Sparse Approximate Inverse Preconditioning for Highly Parallel Implicit Solvers in Computational Fluid Dynamics

Damien Mancy

Department of Mechanical Engineering

McGill University
Montreal, Quebec
November 2020

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering

Table of Contents

List of Figures

List of Tables

Abstract

Numerical simulations of fluid flow with the Navier-Stokes equations, discretized with the finite volume method, ultimately lead to nonlinear equations. The solution of these equations with an implicit time-stepping scheme is obtained with Newton's method, which requires the solution of a linearized problem at each step. Unfortunately, the intermediate systems of linear equations are often severely ill-conditioned. Given the dimension of the systems, iterative solvers are preferred due to their low memory requirements. The use of preconditioning is crucial to improve the conditioning of these linear systems.

The primary drawback of classical preconditioners, such as the incomplete LU factorization (ILU) and successive over-relaxation (SOR), is their strong sequential nature which makes them unsuitable for parallel machines. The use of a domain decomposition method, a divide-and-conquer framework to parallelize these sequential preconditioners, fits naturally into multiblock flow solvers. Regrettably, these strategies generally scale poorly on the ever-expanding massive parallel architectures of current supercomputers.

In this work, we investigate the inherently parallel preconditioner known as Sparse Approximate Inverse (SPAI) applied to highly parallel implicit solvers for computational fluid dynamics (CFD). The SPAI method was the topic of several theoretical and comparative studies on rather small elliptical problems, on which it proved to be robust. Nonetheless, SPAI was never extensively studied for high Reynolds number compressible viscous flows over three-dimensional geometries. In this thesis, a framework is introduced to optimize the SPAI computations within the Newton-Krylov method. It is shown that the convergence of the simulations can be reduced by a factor of almost three with appropriate SPAI recomputations compared to a naive application of SPAI. In addition, comparisons with the traditional domain decomposition methods are conducted. Whereas SPAI is more robust for excessively split domains, domain decomposition preconditioning benefits from faster convergence within the partitioning limit imposed by the CFD solver.

Abrégé

Les simulations numériques d'écoulements de fluides par les équations de Navier-Stokes, discrétisées à partir de la méthode des volumes finis, aboutissent à des systèmes d'équations non linéaires. La résolution de ces équations au moyen d'un schéma implicite est réalisée avec la méthode de Newton, qui nécessite la solution d'un système d'équations linéaires à chaque pas. Malheureusement, ces systèmes linéaires intermédiaires sont souvent très mal conditionnés. Étant donné les dimensions des systèmes d'équations en jeu, les méthodes itératives de résolution sont privilégiées du fait de leur faible consommation de mémoire. L'utilisation d'un préconditionnement pour améliorer la convergence de ces méthodes est fondamentale.

Le principal inconvénient des méthodes de préconditionnement classiques, telles que la factorisation incomplète LU (ILU) et la méthode de surrelaxation successive (SOR), est leur forte nature séquentielle, peu adaptée à une utilisation sur des machines parallèles. Le recours à des méthodes de parallélisation par décomposition de domaines, se basant sur la stratégie de « diviser pour mieux régner », s'intègre naturellement dans les solveurs multiblocs. Cependant, ces stratégies sont peu robustes lorsqu'elles sont employées sur des superordinateurs massivement parallèles.

Dans ce travail, nous étudions le préconditionnement intrinsèquement parallèle connu sous le nom de Sparse Approximate Inverse (SPAI), pour une utilisation dans un solveur implicite parallèle, appliqué à la mécanique des fluides numérique (CFD). La méthode SPAI a fait l'objet de plusieurs études théoriques et comparatives sur des problèmes elliptiques de taille réduite, pour lesquels elle s'est avérée robuste. Néanmoins, le préconditionnement SPAI n'a jamais été étudié de manière approfondie pour les écoulements visqueux compressibles à nombre de Reynolds élevé, en géométrie tridimensionnelle. Dans ce mémoire, une stratégie pour l'application du préconditionnement SPAI au sein d'une méthode de Newton-Krylov est introduite à des fins d'optimisation de calcul. Du reste, il est établi que la convergence des simulations peut être accélérée d'un facteur de presque

trois avec des recalculs judicieux du préconditionneur SPAI par rapport à une application naïve. Enfin, des comparaisons avec les préconditionnements traditionnels de décomposition de domaines sont également menées. Si d'un côté la méthode SPAI est plus robuste pour des maillages excessivement partitionnés, les préconditionnements par décomposition de domaines bénéficient souvent d'une convergence plus rapide dans la limite du partitionnement imposée par le solveur.

Acknowledgements

First, I would like to express my deepest gratitude to my supervisor Prof. Siva Nadarajah for all the help throughout the two years of my Master's degree at McGill. The feedback, devotion and the constant follow-up have been vital to the completion of this thesis.

I would like to extend my gratitude to all my friends and colleagues at the Computational Aerodynamics Group at the Department of Mechanical Engineering for the guidance, the research advice and more general discussions that have greatly helped me move forward. I would like to thank especially Dr. Aditya Kashi for taking the time to help me understand all the new programming tools and mathematical concepts I was not acquainted with.

I am also thankful to Bombardier for allowing me to use their in-house finite volume flow solver FANSC Lite in order to carry out my numerical experiments. Additionally, I am grateful to Calcul Québec and Compute Canada for helping me install the software and scientific libraries on the superclusters, as well as making it possible to run CFD tests on the Béluga supercomputer. I especially thank Bart Oldeman of Calcul Québec for his constant help with the computer clusters used for my numerical simulations.

Lastly, I would like to thank my parents, family and friends for their continued support throughout my studies who gave me strength and motivation.

Contributions

This thesis focuses on the study of parallel preconditioning methods; specifically the SPAI preconditioning method. The principal contributions of the author of this thesis, Damien Mancy, are summarized in the following list:

- An implementation of the block SPAI preconditioner in the scientific library PETSc.
- A thorough study of the SPAI and domain decomposition preconditioners for highly partitioned systems of linear equations arising from the discretization of the compressible Navier-Stokes equations.
- The design of a framework to optimize the use of SPAI preconditioning within a Newton-Krylov method.
- A study of SPAI preconditioning and domain decomposition preconditioners within a flow solver with low-order Jacobians and high-order matrix-free Jacobians.

Nomenclature

Abbreviations

ASM	(Restricted) Additive Schwarz Method
Bi-CGSTAB	Biconjugate Gradient Stabilized Method
CFD	Computational Fluid Dynamics
CFL	Courant–Friedrichs–Lewy condition
COO	Coordinate format
CPU	Central Processing Unit
CSC	Compressed Sparse Column format
CSR	Compressed Sparse Row format
GMRES	Generalized Minimal Residual Method
GPU	Graphics Processing Unit
ILU	Incomplete LU
MPI	Message Passing Interface
nnz	Number of nonzeros
RANS	Reynolds-Averaged Navier–Stokes
SA	Spalart–Allmaras one-equation model
SOR	Successive Over-Relaxation
SPAI	Sparse Approximate Inverse

Chapter 1

Introduction

1.1 Motivation

Fluid dynamics simulations have become an increasingly important tool not only for the estimation of aerodynamic performance but the design of complete aircraft geometries. In this regard, the need for reliable and fast CFD solvers to solve the highly nonlinear Navier-Stokes equations is essential. The simulation of turbulent and high Reynolds number flows is usually very complex, and requires the solution of a system of nonlinear equations.

The advent of parallel computing and the generalization of multi-core clusters have progressively allowed the use of more demanding numerical methods, such as quasi-Newton methods, for the solution of the discretized equations. The need for efficient parallel methods that can take advantage of these architectures have become increasingly important. In addition, the research focus on linear solvers has gradually shifted from the design of linear solvers to the development of robust preconditioning methods to increase their rate of convergence. Preconditioning techniques for the solution of linear systems have been studied since the early 1970s. Nonetheless, the most common methods are directly derived from basic iterative methods or matrix factorization algorithms, which remain inherently sequential.

1.2 Parallel Computing

The design of algorithms has evolved in close connection with the development of computer architectures, namely CPUs and recently GPUs. For most of the computer age, the

increased computational power of processors was mainly driven by the size reduction of transistors according to Moore's law: the number of transistors on a chip approximately doubled every year [1]. The higher density of transistors on the chips explained most of the speedup in computations throughout the 20^{th} century. Increasing the clock frequency is another way to improve the performance of a processor. Over the past decades, the global trend has been an average increase of 25% of the clock frequency per year [2].

Nevertheless, a physical limit called the "power wall" is seriously hindering the further improvement of the CPU clock frequency over a few gigahertz [3]. The simultaneous switching on of the transistors on a single chip produces a significant amount of heat that needs to be properly cooled down so as not to interfere with its operation. A further increased clock frequency and reduction in the transistor size would not be compatible with an appropriate energy consumption of the CPUs. Thus, the trend in the computer industry has shifted towards multiprocessor architectures to keep improving computer performances, while the enhancement of the single cores is slowing down [4]. This has led to an important change of course in the computer industry and the programming philosophy. Today, scientific computation is striving for harnessing the full potential of the massively parallel computing resources.

Shared Memory Architecture

The two major parallel frameworks are the shared memory architecture and the distributed memory architecture [5]. In the first configuration, the different processors directly share the same central memory (cf. Figure 1.1). On request, data from the main memory is copied to local caches associated with the processors. The data access is straightforward for processors but conflicts may arise in return when different processors request the same data in memory. In addition, the increase in the number of processors ultimately lead to an increasingly complex hierarchy of memories with additional connections and memory levels [6]. As a result, the system can become very complicated, hindering the overall performance enhancement of the added cores. OpenMP is the most generally used programming interface for this type of architecture [7]. It uses compiler directives to highlight parallelism in the code.

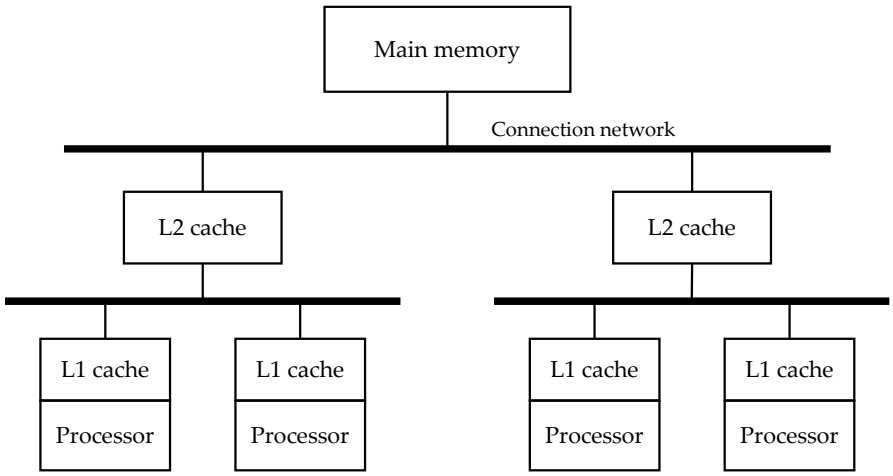


Figure 1.1: Memory hierarchy in a shared memory architecture.

Distributed Memory Architecture

A way to avoid the possible risks of memory concurrency that can occur with shared memory architectures is to restrict the memory access of the processors to their own local memory [3]. This kind of framework is referred to as the distributed memory architecture. Communications between processors are achieved through a communication network and messages between processors (cf. Figure 1.2).

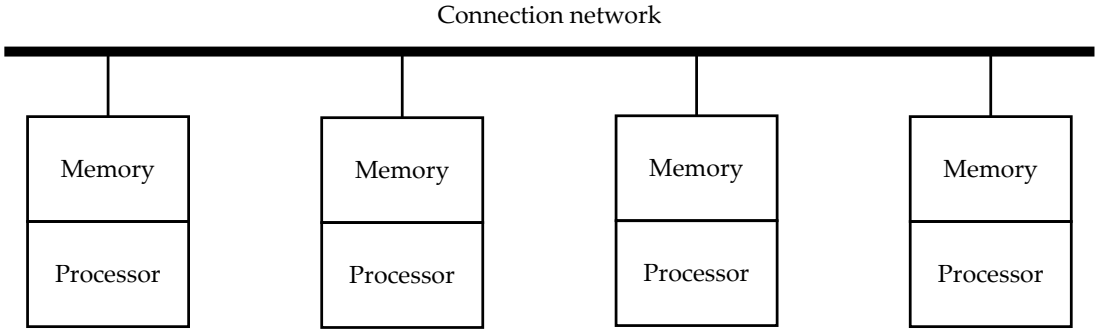


Figure 1.2: Distributed memory architecture.

The distributed memory architectures are easily scalable as the addition of more processors only requires their connection to the network, without remodeling the entire memory hierarchy. Yet, the communication of data between processors must now directly be handled by the user with communication routines in the code. In distributed

memory architectures, the standard protocol for communications between processors is the Message Passing Interface standard (MPI) [8]. It defines a set of generic routines for the synchronizations and the communications between the different processors.

Hybrid Memory Architecture

In practice, both memory architectures are usually combined in a hybrid architecture in large-scale supercomputers [6]. At the highest level, processors are gathered into different nodes according to the distributed memory paradigm. The nodes are linked together by a communication network and the memory is distributed across the different nodes (cf. Figure 1.3). On the other hand, multicore CPUs are generally assembled in a socket within an individual node and operate on the shared memory model. This two-level parallel architecture, between and within the nodes, must be explicitly used in the code to exploit the full potential of the supercomputers.

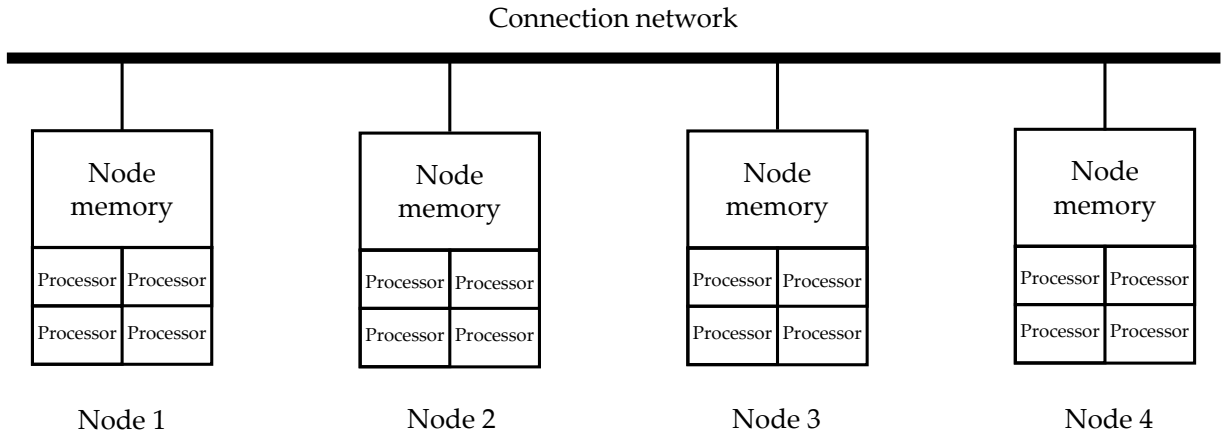


Figure 1.3: Hybrid memory architecture with four nodes.

Since the early 2000s and the development of multiprocessor CPUs for the general public, clusters made of tens, then hundreds and thousands of processors for scientific computing have rapidly become widespread [9].

1.3 Review of Solution Methods in Flow Solvers

Historically, explicit time-marching methods have been the preferred choice to converge towards a steady-state solution of the discretized Navier-Stokes equations. They offer low memory requirements and are fairly simple to implement [10]. Nevertheless, these methods rapidly become ineffective for the simulation of very complex flows on large grids, requiring very short time steps to ensure convergence [11].

The restrictions on the time step can be overcome by using implicit methods. Thanks to larger time steps, these methods are able to achieve convergence in fewer steps. Yet, the computational cost of each step is considerably higher [12]. In this regard, a large system of linear equations must be solved at each time step. One of the first implementation of implicit methods is known to be the Alternating Direction Implicit scheme in the 1950s [13]. This method employs an approximate factorization of the coefficient matrices to solve the linear systems.

The emergence of new computational resources during the 1980s led researchers to investigate Newton's method. Its main advantages are its robustness and prospective quadratic convergence. However, the required exact solutions of large systems of linear equations are prohibitive. Consequently, the simulation results in a very slow convergence in terms of computational time. The use of quasi-Newton methods, in which the subsequent linear systems are approximately solved with iterative solvers [14], rapidly yielded promising results. These methods are able to take advantage of the robustness of Newton's method for nonlinear equations and the reduced costs of iterative solvers for the linear equations compared to exact linear solvers.

Classical iterative methods, such as Jacobi and Gauss-Seidel methods, for the solution of linear equations are often inefficient for the stiff systems arising from CFD problems. The popular Krylov subspace methods can overcome this lack of robustness provided they are coupled with an effective preconditioner [15, 16]. In these methods, the approximate solution is searched within a Krylov subspace. The first implementation of a Krylov method dates back to the 1950s with the Conjugate Gradient method for symmetric positive definite matrices [17]. In the following years, a wide variety of Krylov methods were developed for symmetric and then more general matrices. Generalized Minimal Residual

(GMRES) [18] and Biconjugate Gradient Stabilized Method (Bi-CGSTAB) [19] are the most popular solution methods for general large systems of linear equations. A preconditioning method to transform the linear equations into an easier form is generally mandatory, if only to achieve convergence. In current implicit CFD solvers using Newton-Krylov methods, the solution of the systems of linear equations can account for between 70% to 90% of the overall simulation time [20]. Therefore, the improvement of existing solution methods has become an active research field over the last decades. To this end, the optimization of preconditioning methods has gained more and more importance compared to the improvement of Krylov methods [21]. In this thesis, we focus on parallel preconditioners that we believe will be of paramount importance in the coming years with the improvement of fine-grained parallelism in computer architectures.

1.4 Organization of the Thesis

The chapters in this thesis are concisely summarized in this section. An overview of the solution methods for systems of linear and nonlinear equations is given in Chapter 2. Afterwards, the Navier-Stokes equations and discretization schemes used in CFD are reviewed in Chapter 3. The parallel preconditioning method SPAI as well as its implementation are discussed in Chapter 4. In addition, a comparative study between the sequential preconditioners SOR, ILU and the parallel SPAI is conducted on benchmark matrices of modest sizes. In the same chapter, a preconditioning framework is proposed to use SPAI preconditioning within a Newton-Krylov method. Lastly, Chapter 5 gathers the results of SPAI preconditioning and domain decomposition preconditioners for systems of linear equations on highly partitioned domains from an implicit CFD solver. In the last section, SPAI and domain decomposition preconditioners are employed to precondition the linear systems in the flow solver FANSC.

Chapter 2

Methods for Solving Systems of Linear and Nonlinear Equations

This chapter begins with the introduction of solvers for the solutions of systems of linear equations in Section 2.1 with an emphasis on Krylov subspace methods via the GMRES method. The concept of preconditioning in order to increase their rates of convergence is presented in Section 2.2. Finally, the inexact Newton methods for solving nonlinear equations based on the approximate solutions of successive systems of linear equations are detailed in Section 2.3.

2.1 Solving Linear Systems

Many discretization methods such as the finite difference, the finite volume and the finite element methods can be used to transform a set of partial differential equations into a set of discretized algebraic equations. The core of their application generally consists of solving large and sparse systems of linear equations. As a consequence, a broad variety of algorithms to efficiently solve such systems of equations have been designed in the second half of the 20^{th} century, along with the development of computers. A historical survey on the development of linear solvers is presented in [22]. Classical numerical algorithms were initially developed at a time when computer architectures were highly sequential, for which the concern for parallelism is not central.

Typically, a system of linear equations is denoted via the matrix notation

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{2.1}$$

The n unknowns in the vector $\mathbf{x} \in \mathbb{R}^n$ satisfy n linear equations defined by the coefficients of the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a right-hand side vector $\mathbf{b} \in \mathbb{R}^n$. Provided that matrix $\mathbf{A}$ is nonsingular, the solution of (2.1) is guaranteed to exist and be unique. We denote $\mathbf{x}_*$ the exact solution of the linear system. The choice of the appropriate method to solve the linear system depends on the properties of the coefficient matrix $\mathbf{A}$ such as its size, density, symmetrical properties and positive definiteness.

Historically, direct methods for the solution of such systems were first developed. These methods solve exactly the linear systems, down to an inherent floating point error, in a finite number of operations [23]. In general, the coefficient matrix $\mathbf{A}$ is transformed into a more suitable form, such as a factorization of triangular matrices based on the Gaussian elimination, which can be more easily solved [24]. We can mention the Cholesky factorization for positive definite matrices and the LU factorization in the general case. Today, direct methods are mainly used for the solution of dense systems of equations. They tend to be very robust and barely sensitive to perturbations. Nonetheless, they rarely take a real advantage of the sparsity of the coefficient matrix and their computational cost and memory requirements increase dramatically with the number of unknowns. Some paralellizations of the direct methods have proven to be efficient for problems up to a few millions of unknowns in the accurate inversion of the same coefficient matrix for several different right-hand sides [25]. In these methods, a matrix factorization is only computed once with a direct method and quickly applied to the numerous right-hand sides. In practice, the most common parallel approaches involve either a multifrontal Gaussian elimination, such as in the parallel scientific library MUMPS [26], or a supernodal technique [27]. These direct methods are still current in certain fields such as electromagnetic geophysical simulations [28–30]. An in-depth review of the sequential and parallel direct methods for sparse matrices is available in the recent survey [31].

Unlike direct methods, iterative methods compute successive approximations of the solution of the linear systems converging to the sought solution. They can be classified into two categories: the stationary and nonstationary iterative methods [32]. The stationary methods are easier to use, as the approximate solution is calculated from a simple recurrence relation. Yet, they usually yield slow convergence towards the solution of the problem and may even diverge for particularly complex problems. On the other hand,

the nonstationary methods are based on the approximation of the solution with projections on suitable subspaces. They gather the well-known Krylov subspace methods and are usually the only tractable choice for industrial problems. A historical review of the iterative methods is available in [33].

The most common iterative methods are now very matured methods with a sound mathematical background. In many scientific applications, the discretization process produces very ill-conditioned coefficient matrices for which convergence might be slow or not even ensured. Hence, the initial systems are often transformed into an equivalent system with more suitable spectrum properties through a process more commonly referred to as preconditioning. The additional construction and application cost of the preconditioning in the iterative method is then overcome by the gain in the convergence rate.

In the following section, although being rarely used as linear solvers in practice, the stationary methods are first presented, since they are employed later as preconditioning methods. Thereafter, the Krylov subspace methods, the standard bearers of the nonstationary methods, are extensively explained with the GMRES method.

2.1.1 Stationary Methods

In stationary methods, the solution of the system (2.1) is approached via the construction of a sequence of vectors $\mathbf{x}^{(p)} \in \mathbb{R}^n$ recursively calculated with the relation

$$\mathbf{x}^{(p+1)} = \mathbf{M}\mathbf{x}^{(p)} + \mathbf{c}, \tag{2.2}$$

with $\mathbf{M} \in \mathbb{R}^{n\times n}$ being the iteration matrix, and $\mathbf{c}$ a vector in $\mathbb{R}^n$, both defined from the coefficient matrix $\mathbf{A}$ and $\mathbf{b}$ [34]. Iterative methods become competitive with direct methods when the required number of steps to converge to the solution for a prescribed tolerance is sufficiently low.

The simplest form of a stationary method is the Richardson iteration [35] that defines a fixed-point iteration with $\mathbf{M} = \mathbf{I} - \omega\mathbf{A}$ and $\mathbf{c} = \omega\mathbf{b}$. The constant $\omega \neq 0$ is an acceleration parameter. In this way, the recurrence relation is

$$\mathbf{x}^{(p+1)} = (\mathbf{I} - \omega\mathbf{A})\mathbf{x}^{(p)} + \omega\mathbf{b} = \mathbf{x}^{(p)} + \omega\left(\mathbf{b} - \mathbf{A}\mathbf{x}^{(p)}\right) = \mathbf{x}^{(p)} + \omega\mathbf{r}^{(p)}, \tag{2.3}$$

where $\mathbf{r}^{(p)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(p)}$ denotes the residual of the iterate $\mathbf{x}^{(p)}$. The fixed point of the recurrence relation in equation (2.3) is the solution of the linear system (2.1).

Convergence of Stationary Iterative Methods

Many general theorems have been found in regards to the convergence of such iterative methods. In particular, the convergence rate of iterative methods is characterized by the spectral radius of the iteration matrix $\rho(\mathbf{M})$, given by

$$\rho(\mathbf{M}) = \inf\{||\mathbf{M}||, \text{ where } ||\cdot|| \text{ is a matrix norm}\}. \tag{2.4}$$

Theorem 1 gives a very general rule on the convergence of iterative methods [34].

Theorem 1 *The iteration process defined by (2.2) is convergent, for all starting vectors* $\mathbf{x}_0 \in \mathbb{R}^n$, *if and only if* $\rho(\mathbf{M}) < 1$.

Different iterative methods yield various iterative matrices $\mathbf{M}$ with different convergence properties.

Splitting Methods

Classical iterative methods are based on the splitting of the coefficient matrix $\mathbf{A} = \mathbf{P} - \mathbf{N}$ to determine the iterative matrix $\mathbf{M}$, with $\mathbf{P}$ a nonsingular matrix [35]. The splitting defines the fixed point iteration

$$\mathbf{x}^{(p+1)} = \mathbf{P}^{-1}\mathbf{N}\mathbf{x}^{(p)} + \mathbf{P}^{-1}\mathbf{b}. \tag{2.5}$$

Noting that $\mathbf{P}^{-1}\mathbf{N} = \mathbf{P}^{-1}(\mathbf{P} - \mathbf{A}) = \mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$, the iteration process in (2.5) can be rewritten

$$\mathbf{x}^{(p+1)} = \mathbf{x}^{(p)} + \mathbf{P}^{-1}\mathbf{r}^{(p)}, \tag{2.6}$$

with $\mathbf{r}^{(p)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(p)}$ denoting the residual of the iterate $\mathbf{x}^{(p)}$.

We list in Table 2.1 the most common matrix splittings for stationary iterative methods. The matrix $\mathbf{A}$ is decomposed into the sum $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$, where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$, and $\mathbf{L}$ and $\mathbf{U}$ are the strictly lower and upper triangular parts of $\mathbf{A}$ respectively.

The simple Jacobi and Gauss-Seidel methods often lead to slow convergence, if not divergence. In order to increase the rate of convergence, a relaxation parameter ω can be used to weight the correction obtained with the Gauss-Seidel method.

Table 2.1: Classical splitting methods.

Splitting Method	Matrix $\mathbf{P}$	Matrix $\mathbf{N}$
Richardson	$\omega^{-1}\mathbf{I}$	$\omega^{-1}\mathbf{I} - \mathbf{A}$
Jacobi	$\mathbf{D}$	$\mathbf{L} + \mathbf{U}$
Gauss-Seidel	$\mathbf{D} - \mathbf{L}$	$\mathbf{U}$
Damped Jacobi	$\omega^{-1}\mathbf{D}$	$(\omega^{-1} - 1)\mathbf{D} + \mathbf{L} + \mathbf{U}$
Successive Over-Relaxation (SOR)	$\omega^{-1}\mathbf{D} - \mathbf{L}$	$(\omega^{-1} - 1)\mathbf{D} + \mathbf{U}$
Symmetric SOR	$\frac{(\mathbf{D} - \omega\mathbf{L})\mathbf{D}^{-1}(\mathbf{D} - \omega\mathbf{U})}{\omega(2-\omega)}$	$\frac{[(1-\omega)\mathbf{D} + \omega\mathbf{L}]\,\mathbf{D}^{-1}\,[(1-\omega)\mathbf{D} + \omega\mathbf{U}]}{\omega(2-\omega)}$

2.1.2 Krylov Subspace Methods

The stationary iterative methods just presented compute a converging sequence of approximations towards the solution with a relatively simple recurrence relation. This simplicity comes at the expense of slow convergences, making them unsuitable for tackling poorly conditioned systems.

In contrast, the Krylov subspace methods designate a large class of iterative methods for linear systems of equations. They involve the projection of the approximate solution onto a relevant subspace which is extended at each step [36]. The different Krylov subspace methods differ in the range of matrices they are able to treat. For instance, some Krylov methods can only solve positive definite systems of equations, while others assume nothing about the coefficient matrix [37]. In addition, their computational cost and storage requirements are also substantially different. These methods are extensively used for the solution of large and sparse linear systems of equation arising from industrial applications. The most popular Krylov subspace methods are reviewed from an algorithmic point of view by Barrett et al. in [32], including the GMRES method that we present in detail in this section.

First, let us introduce some general notations to present the Krylov subspace methods. Along the algorithm, a sequence of spaces with an increasing dimension, called the Krylov subspaces and denoted as $\mathcal{K}_k$, is constructed. At the k-th step of the iterative process, the Krylov subspace denotes the space

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{k-1}\mathbf{r}_0\}, \tag{2.7}$$

with $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ being the residual corresponding to the initial guess $\mathbf{x}_0$. The k-th iterate $\mathbf{x}_k$ is calculated as a projection onto the Krylov subspace:

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0). \tag{2.8}$$

A Krylov subspace method provides a framework to iteratively compute the basis vectors $\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\}$ of the subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ and find the projection of $\mathbf{x}_k$ on $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. The set $\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{k-1}\mathbf{r}_0\}$ forms a natural basis for the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ but its vectors usually become almost linearly dependent as k increases. In effect, the sequence of vectors $\left(\mathbf{A}^k\mathbf{r}_0\right)_{k\in\mathbb{N}}$ converges towards an eigenvector corresponding to the greatest eigenvalue of $\mathbf{A}$ in absolute norm according to the power iteration algorithm [38]. The Lanczos and Arnoldi methods, based on the Gram-Schmidt orthogonalization, are two main iterative processes to build an orthogonal basis $\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\}$ for the subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ and are at the core of many Krylov methods [39]. In both methods, the extension of the subspace is carried out by subsequent orthogonalizations of $\mathbf{A}^k\mathbf{r}_0$ against the subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\}$.

In most of the Krylov methods, the matrix $\mathbf{A}$ is only required through its application in sparse matrix-vector multiplications. It allows the use of matrix-free methods, for which the matrix $\mathbf{A}$ is implicitly provided by a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n, \mathbf{x} \longmapsto \mathbf{A}\mathbf{x}$, without the explicit storage of the coefficients in $\mathbf{A}$.

Arnoldi Method

Let us consider that we already have k orthogonal basis vectors $\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\}$ spanning the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\} = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, ..., \mathbf{A}^{k-1}\mathbf{r}_0\}$. Hence,

the vector $\mathbf{q}_k$ can be written as a linear combination of the vectors $(\mathbf{A}^j\mathbf{r}_0)_{j\in\{0,\dots,k-1\}}$:

$$\mathbf{q}_k = \sum_{j=0}^{k-1} \alpha_j \mathbf{A}^j \mathbf{r}_0. \tag{2.9}$$

It is straightforward to show that $\mathbf{A}\mathbf{q}_k$ is in the Krylov subspace $\mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0)$:

$$\mathbf{A}\mathbf{q}_k = \sum_{j=0}^{k-1} \alpha_j \mathbf{A}^{j+1}\mathbf{r}_0 = \sum_{j=0}^{k-2} \alpha_j \mathbf{A}^{j+1}\mathbf{r}_0 + \alpha_{k-1}\mathbf{A}^k\mathbf{r}_0 \ \in \ \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{r}_0). \tag{2.10}$$

The Arnoldi vector $\mathbf{q}_{k+1}$ is obtained with the orthogonalization of $\mathbf{A}\mathbf{q}_k$ against $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, whose component $\mathbf{r}_k$, orthogonal to $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, is given by

$$\mathbf{r}_k = \mathbf{A}\mathbf{q}_k - \sum_{j=1}^{k} (\mathbf{q}_j^T \mathbf{A}\mathbf{q}_k)\mathbf{q}_j. \tag{2.11}$$

If $\mathbf{r}_k = \mathbf{0}$, then the process is interrupted meaning that $\forall l \geq k$, $\mathcal{K}_l(\mathbf{A}, \mathbf{r}_0) = \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. Consequently, the Krylov subspace is invariant by additional Arnoldi steps and the exact solution $\mathbf{x}_*$ is in $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. Otherwise, $\mathbf{r}_k$ is normalized to fetch $\mathbf{q}_{k+1}$:

$$\mathbf{q}_{k+1} = \frac{\mathbf{r}_k}{||\mathbf{r}_k||_2}. \tag{2.12}$$

Besides, since $\mathbf{q}_{k+1}$ is orthogonal to $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, we have

$$\mathbf{q}_{k+1}^T \mathbf{r}_k = ||\mathbf{r}_k||_2 = \mathbf{q}_{k+1}^T \mathbf{A}\mathbf{q}_k. \tag{2.13}$$

Then, the Arnoldi basis satisfies

$$\mathbf{A}\mathbf{q}_k = \sum_{j=1}^{k+1} h_{j,k}\mathbf{q}_j, \tag{2.14}$$

where $h_{j,k} = \mathbf{q}_j^T \mathbf{A}\mathbf{q}_k$. Therefore, $\mathbf{H}_k = \left(h_{i,j}\right)_{i,j\in\{1,\dots,k\}}$ defines a $k \times k$ upper Hessenberg matrix, since $\forall j$, $\mathbf{A}\mathbf{q}_j \in \mathcal{K}_{j+1}(\mathbf{A}, \mathbf{r}_0) \ \perp \ \mathbf{q}_{j+2}$. Hence, denoting $\mathbf{Q}_k$, the $n \times k$ matrix

$\begin{bmatrix}\mathbf{q}_1 & \cdots & \mathbf{q}_k\end{bmatrix}$, the matrix form of the step (2.14) is written as

$$\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_k\mathbf{H}_k + \begin{bmatrix}\mathbf{0} & \cdots & \mathbf{0} & h_{k+1,k}\mathbf{q}_{k+1}\end{bmatrix} = \mathbf{Q}_{k+1}\tilde{\mathbf{H}}_k, \tag{2.15}$$

with $\tilde{\mathbf{H}}_k = \begin{bmatrix}\mathbf{H}_k \\ \hline 0 & \cdots & 0 & h_{k+1,k}\end{bmatrix}$, a $(k+1)\times k$ matrix. The pseudo-code for the Arnoldi iteration is given in Algorithm 1.

Algorithm 1 Arnoldi Iteration.

Require: An initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, and the maximum number of iterations N_{iter}.

