solvers.

A clear cut comparison of the two is difficult as their performance is often problem dependent. As a general rule we may state that the approximate Schur complement system has generally a better preconditioning property at the price of an higher cost "per iteration". It performs better when the ratio unknowns/number of subdomains is "low". Otherwise, the computational cost linked to the solution of the internal problems (which in most cases scales with the square of the number of the local degrees of freedom) may degrade the effectiveness of the preconditioner. It may be attractive also if the ratio between computational and communication speed is high, for example when the processors are connected through a slow network. The smaller number of iterations to converge imply less communication, and in this case it may overcome the higher cost spent at local level.

The Schwarz preconditioner is often the matter of choice of many parallel linear algebra packages, because of its rather simple implementation. The minimal overlap variant is also rather attractive in term of memory usage. Yet it needs a coarse operator to obtain scalability. To this aim, here we have described an agglomeration procedure that has the advantage of generality and of a simple set up. The cost per iteration is smaller, since we need to solve the local problem only once. Yet its preconditioning properties are usually less marked, and this imply a slower convergence.

It is important to notice that all efficient DD preconditioners consist of a *local* and a *global* component. The local part, acts at the subdomain level and may possibly capture the coupling between neighboring subdomains through the interface nodes; the global part provides instead an overall communication among far away subdomains. In the Schur complement based methods, the global part is the solution of the Schur complement system itself, in the Schwarz technique this task is played by the coarse operator.

4.7 Acknowledgments

Part of this work is the result of a research carried out under the frame of the European project IDeMAS (contract number BRPR-CT97-0591). Other research agencies, such us the Italian CNR and MURST are also acknowledged for they financial support.

References

1. J. Bramble, J. Pasciak, and X. Zhang. Two-level preconditioners for 2^{nd} order elliptic finite element problems. *East-West J. Numer. Math.*, 4:99–120, 1996.
2. T. Chan, B.Smith, and J. Zou. Overlapping Schwarz methods on unstructured meshes using non-matching coarse grids. *Numer. Math.*, 73:149–167, 1996.
3. T. Chan, S. Go, and J. Zou. Boundary treatments for multilevel methods on unstructured meshes. *SIAM J. Sci. Comput.*, 21(1):46–66, 1999.
4. T. Chan and T. Mathew. The interface probing technique in domain decomposition. *SIAM Journal on Matrix Analysis and Applications*, 13(1):212–238, 1992.
5. T. Chan and T. Mathew. Domain decomposition algorithms. *Acta Numerica*, pages 61–163, 1993.
6. T. Coffey, C. Kelley, and D. Keyes. Pseudotransient continuation and differential-algebraic equations. *SIAM Journal on Scientific Computing*, 25(2):553–569, 1996.
7. H. Deconinck, H. Paillère, R. Struijs, and P. Roe. Multidimensional upwind schemes based on fluctuaction splitting for systems of conservation laws. *Comput. Mech.*, 11:323–340, 1993.
8. M. Dryja, B. Smith, and O. Widlund. Schwarz analysis of iterative substructuring algorithms for elliptic problems in three dimensions. *SIAM J. Numer. Anal.*, 31(6):1662–1694, 1993.
9. M. Dryja and O. Widlund. Domain decomposition algorithms with small overlap. *SIAM J. Sci.Comput.*, 15(3):604–620, 1994.
10. C. Farhat and F. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Meth. Engrg.*, 32:1205–1227, 1991.
11. A. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel Distrib. Technol.*, 1:12–21, August 1993.
12. F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde. Parallel geometric multigrid. In A. M. Bruaset, P. Bjørstad, and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume X of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 2005.
13. K. Hwang. *Advanced Computer Architecture: Parallelism Scalability, Programmability*. McGraw Hill, New York, 1993.
14. K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, Inc., New York, NY, USA, 1998.
15. G. Karypis and V. Kumar. METIS: Unstructured graph partitining and sparse matrix ordering system. Technical Report 98-036, University of Minnesota, Department of Computer Science, 1998.
16. P. Lin, M. Sala, J. Shadid, and R. Tuminaro. Performance of fully-coupled algebraic multilevel domain decomposition preconditioners for incompressible flow and transport. *submitted to International Journal for Numerical Methods in Engineering*, 2004.
17. A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Oxford, 1999.
18. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Thompson, Boston, 1996.
19. M. Sala. Amesos 2.0 reference guide. Technical Report SAND-4820, Sandia National Laboratories, September 2004.
20. M. Sala. Analysis of two-level domain decomposition preconditioners based on aggregation. *Mathematical Modelling and Numerical Analysis*, 38(5):765–780, 2004.
21. B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1996.

22. X.-H. Sun. Scalability versus execution time in scalable systems. *J. Parallel Distrib. Comput.*, 62(2):173–192, 2002.
23. X.-H. Sun and D. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Parallel Distrib. Systems*, 5:599–613, June 1994.
24. P. L. Tallec. Domain decomposition methods in computational mechanics. *Computational Mechanics Advances*, 1:121–220, 1994.
25. A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, New York, 2005.
26. H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.
27. H. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Num. Lin. Alg. Appl.*, 1:369–386, 1994.
28. P. Vanek, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56:179–196, 1996.
29. U. M. Yang. Parallel algebraic multigrid methods - high performance preconditioners. In A. M. Bruaset, P. Bjørstad, and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume X of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 2005.