$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{q}_1 \leftarrow \mathbf{r}_0/||\mathbf{r}_0||_2$
$k \leftarrow 1$
while $k \leq N_{iter}$ **do**
    $\tilde{\mathbf{q}} \leftarrow \mathbf{A}\mathbf{q}_k$
    for j from 1 to k **do**
        $h_{j,k} \leftarrow \mathbf{q}_j^T\tilde{\mathbf{q}}$
        $\tilde{\mathbf{q}} \leftarrow \tilde{\mathbf{q}} - h_{j,k}\mathbf{q}_j$
    end for
    $h_{k+1,k} \leftarrow ||\tilde{\mathbf{q}}||_2$
    $\mathbf{q}_{k+1} \leftarrow \tilde{\mathbf{q}}/h_{k+1,k}$
    $k \leftarrow k+1$
end while

GMRES Method

The Generalized Minimal Residual (GMRES) method, first proposed by Saad and Schultz [40] in 1985, is a widely used Krylov subspace method to solve general unsymmetric systems of equations. After k steps, the iterate $\mathbf{x}_k$ minimizes the residual in the Euclidean norm over the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$:

$$\mathbf{x}_k = \underset{\mathbf{x}\in\mathcal{K}_k(\mathbf{A},\mathbf{r}_0)}{\arg\min}\ ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2^2. \tag{2.16}$$

In the absence of symmetry, GMRES employs the general Arnoldi method to construct a sequence of orthogonal basis vectors for the Krylov subspace. As a consequence, the

basis vectors $\mathbf{q}_k$ must be kept in memory and the basis is expanded with a complete orthogonalization of $\mathbf{A}\mathbf{q}_k$ against $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. Hence, the memory and computational requirements grow linearly with the iteration count as all the orthogonal vectors accumulate in the memory. This major drawback can prevent the use of the GMRES method when the available storage is insufficient.

In GMRES, the k-th step starts with the orthogonalization of the vector $\mathbf{A}\mathbf{q}_k$ against the current Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ following the Arnoldi process, yielding the new computed Arnoldi vector $\mathbf{q}_{k+1}$. Consequently, any vector $\mathbf{x}$ on the subspace $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ can be written as

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}, \tag{2.17}$$

with $\mathbf{y}$, a vector of size k, defining the coefficients of $\mathbf{x}$ on the Krylov subspace. The Euclidean norm of the residual defined by $\mathbf{x}$ can be written as

$$\begin{aligned} ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 &= ||\mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{Q}_k \mathbf{y})||_2 \\ &= ||\mathbf{r}_0 - \mathbf{A}\mathbf{Q}_k \mathbf{y}||_2 \\ &= ||\beta \mathbf{q}_1 - \mathbf{Q}_{k+1} \tilde{\mathbf{H}}_k \mathbf{y}||_2 \\ &= ||\mathbf{Q}_{k+1}(\beta \mathbf{e}_1 - \tilde{\mathbf{H}}_k \mathbf{y})||_2 \\ ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 &= ||\beta \mathbf{e}_1 - \tilde{\mathbf{H}}_k \mathbf{y}||_2, \end{aligned} \tag{2.18}$$

with $\beta = ||\mathbf{r}_0||_2$. The last equality comes from the orthogonality of the basis vectors $\mathbf{q}_i$.

The minimization of the residual in (2.16) is equivalent to finding the vector $\mathbf{y}$ that minimizes the norm (2.18). This is a simple linear least-squares problem of dimension $(k+1) \times k$ for $\mathbf{y}$, which is generally far less than the dimension of the linear system. Moreover, because of the peculiar structure of the matrix $\tilde{\mathbf{H}}_k$, which is almost triangular, the least-squares problem is solved with successive inexpensive Givens rotations to triangularize the system [41, 42]. The solution $\mathbf{y}_k$ of the least-squares problem yields the approximate solution of the system of linear equations: $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k \mathbf{y}_k$. Moreover, the residual norm directly comes as a byproduct of the triangularisation of $\tilde{\mathbf{H}}_k$ in solving the least-squares problem [43]. As a result, the convergence check is directly performed and the approximate solution $\mathbf{x}_k$ is only explicitly built at the end of the GMRES process.

Computational Cost and Storage Requirements of GMRES

The number of operations to orthogonalize the new search directions and the storage requirements for Krylov vectors increase linearly with the iteration count. In order to overcome this main issue, the GMRES algorithm is almost always used in a restarted version GMRES(m), where the iterations are periodically restarted every m iterations, until the convergence is achieved [44]. After m iterations, the saved Krylov vectors are erased from memory and the GMRES algorithm is restarted with the latest iterate as the initial guess. Nevertheless, the convergence to the solution is in theory no longer ensured with this modification. Besides, there is no general rule for the determination of an appropriate restart value m [45]. As a matter of fact, a large value for m would require a prohibitive amount of memory to store all the vectors $\mathbf{q}_k$, and the orthogonalization stage would be unreasonably computationally expensive. On the contrary, the method can stall or even diverge when m is taken too low.

In terms of operation count, each step i, counted since the last restart, requires $i+1$ dot products, $i+1$ AXPY [1] operations, one matrix-vector product and a preconditioner application [32]. In addition, a new vector $\mathbf{q}_{i+1}$ is kept in memory.

One of the interesting features of the GMRES method is that the coefficient matrix is only involved through a matrix-vector multiplication. The sparsity of $\mathbf{A}$ makes this operation all the more inexpensive. The pseudo-code of the restarted GMRES(m) is given in Algorithm 2.

Alternative Krylov Methods

The Biconjugate Gradient Stabilized (Bi-CGSTAB) algorithm is certainly the most popular Krylov method after the GMRES algorithm for general nonsymmetric matrices. This method has been widely tested for the solutions of linear systems derived from the linearization of the Navier-Stokes equations [46]. Unlike GMRES, Bi-CGSTAB does not need to store the Krylov vectors but requires two sparse matrix-vector multiplications and two preconditioner applications per step. In some cases, Bi-CGSTAB may perform better than the GMRES solver [47]. In our preconditioning tests, we occasionally compared the pre-

[1] The AXPY operation denotes the basic vector addition $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$, with α a scalar.

Algorithm 2 Restarted GMRES(m) iterative method.

Require: An initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, the tolerance $\tau \in \mathbb{R}$, and the maximum number of iterations N_{iter}.

$j \leftarrow 1$
while $j \leq N_{iter}$ **do**
  $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$
  $\beta \leftarrow ||\mathbf{r}||_2$
  $\mathbf{q}_1 \leftarrow \mathbf{r}/\beta$
  for i from 1 to m **do**
    $j \leftarrow j + 1$
    $\tilde{\mathbf{q}} \leftarrow \mathbf{A}\mathbf{q}_i$
    for k from 1 to i **do**
      $h_{k,i} \leftarrow \mathbf{q}_k^T \tilde{\mathbf{q}}$
      $\tilde{\mathbf{q}} \leftarrow \tilde{\mathbf{q}} - h_{k,i}\mathbf{q}_k$
    end for
    $h_{i+1,i} \leftarrow ||\tilde{\mathbf{q}}||_2$
    $\mathbf{q}_{i+1} \leftarrow \tilde{\mathbf{q}}/h_{i+1,i}$
    $\rho \leftarrow \min_{y_i} ||\beta\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y}_i||_2$ ▷ Solve the least-squares problem
    if $\rho < \tau$ or $j > N_{iter}$ **then**
      $\mathbf{x}_i \leftarrow \mathbf{x}_0 + \mathbf{Q}_i\mathbf{y}_i$ ▷ Build the solution
      Exit the loops
    end if
  end for
end while

conditioned GMRES and Bi-CGSTAB methods without noticing substantial differences in the performances.

2.2 Preconditioning

The discretization of scientific problems into systems of linear equations often yields ill-conditioned systems with a low rate of convergence when solved with iterative methods. The rate of convergence of these methods is generally related to the spectral properties of the coefficient matrix **A** such as its eigenvalue distribution and its condition number [34, 48]. Well-conditioned matrices should have clustered eigenvalues around the unity on the complex plane and a low condition number to attain fast convergence in iterative methods.

To this end, a preconditioning step of transforming the original system into an equivalent system with more appropriate properties is required before using an iterative method. The preconditioner denotes the matrix $\mathbf{M}$ that applies the preconditioning transformation. The left-preconditioned system is written as

$$\mathbf{MAx} = \mathbf{Mb}. \tag{2.19}$$

It is clear that the solution of the linear system (2.19) is the same as the original set of linear equations (2.1) but the spectral properties of the preconditioned system $\mathbf{MA}$ can be more favorable by clustering the eigenvalues of $\mathbf{A}$ within a sufficiently small neighborhood around unity [35]. Thus, to be an effective preconditioner, the matrix $\mathbf{M}$ would have to be a suitable approximation of the inverse $\mathbf{A}^{-1}$ in some sense. Using as a preconditioner $\mathbf{M} = \mathbf{A}^{-1}$ might seem to be an optimal preconditioner, solving the linear system in one step as $\mathbf{x} = \mathbf{Mb}$, but the computation of the matrix inverse $\mathbf{A}^{-1}$ is far from evident and its computational cost is equivalent to using a direct method.

In practice, the product $\mathbf{MA}$ in (2.19) is never explicitly computed, since only the action of the coefficient matrix is required in Krylov subspace methods [21]. The preconditioning of the system of linear equations entails an additional sparse-matrix product $\mathbf{y} = \mathbf{Mx}$ in GMRES. The pseudo-code of the preconditioned GMRES(m) Krylov method is presented in Algorithm 3.

So far, only explicit preconditioners $\mathbf{M}$, which are approximations of the inverse matrix $\mathbf{A}^{-1}$, have been presented. Similarly, an implicit preconditioner $\mathbf{P}$ can directly approximate the matrix $\mathbf{A}$. Consequently, the step $\tilde{\mathbf{q}} = \mathbf{MAq}_i$ in the GMRES algorithm is substituted by solving an inexpensive system of linear equations $\mathbf{P}\tilde{\mathbf{q}} = \mathbf{Aq}_i$, for $\tilde{\mathbf{q}}$.

There are three different ways to precondition a linear system:

- The left preconditioned system is obtained by multiplying the equation $\mathbf{Ax} = \mathbf{b}$ on the left side by $\mathbf{M}$, yielding $\mathbf{MAx} = \mathbf{Mb}$.
- For right preconditioning, the system becomes $\mathbf{AMy} = \mathbf{b}$, with $\mathbf{x} = \mathbf{My}$.
- Finally, the two-sided preconditioning employs a split preconditioner as follows: $\mathbf{M_l A M_r y} = \mathbf{M_l b}$, with $\mathbf{x} = \mathbf{M_r y}$.

Algorithm 3 Preconditioned restarted GMRES(m) iterative method.

Require: An initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, the tolerance $\tau \in \mathbb{R}$, and the maximum number of iterations N_{iter}.

$j \leftarrow 1$
while $j \leq N_{iter}$ **do**
  $\mathbf{r} \leftarrow \mathbf{M}(\mathbf{b} - \mathbf{A}\mathbf{x}_0)$
  $\beta \leftarrow ||\mathbf{r}||_2$
  $\mathbf{q}_1 \leftarrow \mathbf{r}/\beta$
  for i from 1 to m **do**
    $j \leftarrow j + 1$
    $\tilde{\mathbf{q}} \leftarrow \mathbf{M}\mathbf{A}\mathbf{q}_i$
    for k from 1 to i **do**
      $h_{k,i} \leftarrow \mathbf{q}_k^T \tilde{\mathbf{q}}$
      $\tilde{\mathbf{q}} \leftarrow \tilde{\mathbf{q}} - h_{k,i}\mathbf{q}_k$
    end for
    $h_{i+1,i} \leftarrow ||\tilde{\mathbf{q}}||_2$
    $\mathbf{q}_{i+1} \leftarrow \tilde{\mathbf{q}}/h_{i+1,i}$
    $\rho \leftarrow \min_{y_i} ||\beta\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y}_i||_2$ ▷ Solve the least-squares problem
    if $\rho < \tau$ or $j > N_{iter}$ **then**
      $\mathbf{x}_i \leftarrow \mathbf{x}_0 + \mathbf{Q}_i\mathbf{y}_i$ ▷ Build the solution
      Exit the loops
    end if
  end for
end while

Yet, the choice of $\mathbf{M}$ as a preconditioner cannot be motivated solely by its approximation of $\mathbf{A}^{-1}$. The construction, application and storage costs of the preconditioner $\mathbf{M}$ come with a significant additional cost and memory requirements that need to be taken into account to devise adequate preconditioners. In addition to its intrinsic performance, the parallel efficiency of a preconditioner is also an important factor when considering its use on parallel architectures, as many preconditioning techniques are highly sequential. In the following subsections, we present some of the well-known preconditioning techniques.

2.2.1 Splitting-Based Preconditioners

The classical stationary iterative methods can be used as preconditioners in the Krylov subspace methods to improve their convergence rates. These preconditioning methods are directly based on the splitting of the matrix $\mathbf{A}$. Their advantage is that they do not

require additional storage and have no construction cost, since the information is simply extracted from the coefficients of $\mathbf{A}$ [49]. Their application is easily carried out with a diagonal scaling for the Jacobi preconditioner and a forward or a backward substitution for the triangular splittings. We recall the form of a stationary iterative method defined with the splitting $\mathbf{A} = \mathbf{P} - \mathbf{N}$:

$$\mathbf{x}^{(p+1)} = \mathbf{P}^{-1}\mathbf{N}\mathbf{x}^{(p)} + \mathbf{P}^{-1}\mathbf{b}. \tag{2.20}$$

This recursive relation converges if and only if $\rho(\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}) < 1$. Each of the methods in Table 2.1 defines an implicit preconditioner $\mathbf{P}$. For instance, the Jacobi preconditioner is easily computed as the diagonal $\mathbf{P} = \mathbf{D}$ of the coefficient matrix $\mathbf{A}$. Thanks to their simplicity, splitting-based preconditioners such as Jacobi and Gauss-Seidel are often employed as smoothers in multigrid methods, due to their ability to efficiently remove the high-frequency oscillations in the solution [50]. However, they usually cannot compete with more robust preconditioners in solving large systems of linear equations.

2.2.2 Incomplete Factorization Preconditioners

A broad range of implicit preconditioners is based on the classical matrix factorization algorithms, which consist in decomposing a given matrix into a product of matrix factors [21]. Well-known factorization techniques such as LU or Cholesky are generally employed for the solution of dense linear systems as direct methods. The first one decomposes a matrix $\mathbf{A}$ into the product of two matrices $\mathbf{L}$ and $\mathbf{U}$, which are respectively upper and lower triangular matrices [24].

Yet, the classical factorization algorithms for sparse matrices often come with an important fill-in of the factors, making them unsuitable for computational purposes. In order to preserve sparsity for factored matrices, incomplete factorizations can be performed by discarding some entries in the factorization process. The so-called incomplete LU (ILU) factorization constructs implicit preconditioners of the form $\mathbf{P} = \hat{\mathbf{L}}\hat{\mathbf{U}}$ where $\hat{\mathbf{L}}$, a lower triangular matrix, and $\hat{\mathbf{U}}$, an upper triangular matrix, are approximate sparse

factors. The factorization is written

$$\mathbf{A} = \hat{\mathbf{L}}\hat{\mathbf{U}} + \mathbf{E}, \tag{2.21}$$

with $\mathbf{E}$ being the error matrix introduced by the incomplete factorization.

The accuracy of the preconditioner is determined by the number of nonzero entries allowed in the sparse factors. More entries in the factors $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ yield a better approximation of the matrix $\mathbf{A}$ with a lower error $||\mathbf{E}||$. Consequently, they are likely to drastically reduce the number of iterations in the linear solver. On the other hand, the computational cost of forming the factors and applying the ILU preconditioner rises. Different dropping criteria to monitor the retained entries in $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ have been devised. They are either based on the level of fill-in or a threshold on the magnitude of the entries.

Level of Fill-in

In the first category, the method is denoted ILU(l) for an integer $l > 0$, defining the allowed entries in the factors of the ILU factorization [34, 51, 52]. Initially, an integer $level_{i,j}$ is assigned to all the entries in the matrix $\mathbf{A}$, such as

$$level_{i,j} = \begin{cases} 0 & \text{if } a_{i,j} \neq 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{2.22}$$

These *level* values are updated along the LU factorization, in such a way that every time a nonzero entry $a_{i,j}$ is to be modified, its assigned $level_{i,j}$ is updated. If $level_{i,j} > l$, the corresponding entry is dropped in the ILU factors.

Thus, a larger value for the fill-in parameter l means denser preconditioners. For $l = 0$, or no fill-in, the sparsity patterns of the ILU factors are the same as the sparsity pattern of $\mathbf{A}$ [21]. The choice of a proper value of fill-in is mostly problem dependent. The same value of fill-in parameter l may yield very different preconditioner densities depending on the matrix. We provide the pseudo-code for the ILU(l) factorization in Algorithm 4. Note that the triangular matrices $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ are directly stored in the matrix $\mathbf{A}$.

Algorithm 4 ILU(l) factorization algorithm.

Require: The coefficient matrix $\mathbf{A} = \left(a_{i,j}\right) \in \mathbb{R}^{n \times n}$, and the level of fill-in l.

$level_{i,j} \leftarrow 0$ for all $a_{i,j} \neq 0$, and $level_{i,j} \leftarrow +\infty$ for all $a_{i,j} = 0$
for i from 2 to n **do**
    for k from 1 to $i-1$ **do**
        if $a_{k,k} \neq 0$ and $level_{i,k} \leq l$ **then**
            $a_{i,k} \leftarrow a_{i,k} / a_{k,k}$
            for j from $k+1$ to n **do**
                $a_{i,j} \leftarrow a_{i,j} - a_{i,k} a_{k,j}$
                $level_{i,j} \leftarrow \min\left(level_{i,j},\, level_{i,k} + level_{k,j} + 1\right)$
            end for
        end if
    end for
    for k from 1 to n **do**
        if $level_{i,k} > l$ **then**
            $a_{i,k} \leftarrow 0$
        end if
    end for
end for

Threshold Parameter

The variant ILU(p,τ) controls the amount of nonzero entries with two dropping parameters p and τ. The criterion to decide whether a new entry is dropped in the LU factors is no longer based on the level of fill-in but on the magnitude of the computed entries [53]. Only the p largest entries satisfying the condition $|a_{i,j}| \leq \tau ||\mathbf{a}_i||_2$ are kept for the incomplete factorization [54]. The parameter p ensures that the factors are sufficiently sparse. On the other hand, τ is expected to be stringent enough to discard all the small entries introduced by the LU factorization that are unlikely to make a difference in the preconditioner application. As well as the level of fill-in parameter, there is no general method to find optimal values of p and τ, and their determination is mostly based on user experience.

Data Dependency in ILU

The main shortcoming of the ILU preconditioner is its strong sequential nature. For a compressed sparse row (CSR) matrix, the coefficient matrix is traversed in a row-wise fashion to compute the ILU entries and update the level of fill-in, based exclusively on

the calculations from the previous rows [49]. The Figure 2.1 illustrates the computational order of the entries in the ILU factors for a CSR matrix as well as the data dependency. It highlights the sequential nature of the ILU computation. Other variants, such as a computation by column more adapted for CSC matrices, can easily be derived.

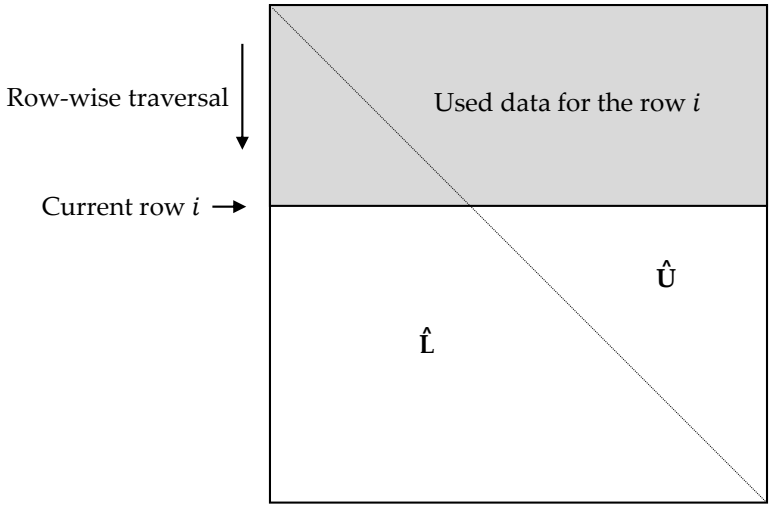


Figure 2.1: Data dependency and computational order of the entries in the ILU factors.

The basic ILU algorithm can be prone to roundoff errors or even fail in the case of a zero pivot in the computations. This problem can be avoided with the use of a pivoting strategy [55, 56]. In addition, the stability of the ILU preconditioner is only ensured for the class of M-matrices which is often not the case of matrices resulting from the finite volume discretization [43]. When the matrix $\mathbf{A}$ is far from being diagonal dominant, the ILU factorization can produce very poor preconditioners which lead to instabilities in the forward and backward substitutions within the linear solvers [57].

2.2.3 Domain Decomposition Preconditioners

Domain decomposition techniques consist in decomposing the solution of partial differential equations over overlapping or non-overlapping subdomains [58]. In accordance with the divide-and-conquer paradigm, the problems can be solved in parallel on each subdomain in order to share the computational cost across processors.

Additive Schwarz

Consider a decomposition of the continuous domain Ω, in which the partial differential equations are solved, into m overlapping subdomains Ω_k:

$$\Omega = \bigcup_{k=1}^{m} \Omega_k. \tag{2.23}$$

Following mesh generation, the discrete domain $\Omega^h = \{x_i \mid 1 \leq i \leq n\} \subset \Omega$ is obtained, with n the number of grid points. The discrete partitioning corresponding to (2.23) reads

$$\Omega^h = \bigcup_{k=1}^{m} \Omega_k^h, \tag{2.24}$$

with $\Omega_k^h = \Omega^h \cap \Omega_k$. We suppose that none of the discrete subdomains Ω_k^h are empty. A given function $u : \Omega \mapsto \mathbb{R}$ is represented by a vector $\mathbf{v} = (v_1, ..., v_n)^T$ in Ω^h, such that $v_i = u(x_i)$. In multiblock CFD solvers, the mesh Ω^h is first divided according to the layout of the non-overlapping subblocks spanning the entire domain. The subblocks are then extended by adding the neighboring grid points within a distance δ from the local subdomains. The parameter δ defines the number of overlapping layers of grid points between the different subdomains.

The decomposition (2.24) naturally defines the restriction operators $\mathbf{R}_k : \mathbb{R}^n \mapsto \mathbb{R}^{|\Omega_k^h|}$ from Ω^h to Ω_k^h, by retaining only the entries of a vector in Ω^h that correspond to grid points in Ω_k^h. Similarly, $\mathbf{R}_k^T : \mathbb{R}^{|\Omega_k^h|} \mapsto \mathbb{R}^n$ is called the prolongation operation. A local vector in Ω_k^h is extended to the global domain Ω^h by appending zero entries for grid points outside of Ω_k^h [59]. These operators are used to restrict the action of the global matrix $\mathbf{A}$ on the local subdomains:

$$\mathbf{A}_k = \mathbf{R}_k \mathbf{A} \mathbf{R}_k^T, \tag{2.25}$$

$\mathbf{A}_k$ being a $|\Omega_k^h| \times |\Omega_k^h|$ square matrix.

An explicit preconditioner based on the overlapping domain decomposition is given by the additive Schwarz method (ASM) [60] as

$$\mathbf{M_{AS}} = \sum_{k=1}^{m} \mathbf{R}_k^T (\mathbf{A}_k)^{-1} \mathbf{R}_k. \tag{2.26}$$

When $\mathbf{M_{AS}}$ is applied to a vector in Ω^h, each term k of the sum in (2.26) can be carried out independently with information from the local and neighboring subdomains. Let us examine in more detail how the additive Schwarz preconditioner is applied within the GMRES algorithm. In the preconditioned GMRES method, from Algorithm 3, the preconditioner is involved in a matrix-vector multiplication of the form:

$$\mathbf{v} = \mathbf{M_{AS}} \mathbf{z}, \text{ with } \mathbf{v} \text{ and } \mathbf{z} \text{ in } \Omega^h. \tag{2.27}$$

The development of the sum yields

$$\mathbf{v} = \sum_{k=1}^{m} \mathbf{R}_k^T (\mathbf{A}_k)^{-1} \mathbf{R}_k \mathbf{z}. \tag{2.28}$$

The vector $\mathbf{v}$ is calculated by applying the matrices to the vector $\mathbf{z}$ from right to left. The first matrix-vector product $\mathbf{R}_k \mathbf{z}$ is simply the restriction of the vector $\mathbf{z}$ into Ω_k^h, denoted $\mathbf{z}_k$, such that

$$\mathbf{v} = \sum_{k=1}^{m} \mathbf{R}_k^T (\mathbf{A}_k)^{-1} \mathbf{z}_k. \tag{2.29}$$

In Figure 2.2, we illustrate the ASM application on a very simple 1D grid Ω^h consisting of 6 grid points, and divided into two overlapping subdomains: Ω_1^h and Ω_2^h. The red entries in the restrictions $\mathbf{R}_i \mathbf{z}$ correspond to the entries communicated between the subdomains.

Thereafter, the matrix-vector products $(\mathbf{A}_k)^{-1} \mathbf{z}_k$ in equation (2.29) lead to several small linear systems of equations to be solved, namely

$$\mathbf{A}_k \mathbf{w}_k = \mathbf{z}_k, \text{ with } \mathbf{w}_k \text{ the unknown vector of size } |\Omega_k^h|. \tag{2.30}$$

When the sizes of the subdomains are sufficiently small, a direct method can be used to exactly solve the subproblems. Otherwise, a local preconditioner $\mathbf{P}_k \approx \mathbf{A}_k$ or an iterative

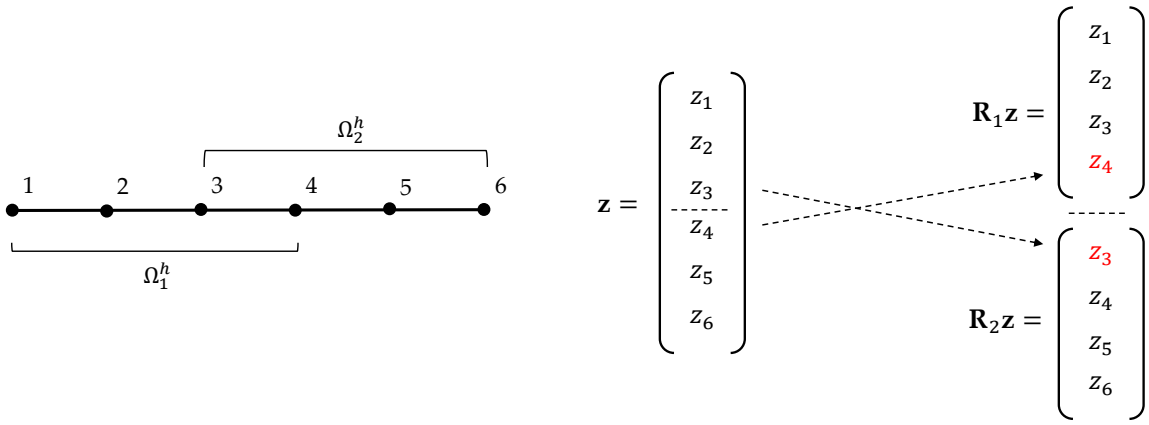


Figure 2.2: Domain Ω^h and subdomains with an overlap of width $\delta = 1$ (left) and application of the restriction operators $\mathbf{R}_i$ on a global vector $\mathbf{z}$ in Ω^h (right). The dashed arrows denote the required communications between the subdomains.

method, such as a preconditioned GMRES algorithm, can be employed to approximate the solution of the local linear systems (2.30). Therefore, the ASM preconditioner can be seen as a parallel generalization of preconditioning methods. For instance, local ILU preconditioners $\mathbf{P}_k$ combined with the ASM preconditioner yields an ILU preconditioning method compatible with coarse-grained parallelism [61]. In our numerical simulations, we employ an ASM-ILU preconditioner, in which the GMRES method, preconditioned with an ILU(0) factorization, approximately solves the local systems of linear equations.

Once the linear systems (2.30) are solved, the local solutions $\mathbf{w}_k$ are scattered across the neighboring overlapping subdomains with the prolongation operator $\mathbf{R}_k^T$:

$$\mathbf{v} = \sum_{k=1}^{m} \mathbf{R}_k^T \mathbf{w}_k. \tag{2.31}$$

In Figure 2.3, the construction of the vector $\mathbf{v}$ from the summations of the local contributions is depicted. The two processors send to each other the red entries, which correspond to the overlapping regions.

The domain decomposition preconditioner $\mathbf{M_{AS}}$ is called the additive Schwarz preconditioner, since all linear systems are solved simultaneously. When defined properly, domain decomposition methods can provide suitable parallel preconditioning methods. As presented, the ASM preconditioner requires two communication steps between the

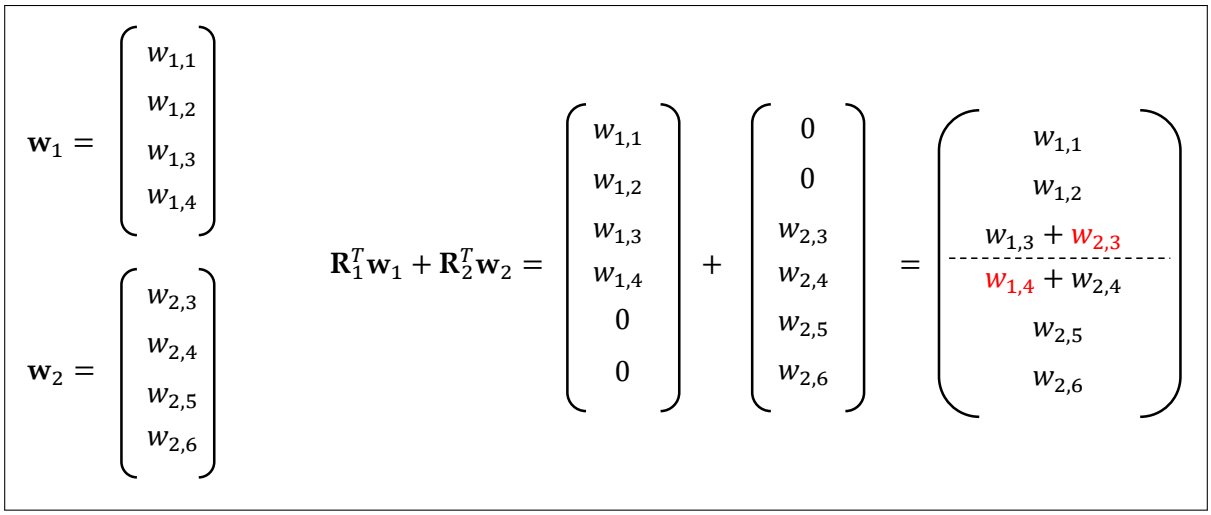


Figure 2.3: Prolongation of the local vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ and summation in Ω^h.

different processors through the application of the restriction and the prolongation operators $\mathbf{R}_k$ and $\mathbf{R}_k^T$ [62]. The entries of the vector $\mathbf{z}$ in Ω^h are originally scattered across the processors, according to the non-overlapping block layout of the discrete domain. Thus, the operation $\mathbf{R}_k\mathbf{z}$ involves the communication of the entries corresponding to the overlapping regions between the processors. In the same vein, the prolongation application $\mathbf{R}_k^T\mathbf{w}_k$ in equation (2.31) requires the communication of entries of $\mathbf{w}_k$ in overlapping layers.

Surprisingly, the additive Schwarz preconditioner in (2.26) often turns out to be a poor preconditioner and its variant called the restricted additive Schwarz method is usually preferred. This method uses a non-overlapping prolongation operator $\tilde{\mathbf{R}}_k^T$. Therefore, restricted additive Schwarz method is more suitable to parallel machines with the removal of one of the communication steps, while counterintuitively giving faster convergence [63, 64]. The preconditioner is given as

$$\mathbf{M}_{\mathbf{RAS}} = \sum_{k=1}^{m} \tilde{\mathbf{R}}_k^T (\mathbf{A}_k)^{-1} \mathbf{R}_k. \tag{2.32}$$

The application of $\tilde{\mathbf{R}}_k^T$ is illustrated for the 1D basic problem in Figure 2.4, in which no communication is performed.

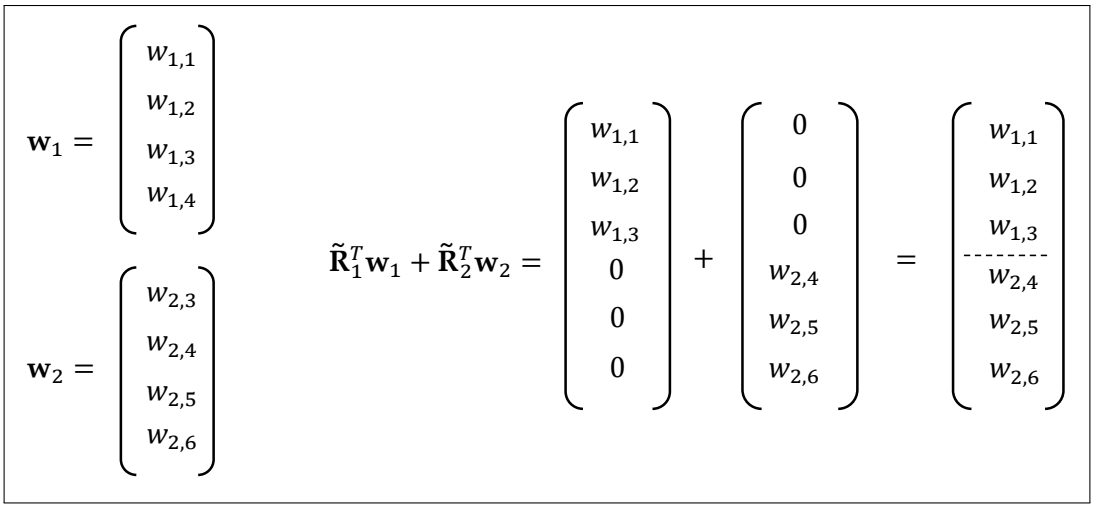


Figure 2.4: Restricted prolongation of the local vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ and summation in Ω^h.

The usage of the ASM preconditioner for the full potential equation and the Euler equations in conjunction with a quasi-Newton method has been the topic of several studies [16, 65–67]. Its suitable parallel feature makes it a privileged choice for parallel solvers.

Block Jacobi

The block Jacobi preconditioner may be seen as a special case of the additive Schwarz preconditioner with no overlap [68]. Consider this time a partitioning of the domain Ω into m subdomains, and the corresponding restrictions $\mathbf{R}_k$:

$$\Omega = \bigcup_{k=1}^{m} \Omega_k, \quad \text{and} \quad \forall i, j \in \{1, ..., m\}, \;\; i \neq j, \;\; \Omega_i \cap \Omega_j = \emptyset. \tag{2.33}$$

The block Jacobi preconditioner reads

$$\mathbf{M_{BJ}} = \sum_{k=1}^{m} \mathbf{R}_k^T (\mathbf{B}_k)^{-1} \mathbf{R}_k, \tag{2.34}$$

with $\mathbf{B}_k$ being square matrices of size $|\Omega_k^h| \times |\Omega_k^h|$. The partitioning of the matrix $\mathbf{A}$ can be written in a block form as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \dots & \mathbf{A}_{1,m} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{m,1} & \dots & \mathbf{A}_{m,m} \end{bmatrix}. \tag{2.35}$$

The block Jacobi preconditioning matrix corresponding to the associated partition is a block diagonal matrix:

$$\mathbf{M_{BJ}} = \begin{bmatrix} \mathbf{B}_1^{-1} & & 0 \\ & \ddots & \\ 0 & & \mathbf{B}_m^{-1} \end{bmatrix}. \tag{2.36}$$

The application of the block Jacobi preconditioner is highly parallel, since there is no longer overlapping subdomains. However, the block Jacobi method tends to yield less effective preconditioners than the ASM method with an overlap. Indeed, the extra communications usually increase the stability and robustness of the preconditioning.

In Krylov subspace methods, the block Jacobi application is a matrix-vector multiplication:

$$\mathbf{v} = \mathbf{M_{BJ}}\,\mathbf{z}, \text{ with } \mathbf{v} \text{ and } \mathbf{z} \text{ in } \Omega^h. \tag{2.37}$$

The computation of $\mathbf{v}$ is carried out with the independent solution of m small linear systems of equations, whose dimensions are defined by the sizes of the subdomains.

$$\mathbf{v} = \begin{bmatrix} \mathbf{B}_1^{-1}\left(\mathbf{R}_1\mathbf{z}\right) \\ \mathbf{B}_2^{-1}\left(\mathbf{R}_2\mathbf{z}\right) \\ \vdots \\ \mathbf{B}_m^{-1}\left(\mathbf{R}_m\mathbf{z}\right) \end{bmatrix}. \tag{2.38}$$

The main pitfall of the domain decomposition preconditioners with parallel solvers is their sensitivity to the number of blocks and so the number of cores. The preconditioning of the matrix $\mathbf{A}$ becomes less efficient with domain decomposition preconditioners for an increasing number of blocks in the partition [69].

2.2.4 Sparse Approximate Inverse Preconditioners

The class of sparse approximate inverse preconditioners aims to approximate the action of $\mathbf{A}^{-1}$ itself. As a consequence, their application within Krylov subspace methods consists of a sparse matrix-vector multiplication $\mathbf{y} = \mathbf{M}\mathbf{v}$.

The wide range of polynomial preconditioners belong to this class. For example, a polynomial preconditioner is derived from the truncation of the Neumann series of $\mathbf{A}$ in [70], written in a splitting form $\mathbf{A} = \mathbf{N}_1 - \mathbf{N}_2$ as

$$\mathbf{M} = \left(\sum_{i=0}^{p-1} \left(\mathbf{I} - \mathbf{N}_1^{-1}\mathbf{A} \right)^i \right) \mathbf{N}_1^{-1}, \tag{2.39}$$

where $p > 0$ and the inverse of $\mathbf{N}_1$ is only approximated. The preconditioner is truncated to obtain a low-degree polynomial of the matrix $\mathbf{A}$. Besides, the preconditioner $\mathbf{M}$ is never assembled explicitly but its application on a vector results in the product with each term in the series (2.39). Nevertheless, polynomial preconditioning methods usually require reliable bound estimates of the eigenvalues of $\mathbf{A}$ to yield sufficiently sound preconditioners [32]. Now rarely used, polynomial preconditioners were mainly investigated due to their suitability for vector processors.

In the next chapter, we present the Sparse Approximate Inverse (SPAI) preconditioner, a highly parallel preconditioner.

2.3 Solving Nonlinear Equations

Many of the early CFD solvers used pressure-based methods, solving sequentially the momentum equations for the velocity and a Poisson pressure equation based on the continuity equation [71]. These methods, albeit not being too demanding in terms of memory storage and computational complexity, are very dependent on the pressure-velocity coupling due to the segregated solution procedure. They could fail on many occasions due to instabilities and a weak coupling between the pressure and velocity.

The rapid development of computers over the last decades has been accompanied by a constant increase in the memory capacity as well as in the computational power. These new computational resources, in conjunction with the development of new linear solver methods for general non-symmetric positive definite matrices, such as GMRES, made it possible to solve the coupled equations [72]. The use of an implicit time-marching scheme for the coupled velocity and pressure equations raised the solver robustness, but also required innovative methods to solve the larger systems of nonlinear equations. In the

same period, the Newton-Krylov method became an algorithm of choice for solving the nonlinear equations from the coupled Navier-Stokes equations. Pueyo gives an extensive description of the development of Newton's method within CFD solvers in [12]. In this respect, a nonlinear GMRES solver was first employed by Wigton et al. in 1985 [73], which is equivalent to solving Newton's linearized systems with a linear GMRES solver.

2.3.1 Newton's Method

Newton's method is an iterative algorithm for solving nonlinear equations. Starting from an initial guess $\mathbf{x}_0$, the method builds a sequence of iterates. The solution of a linear system of equations is necessary at each step to advance the iterate towards the solution.

First, let us consider the general case of a multivariable function $\mathbf{F} : \mathbb{R}^n \longmapsto \mathbb{R}^n$ defining n nonlinear equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$. The Taylor series of $\mathbf{F}$ at a given point $\mathbf{x}$ states that

$$\mathbf{F}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{F}(\mathbf{x}) + \mathbf{J}_\mathbf{F}(\mathbf{x})\delta\mathbf{x} + O(||\delta\mathbf{x}||^2), \tag{2.40}$$

where $\mathbf{J}_\mathbf{F}$ denotes the Jacobian matrix of $\mathbf{F}$ defined by $\mathbf{J}_\mathbf{F}(\mathbf{x})_{i,j} = \dfrac{\partial F_i}{\partial x_j}(\mathbf{x})$, for $i, j \in \{1, ..., n\}$. The idea of Newton's method is to find an appropriate step $\delta\mathbf{x}$ to march in the direction of the root from $\mathbf{x}$. Neglecting the high-order terms in the Taylor series and considering that $\mathbf{x} + \delta\mathbf{x}$ is the sought-after solution, the iterate is updated as

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{J}_\mathbf{F}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x}). \tag{2.41}$$

The main difficulty of Newton's method lies in the fact that the iterative process is only locally convergent, when the starting vector $\mathbf{x}_0$ is sufficiently close to the solution $\mathbf{x}_*$ [74]. Even worse, there is no means to know beforehand if the method will converge from a given starting iterate. An attractive property of the method is its quadratic convergence towards the solution. It means that there exists a positive constant c such that

$$||\mathbf{x}^{(p+1)} - \mathbf{x}^*|| \leq c||\mathbf{x}^{(p)} - \mathbf{x}^*||^2. \tag{2.42}$$

In practice, the inverse of the Jacobian $\mathbf{J_F}$ is never explicitly computed. Instead, the iteration process is carried out with the solutions of successive systems of linear equations:

$$\begin{cases} \mathbf{J_F}(\mathbf{x}^{(p)})\delta\mathbf{x} = -\mathbf{F}(\mathbf{x}^{(p)}), \\ \mathbf{x}^{(p+1)} = \mathbf{x}^{(p)} + \delta\mathbf{x}. \end{cases} \tag{2.43}$$

Hence, Newton's method requires the solution of a linear system at every step. As we already saw in the previous section, the choice of an optimal linear solver is mainly problem dependent. For linear systems for which a direct solver is unsuitable, the choice of an iterative algorithm is preferred. In the case of iterative solvers, the exact solution of the linear system in (2.43) is clearly inaccessible. This could seem to be a major weakness of iterative methods. Nonetheless, the linear systems being merely intermediate steps in Newton's method, their exact solution is unnecessary. In reality, the in-depth convergence of the linear systems is a problem known as oversolving, which must be avoided [75]. Therefore, the intermediate linear systems are approximately solved until the following condition is satisfied:

$$||\mathbf{F}(\mathbf{x}^{(p)}) + \mathbf{J_F}(\mathbf{x}^{(p)})\delta\mathbf{x}|| \leq \tau_{\text{lin}}||\mathbf{F}(\mathbf{x}^{(p)})||, \tag{2.44}$$

where $\tau_{\text{lin}} \in [0, 1[$ is called the forcing term. This parameter prevents the risk of oversolving that can occur during the first iterations of Newton's method [76] and tightens the tolerance as the iterate approaches the root of $\mathbf{F}$.

Hence, two levels of iterations interplay to converge towards the solution of the nonlinear equations. For the sake of clarity, a step p in (2.43) is referred to as a Newton step or a nonlinear iteration in the rest of this thesis. On the other hand, the iterations undertaken within the linear solver are called inner or linear iterations [74]. This Newton-like method is called an inexact or a quasi-Newton method.

2.3.2 Newton-Krylov Methods

A wide variety of Krylov subspace solvers has been devised, depending on the properties of the coefficient matrix $\mathbf{A}$. Here, we only consider Krylov subspace methods for nonsymmetric matrices, such as GMRES and Bi-CGSTAB. The efficiency of Krylov methods lies

in the fact that no other information than the product of the Jacobian matrix $\mathbf{J_F}(\mathbf{x}^{(p)})$ with a given vector is required. This property allows the use of matrix-free Krylov methods for which the action of the Jacobian is only approximated from evaluations of the function $\mathbf{F}$ [77].

2.3.3 Globalization of Newton's Method

One gap that remains to be filled is the lack of robustness of the convergence in Newton's method. The convergence is only ensured locally, with a starting guess in the close neighborhood of the sought-after solution. In order to alleviate this problem, Newton's method can be enhanced with globalization techniques to achieve convergence from virtually any starting point [75].

Line Search

In Newton's method, it may happen that the search direction $\frac{\delta \mathbf{x}}{||\delta \mathbf{x}||}$ is well estimated by the solution of the linear system but the magnitude of the step $||\delta \mathbf{x}||$ overshoots the root location. Line search methods, such as the Armijo rule [78], aim to minimize $||\mathbf{F}||$ along the search direction with successive adjustments of the step length $||\delta \mathbf{x}||$. The determination of an efficient stepsize has been the topic of many papers [79, 80]. The line search methods are rather easy to implement contrary to other globalization techniques but may fail to converge when applied to very stiff cases.

Trust Regions

Other general globalization techniques are the trust-region methods. Instead of searching for a minimizer along a direction, trust-region methods are based on the minimization of a simple modeling of the function $\mathbf{F}$ in a restricted region, generally a ball [81].

Pseudo-Transient

In solving physical differential equations, the previous globalization methods may fail to converge, converge to an unphysical solution or even converge to a point where the Jacobian is nonsingular. In this respect, the pseudo-transient continuation method transforms

the initial nonlinear equations $\mathbf{F}(\mathbf{x}) = 0$ into a transient problem $\frac{d\mathbf{x}}{d\tau} + \mathbf{F}(\mathbf{x}) = 0$, converging to the root of $\mathbf{F}$ [82]. The initial guess of Newton's method becomes equivalent to the initial state of a transient problem, and the path of convergence reproduces the temporal evolution of the solution $\mathbf{x}$ to the steady state.

Unlike line search and trust-region methods, the pseudo-transient continuation strategy is not to guarantee a reduction in the norm $||\mathbf{F}(\mathbf{x})||$ at each step, but rather to ensure the global convergence by following a quasi-physical path. In this endeavor, the pseudo-transient continuation method is able to "climb hills", taking paths along which $\mathbf{F}$ temporarily increases in norm to shorten the overall path length to the solution [83].

Chapter 3

Governing Equations and Flow Solver

The general governing equations used to model the flow are reviewed in Section 3.1. Afterwards, the corresponding spatial discretization and time integration employed with the finite volume method are detailed in Sections 3.3 and 3.4. Finally, we briefly review the flow solver FANSC used for the numerical simulations in Sections 3.6 and 3.7.

3.1 Governing Equations

The compressible Navier-Stokes equations state the governing equations for a viscous compressible flow from the conservation laws of mass, momentum and energy. The integral form of the Navier-Stokes equations on a domain Ω, enclosed by the boundary $\partial\Omega$, reads

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{u}\,\mathrm{d}\Omega+\oint_{\partial\Omega}\left(\mathbf{F_c}\left(\mathbf{u}\right)-\mathbf{F_v}\left(\mathbf{u},\nabla\mathbf{u}\right)\right)\mathrm{d}S=\mathbf{0}, \tag{3.1}$$

where $\mathbf{u}$ is the vector of conservative variables, and $\mathbf{F_c}$ and $\mathbf{F_v}$ are the convective and viscous fluxes respectively. Using the Einstein summation convention, they are defined in three dimensions as

$$\mathbf{u}=\begin{bmatrix}\rho\\ \rho v_x\\ \rho v_y\\ \rho v_z\\ \rho E\end{bmatrix},\quad \mathbf{F_c}(\mathbf{u})=\begin{bmatrix}\rho V\\ \rho v_x V+n_x p\\ \rho v_y V+n_y p\\ \rho v_z V+n_z p\\ \rho H V\end{bmatrix}\quad\text{and}\quad \mathbf{F_v}(\mathbf{u},\nabla\mathbf{u})=\begin{bmatrix}0\\ n_j\tau_{x,j}\\ n_j\tau_{y,j}\\ n_j\tau_{z,j}\\ n_i\left(v_j\tau_{i,j}-q_i\right)\end{bmatrix}, \tag{3.2}$$

with v_x, v_y and v_z being the three Cartesian components of the velocity vector $\mathbf{v}$. Besides, n_x, n_y and n_z denote the Cartesian components of the unit vector $\mathbf{n}$ normal to the surface element $\mathrm{d}S$. As for the remaining variables, ρ, E, H, p, $\tau_{i,j}$ and q_i are respectively the flow density, the total energy per unit mass, the total enthalpy per unit mass, the pressure, the components of the viscous stress tensor $\boldsymbol{\tau}$ and the components of the heat flux vector $\mathbf{q}$. Finally, the contravariant velocity V, given by $V = \mathbf{n}^T\mathbf{v} = n_x v_x + n_y v_y + n_z v_z$, is the velocity component normal to the surface element $\mathrm{d}S$.

The solution of equation (3.1) for a turbulent flow entails solving the Navier-Stokes equations down to the smallest scale involved in the energy cascade. This requires a very low spatial resolution of the mesh with an unreasonable number of grid points [11]. A common approach to overcome this issue is to solve the Navier-Stokes equations for the mean variables via the so-called Reynolds averaging [84], giving the Reynolds-Averaged Navier-Stokes (RANS) equations. These equations are identical to the Navier-Stokes equations (3.1) for the averaged variables aside from the introduction of an additional fictitious viscosity term μ_t in the constitutive equations.

The Navier-Stokes equations in (3.1) contain more unknowns than available equations. Correspondingly, additional equations related to the fluid in question need to be provided to close the equations. For media such as air or water, the general assumption of a Newtonian fluid is employed to relate the viscous stress tensor $\boldsymbol{\tau}$ and the strain rate in the flow:

$$\boldsymbol{\tau} = -\frac{2}{3}(\mu + \mu_t)\nabla \cdot \mathbf{v}\mathbf{I} + (\mu + \mu_t)(\nabla\mathbf{v} + \nabla\mathbf{v}^T), \tag{3.3}$$

where μ denotes the laminar viscosity which is a characteristic feature of the fluid. On the other hand, the turbulent viscosity μ_t has no physical meaning, but is used to incorporate the effects of turbulence in the equations. Its computation requires a turbulence model. Many of them have been designed with different computational complexities and benefits [85]. The one-equation Spalart-Allmaras model is introduced in Section 3.2.

Under the assumption of an ideal gas, the pressure p is related to the density ρ and the total energy per unit mass E according to

$$p = (\gamma - 1)\,\rho\left[E - \frac{||\mathbf{v}||_2^2}{2}\right], \tag{3.4}$$

where γ denotes the heat capacity ratio. In addition, the Sutherland's law is used to calculate the laminar viscosity μ from the flow temperature T, in SI units for air, with the formula:

$$\mu = \frac{1.45\, T^{3/2}}{T + 110} \times 10^{-6}. \tag{3.5}$$

In closing, the Fourier's law for the heat conduction $\mathbf{q}$ is applied as

$$\mathbf{q} = -k\nabla T, \tag{3.6}$$

with k being the thermal conductivity coefficient.

Depending on the phenomena involved and the desired accuracy of the final solution, the governing equations can be simplified to neglect to some extent the complexities of the flow. In some cases such as high Reynolds number flows, the convective component of the flow is typically predominant compared to the viscous term. A common approximation is to consider a purely convective flow and ignore the viscous flux $\mathbf{F_v}$ in the governing equations (3.1) [11]. This simplification gives us the Euler equations:

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{u}\, \mathrm{d}\Omega + \oint_{\partial\Omega} \mathbf{F_c}\left(\mathbf{u}\right)\, \mathrm{d}S = \mathbf{0}. \tag{3.7}$$

The Euler equations can accurately describe the presence of shocks and expansion waves in the flow.

3.2 Turbulence Model

In FANSC, the effects of turbulence in the flow are reflected with the one-equation Spalart-Allmaras (SA) turbulence model [86]. It requires the solution of a transport equation for an eddy viscosity variable $\tilde{\nu}$ from which is calculated the turbulent viscosity μ_t. The transport equation reads

$$\begin{aligned}\frac{\partial \tilde{\nu}}{\partial t} + \mathbf{v}^T \nabla(\tilde{\nu}) = c_{b1}\,(1 - f_{t2})\,\tilde{S}\tilde{\nu} \\ + \frac{1}{\sigma}\left[\nabla \cdot \left[(\nu + \tilde{\nu})\,\nabla\tilde{\nu}\right] + c_{b2}||\nabla\tilde{\nu}||_2^2\right] \\ - \left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\tilde{\nu}}{d_w}\right)^2 + f_{t1}||\Delta\mathbf{v}||_2^2.\end{aligned} \tag{3.8}$$

In equation (3.8), d_w denotes the distance to the closest wall and $\nu = \mu/\rho$ refers to the laminar kinematic viscosity. The distance to the wall must be accurately evaluated to ensure an appropriate prediction of the flow features close to the walls [87]. Once the equation has been solved for $\tilde{\nu}$, the turbulent viscosity μ_t is given by

$$\mu_t = f_{v1}\rho\tilde{\nu}, \tag{3.9}$$

with

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad \text{and} \quad \chi = \frac{\tilde{\nu}}{\nu}. \tag{3.10}$$

The functions f, $\tilde{S}$ and the constant parameters c, σ and κ can be obtained from [11]. This model is often used within CFD software where it yields robust estimations of the turbulent effects for a broad range of flows, without adding a substantial overhead. Other popular turbulent models can sometimes provide more accurate predictions. For instance, the two equation model $k - \epsilon$ is often used, but is more difficult to process and requires finer meshes near the walls [11]. In FANSC, the SA model is only solved on the finest grid, once before every Newton-Krylov step, in a segregated manner.

3.3 Spatial Discretization

The most widespread discretization method for CFD applications is the finite volume method, which directly discretizes the integral form of the conservative equations over the physical domain. This integral formulation naturally ensures the conservation of mass, momentum and energy across the domain throughout the simulation, making it an attractive tool for solving conservative partial differential equations. The accuracy of

the solution depends on the order of accuracy of the spatial discretization scheme. The discretization process yields a set of 5 equations in 3D, for the discretized conservative variables $\mathbf{w}_i^T = \left[\rho_i, (\rho v_x)_i, (\rho v_y)_i, (\rho v_z)_i, (\rho E)_i\right]$ for each cell i:

$$\mathbf{M}_i \frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} + \mathbf{r}(\mathbf{w}_i) = \mathbf{0}. \tag{3.11}$$

In these equations, $\mathbf{M}_i$ is the mass matrix and is determined by the position of the unknowns. In the case of a cell-centered scheme, $\mathbf{M}_i$ becomes a diagonal matrix with the volume of the cell i as the entries. For its part, $\mathbf{r}(\mathbf{w}_i)$ is the numerical flux of the cell i given by the contributions of the convective, dissipation and viscous fluxes. It is a nonlinear function of the state $\mathbf{w}_i$.

The cell discretized equations in (3.11) are gathered into a global nonlinear system of equations, for all the conservative variables, which is written as

$$\mathbf{M} \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} + \mathbf{R}(\mathbf{w}) = \mathbf{0}, \tag{3.12}$$

with $\mathbf{w}, \mathbf{R}(\mathbf{w}) \in \mathbb{R}^N$, where N is the number of cells n times the number of conservative variables. The residual or discretized flux $\mathbf{R}(\mathbf{w})$ contains the successive discretized fluxes of the cells $\mathbf{R}(\mathbf{w})^T = \left[\mathbf{r}(\mathbf{w}_1)^T\ \mathbf{r}(\mathbf{w}_2)^T\ ...\ \mathbf{r}(\mathbf{w}_n)^T\right]$ and $\mathbf{w}^T = \left[\mathbf{w}_1^T\ \mathbf{w}_2^T\ ...\ \mathbf{w}_n^T\right]$.

FANSC uses a central scheme with an artificial dissipation for the discretization of the convective flux. Compared to other more sophisticated spatial discretization schemes, the central scheme is relatively easy to implement in cell-centered schemes. It uses the average of the conservative variables in the adjacent cells to compute the convective flux at the interface. The resulting numerical flux usually yields a very unstable convergence, as it does not take into account the direction of propagation of the waves [88]. In addition, only the two neighboring cells are involved in the flux computation, which may unfortunately lead to some odd-even oscillations and ultimately increase instabilities. Finally, the discretized convective flux favors the appearance of oscillations in the region of high pressure gradient. As a consequence, an artificial dissipation flux is required for stability.

JST Scheme

The different central schemes differ in the formulation of the artificial dissipation scheme. The first implementation was introduced by Jameson, Schmidt and Turkel for the Euler equations with a scalar artificial flux [89]. The scheme is referred to as JST after the authors' name. At the interface $\left(i+\frac{1}{2},j\right)$ between two cells, the JST flux reads

$$\mathbf{F}_{c,i+\frac{1}{2},j} = \mathbf{F}_c\left(\frac{\mathbf{w}_{i,j}+\mathbf{w}_{i+1,j}}{2}\right) - \mathbf{D}_{i+\frac{1}{2},j}, \tag{3.13}$$

where $\mathbf{D}_{i+\frac{1}{2},j}$ is the artificial dissipation flux. It is given as

$$\mathbf{D}_{i+\frac{1}{2},j} = \mathbf{D}^{(2)}_{i+\frac{1}{2},j} - \mathbf{D}^{(4)}_{i+\frac{1}{2},j}, \tag{3.14}$$

$$\mathbf{D}_{i+\frac{1}{2},j} = \Lambda_{i+\frac{1}{2},j}\left[\nu^{(2)}\left(\mathbf{w}_{i+1,j}-\mathbf{w}_{i,j}\right) - \nu^{(4)}\left(\mathbf{w}_{i+2,j}-3\mathbf{w}_{i+1,j}+3\mathbf{w}_{i,j}-\mathbf{w}_{i-1,j}\right)\right]. \tag{3.15}$$

The parameters $\nu^{(2)}$ and $\nu^{(4)}$ are pressure-based sensors to turn on and off the first-order and third-order dissipative fluxes respectively. The third-order component of the dissipation flux tends to introduce deleterious spurious oscillations of the solution around the high gradients and the shocks. As a consequence, the sensor $\nu^{(4)}$ is dominant in smooth regions of the flow but decreases substantially around shocks and discontinuities to favor the more stable first-order flux $\mathbf{D}^{(2)}_{i+\frac{1}{2},j}$. On the contrary, the first-order flux is switched off in smooth regions of the flow to reduce the artificial dissipation as far as possible. The dissipation flux in (3.15) is scaled with the scalar factor $\Lambda_{i+\frac{1}{2},j}$ given by

$$\Lambda_{i+\frac{1}{2},j} = \frac{1}{2}\left(\hat{\lambda}^i_{i,j} + \hat{\lambda}^i_{i+\frac{1}{2},j}\right), \tag{3.16}$$

where $\hat{\lambda}^i_{i,j}$ denotes the scaled spectral radius of the flux Jacobian matrices in the i-direction. The $\nu^{(2)}$ and $\nu^{(4)}$ sensors are defined as

$$\nu^{(2)}_{i,j} = \epsilon^{(2)}\max(\sigma_{i,j},\sigma_{i+1,j}) \quad \text{with} \quad \sigma_{i,j} = \frac{|p_{i+1,j}-2p_{i,j}+p_{i-1,j}|}{p_{i+1,j}+2p_{i,j}+p_{i-1,j}}, \tag{3.17}$$

and

$$\nu_{i,j}^{(4)} = \max\left(0, (\epsilon^{(4)} - \nu^{(2)})\right). \tag{3.18}$$

Finally, $\epsilon^{(2)}$ and $\epsilon^{(4)}$ are both constants generally around the unity and $1/32$ respectively.

Matrix Dissipation Scheme

The accuracy of the JST scheme can be improved with the use of a matrix to scale the dissipation flux. This discretization is called the matrix dissipation (MATD) scheme. In MATD, each equation is scaled with the corresponding eigenvalue of the flux Jacobian instead of the spectral radius [90]. The artificial dissipation flux $\mathbf{D}_{i+\frac{1}{2},j}$ becomes

$$\mathbf{D}_{i+\frac{1}{2},j} = \left|\mathbf{J}_{c,i+\frac{1}{2},j}\right| \left[\nu^{(2)}\left(\mathbf{w}_{i+1,j} - \mathbf{w}_{i,j}\right) - \nu^{(4)}\left(\mathbf{w}_{i+2,j} - 3\mathbf{w}_{i+1,j} + 3\mathbf{w}_{i,j} - \mathbf{w}_{i-1,j}\right)\right], \tag{3.19}$$

with $\mathbf{J}_{c,i+\frac{1}{2},j} = \dfrac{\partial \mathbf{F}_{c,i+\frac{1}{2},j}}{\partial \mathbf{w}}$, the convective flux Jacobian [11]. Nevertheless, the substitution of the spectral radii by the flux Jacobian results in a more expensive central difference scheme.

Viscous Flux Discretization

In FANSC, the gradients in the viscous flux are calculated from the interpolated variables at the vertices of the mesh thanks to the Gauss-Green formula [88].

3.4 Time Integration

Acceleration techniques that allow rapid convergence to the steady-state solution without accurately solving the transient solutions have been a focus in a considerable number of research projects [11]. In these strategies, only the final solution is meaningful and these simplifications result in substantial computational savings. As a result, many CFD simulations concentrate only on solving the stationary nonlinear equations

$$\mathbf{R}(\mathbf{w}) = \mathbf{0}, \tag{3.20}$$

where the residual $\mathbf{R}$ is a vector of size N, with N being the overall number of discretized variables in the mesh.

3.4.1 Pseudo-Transient Continuation

Several different globalization methods to ensure the convergence of Newton's method for nonlinear equations were briefly presented in Subsection 2.3.3. It has been noted that line search and trust-region methods may face numerous issues for the convergence towards the physical solution of nonlinear equations.

A common globalization method to overcome the convergence issue is to employ a pseudo-transient formulation of the problem [82, 91, 92]:

$$\mathbf{M}\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau} + \mathbf{R}(\mathbf{w}) = \mathbf{0}. \tag{3.21}$$

The pseudo-time formulation (3.21) is fundamentally different from the physical time-dependent partial differential equations (3.12). The transformation must be understood merely as an acceleration technique towards the steady-state solution, rather than an accurate time-marching method. Its main idea is to first mimic a transient path before approaching Newton's method as the pseudo-time τ tends towards infinity [93]. The pseudo-time integration is performed using a first-order forward Euler method:

$$\left(\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau}\right)^n = \frac{\mathbf{w}^{n+1} - \mathbf{w}^n}{\Delta\tau} + \mathcal{O}(\Delta\tau). \tag{3.22}$$

In equation (3.22), the superscript n stands for the time level. Thus, $\mathbf{w}^n$ denotes the state vector at the current time τ and $\mathbf{w}^{n+1}$ its value at the time $\tau + \Delta\tau$, with $\Delta\tau$ being the time step.

The distinct spatial and temporal discretizations of the governing equations with the method of lines allow explicit or implicit integration of the discretized equations (3.21) [11]. In explicit time-stepping schemes, the residual $\mathbf{R}$ is evaluated at the current state $\mathbf{w}^n$:

$$\frac{\mathbf{M}\left(\mathbf{w}^{n+1} - \mathbf{w}^n\right)}{\Delta\tau} + \mathbf{R}(\mathbf{w}^n) = \mathbf{0}. \tag{3.23}$$

Hence, as its name implies, the residual can explicitly be calculated. Therefore, the update state $\mathbf{w}^{n+1}$ is easily derived with a mere algebraic calculation. From a practical standpoint, more sophisticated but still straightforward methods, such as the so-called Runge-Kutta multistage schemes, are used to favor stability [94].

On the other hand, in implicit integration schemes, the residual is evaluated at the next time step $\mathbf{R}(\mathbf{w}^{n+1})$. A direct consequence is that the residual $\mathbf{R}$ is now dependent on the unknown state $\mathbf{w}^{n+1}$:

$$\frac{\mathbf{M}\left(\mathbf{w}^{n+1}-\mathbf{w}^{n}\right)}{\Delta\tau}+\ \mathbf{R}(\mathbf{w}^{n+1})=\mathbf{0}. \tag{3.24}$$

In a fluid, the information is propagated through the medium at the wave speeds corresponding to the flow speed and the speed of sound relative to the flow. The so-called CFL condition for stability ensures that no cell misses the information by restricting the maximum allowed time step [11]. The Courant number σ is expressed as the ratio between the computational time step $\Delta\tau$ and the minimum time taken by the waves to travel in a cell. In 1D, the Courant number reads

$$\sigma=\max_{i}\frac{(|\mathbf{v}_i|+c)\Delta\tau}{\Delta x_i}. \tag{3.25}$$

For time-explicit methods, the CFL condition assesses that σ should be bounded, ensuring that the computational domain of dependence embeds the physical domain of dependence [95]. The computational time step $\Delta\tau$ is guaranteed to be smaller than the relevant time scales. However, the CFL condition puts a severe limitation on the use of explicit methods, requiring too short a time step for typical engineering problems. On the other hand, implicit methods avoid the CFL restrictions using the entire computational domain, which allows the use of larger time steps [96]. The stringent upper bound on the time step with the explicit time-marching method gives an advantage in the choice of implicit time integration.

In equation (3.24), the residual $\mathbf{R}(\mathbf{w}^{n+1})$ is linearized with a first-order truncation:

$$\mathbf{R}(\mathbf{w}^{n+1})\approx\mathbf{R}(\mathbf{w}^{n})+\left(\frac{\partial\mathbf{R}}{\partial\mathbf{w}}\right)^{n}\Delta\mathbf{w}^{n}, \tag{3.26}$$

where $\left(\frac{\partial \mathbf{R}}{\partial \mathbf{w}}\right)^n = \frac{\partial \mathbf{R}}{\partial \mathbf{w}}(\mathbf{w}^n)$ is the flux Jacobian evaluated at the current time step. Consequently, the original nonlinear problem (3.20) is substituted for a series of systems of linear equations of the form:

$$\left[\frac{\mathbf{M}}{\Delta\tau} + \left(\frac{\partial \tilde{\mathbf{R}}}{\partial \mathbf{w}}\right)^n\right] \Delta\mathbf{w}^n = -\mathbf{R}\left(\mathbf{w}^n\right), \tag{3.27}$$

$$\mathbf{w}^{n+1} = \mathbf{w}^n + \alpha\Delta\mathbf{w}^n. \tag{3.28}$$

The update of the solution is weighted with an under-relaxation parameter α, between 0 and 1, to increase stability at the expense of a slower rate of convergence. In addition, $\tilde{\mathbf{R}}$ denotes an approximation of the high-order numerical flux $\mathbf{R}$, used to compute the Jacobian matrix $\mathbf{J} = \frac{\partial \tilde{\mathbf{R}}}{\partial \mathbf{w}}$. At each step, $\mathbf{J}$ is evaluated at the latest approximation $\mathbf{w}^n$.

At the outset, the equations are accurately integrated in time with modest time steps to follow the transient physical path. As the approximate solution becomes closer to the steady-state solution, the time step is increased to accelerate convergence at the detriment of temporal accuracy. We note that if the linearization of $\mathbf{R}$ is exact, the implicit time-stepping scheme (3.27) turns into a Newton step when the time step $\Delta\tau$ tends to $+\infty$:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{w}}\left(\mathbf{w}^n\right)\Delta\mathbf{w}^n = -\mathbf{R}\left(\mathbf{w}^n\right). \tag{3.29}$$

In the case of an inexact Jacobian matrix $\frac{\partial \tilde{\mathbf{R}}}{\partial \mathbf{w}}$ or approximately solved linear systems, a linear convergence is generally attained.

3.4.2 Left-Hand Side Operator

It must be noted that only the spatial discretization of the right-hand side residual $\mathbf{R}\left(\mathbf{w}^n\right)$ determines the accuracy of the final solution, since it must ultimately satisfy the nonlinear equations (3.20). On the contrary, the left-hand side operator in equation (3.27) controls the convergence towards this solution. Its purpose is to genuinely drive the approximate solution to the root of the residual. A consistent flux Jacobian $\mathbf{J}$ with the numerical flux $\mathbf{R}$ approaches Newton's method, and favors the rate of convergence but might re-

quire a greater computational cost per step. Different left-hand side operators will differ in terms of consistency with the right-hand side, robustness, computational cost and required number of nonlinear steps to converge.

A compromise is often needed between the accuracy of the left-hand side operator $\mathbf{J} \approx \dfrac{\partial \mathbf{R}}{\partial \mathbf{w}}$ and the computational cost of a linear step. When the right-hand side $\mathbf{R}$ is computed with high-order schemes, as it is often the case in modern simulations, the exact Jacobian matrix becomes relatively dense and is likely to be heavy to both compute and store [97]. Besides, the analytical Jacobian matrix is generally easily calculated by hand for relatively basic spatial discretizations, such as the Steger-Warming flux discretization. Unfortunately, for more accurate discretizations requiring more involved numerical fluxes, the analytical linearization of the residual including the boundary conditions and nonlinear limiters becomes an intractable problem.

Defect Correction Approach

A way to make the left-hand side easier to calculate at the expense of consistency is to use a defect correction approach in which a lower-accurate residual $\mathbf{R}_{low}$ is employed to form the left-hand side Jacobian matrix [98]. In aerodynamic applications, a lower order discretization scheme or an exact flux Jacobian derived from simplified governing equations are often used. Consequently, the approximate Jacobian matrix $\mathbf{J}$ does not include all the effects of the high-order discretization used to compute the numerical flux $\mathbf{R}$. On the other hand, this approximation yields a sparser Jacobian matrix which is therefore computationally easier to handle. The linearized equations (3.27) become

$$\left[\frac{\mathbf{M}}{\Delta\tau} + \left(\frac{\partial \mathbf{R}_{low}}{\partial \mathbf{w}}\right)^n\right] \Delta\mathbf{w}^n = -\mathbf{R}(\mathbf{w}^n). \tag{3.30}$$

The two sides of the equations are inconsistent since $\dfrac{\partial \mathbf{R}_{low}}{\partial \mathbf{w}} \neq \dfrac{\partial \mathbf{R}}{\partial \mathbf{w}}$, which ultimately restricts the length of the time steps $\Delta\tau$ and the convergence rate. Therefore, the number of steps to converge to the same tolerance is likely to increase, but the reduction in time per step can offset it in return. Nonetheless, low-order Jacobian matrices tend to be better con-

ditioned, with a better diagonal dominance. Consequently, the resulting linear systems are easier to precondition and solve.

In the flow solver FANSC, the low-order Jacobian matrix is computed using first-order discretization schemes. If the convective flux is discretized with a JST central scheme, the convective low-order flux Jacobian employs a first-order scalar dissipation flux. Similarly, in the case of a MATD discretization, the first-order upwind Roe scheme is used for the Jacobian matrix [99]. As for the viscous flux, the viscous Jacobian matrix is computed with the thin-shear layer approximation of the governing equations [100].

Nearly Consistent Left-Hand Side

The most widely used iterative solvers for solving large and sparse linear systems such as in equation (3.27) are the Krylov methods in which the Jacobian matrix is only involved through matrix-vector multiplications. This property gives the possibility to use the so-called matrix-free implementations [77], where the action of the Jacobian matrix on a vector is approximated by a forward finite difference:

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{w}}\right)^n \Delta \mathbf{w}^n \approx \frac{\mathbf{R}(\mathbf{w}^n + h\Delta \mathbf{w}^n) - \mathbf{R}(\mathbf{w}^n)}{h}. \tag{3.31}$$

In equation (3.31), the Jacobian-vector multiplication approximation requires the evaluation of the residual $\mathbf{R}$ for two different states. They are computed using the same residual accuracy as the right-hand side, ensuring a better consistency between both sides in (3.27).

The matrix-free approach has some important advantages over the defect correction approach. First, a matrix-free Jacobian matrix clearly exempts the storage of the Jacobian matrix. In addition, this approach provides a high-order linearization of the residual [11]. All the phenomena taken into account in the numerical fluxes in $\mathbf{R}$ are also integrated in the flux Jacobian. Even though the matrix-free flux Jacobian remains an approximation of the exact Jacobian matrix with the influence of extraneous factors such as the step size h and the rounding errors, past studies [77, 101, 102] have shown that the quadratic convergence of Newton's method is still achievable. In addition, the convergence is often indistinguishable with analytical Jacobian matrices. Orkwis et al. concluded in [103] that numerical Jacobian approaches should be preferred over analytical computations due to

their coding simplicity and superior robustness. The forward finite difference is the most commonly used method. Different schemes which differ in computational cost and accuracy could be employed, such as a central finite difference [104].

The step size h must be carefully chosen to mitigate numerical errors. If h is too large, the finite difference is poorly approximated due to the truncation error in the Taylor approximation. On the contrary, too small a value for h soars the floating-point roundoff error in the denominator. Knoll and Keyes reviewed a wide variety of formulas in [77] to determine the step h related to the machine epsilon ϵ and the vectors $\mathbf{w}^n$ and $\Delta\mathbf{w}^n$. The simplest form for the step is

$$h = \frac{\sqrt{\epsilon}}{||\Delta\mathbf{w}^n||_2}, \tag{3.32}$$

and the slightly more elaborated formula from [101] reads

$$h = \frac{\sqrt{\epsilon}}{||\Delta\mathbf{w}^n||_2^2} \max\left[|\mathbf{w}^n \cdot \Delta\mathbf{w}^n|, \text{typ}\left(\mathbf{w}^n \cdot |\Delta\mathbf{w}^n|\right)\right] \text{sign}\left(\mathbf{w}^n \cdot \Delta\mathbf{w}^n\right), \tag{3.33}$$

with ϵ being the machine epsilon and $\text{typ}(\cdot)$ a typical size of the term in brackets. Both forms are implemented within FANSC but they only lead to minor differences in the reported convergences.

However, the preconditioning of matrix-free Krylov methods remains unclear. Most of the commonly used preconditioners, such as SOR and ILU, are directly based on the coefficients of the Jacobian matrix. This obviously poses a problem when the Jacobian matrix is never explicitly calculated. As a consequence, a simplification of the flux Jacobian, which can be easily computed and stored, is used to compute the preconditioner for the Krylov method. In our case, we employ the same first-order approximation of the Jacobian matrix as presented in the defect correction approach. This ensures a suitable sparsity for the Jacobian matrix, and therefore a decent computational cost for the preconditioner application. We should note that some matrix-free preconditioners have been devised to obtain a real matrix-free Newton-Krylov approach, without any matrix storage [77, 105].

3.5 Properties of the Implicit Operator

3.5.1 Structure of the Jacobian Matrix

The discretization process transforms the governing equations into a set of algebraic equations, one for each unknown of each cell in the mesh. These algebraic equations define the dependencies between elements. The discretization stencils are much smaller than the computational domain. Hence the coupling between unknowns is rather weak, meaning that only a few elements, which are usually the immediate neighbors, strongly influence each other. As a result, the Jacobian matrix is a large and sparse matrix.

An adequate ordering strategy of the unknowns is paramount in CFD solvers to ensure stability of the approximate factorization methods and optimize data dependency on highly parallel implementations. The determination of an appropriate ordering is particularly a significant issue for unstructured grids, for which a natural ordering does not exist. In a block structured solver, the conservative unknowns of one cell are usually consecutively stored. The natural row-wise ordering of the mesh cells is employed to assemble the global unknown vector $\mathbf{w}$. For instance, this ordering in 2D yields the following state vector:

$$\mathbf{w}^T = \left[\rho_1, (\rho v_x)_1, (\rho v_y)_1, (\rho E)_1, \rho_2, (\rho v_x)_2, (\rho v_y)_2, (\rho E)_2, ...\right]. \tag{3.34}$$

The discretized algebraic equations follow the same ordering by "grid points" which leads to a block Jacobian matrix, whose block size is the number of conservative variables in a cell. Donato showed in [106] that an ordering based on the equations yields less robust convergence for iterative methods by mitigating the coupling between the unknowns in the same cell.

The flux Jacobian entries are defined as the derivatives of the numerical flux with respect to the discretized conservative variables. The numerical flux in a cell only depends on a small subset of cells within the discretization stencil. As a consequence, the number of nonzero blocks in a row of the Jacobian matrix corresponds to the number of cells in the

stencil of the discretization method. Thus, a high-order discretization method, resulting in more dependencies between cells, entails denser Jacobian matrices.

For instance, let us consider a very small 2D grid containing only six cells depicted in Figure 3.1. In a five-point stencil, involving only the immediate neighboring cells, the "grid point" ordering leads to a sparse block Jacobian matrix with a symmetric pattern. No matter the size of the grid, the interior cells lead to rows with no more than five dense blocks. Moreover, a higher dimension results in a denser Jacobian matrix: the central scheme in 3D with a seven-point stencil entails a block-septadiagonal Jacobian matrix.

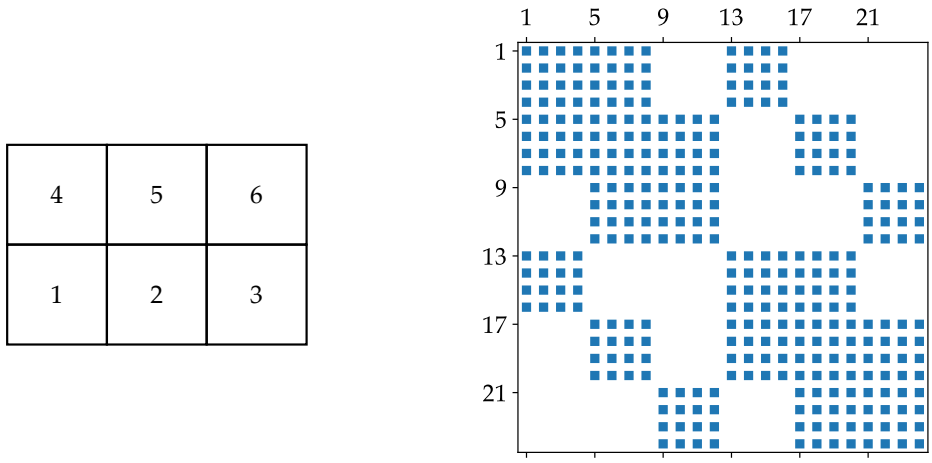


Figure 3.1: 2D structured mesh (on the left) and the resulting pattern of the associated Jacobian matrix with a five-point stencil (on the right). Each blue square is a nonzero entry.

3.5.2 Poor Conditioning

The compressible Navier-Stokes equations model complex flows involving multi-scale phenomena. The strongly coupled variables as well as the propagation of acoustic and shock waves are among the factors that give rise to deeply stiff equations to solve. First, direct methods, known for their robustness, are simply intractable due to the size of the involved matrices in typical simulations. On the other hand, the lack of decent properties of the linearized equations prevents the use of classical iterative methods, which often diverge.

The diagonal dominance of the coefficient matrix tends to increase the stability of the linear solver. A matrix with large magnitudes of diagonal entries compared to the magnitudes of off-diagonal entries leads to more stable factorizations and a faster convergence in the linear solver. In contrast, low diagonal matrices may yield poor subdomain ILU factorizations, introducing instabilities that can hamper the efficiency of the domain decomposition preconditioners [107]. In implicit solvers, the spatial discretization schemes for the convective part of the flux, such as JST and MATD, add a strong skew-symmetric contribution to the Jacobian matrix and greatly mitigate its diagonal dominance. Even though the artificial dissipation flux helps to balance the lack of robustness introduced by the convective flux, the Jacobian matrices generally remain significantly ill-conditioned.

3.6 Flow Solver

The numerical experiments were conducted with Bombardier's in-house flow solver *Full-Aircraft Navier-Stokes Code* or FANSC for block structured meshes [108]. FANSC can solve either the compressible 3D Euler or RANS equations using a cell-centered finite-volume discretization. The solution of the linearized equations (3.27) is the most time-consuming part of the flow solver.

Dozens of scientific libraries are devoted to the parallelization of linear algebra routines. In this research work, we used the Portable, Extensible Toolkit for Scientific Computation (PETSc) developed and maintained by Argonne National Laboratory at the University of Chicago [109]. The library provides a large set of data structures and routines suited for sparse matrix computations and the solving of partial differential equations. These routines can easily be implemented and appended to pre-existing scientific softwares such as CFD solvers. PETSc offers a complete scalable linear equation solver object to use a wide range of Krylov subspace methods in conjunction with several in-built preconditioning techniques, such as SOR, ILU, block Jacobi and ASM. Each Krylov method can be tuned with different parameters, such as the maximum number of iterations and the desired relative tolerance. Besides, some routines are available to let the user define his own matrix and preconditioner classes.

3.7 Parallelism in the CFD Code

The coarse-grained parallelism of the CFD code is achieved by a domain decomposition strategy with the partitioning of the original grid into smaller non-overlapping blocks [110]. In structured block solvers, the mesh blocks and associated data are initially scattered across the processors. Some additional layers of cells, called halo cells, are added to perform the inter-block communications. Along the convergence, the cores process their locally owned data corresponding to their assigned mesh blocks. The necessary communications between neighboring blocks for data exchanges are performed with the MPI standard.

The decomposition of the mesh across the workstation is not straightforward and the partitioning process must take into account the load balancing to ensure that all cores will approximately have the same computational work. With unsuitable decompositions, some cores might possess too many blocks compared to others. In the worst-case scenario, some cores remain idle waiting for the overloaded cores to complete their calculations. Usually, these concerns are addressed by dividing the original mesh into a larger number of smaller blocks than the number of available cores. To this extent, several blocks are assigned to the same core. Nevertheless, an excessive domain splitting comes with a larger number of halo cells that can dramatically increase the overall computational cost and communication count. The block-splitting strategy used in FANSC along with the data communication framework is explained in [111].

In Figure 3.2, we depict the decomposition of a simple 2D mesh into four non-overlapping subblocks on the left. The corresponding stored data with one layer of halo cells surrounding the block boundaries is shown on the right side. The white circles are the halo cells that are used to store the values of the neighboring cells, whereas the filled circles are the inner cells. For instance, the cells pointed by the arrows correspond to only one physical cell, whose flux computations are carried out in the upper left block. Thereafter, its values are shared with the other three subblocks with point-to-point communications.

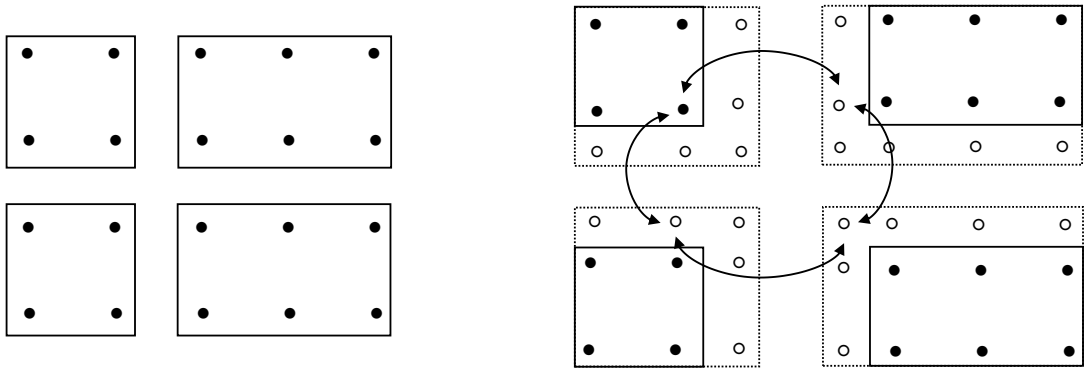


Figure 3.2: Example of a computational domain split into four subblocks (left). The subblocks are padded with a layer of halo cells (right).

The number of layers of halo cells is mainly dependent on the size of the stencil used for the numerical discretization. The example in Figure 3.2 is compatible with a first-order 5-point stencil, in which only the neighboring cell data is needed. In FANSC, the second-order flux discretization requires the storage of two layers of halo cells.

Chapter 4

Parallel Preconditioning with the Sparse Approximate Inverse

Solution methods for the linear and nonlinear equations were reviewed in Chapter 2 with a focus on Krylov subspace methods, and primarily the GMRES approach. The well-established splitting-based and ILU preconditioners were introduced, as well as parallelization frameworks through the block Jacobi and additive Schwarz methods. In particular, their lack of robustness as a function of parallel partitioning was highlighted [112].

As a remedy, we present in this section the parallel Sparse Approximate Inverse (SPAI) preconditioning method, an explicit preconditioner independent of the computational domain partitioning and ordering. This preconditioning method was extensively studied for relatively simple problems of low-dimension linear systems on sequential or weakly parallel architectures. In these settings, SPAI can hardly compete with the sound sequential preconditioning techniques such as ILU. To our knowledge, SPAI preconditioning has been exclusively studied for given linear systems of equations arising from scientific applications such as in electromagnetism [113, 114] or for convective-diffusion equations [115, 116]. Quite surprisingly, SPAI has not been investigated for solving complex nonlinear equations on high-end parallel machines. Nonetheless, in the survey [117], Benzi and Tůma stated, "We think that the highest potential for approximate inverse techniques lies in their use as part of sophisticated solving environments for nonlinear and time-dependent problems". With the advent of large parallel computing resources, we investigated the SPAI preconditioner for parallel CFD solvers, requiring the solution of numerous linear systems of equations. The first step is to devise a sufficiently flexible SPAI algorithm, which can be used within the solver.

In Section 4.1, the SPAI algorithm is first introduced in its historical formulation as presented by Grote and Huckle in [118]. Thereupon, its block version adapted to block structured coefficient matrices is detailed. Finally, we present a preconditioning framework for the SPAI preconditioning suitable for use in an implicit solver in Section 4.2.

4.1 Sparse Approximate Inverse (SPAI) Preconditioning

The convergence of a Krylov subspace method for linear systems of very poorly conditioned equations is mainly determined by the efficiency of the preconditioning techniques.

The first studies conducted on the computation of a sparse approximation of the inverse of a matrix by norm minimization date back to the second half of the 20^{th} century [119, 120]. Thereafter, the sparse approximate inverse preconditioning has been considered throughout the 1990s with the development of massively parallel machines. Early research focused on the computation of the approximate inverse $\mathbf{M}$ from a given sparsity pattern $\mathcal{P}$ provided by the user. The choice of $\mathcal{P}$ was mostly based on empirical observations [113]. The difficulty in finding a satisfying *a priori* sparsity pattern to compute an efficient sparse approximate inverse drove researchers to devise adaptive algorithms to automatically capture the pattern of the inverse of $\mathbf{A}$. In these algorithms, a simple pattern such as a diagonal matrix is iteratively augmented to find the largest entries in $\mathbf{A}^{-1}$. The first adaptive strategy is known to be the one developed by Cosgrove et al. [121]. The most commonly used adaptive algorithm has been proposed by Grote and Huckle [118] and is the one presented here. This variant is often referred to as the Sparse Approximate Inverse (SPAI) preconditioner. A comprehensive comparative study of different sparse approximate preconditioners for Krylov subspace methods on a vector computer, including factorized approximate inverses, was conducted by Benzi and Tůma in their survey [117].

The main benefit of SPAI preconditioning is not only its high degree of parallelism, but also its invariance with respect to the number of cores, unlike domain decomposition preconditioners. The parallelism in the SPAI preconditioning emerges both in its computation and its application. First, each column of the preconditioning matrix $\mathbf{M}$ can be

computed independently of one another. Secondly, the application of the SPAI preconditioner in the Krylov subspace method consists of a sparse matrix-vector multiplication, which can be adequately implemented in parallel architectures [122].

4.1.1 Algorithm

The main idea of the SPAI preconditioning algorithm is to construct a sparse approximation $\mathbf{M}$ of $\mathbf{A}^{-1}$ through the minimization of the following Frobenius norm:

$$\min_{\mathcal{P}(\mathbf{M})\in\mathcal{P}} ||\mathbf{AM}-\mathbf{I}||_F, \tag{4.1}$$

with $\mathbf{A},\ \mathbf{M} \in \mathbb{R}^{n\times n}$ and a prescribed sparsity pattern $\mathcal{P}$ for the matrix $\mathbf{M}$. The parallelism comes from the splitting of the Frobenius norm (4.1) into the sum of n Euclidean norms as

$$||\mathbf{AM}-\mathbf{I}||_F^2 = \sum_{k=1}^{n} ||\mathbf{Am}_k - \mathbf{e}_k||_2^2, \tag{4.2}$$

with $\mathbf{m}_k$ and $\mathbf{e}_k$ being respectively the k-th column of $\mathbf{M}$ and $\mathbf{I}$. Thereafter, it follows that the minimization of the sum in (4.2) is equivalent to the simultaneous minimization of each term of the sum, namely

$$\min_{\mathcal{P}(\mathbf{m}_k)\in\mathcal{P}_k} ||\mathbf{Am}_k - \mathbf{e}_k||_2^2, \ \text{ for } \ k \in \{1,...,n\}. \tag{4.3}$$

In this way, the original Frobenius minimization problem (4.1) is decomposed into n independent linear least-squares problems. The solutions of the least-squares problems are only determined by the chosen sparsity pattern $\mathcal{P}$ for $\mathbf{M}$. Therefore, the pattern $\mathcal{P}$ should be the subject of close attention, as it is the cornerstone of an effective preconditioner. Nonetheless, the use of an adaptive algorithm to automatically expand the pattern mitigates the importance of the *a priori* pattern.

From the prescribed sparsity pattern $\mathcal{P}$, the index set $\mathcal{J}_k$ gathers the nonzero entry indices in the column $\mathbf{m}_k$ as

$$\mathcal{J}_k = \{j \in \{1,...,n\} \mid \mathbf{m}_k(j) \neq 0\}. \tag{4.4}$$

Both sparsities of the coefficient matrix $\mathbf{A}$ and the column $\mathbf{m}_k$ are taken into consideration to reduce the size of the minimization problems as much as possible. First, only nonzero entries of $\mathbf{m}_k$ and the corresponding columns of $\mathbf{A}$ are retained. The residuals $\mathbf{r}_k$ of the reduced least-squares problems (4.3) become

$$\mathbf{r}_k = \mathbf{A}\mathbf{m}_k - \mathbf{e}_k = \mathbf{A}(\,\cdot\,, \mathcal{J}_k)\mathbf{m}(\mathcal{J}_k) - \mathbf{e}_k, \quad \text{for } \; k \in \{1, ..., n\}. \tag{4.5}$$

In the same vein, if a row in $\mathbf{A}(\,\cdot\,, \mathcal{J}_k)$ is identically zero, its corresponding contribution in the matrix-vector product $\mathbf{A}\mathbf{m}_k$ is zero and the row is of no interest in the least-squares problem. The index set of nonzero rows of the matrix $\mathbf{A}(\,\cdot\,, \mathcal{J}_k)$ is denoted $\mathcal{I}_k$, namely

$$\mathcal{I}_k = \{i \in \{1, ..., n\} \mid \mathbf{A}(i, \mathcal{J}_k) \neq 0\}, \tag{4.6}$$

and the final reduced problems are

$$\min_{\mathcal{P}(\mathbf{m}_k) \in \mathcal{P}_k} ||\mathbf{A}(\mathcal{I}_k, \mathcal{J}_k)\mathbf{m}_k(\mathcal{J}_k) - \mathbf{e}_k(\mathcal{I}_k)||_2^2, \quad \text{for } \; k \in \{1, ..., n\}. \tag{4.7}$$

The resulting least-squares problems are of size $|\mathcal{I}_k| \times |\mathcal{J}_k|$ and much smaller than the original problems due to the high sparsity of $\mathbf{A}$ and $\mathbf{M}$. For readability, we use the following notations for the reduced coefficient matrix and vectors: $\hat{\mathbf{A}}_k = \mathbf{A}(\mathcal{I}_k, \mathcal{J}_k)$, $\hat{\mathbf{m}}_k = \mathbf{m}(\mathcal{J}_k)$ and $\hat{\mathbf{e}}_k = \mathbf{e}(\mathcal{I}_k)$. The nonsingularity of $\mathbf{A}$ ensures that the reduced matrix $\hat{\mathbf{A}}_k$ has full rank and that its QR decomposition can be computed. In this way, the problems in (4.7) are solved using the QR factorization $\hat{\mathbf{A}}_k = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, where $\hat{\mathbf{Q}} \in \mathbb{R}^{|\mathcal{I}_k| \times |\mathcal{J}_k|}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{|\mathcal{J}_k| \times |\mathcal{J}_k|}$, computed with Householder reflections:

$$\begin{cases} \hat{\mathbf{c}}_k = \hat{\mathbf{Q}}^T \hat{\mathbf{e}}_k, \\ \hat{\mathbf{m}}_k = \hat{\mathbf{R}}^{-1} \hat{\mathbf{c}}_k. \end{cases} \tag{4.8}$$

The vector $\hat{\mathbf{c}}_k$ is retrieved by multiplication of the successive Householder reflections, while $\hat{\mathbf{m}}_k$ is obtained by means of a backward substitution.

Patterns for the SPAI Matrix

Early implementations of the SPAI preconditioning considered using the pattern of $\mathbf{A}$ to compute $\mathbf{M}$ [21, 123]. Although giving decent results for a good number of linear problems, they can also lead to poor convergence, or even fail [124]. As noted in [52], the pattern $\mathcal{P}(\mathbf{A})$ may yield zero columns in the preconditioner $\mathbf{M}$, when $\mathbf{A}$ contains zeros in its diagonal entries. These conclusions led to the improvement of fixed sparsity patterns for an inverse matrix. For instance, based on the Cayley-Hamilton theorem, the inverse matrix $\mathbf{A}^{-1}$ can be formulated as a linear combination of the powers of $\mathbf{A}$:

$$\mathbf{A}^{-1} = -\frac{1}{p_0} \sum_{i=0}^{n-1} p_{i+1} \mathbf{A}^i. \tag{4.9}$$

The theorem motivated the use of sparsified powers of $\mathbf{A}$ by Chow in [123]. In practice, the pattern $\mathcal{P}$ can be taken as the sparsified pattern of a low power of $\mathbf{A}$. A common way to sparsify the pattern of a matrix is to use a threshold condition in which only the largest entries are retained in the matrix pattern [113].

4.1.2 Adaptive Strategy

The main advantage of the adaptive SPAI method is its ability to easily augment the sparsity of the preconditioner $\mathbf{M}$. Following the adaptive algorithm from Grote and Huckle [118], the SPAI computation for $\mathbf{m}_k$ is repeated with a larger pattern $\mathcal{P}_k$ until the Euclidean norm of the least-squares problem (4.3) is lower than a user-defined tolerance ϵ, such as

$$||\mathbf{A}\mathbf{m}_k - \mathbf{e}_k||_2 < \epsilon. \tag{4.10}$$

The parameter ϵ controls the level of fill-in in the SPAI column $\mathbf{m}_k$. A low ϵ entails a dense SPAI matrix, which more accurately approximates the inverse of $\mathbf{A}$, reducing the number of linear iterations. In return, the SPAI matrix is also more expensive to compute and apply within a Krylov method. The new indices are chosen based on the calculation of a cheap estimate of their contribution to the reduction of the column residual norm $||\mathbf{A}\mathbf{m}_k - \mathbf{e}_k||_2$.

As well as the tolerance ϵ, a maximum number of adaptive steps and new entries per step are provided to offset the sparsity of the approximate inverse and its accuracy as an approximation of $\mathbf{A}^{-1}$. The purpose of the adaptive algorithm is to capture the sparsity pattern of the absolute value of the largest entries of $\mathbf{A}^{-1}$ to expand the sparsity pattern $\mathcal{P}$. However, there is no clear evidence that a pattern based on the largest entries of the inverse yields a sound preconditioner. For instance, it was shown that the adaptive SPAI algorithm may eventually fail to recover the pattern of $\mathbf{A}^{-1}$ while providing a satisfying preconditioner in [125].

At the end of the SPAI computation for a given pattern $\mathcal{P}$, unless the exact inverse column $\mathbf{m}_k$ has been found, the residual $\mathbf{r}_k$ is nonzero. We form the set

$$\mathcal{L}_k = \{l \mid \mathbf{r}_k(l) \neq 0\}, \tag{4.11}$$

which gathers the nonzero entries of $\mathbf{r}_k$. The objective of the adaptive strategy is to find additional nonzero entries in $\mathbf{m}_k$ that can efficiently lower the residual norm $||\mathbf{r}_k||_2$. For each index $l \in \mathcal{L}_k$, the index set

$$\mathcal{N}_l = \{j \mid \mathbf{A}(l,j) \neq 0 \text{ and } j \notin \mathcal{J}_k\} \tag{4.12}$$

is constructed. The set $\mathcal{N}_l$ contains the potential new indices that could be added to $\mathcal{J}_k$ in order to further reduce the magnitude of the l-th entry of $\mathbf{r}_k$. Finally, the new entries are chosen among the union $\tilde{\mathcal{J}}_k = \bigcup_{l \in \mathcal{L}_k} \mathcal{N}_l$.

An inexpensive upper bound of the reduction in the residual induced by each potential candidate $j \in \tilde{\mathcal{J}}_k$ is estimated with the solution of a one-dimensional minimization problem:

$$\min_{\mu_j \in \mathbb{R}} ||\mathbf{A}(\mathbf{m}_k + \mu_j \mathbf{e}_j) - \mathbf{e}_k||_2 \;=\; \min_{\mu_j} ||\mathbf{r}_k + \mu_j \mathbf{A}\mathbf{e}_j||_2, \tag{4.13}$$

where the minimum is attained for $\mu_j^* = -\dfrac{\mathbf{r}_k^T \mathbf{A}\mathbf{e}_j}{||\mathbf{A}\mathbf{e}_j||_2^2}$. Therefore, the new residual norm ρ_j, obtained with adding the index $j \in \tilde{\mathcal{J}}_k$, is

$$\rho_j^2 = ||\mathbf{A}(\mathbf{m}_k + \mu_j^* \mathbf{e}_j) - \mathbf{e}_k||_2^2 = ||\mathbf{r}_k||_2^2 - \frac{(\mathbf{r}_k^T \mathbf{A}\mathbf{e}_j)^2}{||\mathbf{A}\mathbf{e}_j||_2^2}. \tag{4.14}$$

It follows that the index j entails a reduction of $\frac{(\mathbf{r}_k^T \mathbf{A}\mathbf{e}_j)^2}{||\mathbf{A}\mathbf{e}_j||_2^2}$ in the column residual square norm $||\mathbf{r}_k||_2^2$. The most profitable indices with the lowest residuals ρ_j, and thus the highest μ_j^*, are added to $\mathcal{J}_k$.

Certain columns may require numerous adaptive steps to satisfy the condition (4.10). Therefore, a maximum number of adaptive steps is set to prevent inadequate dense SPAI columns. The optimal value for ϵ is highly problem and machine dependent. Ideally, ϵ should lead to a sufficiently sparse SPAI matrix $\mathbf{M}$ to ensure low application costs within the linear solver while providing efficient preconditioners to reduce the number of linear iterations. The presented adaptive strategy is not unique, and other authors have proposed different methods to improve the sparsity pattern of $\mathbf{M}$, such as in [121, 126]. Nonetheless, the adaptive SPAI from Grote and Huckle is the most widely used in SPAI implementations and provides sufficient freedom to adjust the density of $\mathbf{M}$.

4.1.3 Block Generalization of the SPAI Algorithm

The SPAI algorithm can be easily extended to a block version. In Section 3.5, we saw that matrices arising from the discretization of partial differential equations naturally lead to block structured matrices, whose block size is the number of unknowns per cell. When applied to block matrices, the original SPAI algorithm may lose a considerable amount of time reproducing the same computations for SPAI columns belonging to the same block. The following section presents the block version of the SPAI algorithm as first introduced by Barnard and Grote [127].

Let us denote b the constant block size of the block matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. The block SPAI algorithm divides the original minimization problem into n/b independent problems whose unknowns are the block columns $\mathbf{M}_k$ of dimension $n \times b$ as

$$||\mathbf{A}\mathbf{M} - \mathbf{I}||_F^2 = \sum_{k=1}^{n/b} ||\mathbf{A}\mathbf{M}_k - \mathbf{E}_k||_F^2. \tag{4.15}$$

Thus, the individual problems become

$$\min_{\mathcal{P}(\mathbf{M}_k) \in \mathcal{P}_k} ||\mathbf{A}\mathbf{M}_k - \mathbf{E}_k||_F^2. \tag{4.16}$$

In the same way as for the scalar SPAI, a starting block sparsity pattern $\mathcal{P}$ is provided to define a unique solution. The common sparsity structure for all the scalar columns in $\mathbf{M}_k$ entails reduced problems with the exact same coefficient matrix $\hat{\mathbf{A}}_k = \mathbf{A}(\mathcal{I}_k, \mathcal{J}_k)$ but different sparse vectors $\mathbf{e}_i$. Hence, the benefit from the block SPAI formulation comes from the fact that a unique QR decomposition is required to determine the b scalar columns in $\mathbf{M}_k$. As in (4.8), the block column $\mathbf{M}_k$ is retrieved in two successive steps:

$$\begin{cases} \hat{\mathbf{C}}_k = \hat{\mathbf{Q}}^T \hat{\mathbf{E}}_k, \\ \hat{\mathbf{M}}_k = \hat{\mathbf{R}}^{-1} \hat{\mathbf{C}}_k. \end{cases} \tag{4.17}$$

The QR factorization being the most time-consuming routine in the SPAI algorithm, this block alternative helps to drastically decrease the computational cost of the construction of $\mathbf{M}$.

The main difference with the scalar algorithm concerns the dynamic criterion to augment the sparsity pattern of the block column $\mathbf{M}_k$. The set of potential new indices $\tilde{\mathcal{J}}_k$ is constructed in the same way but considering block instead of scalar indices. For a given block index $j \in \tilde{\mathcal{J}}_k$, the determination of its contribution in the reduction in $\|\mathbf{A}\mathbf{M}_k - \mathbf{E}_k\|_F$ would require the expensive solution of the SPAI minimization problem $\min_{\mathcal{P}(\mathbf{M}_k)\cup\{j\}} \|\mathbf{A}\mathbf{M}_k - \mathbf{E}_k\|_F$. Instead, a cheap estimation ρ_j of the reduction induced by the extra block $\mathbf{A}_j\mathbf{M}_{jk}$ is calculated while maintaining all the entries in $\mathbf{M}_k$. The norm ρ_j is given by

$$\rho_j^2 = ||\mathbf{R}_k - \mathbf{A}_j\mathbf{M}_{jk}||_F^2 = ||\mathbf{R}_k||_F^2 + \text{trace}\left(\mathbf{R}_k^T \mathbf{A}_j \mathbf{M}_{jk}\right). \tag{4.18}$$

The minimum reduced residual norm ρ_j is attained for

$$\mathbf{M}_{jk} = -(\mathbf{A}_j^T \mathbf{A}_j)^{-1} \mathbf{A}_j^T \mathbf{R}_k. \tag{4.19}$$

Therefore, the estimates ρ_j can be explicitly computed as

$$\rho_j^2 = ||\mathbf{R}_k||_F^2 - \text{trace}\left(\mathbf{R}_k^T \mathbf{A}_j (\mathbf{A}_j^T \mathbf{A}_j)^{-1} \mathbf{A}_j^T \mathbf{R}_k\right). \tag{4.20}$$

As in the scalar SPAI version, only the most profitable block indices j are added to the sparsity pattern $\mathcal{P}(\mathbf{M}_k)$ from which a new SPAI matrix is calculated.

4.1.4 Implementation of the SPAI Algorithm

An early MPI implementation of the SPAI algorithm in C, called `spai-3.0`, was realized by Barnard et al. whose parallel implementation is extensively discussed in [125, 128]. `spai-3.0` reads a general sparse matrix stored in the coordinate format (COO), converts it into a compressed sparse column (CSC) matrix, and computes a right preconditioner in the CSC format. The common sparse matrix storage frameworks are reviewed in [3, 129].

The SPAI preconditioning matrix is computed with the adaptive algorithm of Grote and Huckle [118] from a diagonal sparsity pattern. Besides, the density of $\mathbf{M}$ is monitored using the following three parameters:

- The SPAI tolerance ϵ controls the density of the SPAI columns along the adaptive algorithm with the condition $||\mathbf{A}\mathbf{m}_k - \mathbf{e}_k||_2 < \epsilon$.
- The maximum number of adaptive steps per column.
- The maximum number of added nonzero entries in the sparsity pattern $\mathcal{P}_k$ per adaptive step.

The `spai-3.0` software is available as an external package in PETSc to enable SPAI as a preconditioner for Krylov subspace methods. Its implementation is quite unsuitable for use in a complex solver, given that the structures are allocated at each new preconditioning computation and the implementation does not offer a lot of flexibility. For instance, the preconditioner is recomputed from scratch at every nonlinear iteration, from a diagonal pattern without reusing any information from the previous computations. Nevertheless, the essence of its parallel programming paradigm is the same as that of our implementation of SPAI used in the numerical experiments.

In the classical formulation of Grote and Huckle [118], the SPAI algorithm is presented for the computation of a right preconditioner, namely with the minimization of $||\mathbf{A}\mathbf{M} - \mathbf{I}||_F$. This algorithm is adapted to matrices stored by columns. In distributed

memory architectures, matrices are usually stored in a compressed sparse row (CSR) format in which continuous rows are assigned to different cores. For instance, Goharian et al. showed in [130] the benefits of using the CSR format to optimize matrix operations such as sparse matrix-vector multiplication. This is the case of the matrices stored in the CFD solver used. Therefore, it becomes more natural to compute a left preconditioner through the minimization of $||\mathbf{MA} - \mathbf{I}||_F$. The main difference with a right SPAI preconditioning lies in the way in which entries in $\mathbf{M}$ are computed. In $||\mathbf{AM} - \mathbf{I}||_F$, the columns of $\mathbf{M}$ are separately computed, whereas the parallelism takes place per row for the left preconditioning. Thus, the right preconditioning lends itself more to a matrix $\mathbf{M}$ stored in the CSC format, while the CSR format is more suitable for a left preconditioned Krylov method.

The same SPAI algorithm can be used for computing both a right CSC and left CSR SPAI preconditioner. This can be seen by understanding that the CSR storage of a matrix $\mathbf{A}$ is actually the CSC storage of its transpose $\mathbf{A}^T$ [131]. Consequently, computing a right SPAI preconditioner with `spai-3.0` from a CSR PETSc matrix $\mathbf{A}$, read as a matrix in CSC format, is actually the same as computing a left SPAI preconditioner $\mathbf{M}$ in the CSR format in PETSc. Thus, `spai-3.0` computes a left SPAI preconditioner for CSR matrices in PETSc. We note that the two minimization problems $||\mathbf{MA} - \mathbf{I}||_F$ and $||\mathbf{AM} - \mathbf{I}||_F$ lead to different sparse approximate inverses, even for the same prescribed sparsity pattern $\mathcal{P}$, which can be of different quality.

Primarily based on the Barnard's version of the SPAI algorithm, a C++ standalone `mspai-1.2` has been developed by Sedlacek at the Technical University of Munich [132]. The code `mspai-1.2` originally includes most of Barnard's `spai-3.0` implementation strategies except the block SPAI version. The software computes the SPAI preconditioner of a matrix stored in the COO format. Like most of the implementations of the SPAI preconditioner, this code lets the user specify the fill-in parameters such as the tolerance ϵ, the number of adaptive steps, and the number of added entries in the set $\mathcal{J}_k$ in each of these steps. Among the innovations, the user can choose the initial sparsity pattern of the SPAI matrix from an external file in the COO format in addition to starting from a diagonal pattern or using the pattern of the matrix $\mathbf{A}$ itself.

In our work, we enhanced `mspai-1.2` with a PETSc interface to use SPAI as a user-defined preconditioner in a Krylov method. The SPAI preconditioning framework, presented in the following section, has been incorporated in a PETSc implementation, to allow the user to control when the preconditioner should be recomputed during the Newton steps. In addition, a complete block version of the SPAI algorithm has been integrated in `mspai-1.2`, following Barnard's method in [127]. Possibilities to use sparsified powers of the matrix $\mathbf{A}$ or a pattern saved in memory as the starting pattern $\mathcal{P}$ were added. Allocation routines have been redesigned to avoid inappropriate reallocations that could occur with the PETSc SPAI external package. The benefit of using an object-oriented SPAI implementation is its reusability. New functionalities can easily be appended to the pre-existing software while keeping all the previously optimized routines.

In the rest of this section, we highlight the common parallelization strategies used in the SPAI implementations `spai-3.0` and `mspai-1.2`.

Load Balancing

In compliance with the distributed memory architecture, the cores process their locally owned columns, or rows, of the preconditioner $\mathbf{M}$. This partitioning is related to the decomposition of the domain into blocks. In order to mitigate the load imbalance, the blocks are scattered across the cores in such a way that the inter-core communications are minimized and the workload is shared equitably. Generally, the cores have more or less the same number of cells, which corresponds to an equivalent number of columns, or rows, in the Jacobian matrix.

Nevertheless, it may happen that some SPAI columns require more adaptive steps, and are therefore more computationally expensive, than the average case. In this situation, a few cores might achieve their computations in advance and remain idle waiting for the other cores. In the `spai-3.0` program, a dynamic strategy is implemented to let the idle cores fetch unprocessed columns from other cores and carry out their computation. This procedure yielded quite poor results for large size matrices on highly partitioned meshes and was thus disabled in our tests.

Communications Between Cores

The SPAI computation of a column in **M** requires access to potentially any entry in the coefficient matrix **A**. Hence, a sophisticated communication framework is sought to efficiently allow communications along the computations in a non-invasive manner. One-sided communications in MPI are a convenient way to fetch data on another core without requiring any action from the other core. On the contrary, within a two-sided communication, both concerned cores, namely the sender and receiver, must call separate routines such as `MPI_Send` and `MPI_Recv` at the same time, introducing implicit synchronization along the computation [133].

At the time when Barnard et al. developed their version of the SPAI implementation, one-sided communications were not yet available in the MPI standard [134]. In order to cope with this issue, they developed an artificial one-sided communication strategy with the use of a communication server that handles the data requests of the different cores. In practice, the SPAI computations on the cores are conducted without any synchronization. Whenever a core requires remote data from another core, a message specifying the desired data is submitted with the nonblocking routine `MPI_Isend`, letting the core proceed its computations while the request is sent to the target core. All the cores intermittently check if a request has been sent from another core for its local data.

Latency Hiding

The inter-core communications are usually very slow compared to the inner-core operations in a distributed memory architecture [3]. Consequently, the code execution is substantially slowed down whenever a core submits a data request to another core. A common way to overcome this issue is to overlap the communications and operations within a core. This strategy is commonly known as "latency hiding". In `spai-3.0`, the latency hiding is ensured with two different strategies. First, once a core submits an asynchronous request to retrieve remote data, it repeatedly checks for messages from other cores concerning its local data. In this way, the cores are able to handle incoming requests meanwhile their own requests are processed. In addition to this asynchronous communication strategy, a hash table was implemented to cache any received data and avoid

repeated requests to the same remote data. As a matter of fact, consecutive SPAI columns $\mathbf{m}_k$, processed by the same core, often need access to the same remote data [125].

4.1.5 Numerical Result of the Preconditioning Methods

In this subsection, we study the convergence of a few small linear systems, listed in Table 4.1 with the splitting-based SOR preconditioner, the ILU factorization preconditioner and different versions of the SPAI preconditioner. The sparse coefficient matrices belong to the SuiteSparse matrix collection [135], formerly named the Florida Sparse Matrix Collection, which contains a wide range of matrices from scientific applications. Some of the coefficient matrices are provided with a right-hand side vector. When this is not the case, the right-hand side is taken as a vector full of ones, and a " $*$ " is written next to the matrix name in Table 4.1.

Table 4.1: Test matrix characteristics.

Matrix name	Problem	n	nnz	Condition Number	Diag. Dom.
ORSIRR_1 $*$	Oil reservoir simulation	1030	6858	7.71×10^4	Yes
ORSIRR_2 $*$	Oil reservoir simulation	886	5976	6.33×10^4	Yes
SHERMAN1	Black Oil simulator, Shale Barriers	1000	3750	1.56×10^4	No
SHERMAN3	Black Oil IMPES simulator	5005	20,033	5.01×10^{17}	Yes
SHERMAN5	Fully implicit Black Oil simulator	3312	20,793	1.88×10^5	No
PORES_2 $*$	Reservoir modeling	1224	9613	1.09×10^8	no
ORSREG_1 $*$	Oil reservoir simulation	2205	14,133	6.75×10^3	Yes

Nearly all the past SPAI experimental studies have focused on the convergence of linear systems from the SuiteSparse collection such as the matrices in Table 4.1 [117, 118, 136]. These systems are generally far from both the size and the complexity of the problems arising from industrial CFD simulations. On the other hand, they provide an accurate picture of the inherent preconditioning efficiency, as the problem sizes allow the study of metrics such as the condition number of the preconditioned systems and their eigenvalue distributions.

The tests are conducted using the restarted GMRES(30) linear solver from the library PETSc. The convergence is terminated when the relative residual norm $\frac{||\mathbf{b}-\mathbf{A}\mathbf{x}^{(p)}||_2}{||\mathbf{b}-\mathbf{A}\mathbf{x}^{(0)}||_2}$ has been reduced by at least four orders of magnitude or when the number of linear iterations reaches 10,000 iterations. In the latter case, the convergence is deemed to have failed. The initial guess for the linear solver is a zero vector.

The linear solvers were carried out on a single core of an Intel Xeon E5-2638 v4 Broadwell processor, with a clock frequency of 2.1 GHz. The SPAI preconditioner is not expected to compete with classical preconditioning methods, as the SPAI algorithm is computationally expensive and its inherent parallelism is not exploited with a single core.

First, the linear systems are solved without preconditioning methods. These results are reported in Table 4.2. Unsurprisingly, most of the convergences fail to converge in fewer than 10,000 iterations. In fact, the residual norm often stalls and the GMRES algorithm cannot reach the prescribed relative tolerance reduction. Convergence is only achieved for diagonally dominant matrices with relatively low condition numbers.

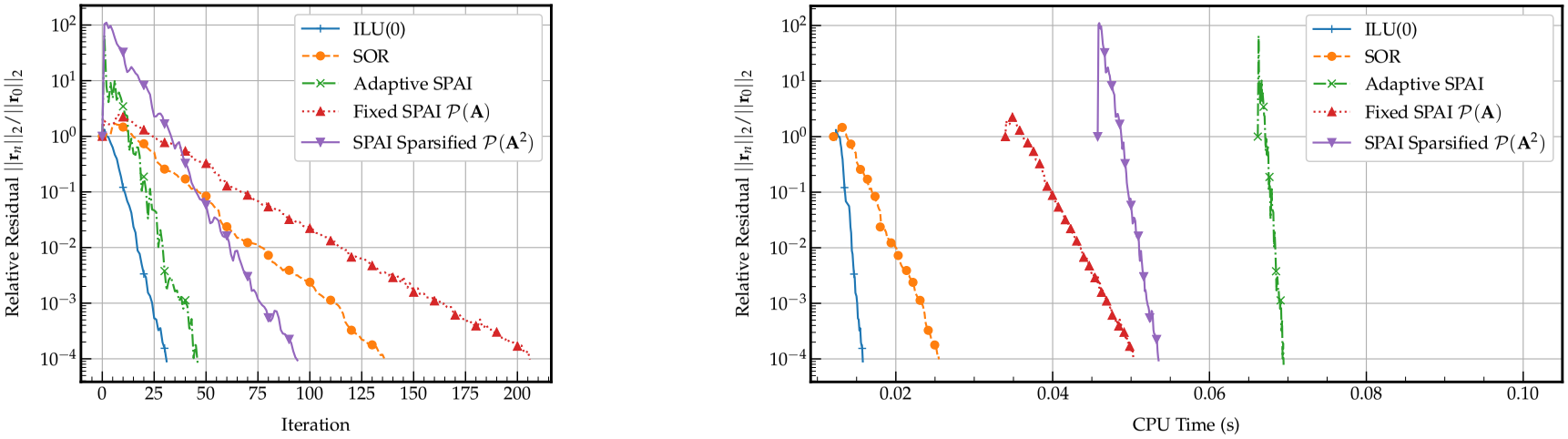
Table 4.2: Comparison of the number of GMRES iterations without preconditioning.

Matrix name	Iteration number	CPU time (s)
ORSIRR_1	2083	0.35
ORSIRR_2	940	0.103
SHERMAN1	Stalls	-
SHERMAN3	Stalls	-
SHERMAN5	Stalls	-
PORES_2	Stalls	-
ORSREG_1	74	0.017

Table 4.3 summarizes the results from the numerical experiments solved with a left preconditioned GMRES(30) method. In addition, the convergence of the residuals for ORSIRR_2 is shown in Figure 4.1 in terms of both the number of GMRES iterations and the CPU time. Note that the convergences begin once the preconditioners are computed, which explains the offsets in Subfigure 4.1b.

Table 4.3: Comparison of the CPU times and number of GMRES iterations for different left preconditioners.

	ILU(0)		SOR		Adaptive SPAI			Fixed SPAI $\mathcal{P}(\mathbf{A})$			SPAI Sparsified $\mathcal{P}(\mathbf{A}^2)$		
Matrix	Iter. count	Overall time	Iter. count	Overall time	Iter. count	Prec. time	Overall time	Iter. count	Prec. time	Overall time	Iter. count	Prec. time	Overall time
ORSIRR_1	31	0.017	121	0.025	38	0.053	0.062	172	0.034	0.051	90	0.047	0.058
ORSIRR_2	31	0.016	136	0.025	46	0.065	0.069	206	0.032	0.050	94	0.044	0.054
SHERMAN1	28	0.011	110	0.014	118	0.028	0.036	65	0.020	0.024	49	0.021	0.025
SHERMAN3	93	0.026	476	0.100	303	0.047	0.103	499	0.033	0.121	326	0.097	0.156
SHERMAN5	28	0.013	42	0.015	156	0.067	0.090	131	0.061	0.086	63	0.084	0.095
PORES_2	41	0.021	6024	0.463	462	0.077	0.111	1677	0.037	0.1655	590	0.051	0.103
ORSREG_1	29	0.020	57	0.022	31	0.052	0.057	102	0.043	0.061	76	0.070	0.083



(a) Convergence history in terms of iterations.

(b) Convergence history in terms of CPU time.

Figure 4.1: Convergence for ORSIRR_2 with different preconditioners.

We provide the number of nonzero entries (nnz) in the adaptive SPAI matrices in Table 4.4.

Table 4.4: Number of nonzero entries in the resulting adaptive SPAI matrix.

Matrix name	nnz
ORSIRR_1	5150
ORSIRR_2	4328
SHERMAN1	4606
SHERMAN3	17,177
SHERMAN5	24,314
PORES_2	10,454
ORSREG_1	11,025

We note that the adaptive SPAI generally achieves convergence in fewer linear iterations than a fixed SPAI preconditioner, computed from the sparsity pattern of $\mathbf{A}$ or the sparsified pattern of $\mathbf{A}^2$. Yet, some examples, such as the SHERMANN5 matrix, show the opposite. On the other hand, the adaptive SPAI preconditioner is usually more expensive to calculate, so that the overall static SPAI time, including SPAI calculation and convergence, is often faster. The adaptive SPAI algorithm is driven by many different parameters to control the amount of fill-in in the SPAI preconditioner. The lack of a clear relationship between the largest entries of the inverse of $\mathbf{A}$ and their importance in the matrix-vector product may explain why the quality of the SPAI is very case dependent.

Besides, thanks to the low dimension of the baseline matrices, it is computationally possible to explicitly calculate their condition number. Table 4.5 gathers some of the condition numbers of $\mathbf{MA}$ and $\mathbf{AM}$ for the adaptive SPAI matrices resulting from the minimization of $||\mathbf{MA} - \mathbf{I}||_2$. We note that the left SPAI matrix effectively lowers the condition number $\kappa(\mathbf{MA})$ for left preconditioned systems. On the other hand, the quality of the dynamic SPAI as a right preconditioner is more erratic and $\mathbf{M}$ sometimes fails to reduce the condition number. In contrast, the ILU(0) preconditioner proves to be a more robust preconditioner with a more efficient reduction of the condition number (cf. Table 4.6).

By using a SPAI preconditioner, it is implicitly anticipated that the inverse of a sparse matrix $\mathbf{A}$ can be adequately approximated by a sparse matrix. This is essential to obtain a sufficient approximation of the multiplication $\mathbf{A}^{-1}\mathbf{x}$ with a sparse matrix-vector prod-

Table 4.5: Condition numbers with adaptive SPAI preconditioning.

Matrix name	Condition number $\kappa(\mathbf{M_{SPAI}A})$	Condition number $\kappa(\mathbf{AM_{SPAI}})$
ORSIRR_1	69.62	88.36
ORSIRR_2	67.97	7.53×10^4
SHERMAN1	4115	1.09×10^7
SHERMAN5	2318	4.61×10^5

Table 4.6: Condition numbers with ILU(0) preconditioning.

Matrix name	Condition number $\kappa(\mathbf{M_{ILU}A})$	Condition number $\kappa(\mathbf{AM_{ILU}})$
ORSIRR_1	49.49	59.41
ORSIRR_2	49.49	59.40
SHERMAN1	129.31	129.31
SHERMAN5	248.67	4.61×10^4

uct in the Krylov methods. Yet, nothing really seems to justify this *a priori* assumption since the inverse of a sparse matrix is usually dense. Nonetheless, it is noted in [137] that for particular sparse diagonally dominant matrices, the entries of the inverse decay exponentially away from the diagonal. As a result, some inverses of sparse matrices may have only a handful of large entries. Therefore, the action of the inverse could be well approximated with a sparse matrix.

Let us consider ORSIRR_2 from the SuiteSparse collection. This matrix was generated from an oil reservoir simulation resulting from a finite element discretization. ORSIRR_2 is sparse and diagonally dominant. Its pattern is displayed in Figure 4.2. Due to its rather small dimension, its inverse matrix can be exactly computed. Although being dense, the inverse has only a handful of large magnitude entries. Coefficients larger than the threshold $\tau = 10^{-3}$ in $\mathbf{A}^{-1}$ are plotted on the right of Figure 4.2. The sparsity pattern of the adaptive SPAI preconditioner with a tolerance $\epsilon = 0.2$ is displayed on the left of Figure 4.3. The large entries of $\mathbf{A}^{-1}$ are efficiently captured by the adaptive SPAI algorithm. As a consequence, the convergence associated with the adaptive SPAI preconditioner is effectively accelerated with a number of steps slightly larger than for the ILU preconditioner, but much less than with the fixed sparsity pattern of $\mathbf{A}$ (cf Table 4.3). The pattern

obtained from the matrix $\mathbf{A}^2$ is also presented in Figure 4.3. The sparse inverse pattern is less accurately recovered but most of the entries corresponds to large entries of the inverse.

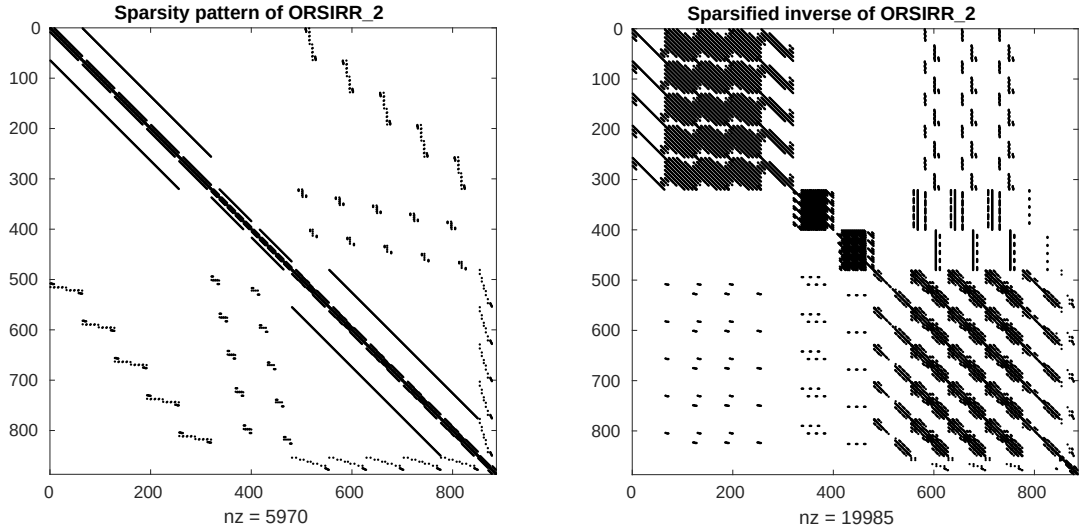


Figure 4.2: Sparsity pattern of ORSIRR_2 (left) and largest entries in the inverse of ORSIRR_2 (right).

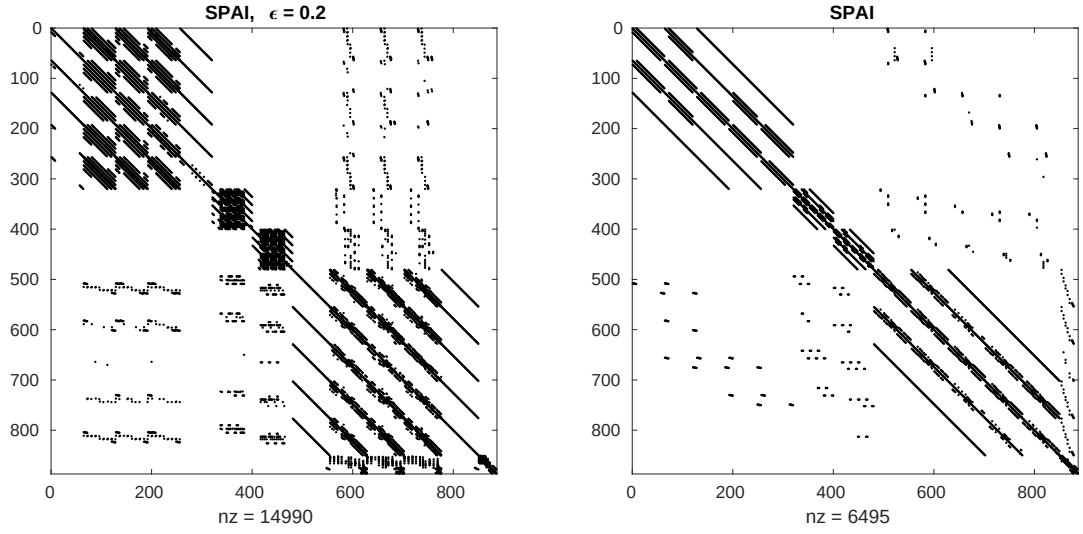


Figure 4.3: Adaptive SPAI with $\epsilon = 0.2$ (left) and fixed SPAI from a sparsified $\mathcal{P}(\mathbf{A}^2)$ (right) for ORSIRR_2.

In addition, an eigenvalue study of the resulting preconditioned systems is conducted. Figure 4.4 shows the impact of the tolerance ϵ on the eigenvalue distribution. A tighter

SPAI tolerance ϵ tends to gather the eigenvalues of the preconditioned system around unity and away from zero, providing better conditioned matrices.

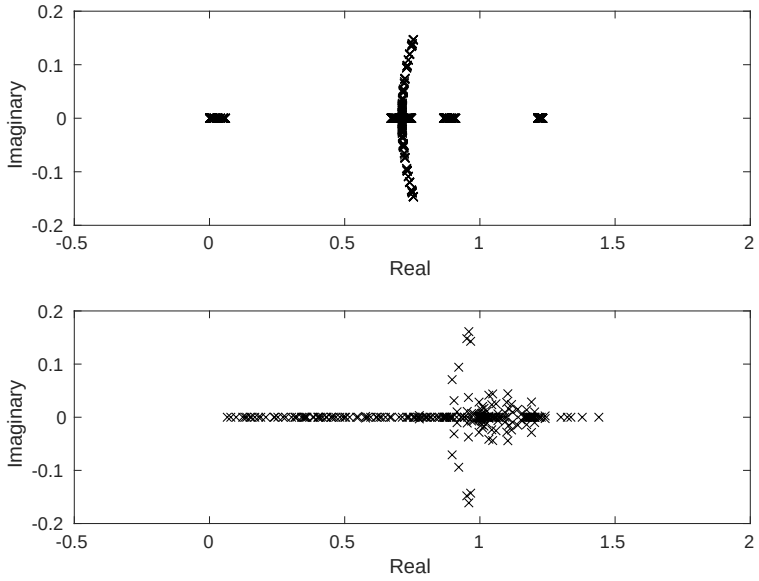


Figure 4.4: Eigenvalue distributions for $\mathbf{M_{SPAI}A}$, computed with a SPAI tolerance $\epsilon = 0.6$ in the upper plot, and $\epsilon = 0.2$ for the bottom distribution.

In most of the numerical experiments, the factorization-based preconditioners show more robust and faster convergences than the sparse approximate preconditioners, at least in terms of iteration count and spectral properties. However, factorization-based preconditioners do not scale efficiently on massively parallel architectures.

Left and Right Preconditioning

As described in Section 2.2, the preconditioner can be applied to the right or left side of the system of linear equations. We noted in Subsection 4.1.4 that the implementation of SPAI is compatible with a left preconditioned Krylov method only. In this regard, the use of the left computed SPAI as a right preconditioner yielded poor conditioning (cf. Table 4.5). In this subsection, the preconditioning side and its influence on the GMRES algorithm are discussed. A measure of the quality of the current approximate solution $\mathbf{x}$ to the solution $\mathbf{x}_*$ is required within the linear solver to decide whether the convergence should proceed. Since we look for the solution of the linear system, the interesting metric is the norm of

the error $\mathbf{e}$, given by $||\mathbf{e}|| = ||\mathbf{x}_* - \mathbf{x}||$. Unfortunately, the error calculation requires the unknown solution $\mathbf{x}_*$ and is thus inaccessible. On the other hand, the residual $\mathbf{r}$ gives an estimate of the extent to which the approximate solution satisfies the linear system with

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}. \tag{4.21}$$

The condition $\mathbf{r} = \mathbf{0}$ is equivalent to $\mathbf{e} = \mathbf{0}$. Nonetheless, a small norm of the residual does not always guarantee that the error is small in norm as well. The inequalities (4.22) from [138] provide relationships between the error, the residual and the condition number of the coefficient matrix $\kappa(\mathbf{A}) = ||\mathbf{A}||_2||\mathbf{A}^{-1}||_2$:

$$\frac{1}{\kappa(\mathbf{A})}\frac{||\mathbf{r}||_2}{||\mathbf{b}||_2} \leq \frac{||\mathbf{e}||_2}{||\mathbf{x}_*||_2} \leq \kappa(\mathbf{A})\frac{||\mathbf{r}||_2}{||\mathbf{b}||_2}. \tag{4.22}$$

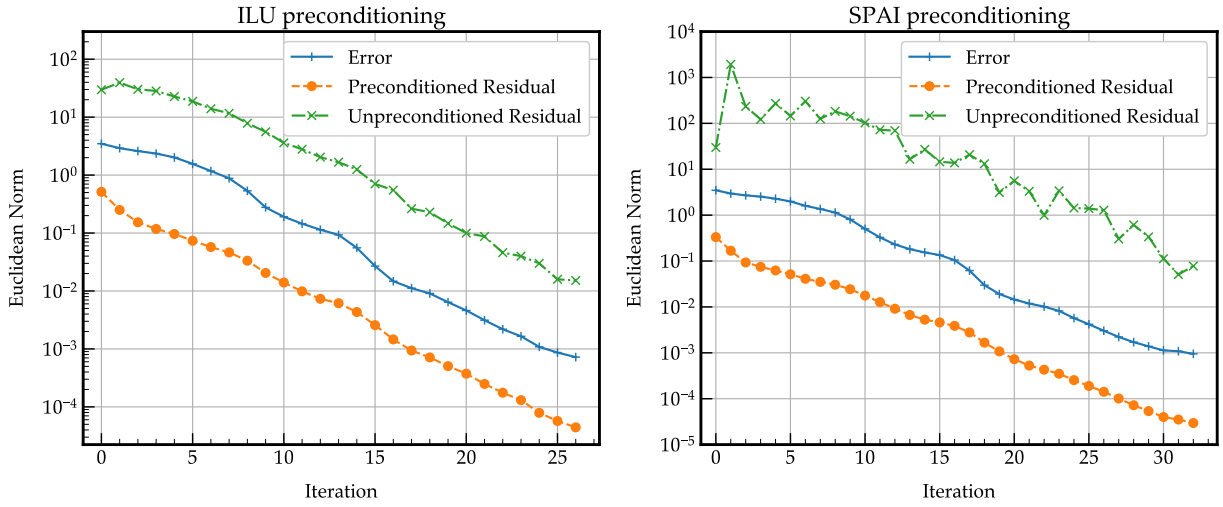
From these inequalities, it follows that a small relative residual $\frac{||\mathbf{r}||_2}{||\mathbf{b}||_2}$ implies a small relative error $\frac{||\mathbf{e}||_2}{||\mathbf{x}_*||_2}$ provided that the coefficient matrix $\mathbf{A}$ is well-conditioned.

In Krylov subspace methods such as GMRES, the residual (4.21) is never explicitly computed. In practice, the residual norm comes as a by-product with the solution of the least-squares problem (2.18). When the linear system is solved with a right preconditioning method, the equivalent system to be solved is $\mathbf{AMy} = \mathbf{b}$ and $\mathbf{x} = \mathbf{My}$, for $\mathbf{x}$. In GMRES, the termination criterion employs the norm of the residual $\mathbf{r} = \mathbf{b} - \mathbf{AM}(\mathbf{M}^{-1}\mathbf{x}) = \mathbf{b} - \mathbf{Ax}$, which is independent of the chosen right preconditioner. Hence, right preconditioning is usually preferred when different preconditioners are compared as explained in [42]. In this way, the stopping criterion is based on the minimization of the same independent quantity $||\mathbf{b} - \mathbf{Ax}||_2$. This quantity is now referred to as the unpreconditioned norm.

On the contrary, left preconditioning alters the residual of the linear system $\mathbf{MAx} = \mathbf{Mb}$. In lieu of the residual $\mathbf{b} - \mathbf{Ax}$, the GMRES termination criterion is naturally calculated with the norm of the preconditioned residual $\mathbf{r_{prec}} = \mathbf{M}(\mathbf{b} - \mathbf{Ax})$, readily available in memory. The computation of the unpreconditioned residual norm $||\mathbf{b} - \mathbf{Ax}||_2$ is practicable but requires an additional explicit computation, which is thus avoided. As a consequence, the quality of the preconditioning matrix influences the termination criterion. In

the case where $\mathbf{M}$ is a sound approximation of $\mathbf{A}^{-1}$, the preconditioned residual becomes a suitable estimation of the error $\mathbf{e}$, as $||\mathbf{M}(\mathbf{b} - \mathbf{A}\mathbf{x})||_2 \approx ||\mathbf{A}^{-1}\mathbf{b} - \mathbf{x}||_2 = ||\mathbf{e}||_2$. Newton-Krylov methods often employ a left preconditioner to use an error estimate of the update vector $\Delta\mathbf{w}$ in the termination criterion [139].

In Figure 4.5, the convergence of the error, the preconditioned norm and the unpreconditioned norm along the convergence of the linear system defined by ORSIRR_2 are plotted. The system is solved with the left preconditioned GMRES(30) using the preconditioners ILU(0) and SPAI respectively for a relative tolerance of $\tau = 10^{-4}$. Therefore the natural norm used in the stopping criterion is the preconditioned residual norm. This output is specific to this case. Yet, we notice that both the preconditioned and unpreconditioned norms are generally related in such a way that a decrease in one of them leads to a decrease of about the same order of magnitude in the other.



(a) ILU(0) left preconditioning.

(b) Adaptive SPAI left preconditioning.

Figure 4.5: Convergence histories of the error, preconditioned residual and unpreconditioned residual norms for the ORSIRR_2 matrix.

4.2 SPAI Preconditioning Strategy in the Newton-Krylov Framework

The solution of nonlinear equations with Newton's method requires successive solutions of several slightly different linear systems of equations at each time step. At the k-th Newton step, the linear system to solve reads

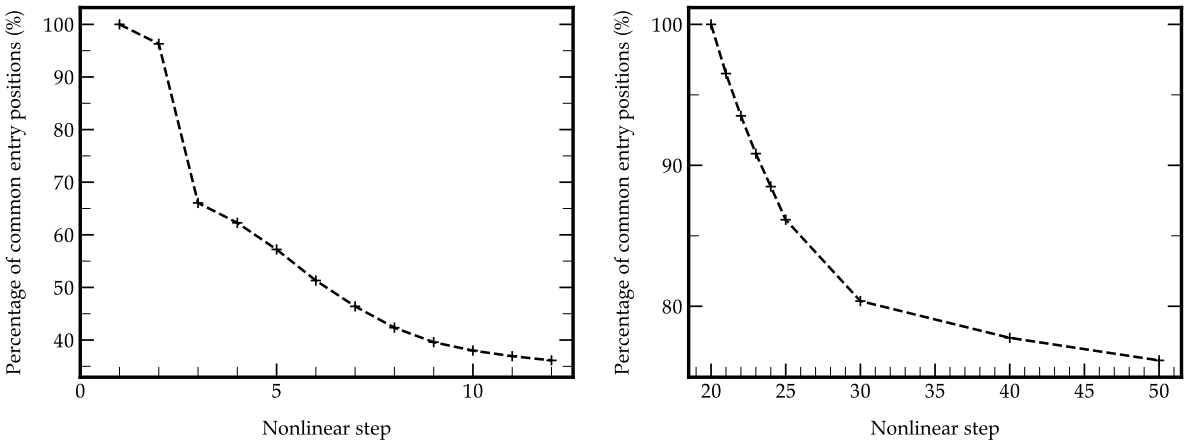
$$\mathbf{A}^{(k)}\mathbf{x} = -\mathbf{R}^{(k)}, \tag{4.23}$$

with $\mathbf{x}$ being the unknown vector, $\mathbf{R}^{(k)}$ the nonlinear residual, and $\mathbf{A}^{(k)}$ the Jacobian matrix. The linear system solver, including the preconditioner computation, is generally the most time-consuming part of a flow solver. To this extent, the need for fast and scalable linear solvers is paramount for efficient implicit CFD solvers. Although the SPAI preconditioner shows a high degree of coarse-grained parallelism, its computation is expensive. Therefore, we seek to damp the cost of the SPAI algorithm for successive systems of linear equations, even if it might come at the expense of the quality of the preconditioner. In this section, we examine ways to optimize the SPAI preconditioner in the Newton-Krylov method.

4.2.1 Reusing the SPAI Pattern

In implicit coupled solvers, the fixed discretization stencil ensures that all flux Jacobians $\mathbf{A}^{(k)}$ possess a fixed block sparsity pattern. Only their entry values vary from one step to another as the approximate solution converges to the steady-state solution. Hence, we can expect moderate changes in the sparsity pattern obtained from the SPAI adaptive algorithm for subsequent Jacobian matrices. Following a remark in [118], we consider the idea of reusing a former SPAI pattern to compute a static SPAI. In this way, the preconditioner is alternatively computed with the more expensive adaptive SPAI algorithm from the diagonal start pattern and the fixed SPAI preconditioner from the last computed SPAI pattern.

To illustrate the relevance of reusing a SPAI pattern for consecutive Newton steps in equation (4.23), let us consider the simulation in FANSC of a transonic flow around a NACA 0012 airfoil with a Mach number of $M = 0.8$. The solution of the Reynolds-Averaged Navier-Stokes equations is carried out with a Newton-GMRES(30) method, preconditioned with a block adaptive SPAI preconditioner. The SPAI matrix is computed with a SPAI tolerance $\epsilon = 0.2$ and a maximum number of 7 adaptive steps, each adding a maximum of 5 new entries. The evolution of the sparsity pattern of the resulting SPAI preconditioners along Newton's method is studied. In Figure 4.6a, we report the percentages of common entry positions of the newly computed SPAI with respect to the SPAI sparsity pattern calculated at the first nonlinear step $k = 1$.



(a) With respect to the first Newton step.

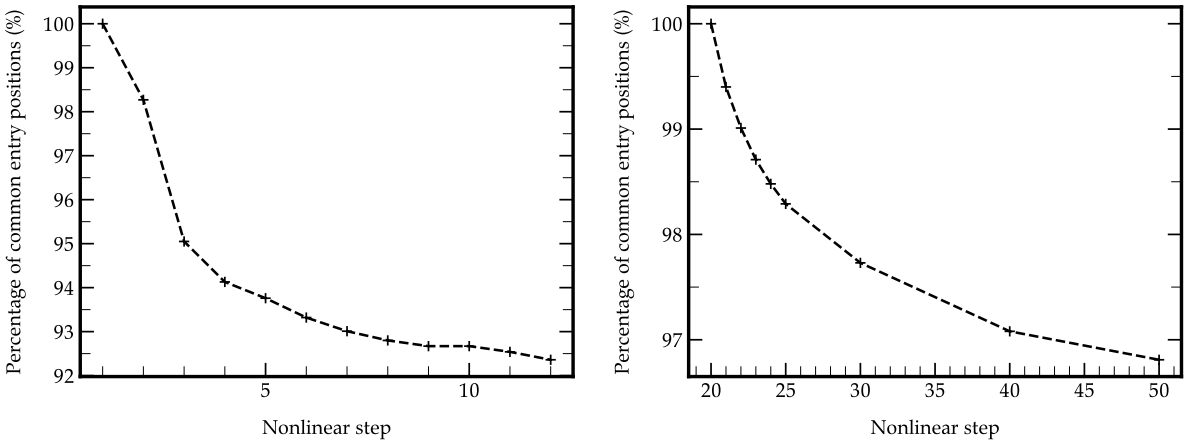
(b) With respect to the 20^{th} Newton step.

Figure 4.6: Percentage of common entry positions with the first SPAI pattern of the successive SPAI preconditioners in the NACA 0012 test case.

It is common to witness two different stages of convergence in a steady-state CFD simulation. In the first nonlinear iterations, the approximate solution is likely to be far from the steady-state solution. Therefore, a highly unsteady first stage is characterized by substantial variations in the coefficient matrix entries and large updates are applied to the approximation $\mathbf{w}$. Thus, the SPAI patterns computed with the adaptive algorithm are divergent after a couple of iterations. Once the convergence is established, the instationary iterations give way to a steady linear convergence towards the solution. In this second phase, the Jacobian matrices evolve more slowly. Therefore, if we compare the SPAI pat-

terns of subsequent iterations later in the convergence, the variation is less stringent. As an example, around 75% of the entry positions in the 50^{th} SPAI matrix are already present in the 21^{st} SPAI matrix (cf. Figure 4.6b). The shapes of the curves tend to suggest that the SPAI sparsity patterns of successive Jacobian matrices have a common base to which are added entries sensitive to small variations in the values of the Jacobian coefficients.

The same study was conducted with the ONERA M6 viscous test case, the results of which are reported in Figure 4.7. The SPAI sparsity patterns are even more consistent in this case where only 3% of the entry coordinates are different between the 21^{st} and the 50^{th} SPAI matrices.



(a) With respect to the first nonlinear step.

(b) With respect to the 20^{th} nonlinear step.

Figure 4.7: Percentage of common entry positions with a given SPAI pattern for the successive SPAI preconditioners in the ONERA M6 test case.

4.2.2 SPAI Strategy in Newton's Method

When the Jacobian matrix does not change significantly from one iteration to another, it might be advantageous to freeze the preconditioner and use the same SPAI preconditioning matrix as stored for several subsequent Newton steps. As explained in [20], Birken et al. failed to design sophisticated criteria to automatically decide whether to recalculate the preconditioner based on the convergence of the linear problems along the Newton steps. They noted that different phases of the solving process lead to different convergence properties for the linear solvers. Therefore, it is not evident whether a higher

number of linear iterations is primarily the consequence of a low-quality frozen preconditioner that needs to be updated. As a matter of fact, the number of linear iterations also depends on the evolution of the CFL number and the forcing term in Newton's method.

The strategy adopted in this work follows a periodic recalculation of the adaptive SPAI preconditioner, determined by a period of nonlinear iterations, as well as a combination with fixed SPAI preconditioners computed from a saved pattern and frozen SPAI preconditioners. Frozen means that the SPAI preconditioner **M** is directly reused as it is stored in memory, without any calculation. The main objective is to study the efficiency of SPAI preconditioning in comparison with other parallel preconditioners such as the domain decomposition block Jacobi and ASM. An example of SPAI computations during Newton's method with this strategy is described in Figure 4.8. The nonlinear iterations are decomposed into a succession of periods, gathering a fixed number of consecutive Newton steps.

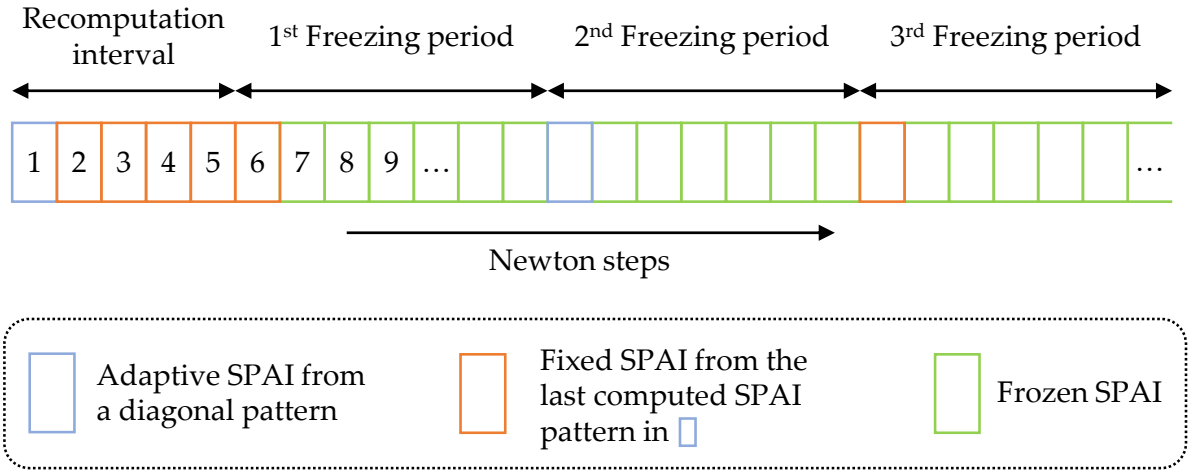


Figure 4.8: SPAI computation strategy during the nonlinear steps of Newton's method.

At the beginning of each of these periods, the SPAI preconditioner is recomputed from scratch with the adaptive SPAI algorithm or from the last known SPAI pattern, performing a static SPAI computation. The fixed SPAI steps are represented with an orange box in Figure 4.8, whereas the adaptive SPAI computations correspond to a blue box. In the remaining iterations within the periods, the SPAI preconditioning matrix is frozen and directly reused in the GMRES process. These steps are represented with a green box. The

ensuing increase in the number of linear iterations should be offset by the time saved from freezing the preconditioner. Since the Jacobian matrix is expected to change rapidly at the beginning of the convergence, we decided to include a first interval in which the SPAI preconditioner is recomputed at every iteration. The first SPAI computation of the first nonlinear iteration is computed from scratch, using the adaptive SPAI algorithm from the diagonal pattern. This step is by far the most time-consuming SPAI computation since all the memory allocations are performed throughout the first SPAI computation. Finally, the SPAI pattern is recomputed from the diagonal pattern after a number of freezing periods, referred to as the *update pattern rate*. In the example in Figure 4.8, the SPAI freezing period is set to 7. In addition, the recomputation interval length is 5 and the update pattern rate is 2.

The SPAI preconditioning strategy within the flow solver is depicted in the flowchart in Figure 4.9. As it is shown in Chapter 5, the optimization of the freezing period, the update pattern rate and the SPAI density parameters is accompanied by a substantial improvement in the SPAI application timings within a Newton-Krylov method.

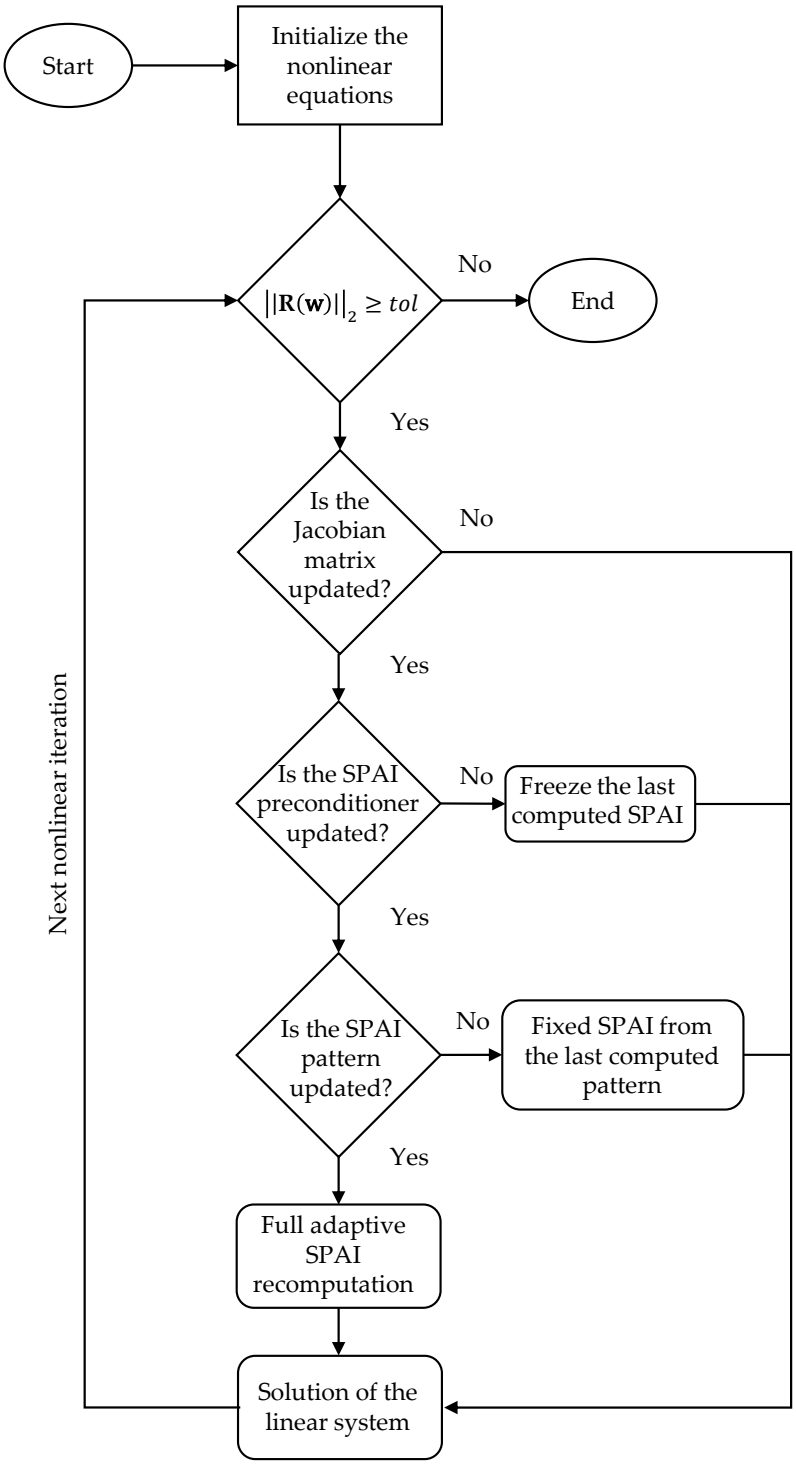


Figure 4.9: SPAI computation strategy in the nonlinear solver.

Chapter 5

Results

This chapter is divided into two parts. First, in Section 5.1, a scalability and performance study of the parallel preconditioning methods is carried out using a highly partitioned system of linear equations from a CFD problem. Different versions of the SPAI preconditioner are tested and convergence rates are compared to the block Jacobi and restricted additive Schwarz preconditioners. Secondly, in Section 5.3, we present the results for the Navier-Stokes equations with FANSC, an in-house flow solver at Bombardier. In the remainder of the thesis, ASM refers exclusively to the restricted additive Schwarz method, and ILU to ILU(0).

We conducted the numerical experiments on general-purpose clusters maintained by Compute Canada. In the first part, the tests were performed on the supercomputer Cedar located at Simon Fraser University [140]. The 768 available nodes contain respectively two Intel Platinum 8160F Skylake CPUs with a total of 48 cores, each with 4 GB of RAM and running at a frequency of 2.4 GHz. Secondly, the numerical simulations on FANSC were carried out on the supercluster Béluga located at the École de technologie supérieure in Montreal, managed by Calcul Québec and Compute Canada [141]. The cluster consists of 872 nodes with two Intel Gold 6148 Skylake processors, each with 20 cores and a clock frequency of 2.4 GHz. The overall memory of the nodes ranges from 92 GB to 186 GB depending on the node. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), Ministère de l'Économie, des Sciences et de l'Innovation du Québec (MESI) and le Fonds de recherche du Québec – Nature et technologies (FRQ-NT).

5.1 Parallel Scaling of SPAI and Domain Decomposition Preconditioners

Before testing the SPAI preconditioner to solve a system of nonlinear equations such as the discretized Navier-Stokes equations, we seek to study the efficiency of preconditioning methods on a highly partitioned system of linear equations from an implicit solver. Unfortunately, test matrices available in the SuiteSparse collection are generally fairly small and rarely representative of the complex matrices arising from large-scale industrial simulations. In addition, they rarely exhibit the typical block structure of Jacobian matrices from the implicit discretization of the flow equations. Besides, they are hardly ever provided with a right-hand side vector.

For this purpose, some matrices were collected during the convergence from the flow solver *Finite Volume Euler and Navier-Stokes* (FVENS) [142]. FVENS is an open-source 2D cell-centered finite volume solver for the compressible Euler and Navier-Stokes equations written in C++. The baseline test case is a NACA 0012 airfoil in a subsonic Mach number regime of $M = 0.5$ [143]. The convective part of the flux is discretized with the Roe upwind scheme, whereas the diffusive flux uses the least-squares approach to compute the gradients. More details on the convective and viscous flux discretization schemes are available in [144] and [88] respectively. A series of three unstructured meshes consisting of triangles have been used to discretize the governing equations. Table 5.1 summarizes the test conditions, the properties of the mesh and the number of nonzero entries in the studied Jacobian matrices, with Re being the Reynolds number and α the angle of attack. The steady-state Mach number contour plot around the airfoil is shown in Figure 5.1.

Table 5.1: Parameters of the test cases used in FVENS.

Case name	Airfoil	M	α	Re	Grid cells	nnz in the Jacobian matrix
visc_1	NACA 0012	0.5	0.0°	5000	52,624	3.62×10^6
visc_2	NACA 0012	0.5	0.0°	5000	210,496	14.50×10^6
visc_3	NACA 0012	0.5	0.0°	5000	841,984	58.03×10^6

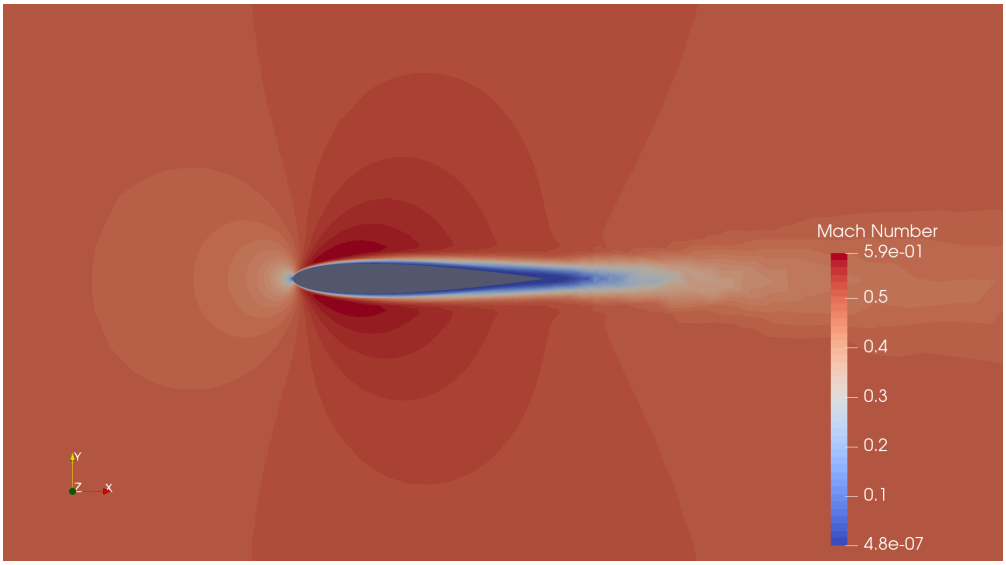


Figure 5.1: Steady-state Mach number contours around the NACA 0012 airfoil.

Since the 2D equations are solved, each cell contains the four discretized variables $\left[\rho_i, (\rho v_x)_i, (\rho v_y)_i, (\rho E)_i\right]$ and the Jacobian matrix $\mathbf{A}$ has a natural block size of 4. The same problem is solved on three meshes of different sizes. This allows us to study the impact of the grid size and the number of subdomains on the preconditioning performances.

The Jacobian matrices are stored in the CSR format in the computer code. Consequently, the matrices are partitioned into as many chunks of rows as there are cores. In compliance with the distributed memory approach, each core has a direct access to its own chunk of rows. This partitioning of the matrices into rows is closely related to the mesh partitioning, since each core is assigned a block. Hence, the number of cores uniquely determines the domain partition as illustrated in Figure 5.2.

According to the number of required cores or blocks, the least possible number of nodes was used to reduce the latency of communications between cores from one node to another, and favor communications within the nodes. This especially benefits ASM and SPAI preconditioners which require inter-core communications within the Krylov application, contrary to the parallel block Jacobi approach. The solutions of the linear systems were carried out with the scientific library PETSc via the left preconditioned GMRES(30) method. As a result, convergence is based on the use of the preconditioned residual. However, the unpreconditioned residual is the most significant metric for comparing the

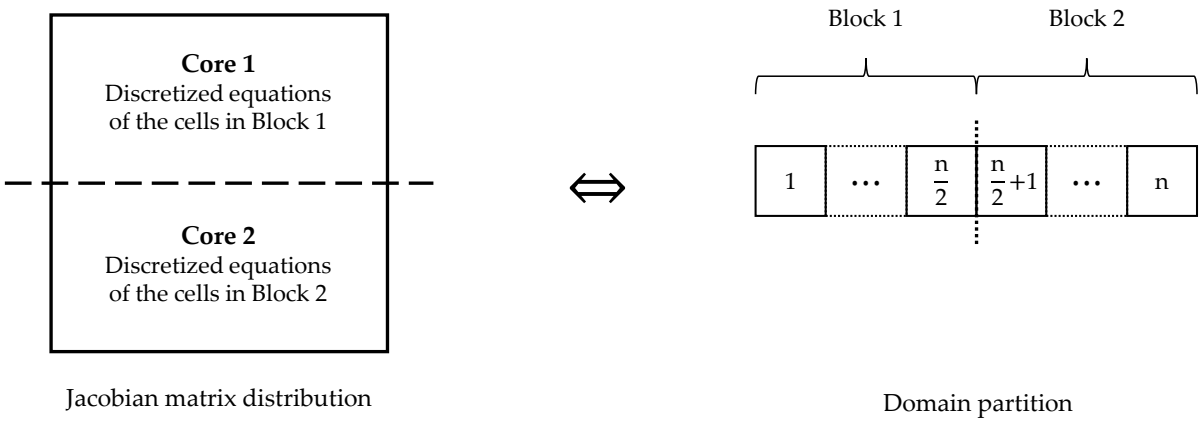


Figure 5.2: Jacobian matrix distribution across the cores (left), and corresponding domain decomposition (right).

efficiency of different preconditioners as noted in Subsection 4.1.5. In this respect, the residual norm $||\mathbf{b} - \mathbf{Ax}||_2$ is explicitly computed at each step. This additional computational time is deducted from the timings, since the unpreconditioned residual is normally not constructed as part of the GMRES process. Thus, all the following residual norms refer to the same metric and can legitimately be compared. Longer restart periods have been considered without leading to real improvements in the convergence.

5.1.1 Comparison of SPAI Preconditioning Variants

We investigate the sensitivity of various parameters on the density of the preconditioning matrix for both scalar and block versions of the SPAI preconditioners. We perform this study for the smaller Jacobian matrix *visc_1*. The block SPAI takes advantage of the block structure of the Jacobian matrix, with a reduced setup cost. The adaptive versions of the preconditioning method use the adaptive algorithm from a diagonal sparsity pattern, or a block diagonal for the block SPAI. The maximum number of improvement steps is set to 5, with a maximum of 5 new indices per step. These parameters are sufficient to act as a safeguard, preventing unreasonable fillings in the SPAI columns, while allowing the SPAI tolerance ϵ to efficiently monitor the density of the preconditioner. As a matter of fact, most of the SPAI column computations are stopped before reaching this upper limit due to the residual condition enforced by the SPAI tolerance ϵ (cf. equation (4.10)).

The fixed version of the SPAI preconditioners calculates the SPAI matrix only from a prescribed sparsity pattern. In our cases, it is realized either with $\mathcal{P}(\mathbf{A})$ or a sparsified version of $\mathcal{P}(\mathbf{A}^2)$, with the same number of nonzero entries as in $\mathbf{A}$. Finally, two hybrid versions combining the fixed and adaptive SPAI algorithms are tested. The starting patterns $\mathcal{P}(\mathbf{A})$ or $\mathcal{P}(\mathbf{A}^2)$ are extended with a few improvement steps according to the processes described in the following list.

- Scalar hybrid 1: The starting SPAI pattern is the sparsified pattern of $\mathbf{A}^2$, which is then improved with three steps of the adaptive algorithm, where each adds a maximum of 5 new entries with a SPAI tolerance $\epsilon = 0.2$.
- Block hybrid 2: The pattern of $\mathbf{A}$ is extended with one additional adaptive step adding a maximum of 5 new blocks. The SPAI tolerance ϵ is set to 0.4.

Table 5.2 summarizes the densities and computational times of the resulting SPAI preconditioners. It is important to note that we only measure the total time taken by the SPAI computation and the linear solver convergence. The CPU runtimes are reported for a tolerance drop of four orders of magnitude $\tau_{\text{lin}} = 10^{-4}$ in the unpreconditioned residual. For clarity, Figure 5.3 shows the convergence history of the different preconditioned GMRES solves from Table 5.2.

Table 5.2: Results for SPAI preconditioning for *visc_1*. The runtimes are recorded for 8 cores and a linear tolerance $\tau_{\text{lin}} = 10^{-4}$.

SPAI Parameters	nnz	SPAI Computational Time (s)	Convergence Time (s)	Total Time (s)
Block Adaptive SPAI $\epsilon = 0.4$	7.30×10^6	0.79	16.63	18.34
Block Fixed SPAI from $\mathcal{P}(\mathbf{A})$	3.62×10^6	0.14	Stalls	Stalls
Scalar Adaptive SPAI $\epsilon = 0.4$	3.55×10^6	11.52	Stalls	Stalls
Scalar Fixed SPAI from $\mathcal{P}(\mathbf{A})$	3.62×10^6	0.44	Stalls	Stalls
Scalar Fixed SPAI from sparsified $\mathcal{P}(\mathbf{A}^2)$	3.62×10^6	0.79	Stalls	Stalls
Scalar Adaptive SPAI $\epsilon = 0.2$	8.03×10^6	34.44	11.74	46.37
Scalar Adaptive SPAI $\epsilon = 0.1$	11.19×10^6	77.56	8.19	85.99
Scalar Hybrid 1 SPAI	6.69×10^6	20.28	7.92	28.70
Block Hybrid 2 SPAI	7.45×10^6	20.28	16.51	18.12

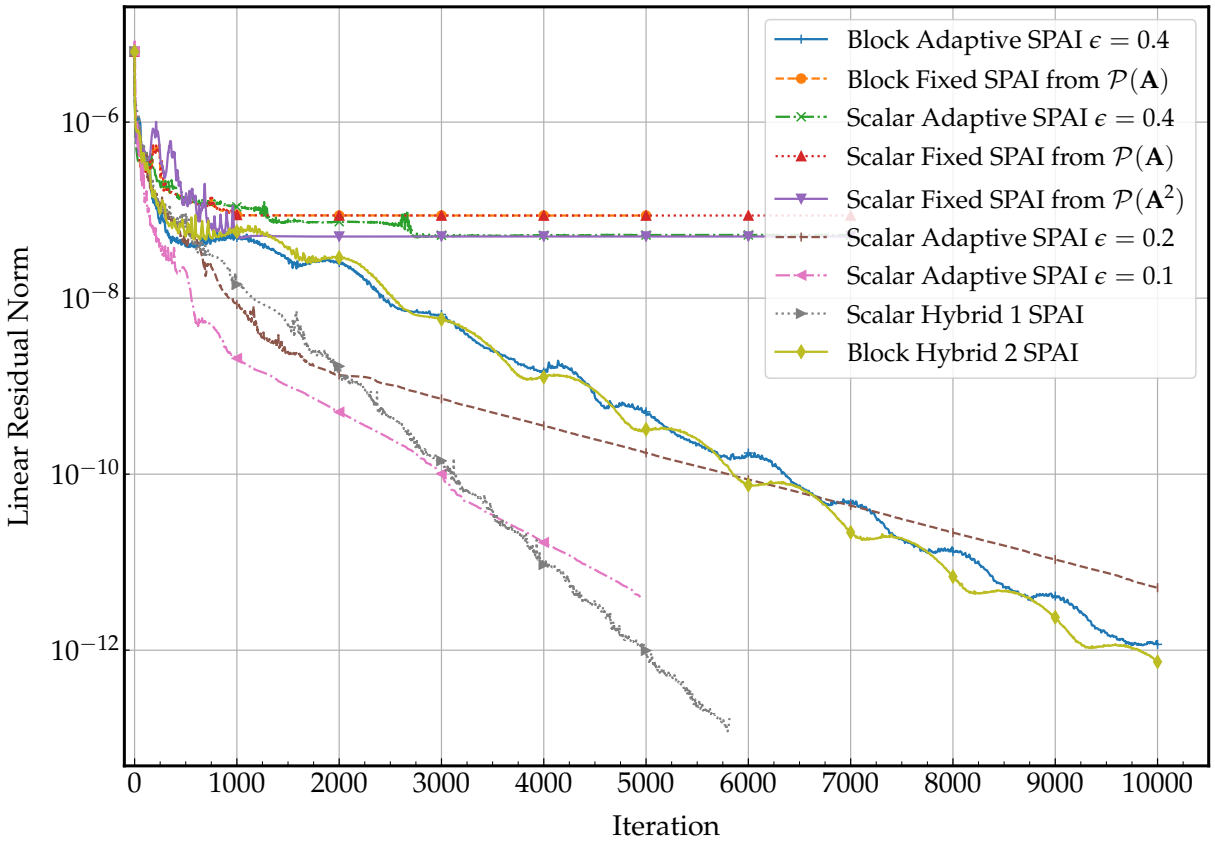


Figure 5.3: Convergence histories with different SPAI settings for *visc_1*.

From the different tests, we can see that the GMRES processes stall below a certain number of entries in the SPAI preconditioner. The block SPAI entails a denser preconditioner than the scalar algorithm in the same number of steps. Every time an index is added in the adaptive block algorithm, a 4×4 nonzero block is added to the pattern of the SPAI matrix. Hence, the block SPAI quickly fills in the sparsity pattern of the preconditioning matrix. In contrast, the patterns of $\mathbf{A}$ and $\mathbf{A}^2$ are too sparse to yield preconditioners of high quality without improving them with adaptive steps. As a consequence, the convergence rapidly stalls for $\mathbf{M}$ resulting from the fixed SPAI based on $\mathcal{P}(\mathbf{A})$ and $\mathcal{P}(\mathbf{A}^2)$, whereas they ensured convergence for the benchmark matrices from the SuiteSparse collection in Subsection 4.1.5.

The block adaptive SPAI with a tolerance $\epsilon = 0.4$ produces a preconditioner with 7.30×10^6 nonzero entries which is sufficient to guarantee a decrease of 6 orders of magnitude in the residual norm. To attain about the same number of nonzero entries, the

scalar version requires a very low tolerance of $\epsilon = 0.2$. However, the computation of the SPAI is prohibitively expensive in this case, taking more than half a minute to perform. The GMRES method preconditioned with the block hybrid 2 SPAI has a faster convergence rate than the adaptive preconditioners, while producing a less dense SPAI matrix $\mathbf{M}$. The convergence histories in terms of CPU execution time are shown in Figure 5.4. The plots are offset to start once the SPAI computations are achieved. As a consequence, none of the curves actually start at the origin. The adaptive scalar computations are considerably more expensive compared to the block SPAI computations for the same number of nonzero entries in the SPAI matrix.

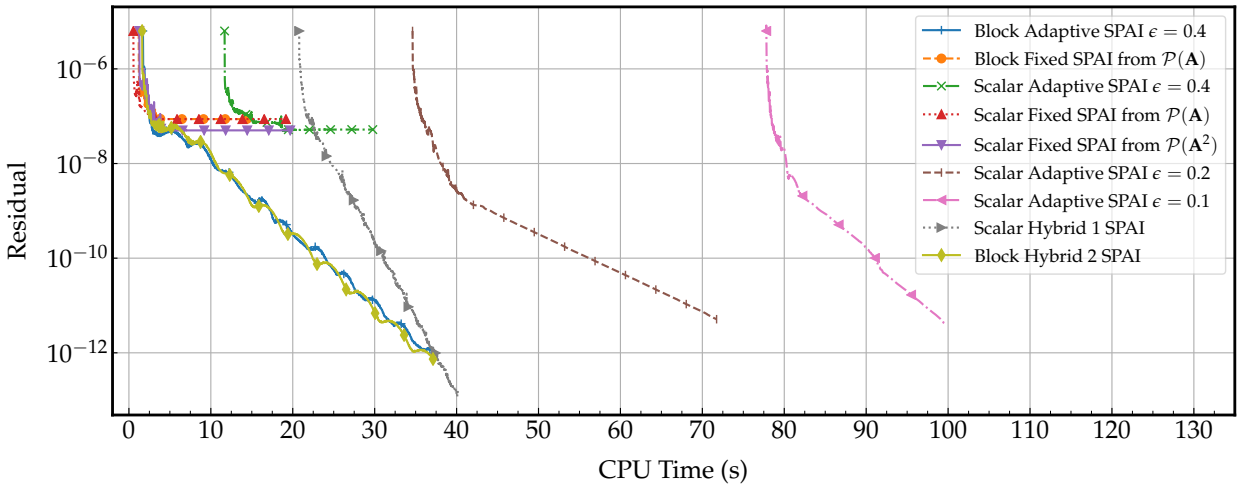


Figure 5.4: Residual norm as a function of the CPU runtime for different SPAI settings for *visc_1* with 8 cores. The beginnings of the convergences were shifted to account for the SPAI computational time.

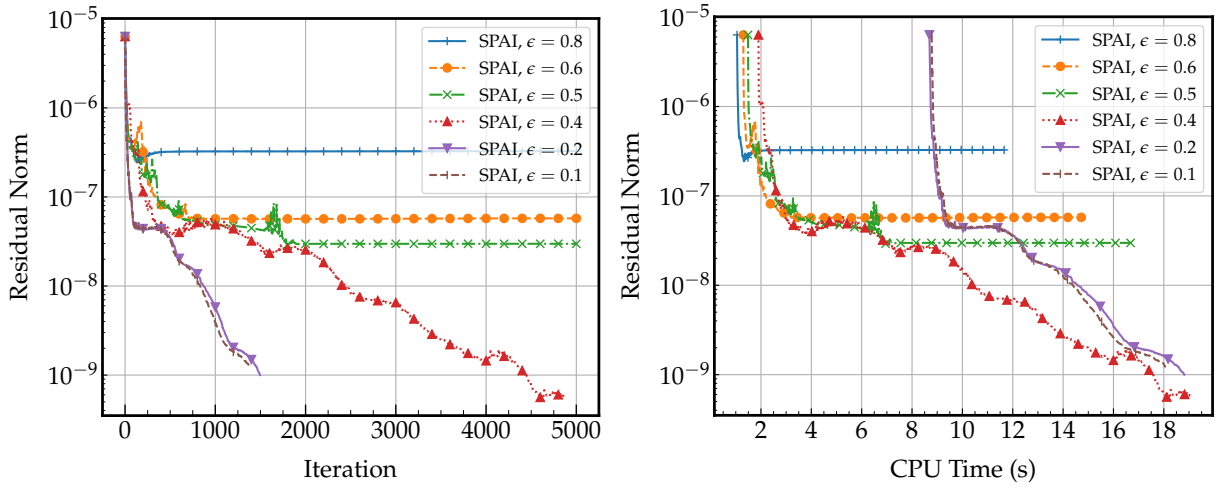
5.1.2 Study of the Adaptive SPAI Preconditioning

It is important to ensure the convergence of the linear systems at each step of Newton's method. The adaptive SPAI is the most flexible SPAI preconditioner. Indeed, the preconditioner density can easily be increased by tightening the tolerance ϵ if the preconditioner is not effective in converging the linear systems. On the other hand, the performance of the fixed preconditioners seems to be more problem dependent, with mixed results for different problems. The performance of the block adaptive SPAI is studied with differ-

ent density settings, for a tolerance ϵ ranging from 0.1 to 0.8. The results are reported in Table 5.3 for 8 cores.

Table 5.3: Results for the block adaptive SPAI preconditioning with a forcing term $\tau_{\text{lin}} = 10^{-3}$ for *visc_1* with 8 cores.

Tolerance ϵ	nnz	SPAI Computational Time (s)	Convergence Time (s)	Total Time (s)
0.8	0.84×10^6	0.05	Stalls	Stalls
0.6	3.61×10^6	0.26	Stalls	Stalls
0.5	4.64×10^6	0.41	Stalls	Stalls
0.4	7.30×10^6	0.76	10.61	12.51
0.2	19.64×10^6	7.44	6.68	15.36
0.1	19.77×10^6	7.53	6.17	14.95



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of CPU time.

Figure 5.5: Study of the adaptive block SPAI preconditioners with different tolerances ϵ for *visc_1*.

The results in Table 5.3 tend to show that the lower the tolerance ϵ, the closer the SPAI preconditioner $\mathbf{M}$ is to $\mathbf{A}^{-1}$. A more accurate preconditioner is expected to reduce the number of iterations in the linear solver, also referred to as linear iterations. Yet, each iteration becomes increasingly more expensive due to the increased density of the SPAI preconditioner, since the application cost in the matrix-vector multiplication grows with the number of nonzeros in the preconditioner $\mathbf{M}$. Regardless of the tolerance value, all

the preconditioners efficiently reduce the residual norm by at least one order of magnitude after a few dozen iterations. However, the linear solver convergence quickly stalls for SPAI preconditioning with a tolerance of $\epsilon \geq 0.5$. Hence, the preconditioning method is very sensitive to the tolerance parameter. Even if small tolerances lead to convergence with a reduced number of linear iterations, their increased computational and application costs often outweigh the gain due to the reduction in the number of iterations. For example, the reduction of four orders of magnitude in the preconditioned residual is attained in around 1500 iterations with a tolerance $\epsilon = 0.1$, whereas about 5000 iterations are required with a tolerance $\epsilon = 0.4$. However, their CPU timing is approximately the same to meet this tolerance (cf. Figure 5.5b).

5.1.3 Comparison with Domain Decomposition Preconditioners

In this subsection, we compare the SPAI preconditioner to domain decomposition preconditioners, namely the block Jacobi and restricted additive Schwarz (ASM) preconditioners. The block SOR and block ILU(0) respectively denote the block Jacobi preconditioner where the local submatrices $\mathbf{B}_i$ are simply the SOR and ILU factorizations of the restrictions of $\mathbf{A}$ on the local subdomains. Their application consists of a single application of the local SOR and ILU matrices. The block Jacobi-SOR and block Jacobi-ILU(0) methods refer to a block Jacobi preconditioner with local GMRES(30) solvers preconditioned with SOR and ILU respectively. In every iteration, the local subsystems are solved down to a relative tolerance of $\tau_{loc} = 10^{-2}$. These preconditioners are expected to be more robust than the mere applications of the local SOR and ILU matrices. Finally, ASM-ILU(0) denotes the restricted ASM preconditioning approach combined with local preconditioned GMRES(30)-ILU(0) subsolvers. The width of overlap between local subdomains is set to a single layer $\delta = 1$. As with the block Jacobi preconditioner, the relative tolerance of the local systems is $\tau_{loc} = 10^{-2}$. The overlap increases the stability of the preconditioning methods but additional expensive inter-core communications are then required.

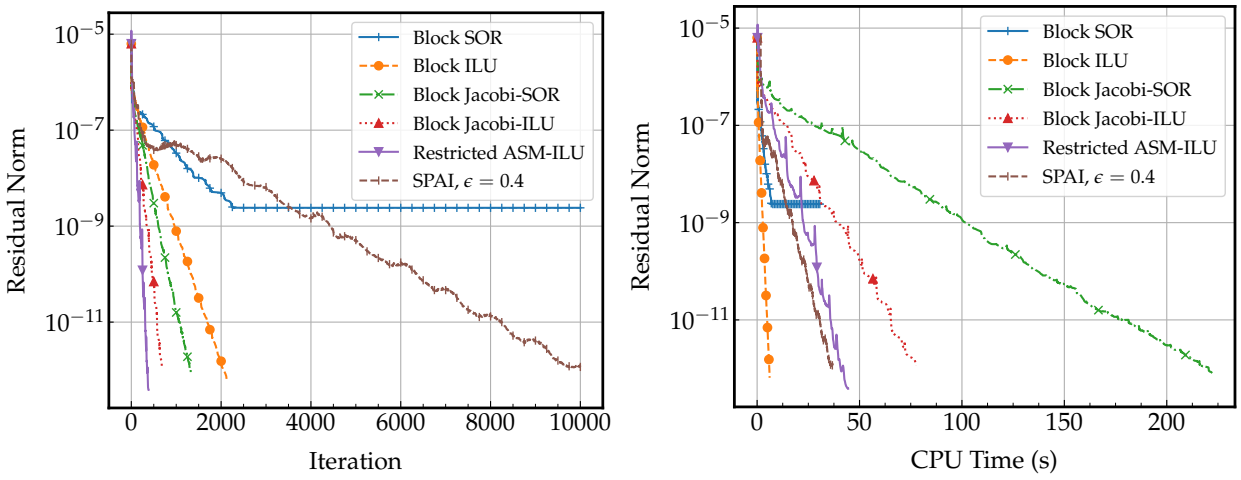
Convergence of the Domain Decomposition and SPAI Preconditioners

The results of the domain decomposition preconditioners as well as those of the adaptive block SPAI are reported in Table 5.4 for 8 cores, 46 cores and 208 cores respectively. The number of iterations and CPU timings are reported for a drop of four orders of magnitude in the unpreconditioned residual norm. The corresponding convergences for 8 cores and 208 cores are shown in Figure 5.6 and Figure 5.7. As explained in Subsection 2.2.3, larger subdomains provide for more robust block Jacobi and ASM preconditioners with denser preconditioners. Unlike domain decomposition preconditioning, the rate of convergence of linear solvers preconditioned with SPAI does not depend on the distribution of the matrix across processors.

Table 5.4: Results for the problem *visc_1* with a relative tolerance $\tau_{\text{lin}} = 10^{-4}$.

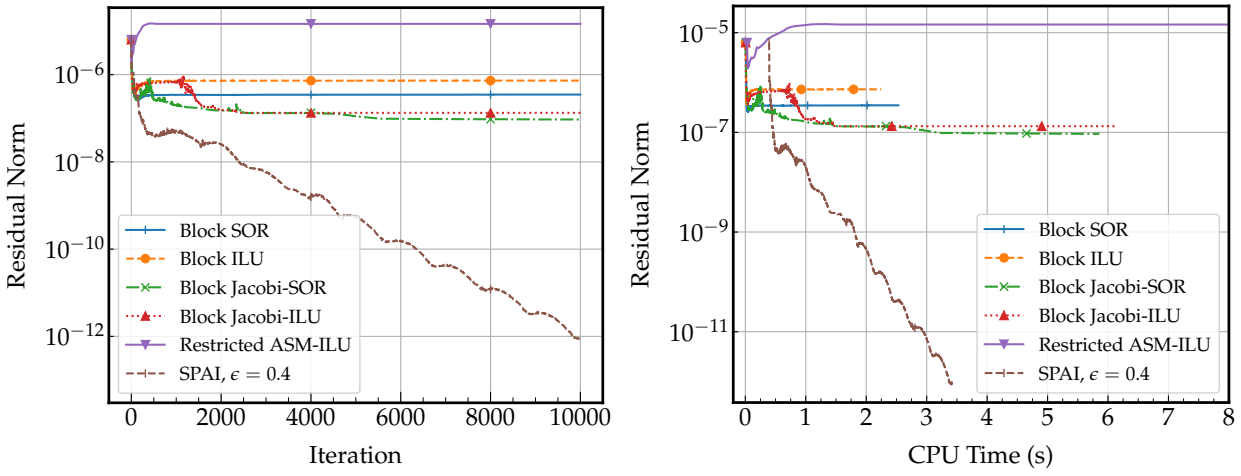
Partitioning	Preconditioning	Number of iterations	CPU Runtime (s)
8 cores 6578 cells per core	Block SOR	Stalls	Stalls
	Block ILU(0)	1041	3.00
	Block Jacobi-SOR	653	109.41
	Block Jacobi-ILU(0)	383	43.12
	Restricted ASM-ILU(0)	242	28.27
	Block Adaptive SPAI, $\epsilon = 0.4$	4660	18.05
46 cores 1144 cells per core	Block SOR	Stalls	Stalls
	Block ILU(0)	Stalls	Stalls
	Block Jacobi-SOR	2816	29.43
	Block Jacobi-ILU(0)	Stalls	Stalls
	Restricted ASM-ILU(0)	1442	20.00
	Block Adaptive SPAI, $\epsilon = 0.4$	4617	6.22
208 cores 253 cells per core	Block SOR	Stalls	Stalls
	Block ILU(0)	Stalls	Stalls
	Block Jacobi-SOR	Stalls	Stalls
	Block Jacobi-ILU(0)	Stalls	Stalls
	Restricted ASM-ILU(0)	Stalls	Stalls
	Block Adaptive SPAI, $\epsilon = 0.4$	4596	1.85

The block SOR preconditioning is not robust enough to guarantee the convergence of the GMRES algorithm with a partitioning into 8 subdomains. The convergence stalls after a reduction of three orders of magnitude. On the contrary, other domain decompo-



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of CPU time.

Figure 5.6: Convergence results for *visc_1* on 8 cores.



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of time.

Figure 5.7: Convergence results for *visc_1* on 208 blocks.

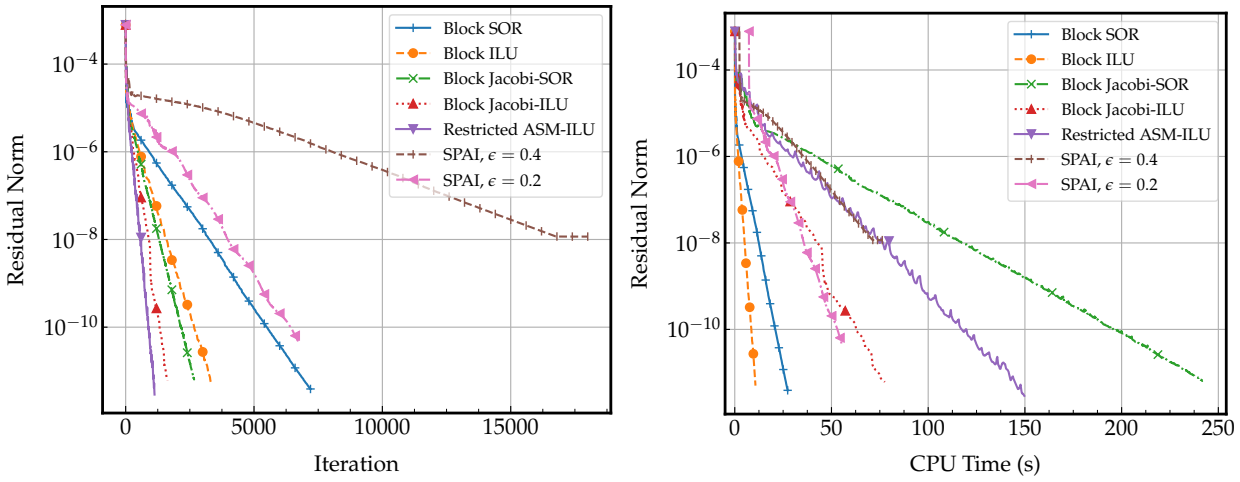
sition preconditioners reach the prescribed relative tolerance within the allowed number of linear iterations. The efficiency of parallel domain decomposition preconditioners declines rapidly as the partitioning is increased to 46 and then 208 cores. In the latter case, the only preconditioner ensuring the convergence of the GMRES method is the adaptive SPAI preconditioner. The scalability of the ASM-ILU(0) preconditioner reaches its limit, as subdomains contain too few cells to entail sufficiently robust preconditioners. In contrast, SPAI preconditioning generally requires more linear iterations than domain decomposition preconditioners to converge. The same SPAI preconditioning matrix $\mathbf{M}$ is computed in each partitioning and SPAI yields the same convergence in all the cases. The minor differences in the number of linear iterations for the SPAI preconditioner with the different number of cores are attributed to the roundoff errors.

We next compare the preconditioners for the larger matrix *visc_2*, which is obtained from a mesh containing four times as many cells as with *visc_1*. The results of the different preconditioning methods are gathered in Table 5.5 for 46 and 208 cores respectively. In addition, the convergences are shown in Figure 5.8 and Figure 5.9.

Table 5.5: Results for the problem *visc_2* for a relative tolerance $\tau_{\text{lin}} = 10^{-4}$.

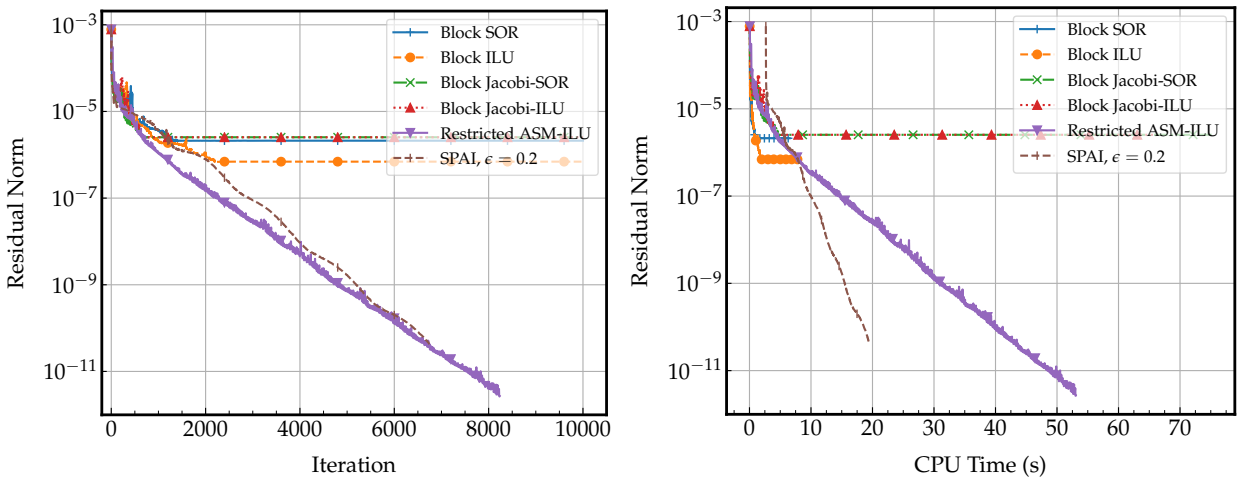
Partitioning	Preconditioning	Number of iterations	CPU Runtime (s)
46 cores 4576 cells per core	Block SOR	2228	8.54
	Block ILU(0)	1094	3.53
	Block Jacobi-SOR	938	85.14
	Block Jacobi-ILU(0)	628	29.93
	Restricted ASM-ILU(0)	426	56.47
	Block Adaptive SPAI, $\epsilon = 0.4$	12,985	55.74
	Block Adaptive SPAI, $\epsilon = 0.2$	3110	29.51
208 cores 1012 cells per core	Block SOR	Stalls	Stalls
	Block ILU(0)	Stalls	Stalls
	Block Jacobi-SOR	Stalls	Stalls
	Block Jacobi-ILU(0)	Stalls	Stalls
	Restricted ASM-ILU(0)	2491	16.09
	Block Adaptive SPAI, $\epsilon = 0.4$	13,050	18.21
	Block Adaptive SPAI, $\epsilon = 0.2$	3108	10.41

As the matrix size is larger, each individual core contains sufficient cells to yield robust convergences even with 46 subdomains, whereas most of them were stalling for the



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of time.

Figure 5.8: Convergence results for *visc_2* on 46 blocks.



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of time.

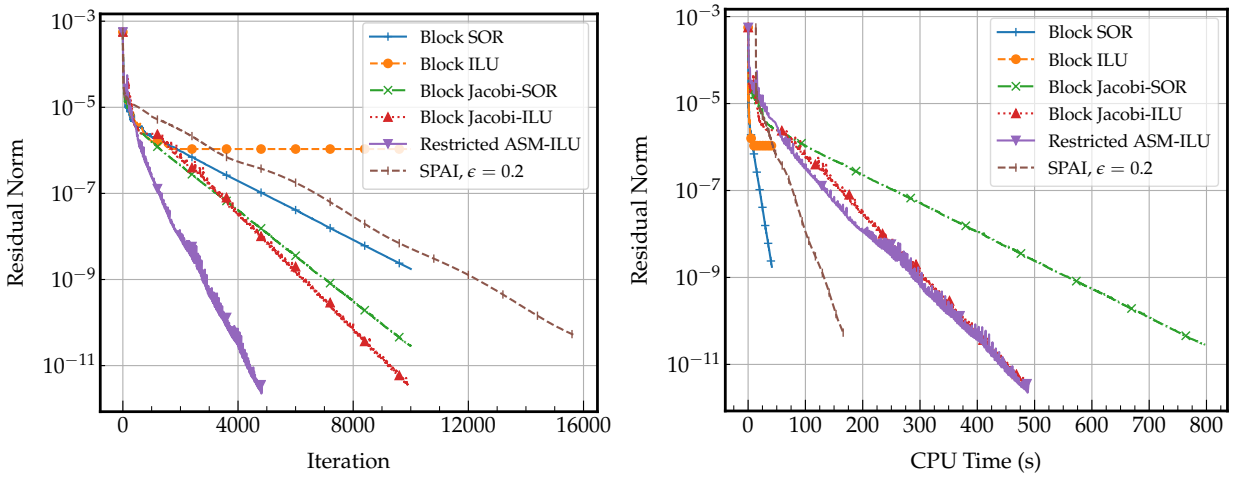
Figure 5.9: Convergence results for *visc_2* on 208 blocks.

same number of cores with *visc_1*. The SPAI tolerance ϵ of 0.4 is no longer enough to sufficiently reduce the residual. A tolerance of $\epsilon = 0.3$ seems more appropriate. For 46 subdomains, the GMRES method preconditioned with the adaptive SPAI requires more iterations than the other domain decomposition preconditioners. Yet, in terms of CPU runtime, the SPAI outshines all block Jacobi and ASM preconditioners. The block SOR and block ILU with their low application costs are competitive against SPAI. Both block SOR and ILU are known to be very sensitive to the increase in the number of nodes. Hence, for a relatively low block partitioning, they can hardly be outperformed by more complex domain decomposition preconditioning, such as restricted ASM-ILU(0), or the parallel SPAI preconditioner. With a partitioning into 208 subdomains, the SPAI preconditioner shows to be an efficient alternative for highly divided meshes. In contrast, the domain decomposition approaches can hardly reduce the residual for more than a few orders of magnitude (cf. Figure 5.9).

Finally, let us consider the larger mesh with 841,984 cells and the Jacobian matrix *visc_3*. The convergence timings for 208 and 506 cores respectively are listed in Table 5.6 for a relative tolerance $\tau_{\text{lin}} = 10^{-4}$. The unpreconditioned residual norms as a function of the iteration count and CPU time are plotted in Figure 5.10 and Figure 5.11.

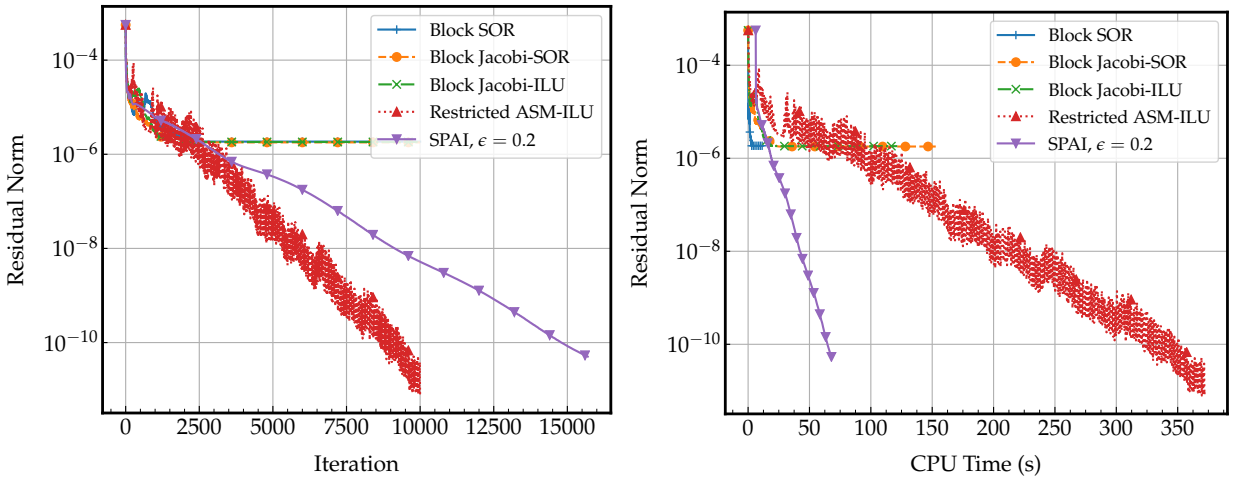
Table 5.6: Results for the problem *visc_3* for a relative tolerance $\tau_{\text{lin}} = 10^{-4}$.

Partitioning	Preconditioning	Number of iterations	CPU Runtime (s)
208 cores 4048 cells per core	Block SOR	5603	23.36
	Block ILU(0)	Stalls	Stalls
	Block Jacobi-SOR	3726	294.34
	Block Jacobi-ILU(0)	3633	178.04
	Restricted SM-ILU(0)	1412	150.44
	Block Adaptive SPAI, $\epsilon = 0.2$	7328	83.81
506 cores 1664 cells per core	Block SOR	Stalls	Stalls
	Block ILU(0)	Stalls	Stalls
	Block Jacobi-SOR	Stalls	Stalls
	Block Jacobi-ILU(0)	Stalls	Stalls
	Restricted ASM-ILU(0)	5071	185.70
	Block Adaptive SPAI, $\epsilon = 0.2$	7310	35.14



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of time.

Figure 5.10: Convergence results for *visc_3* on 208 blocks.



(a) Convergence history in terms of iterations. **(b)** Convergence history in terms of time.

Figure 5.11: Convergence results for *visc_3* on 506 blocks.

At first, the domain is decomposed into 208 equivalent subdomains. For such a large matrix, a partitioning of the computational domain into 208 blocks is not problematic, since there are more cells to feed the cores. The block SOR preconditioner works well in this configuration and again cannot be outperformed by any other preconditioner in terms of CPU runtime, even though more linear iterations are needed. Lastly, this system of linear equations is tested with a high partitioning of 506 subdomains. The number of cells per block is now only 1664. Restricted ASM-ILU(0) is the only domain decomposition preconditioner allowing the GMRES solver to achieve convergence. Yet, in this case, the convergence is hardly stable with spurious oscillations, while the SPAI preconditioned method converges rapidly.

Conclusion

We can draw a few conclusions from the test cases presented. As the matrix dimension increases, domain decomposition preconditioners become more robust for a given number of cores and subdomains. These preconditioners are more sensitive to the number of cells per block than the actual size of the Jacobian matrix, and tend to perform better with larger subdomains. On the other hand, when a given problem is solved with an increasing number of cores, domain decomposition preconditioning methods quickly reach their scalability limit. Robustness could be increased with additional overlapping layers for the restricted ASM preconditioner, but the communication overhead would increase significantly.

As for the sparse approximate inverse, a larger Jacobian matrix requires denser SPAI matrices to provide effective preconditioning. The density parameters suitable for small matrices work poorly for larger grids. The most salient examples are the fixed versions of SPAI, which worked successfully for the small matrices in the SuiteSparse collection but are inoperable for the matrices presented in this subsection. The major drawback of the SPAI preconditioner is the difficulty of finding proper parameters for a given coefficient matrix. Yet, unlike domain decomposition preconditioners, SPAI is not dependent on the number of subdomains in the matrix, and can succeed with dense preconditioners when block Jacobi and ASM preconditioning methods fail because of their lack of scalability.

Finally, it is important to note that the linear systems arising from the quasi-Newton method in the CFD solvers are rarely solved for more than a few orders of magnitude. Thus, even if the preconditioning method results in a stalling convergence, the reduction in the residual norm may still be sufficient to satisfy the prescribed relative tolerance, ensuring an overall convergence of the nonlinear problem.

5.1.4 Strong and Weak Scaling

In this section, we present both the strong and weak scaling performances of the SPAI preconditioner. These two studies provide a measure of the parallel performance of an algorithm. As a rule, a given algorithm is said to be scalable if its efficiency does not degrade with the number of cores [145]. For completeness, we compare the adaptive SPAI preconditioning with the restricted ASM-ILU(0) method, which has proven to be the most robust domain decomposition preconditioner in the preceding tests.

Strong Scaling

In strong scaling studies, the same problem is run for an increasing number of cores. Consequently, the workload per core gradually decreases. Ideally, the scaling is studied in comparison with a serial case run. However, it was not possible to test the SPAI preconditioning on a single core for the larger case due to memory restrictions. Therefore, we use 64 cores as the baseline run to measure the strong scalability. The parallel speedup is defined as

$$Sp(n_p) = \frac{t_{n_0}(\text{dofs})}{t_{n_p}(\text{dofs})}, \tag{5.1}$$

with t_{n_p} the time taken to execute the algorithm on n_p cores for a fixed size problem, namely a fixed number of degrees of freedom (dofs) [146]. Besides, the reference number of cores is taken as n_0. Similarly, the strong scalability efficiency is given by

$$E_{\text{Strong Scalability}} = Sp(n_p) \times \frac{n_0}{n_p}. \tag{5.2}$$

The optimal speedup is $Sp(n_p) = n_p/n_0$, which denotes that the execution on n_p cores is exactly n_p/n_0 times faster than on n_0 cores, whereas the ideal efficiency is 1. The strong scalability plots for the Jacobian matrix *visc_3* are given in Figure 5.12.

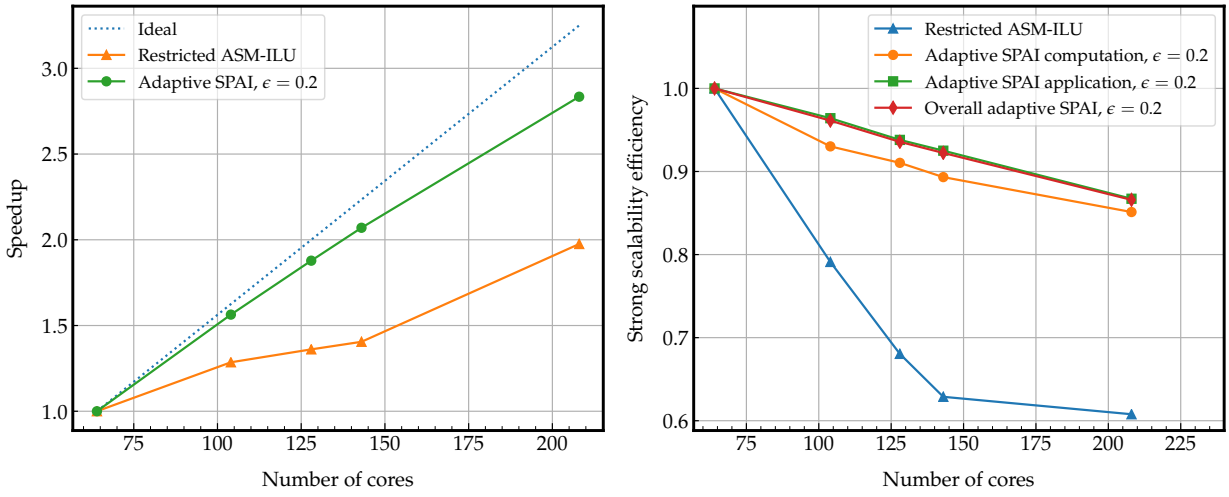


Figure 5.12: Speedup with an increasing number of cores (on the left) and strong scaling efficiency (on the right) for *visc_3*.

In Figure 5.12, "Adaptive SPAI computation" refers to the computational time of the SPAI preconditioner only, while "Adaptive SPAI application" refers to the time spent within the GMRES process, including the application of the preconditioner. SPAI preconditioning features a better overall strong scaling than the ASM-ILU(0) preconditioner. This is due to the independence of SPAI preconditioning from the number of cores. Thus, the decrease in efficiency is essentially due to the increase in the number of inter-core communications. The SPAI applications in the Krylov method require the communication of the vector entries, whereas in the SPAI computation, block columns of $\mathbf{A}$ are exchanged. On the contrary, the ASM-ILU(0) applications are less expensive for each iteration as the subdomains become increasingly smaller. Nonetheless, their overall efficiency decreases considerably due to the growing number of linear iterations required.

We note that the application portion of the SPAI preconditioner has a better strong scaling. This is probably due to the fact that a larger number of long communications is required throughout the SPAI computations. In contrast, the matrix-vector multiplications and dot products in the GMRES process respectively require easily parallelized

operations such as the scattering of vector entries and some reduction operations [147–149].

Weak Scaling

In practice, the increase in computing power is usually used to increase the amount of data to be processed, for example with more accurate simulations and larger grids in CFD applications. To this extent, the weak scalability efficiency measures the performance of an algorithm when both the number of cores and the size of the problem increase at the same rate. In this way, the problem size per parallel core is kept fixed for an increasingly larger problem [150]. The weak scalability efficiency is defined as

$$E_{\text{Weak Scalability}} = \frac{t_{n_0}(\text{dofs})}{t_{n_p}(n_p/n_0 \cdot \text{dofs})}. \tag{5.3}$$

The study of the weak scaling for the parallel preconditioning methods is conducted solving the discretized equations on the three meshes of increasing size presented in Table 5.1, for 8, 32 and 128 cores respectively. In this manner 6578 cells are assigned to each core in the three different runs.

The SPAI preconditioner is used in its adaptive version with a fixed tolerance of $\epsilon = 0.2$. Even though the SPAI preconditioner is computed with the same density parameters, the resulting preconditioning matrices exhibit different densities (cf. Table 5.7). Consequently, the number of linear iterations to achieve the same decrease in the residual norm deteriorates as the size of the overall problem increases. Thus, the efficiency of both the ASM-ILU(0) and SPAI application for the weak scalability is shown per iteration in Figure 5.13. In contrast, the workload per core is approximately the same along the SPAI computations as the same number of columns is assigned to each core in all test cases.

Table 5.7: Properties of the block adaptive SPAI preconditioners for the matrices *visc*.

Case	SPAI Tolerance ϵ	nnz	Density	Iteration count
visc_1	0.2	19.64×10^6	4.4×10^{-4}	1571
visc_2	0.2	79.51×10^6	1.1×10^{-4}	3108
visc_3	0.2	319.73×10^6	2.8×10^{-5}	7311

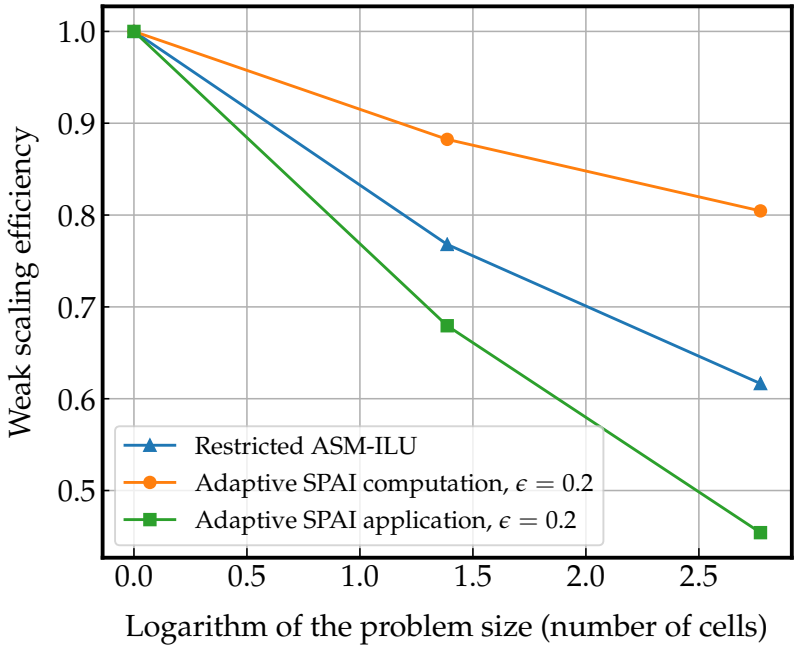


Figure 5.13: Weak scalability efficiencies of the block adaptive SPAI and ASM-ILU(0) preconditioners for the matrices *visc*.

5.2 Test Cases for the Euler and Navier-Stokes Equations

Different test cases were used to simulate different flow types and optimize the parameters of the preconditioners.

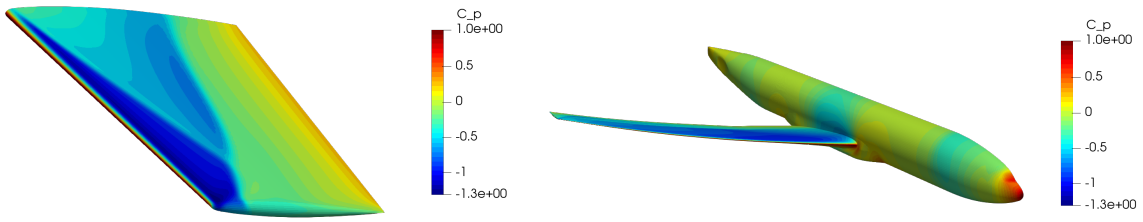
5.2.1 Test Cases

In Table 5.8, we present the different test cases solved with FANSC in Section 5.3. They depict transonic cases of different geometry and size. Each of them is characterized by the free-stream Mach number M, the angle of attack α and the Reynolds number Re.

Table 5.8: Parameters for the test cases used in FANSC.

Name	M	α	Re	Dimension	Grid cells	Number of Blocks
NACA 0012 [143]	0.80	1.25°	inviscid	2D	3.17×10^4	4
ONERA M6 [151]	0.84	3.06°	11.72×10^6	3D	1.85×10^6	288
CRM-L3 [152]	0.84	2.11°	5.0×10^6	3D	7.74×10^6	256

The test cases include an inviscid 2D case for the NACA 0012 airfoil with a transonic flow at a Mach number 0.8. The ONERA M6 wing as well as the CRM wing/body are both turbulent and 3D cases. The NACA 0012 airfoil and ONERA M6 wing cases were used in Subsection 4.2.1 to study the similarities in subsequent adaptive SPAI patterns. The initialization of the flow on the mesh is achieved with a nested iteration strategy in which the initial guess is improved with inexpensive solves on successive nested grids of small size [138]. The solution is then interpolated to be used as an initial guess for the next finer grid. In this way, it is possible to start the simulation on the fine grid with a more accurate initial solution. The original grids were split to obtain the maximum degree of parallelism. The splitting strategy used in FANSC to ensure an adequate load balancing is described in [111]. The pressure contours of the steady-state solution over the ONERA M6 and CRM geometries are shown in Figure 5.14.



(a) ONERA M6 Wing. **(b)** CRM Wing/Body.

Figure 5.14: Surface pressure contours calculated with FANSC over the ONERA M6 wing and CRM wing/body.

5.2.2 Methodology for Comparing Algorithms

The convergence of the flow solver is monitored with the nonlinear residual r_{nl} based on the normalized mass flux norm over the cells, given at the n-th nonlinear iteration by

$$r_{nl}^{n} = \frac{|| \left(\rho \mathbf{v}\right)^{n} ||_2}{|| \left(\rho \mathbf{v}\right)^{0} ||_2}. \tag{5.4}$$

The main unit of interest for comparing the efficiency of different algorithms is the CPU execution time. However, its measurement is far from perfect as it is dependent on the architecture of the computer and the compiler. The memory bandwidth and the cache size can have a significant impact on the efficiency of an algorithm. On the contrary, units specific to the solving method, such as the number of nonlinear and linear iterations in a Newton-Krylov method, are indicators of the inherent performance of the algorithm. Nevertheless, they cannot provide the overall picture of the algorithm efficiency. For instance, a convergence in fewer Newton steps usually comes with a higher computational cost per step. For the tests, we report the CPU execution times as well as the number of linear iterations, nonlinear steps and eventually the number of preconditioning evaluations. All the following tests were performed on the supercluster Béluga managed by Calcul Québec, and were therefore executed on the same CPU architecture.

5.3 Reynolds-Averaged Navier-Stokes Results and Discussion

In this section, the preconditioning methods are tested to solve the compressible RANS equations with a Newton-GMRES algorithm within the flow solver FANSC. The effects of SPAI preconditioning on the convergence of the nonlinear problems and its performance in comparison with domain decomposition preconditioners are studied.

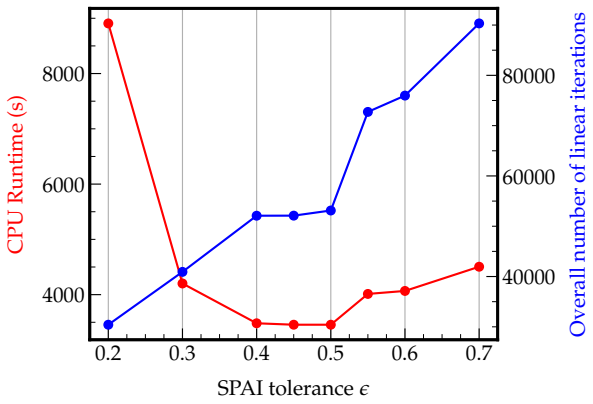
5.3.1 Optimization of the SPAI Preconditioning Strategy

Before comparing SPAI with domain decomposition preconditioning methods, we first investigate in depth the optimization of the SPAI preconditioning strategy introduced in Section 4.2. The optimization of the recomputation and SPAI density parameters are paramount for the efficiency of SPAI. The following tests were carried out with the CRM-L3 grid on 160 cores for load balancing purposes. The same type of optimization was conducted with the ONERA M6 wing. The CRM case is solved by means of a Newton-GMRES(30) method with a defect correction approach where the relative inner tolerance τ_{lin} for the linear systems is set to 10^{-2}. Besides, the CFL number is linearly increased at every nonlinear update. The convergence stops once the nonlinear residual r_{nl} has been reduced by five orders of magnitude. Four parameters were investigated to optimize the SPAI preconditioning:

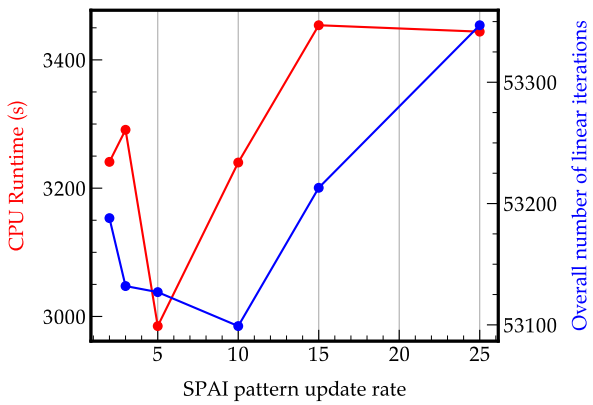
- The SPAI tolerance ϵ, which defines the stopping criterion within the SPAI adaptive strategy (cf. equation (4.10)).
- The update pattern rate, which sets the rate at which the SPAI pattern is recomputed from scratch with the adaptive SPAI algorithm, i.e. the number of reused patterns with a fixed SPAI (orange cell in Figure 4.8) before performing a fresh adaptive computation from a diagonal pattern (blue cell in Figure 4.8).
- The SPAI freezing period, which specifies the length of the freezing periods in Figure 4.8.
- The number of Newton steps before the onset of the freezing periods, i.e. the stride of orange cells in Figure 4.8 before the first freezing period.

The final CPU timings as well as the total number of linear iterations are reported. We note that in a given parameter study, all other parameters are maintained constant, but not necessarily at their optimal value. The objective is to examine the possible speedup obtained with appropriate SPAI parameters. The results are presented in Figure 5.15.

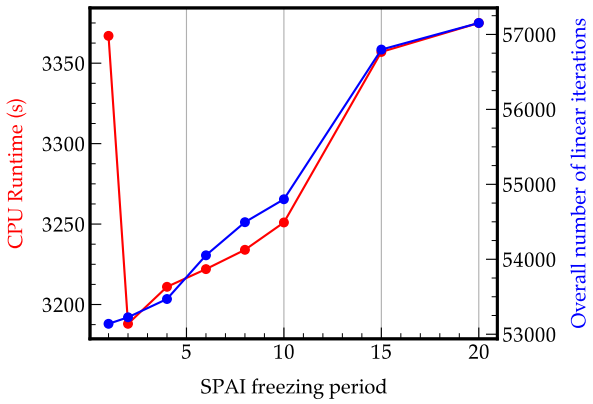
In Figure 5.15a, the performance of the preconditioning is reported with a SPAI tolerance ϵ ranging from 0.2 to 0.7. The fixed version of the SPAI preconditioner is recalculated



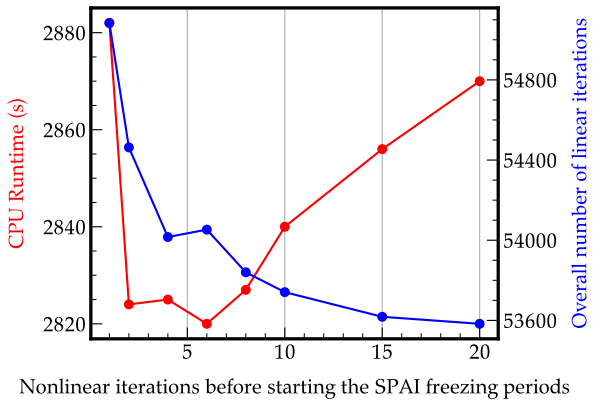
(a) Influence of the SPAI tolerance ϵ.



(b) Influence of the update pattern rate.



(c) Influence of the SPAI period length.



(d) Influence of the interval length before the freezing periods.

Figure 5.15: CPU runtimes and number of nonlinear iterations for different SPAI parameters for the CRM test case.

every iteration and the SPAI pattern is updated every five iterations. A steady reduction in the number of linear iterations is achieved with a lower tolerance ϵ as the preconditioner becomes a better approximation of the inverse of the Jacobian matrix. As a result, the decrease in the number of linear iterations is first accompanied by a reduction in the CPU runtime until the application of the preconditioner becomes too computationally expensive. The optimal tolerance ϵ would be between 0.4 and 0.5. In Figure 5.15b, the SPAI tolerance ϵ is set to 0.5 while the update pattern rate is adjusted.

In the freezing period study in Figure 5.15c, the SPAI preconditioner is frozen for successive nonlinear iterations. In this case, frequent recomputations of the SPAI entries result in a substantial acceleration. Finally, the impact of the length of the initial successive recomputations of the SPAI preconditioner is studied in Figure 5.15d. The different studies show that many parameters have to be taken into account to ensure an efficient SPAI preconditioner, and prevent unreasonable expensive computations. In summary, the most optimal SPAI preconditioner produced a threefold increase in the convergence of the nonlinear solver.

5.3.2 Comparison with Domain Decomposition Preconditioners

Low-Order Jacobian

In the first numerical experiments, the Newton-Krylov steps are carried out with a low-order Jacobian matrix as the left operator, according to the defect correction approach presented in Subsection 3.4.2. The low-order Jacobian matrices are sparser due to their reduced discretization stencil. On the other hand, the convergence requires a large number of nonlinear iterations to reach the desired reduction in the nonlinear residual r_{nl}.

The flow around the ONERA M6 wing is solved with FANSC for a decomposition of the mesh into 288 blocks. This is the maximum we were able to obtain from the original mesh to guarantee a sufficient number of cells on the coarsest mesh. Even though the mesh is made of 288 blocks, the simulations are carried out on 252 cores to ensure a satisfying load balance between the cores. We study the convergence for different inner linear tolerances τ_{lin}, ranging from 10^{-1} to 10^{-4} and compare the following preconditioners: block SOR, block Jacobi-ILU(0), restricted additive Schwarz-ILU(0) and SPAI. In line

Table 5.9: Results of the ONERA M6 case with domain decomposition preconditioning for different linear relative tolerances τ_{lin}.

288 blocks - ONERA M6 Wing					
Preconditioner	τ_{lin}	Nonlinear iterations	Lin. Iters.	Krylov time (s)	Total CPU time (s)
Block SOR	10^{-1}	497	7052	50.01	87.96
	10^{-2}	495	17,246	121.31	157.55
	10^{-3}	495	52,197	375.07	419.9
	10^{-4}	495	125,197	983.95	1021.03
Block Jacobi ILU(0)	10^{-1}	496	3119	82.12	118.38
	10^{-2}	497	7636	185.28	224.76
	10^{-3}	496	28,389	692.38	737.51
	10^{-4}	496	67,658	1833.08	1876.82
Restricted Additive Schwarz-ILU(0)	10^{-1}	495	2303	89.5	125.54
	10^{-2}	498	7245	262.19	302.42
	10^{-3}	496	2454	950.33	996.12
	10^{-4}	Diverges			

with the SPAI strategy study carried out for the CRM case in the preceding subsection, the following SPAI results are obtained with the optimal parameters, namely with a SPAI tolerance $\epsilon = 0.5$, a freezing period of length 6 and a first update interval of length 6. The corresponding results are reported in Table 5.9 and Table 5.10.

In Figure 5.16, an instance of the convergence histories is plotted for four different linear tolerances τ_{lin} with the block Jacobi-ILU(0) preconditioner. It turns out that the number of nonlinear iterations is barely sensitive to the reduction of the linear tolerance τ_{lin}. This is due to the use of an inaccurate Jacobian matrix, which is detrimental to the consistency of the linear systems. Therefore, deep convergence of linear systems is not only unnecessary in this case, but also leads to oversolving of the linear systems. In addition, a large number of nonlinear iterations are required to achieve convergence.

Figure 5.17 illustrates the convergence of the nonlinear residual r_{nl} with the block Jacobi-ILU(0), ASM-ILU(0) and SPAI for different inner tolerances. We can observe that the SPAI preconditioner becomes more advantageous when the linear systems need to be solved more precisely. This is due to the fact that the application of the SPAI preconditioner is rather inexpensive compared to sophisticated block Jacobi-ILU(0) or ASM-ILU(0) preconditioners. As a result, the SPAI computational cost before the Krylov method is bet-

Table 5.10: Results of the ONERA M6 case for different linear relative tolerances τ_{lin} with the adaptive block SPAI.

Adaptive Block SPAI - 288 blocks - ONERA M6 wing						
τ_{lin}	Nb. of SPAI comp.	Nonlin. Iters.	Lin. Iters.	Krylov time (s)	SPAI time (s)	Total CPU time (s)
10^{-1}	52	626	24,967	161.68	12.57	205.45
10^{-2}	49	492	32,447	203.07	11.32	241.48
10^{-3}	49	496	88,549	547.94	11.33	588.37
10^{-4}	49	497	169,587	1163.09	11.23	1202.82

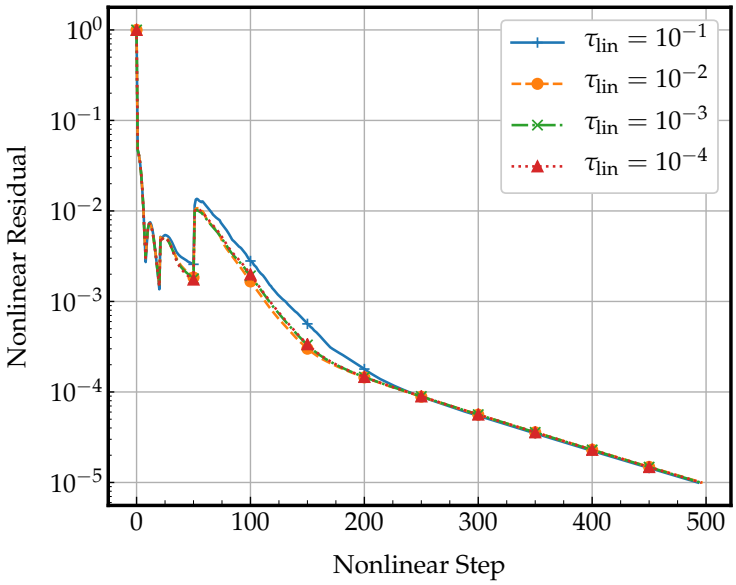
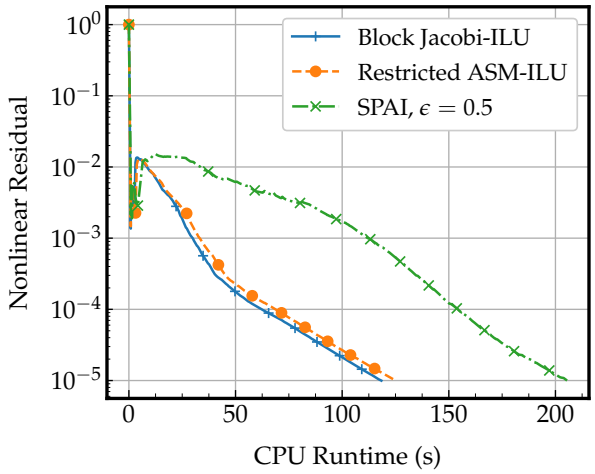
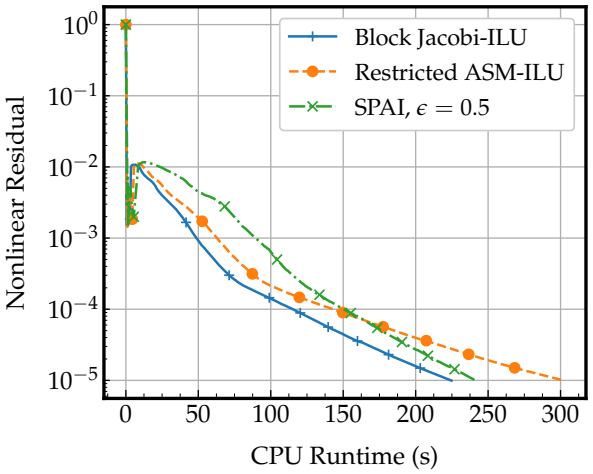


Figure 5.16: Convergence histories of the nonlinear residual r_{nl} for different forcing terms τ_{lin} with block Jacobi-ILU(0) preconditioning in the ONERA M6 test case.

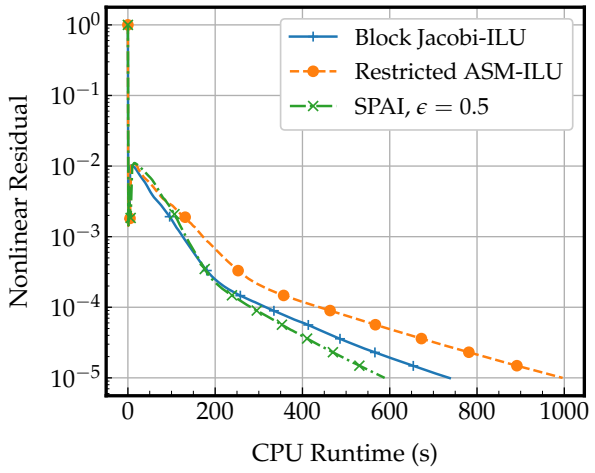
ter amortized over the different linear iterations. Besides, as we can see in Table 5.12, the overall SPAI computational time is bounded, since the number of nonlinear iterations remains approximately the same in all cases. On the contrary, the Krylov application time increases due to the growing number of linear iterations. In contrast, the efficiency of block Jacobi and ASM decreases as τ_{lin} is lowered. SPAI preconditioning becomes more efficient than the domain decomposition preconditioners for inner tolerances lower than $\tau_{\text{lin}} = 10^{-3}$. For the last relative tolerance $\tau_{\text{lin}} = 10^{-4}$, the ASM preconditioned method does not converge probably because the local GMRES processes are only carried out down to a relative tolerance of 10^{-2}, which is too loose.



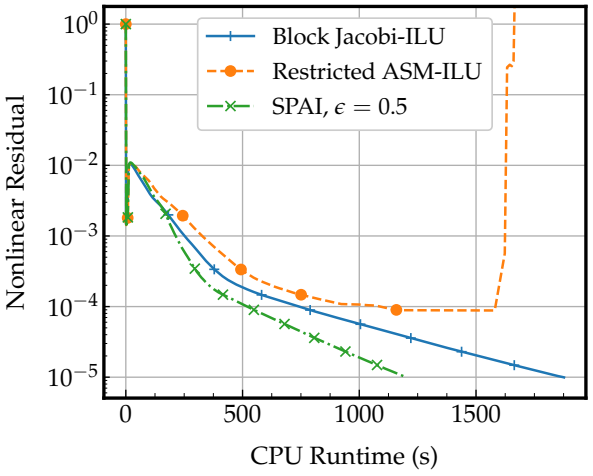
(a) $\tau_{\text{lin}} = 10^{-1}$.



(b) $\tau_{\text{lin}} = 10^{-2}$.



(c) $\tau_{\text{lin}} = 10^{-3}$.



(d) $\tau_{\text{lin}} = 10^{-4}$.

Figure 5.17: Convergence histories for the ONERA M6 test case with different forcing terms τ_{lin} using block Jacobi-ILU(0), ASM-ILU(0) and SPAI preconditioners.

The same study is carried out with the CRM-L3 test case for two different inner tolerances $\tau_{\text{lin}} = 10^{-2}$ and $\tau_{\text{lin}} = 10^{-3}$. The latter implies a deeper convergence for the linearized systems arising from Newton's method. Once again, we consider block SOR, ASM and SPAI preconditioning techniques. The SOR preconditioner is directly based on the coefficients of the Jacobian matrix and therefore does not require any calculation before its application, unlike the ILU(0) and SPAI preconditioners. SPAI preconditioning is employed with the tolerance $\epsilon = 0.5$ and a freezing period of length 2. Table 5.11 shows the results of these tests with the block SOR and ASM-ILU(0) methods. The results for the SPAI preconditioner are gathered in Table 5.12. The CRM simulations are executed on 160 cores with a mesh decomposed into 256 blocks.

Table 5.11: Results of the CRM test case with the domain decomposition preconditioners for different linear tolerances τ_{lin}.

Preconditioning	τ_{lin}	Nonlinear iterations	Lin. Iters.	Krylov time (s)	Total CPU time (s)
Block SOR	10^{-2}	464	20,359	932.88	1100.11
	10^{-3}	461	51,570	2423.65	2596.46
Restricted ASM-ILU(0)	10^{-2}	461	4377	3335.14	3506.38
	10^{-3}	450	12,166	4741.47	4917.47

Table 5.12: Results of the CRM test case with the adaptive block SPAI for different linear tolerances τ_{lin}.

Adaptive Block SPAI - CRM L3 grid						
τ_{lin}	Nb. of SPAI comp.	Nonlin. Iters.	Lin. Iters.	Total CPU time (s)	Krylov time (s)	SPAI time (s)
10^{-2}	83	457	53,221	2305.73	2120.67	152.84
10^{-3}	84	460	252,892	10,190.86	9999.99	158.82

The CPU timings for the three different preconditioning methods are shown in Figure 5.18. The SOR preconditioner yields the best results, converging in about 1000 seconds, less than half of what is required for the SPAI to converge and one third of the runtime for the ASM preconditioner. The computational domain is not sufficiently partitioned to negatively affect the convergence with block SOR preconditioning.

Let us have a better insight into the convergence of the linear systems provided by Newton's method. In Figure 5.19a, the numbers of linear iterations to satisfy the termi-

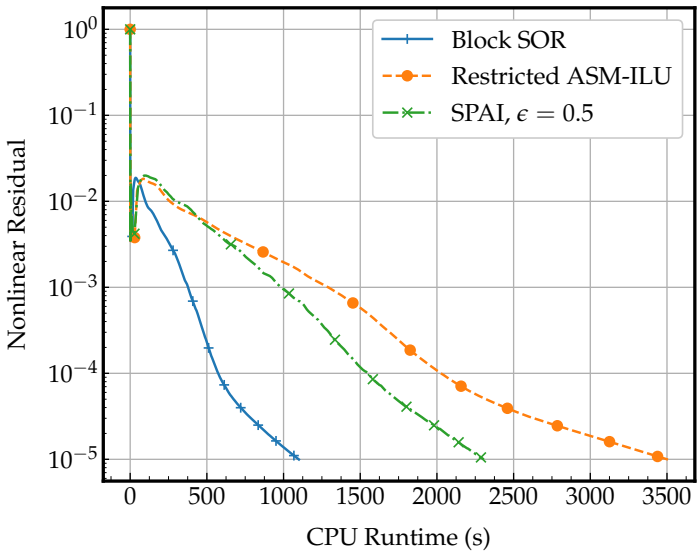
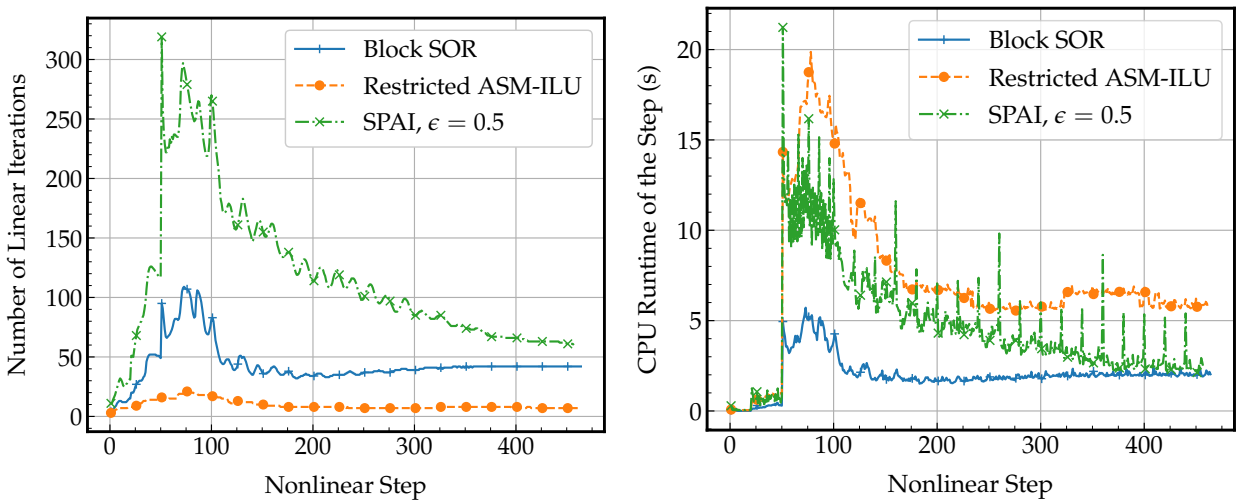


Figure 5.18: CPU runtime for block SOR, ASM-ILU(0) and SPAI preconditioning for the CRM test case with $\tau_{\text{lin}} = 10^{-2}$.

nation criterion for the successive linear solvers are shown. The SPAI preconditioned method requires more linear iterations to converge than with other preconditioners. SPAI is an explicit preconditioning method approximating the inverse of the Jacobian matrix. On the contrary, SOR and ILU(0) belong to the class of implicit preconditioners and directly approximate the Jacobian matrix. As a consequence, the inverses of SOR and ILU are dense approximations of the inverse of the flux Jacobian. As a result, they generally achieve convergence in fewer linear iterations. On the other hand, their application requires backward and forward substitutions when the application of the SPAI preconditioner is simply a sparse matrix-vector product. The advantage of the SPAI comes from the low cost of its application which is offset by a higher number of linear iterations.

In Figure 5.19b, we plot the time required to solve the linear system involved in each nonlinear step for different preconditioning methods. Besides, the computational time to form the SPAI and ILU(0) preconditioners have been added. We note that SPAI preconditioning often results in faster convergence for the linear solvers compared to ASM preconditioning, except for some very time-consuming linear solvers giving regularly spaced spikes in Figure 5.19b. The complete recomputations of the SPAI preconditioner, namely using the adaptive algorithm from a diagonal matrix yield the highest peaks.



(a) Number of linear steps for the successive linear solvers.

(b) Time spent in the successive linear solvers

Figure 5.19: Study of the linear solvers along the simulation of the flow around the CRM test case for a linear tolerance $\tau_{\text{lin}} = 10^{-2}$

These nonlinear steps are the most expensive to carry out. In contrast, updating the SPAI preconditioner from the last computed pattern yields the intermediate spikes, which are approximately as expensive as ASM-ILU(0) preconditioning. Finally, the lower parts of the SPAI plot correspond to frozen SPAI applications. Therefore, the SPAI computations must be performed with care in order to avoid certain unnecessarily costly calculations.

Matrix-Free Jacobian

The left hand-side operator within the Krylov method is now computed with a matrix-free approach as presented in Subsection 3.4.2. A forward finite difference in the same way as in equation (3.31) is utilized. Consequently, an application of the Jacobian matrix within the GMRES process requires an additional residual evaluation for a perturbed state $\mathbf{R}(\mathbf{w}^n + h\delta\mathbf{w})$. The low-order Jacobian matrix used in the defect correction approach is still explicitly computed to construct the preconditioners.

As we shall see, this improvement in consistency between the left hand side Jacobian and the right-hand side residual $\mathbf{R}(\mathbf{w}^n)$ is accompanied by a sharp reduction in the number of nonlinear steps to reach the prescribed nonlinear tolerance for r_{nl}. In return,

the linear convergences become more demanding with a greater number of linear iterations required. As in the defect correction approach, a nested iteration strategy with three meshes is performed to compute a satisfying initial guess on the fine mesh of interest. The CFL number is increased exponentially to accelerate the convergence. A pre-study was conducted to determine the appropriate parameters for the SPAI preconditioning strategy. The linear solvers employ left preconditioners, and the linear tolerance τ_{lin} of the GMRES processes is set to 0.1. The corresponding convergence histories are shown in Figure 5.20. We observe that the solver termination criterion is achieved in far fewer Newton steps than with a low-order Jacobian matrix. Approximately 100 Newton steps are necessary to attain a decrease of five orders of magnitude, whereas around 500 steps were required with the defect correction approach (cf. Figure 5.16). In Figure 5.20, we can easily observe the convergence on the three meshes which start respectively at the 1^{st}, 21^{st}, and 51^{st} nonlinear iteration. The simulations are conducted for a drop of seven orders of magnitude in the nonlinear residual r_{nl}.

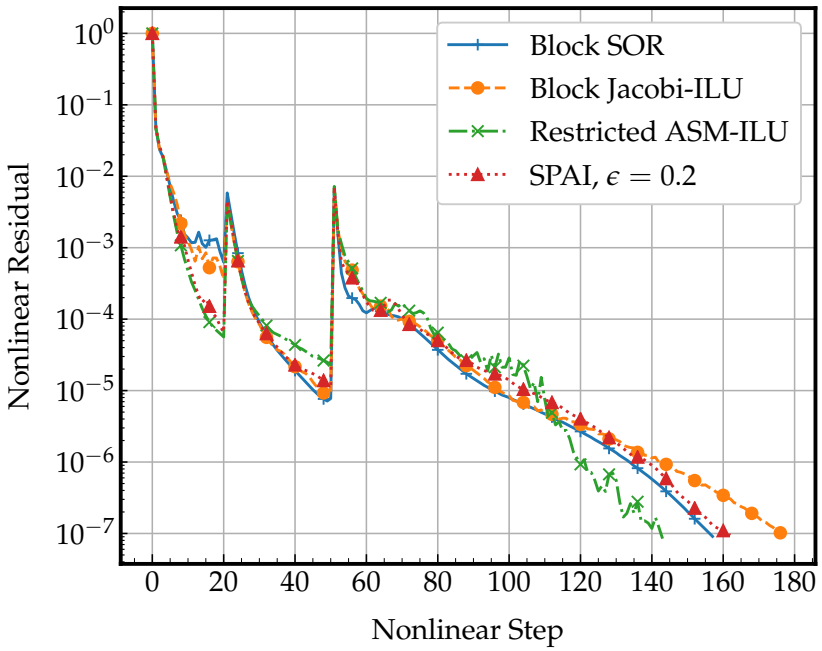
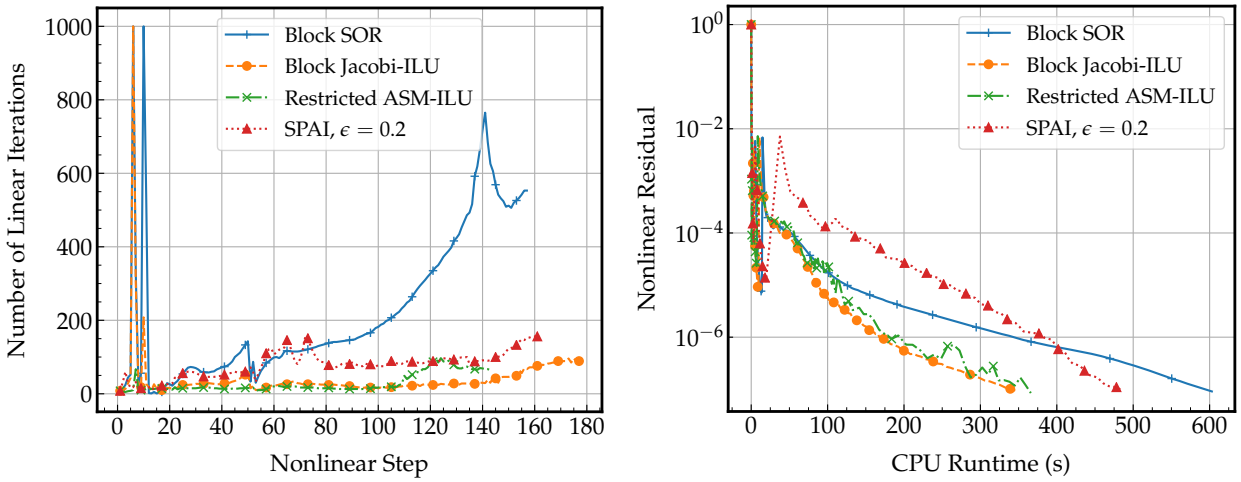


Figure 5.20: Convergence histories with a matrix-free approach for the ONERA M6 case.

A dense SPAI matrix is essential to obtain satisfactory results with the matrix-free method since the linear systems are more demanding to solve. Therefore, the optimal con-



(a) Number of linear iterations per nonlinear step.

(b) CPU runtimes for the different preconditioners.

Figure 5.21: Results for the ONERA M6 case solved with a matrix-free approach.

vergence is obtained with a tolerance $\epsilon = 0.2$, whereas 0.5 was sufficient for a low-order Jacobian matrix. We emphasize that the SPAI matrix used is the approximate inverse of the low-order Jacobian matrix, employed to precondition the high-order matrix-free Jacobian. Thus, even though the linearized system solved is closer to an exact Newton step, there is still a lack of consistency between the preconditioner and the coefficient matrix, which may impede convergence of the GMRES solver. With these settings, the linear systems converge with a relatively low number of linear steps (cf. Figure 5.21a). The number of linear iterations progressively increases as the CFL number is rapidly raised.

Nevertheless, the SPAI preconditioned implicit solver lags behind in terms of CPU time compared to the block Jacobi and ASM with ILU(0) preconditioning for local subsystems (Figure 5.21b). To explain this discrepancy between SPAI and domain decomposition methods, we study in more detail the cost of applying the preconditioning methods. In Figure 5.22, the y-axis displays the CPU runtime of a single linear iteration within a given nonlinear step. Since the GMRES computational cost grows linearly with the number of linear iterations without restart, the average CPU times of the nonlinear steps are reported. On the one hand, the block Jacobi preconditioning applications are purely parallel, but involve fairly expensive backward and forward local substitutions. The SOR splitting and ILU(0) factorization are performed locally on subdomains owned by the

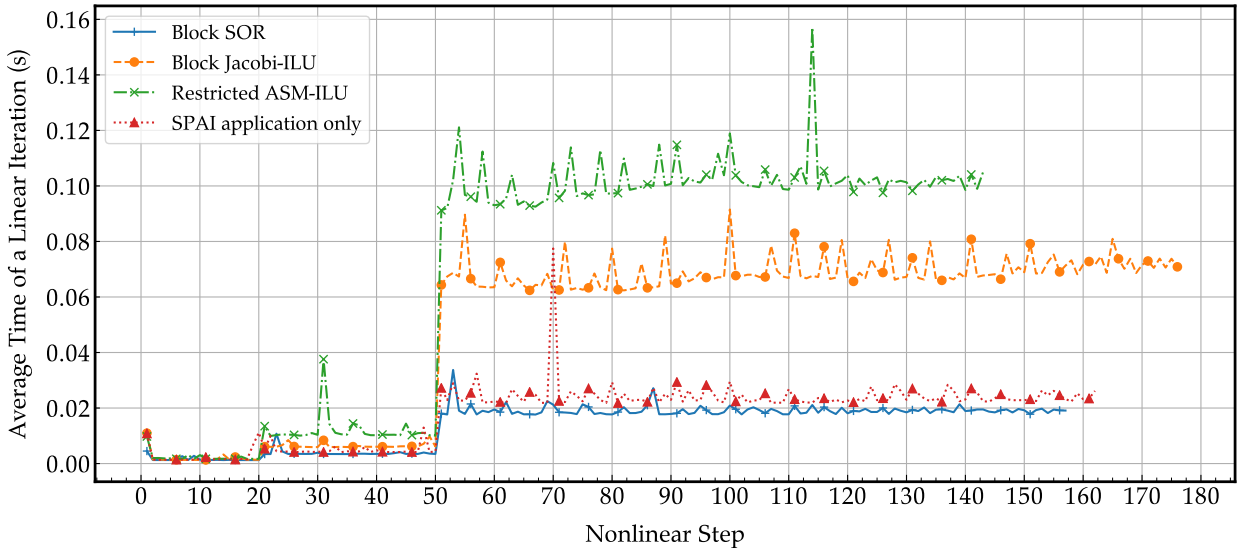


Figure 5.22: Average time of a linear iteration using domain decomposition and SPAI preconditioners along the convergence for the ONERA M6 test case.

cores, without any communications. Contrary to the ILU preconditioner, the SOR preconditioner does not require any construction step, but directly uses the entries of the coefficient matrix. On the other hand, the SPAI applications are mere sparse matrix-vector multiplications, which are inexpensive in terms of computation but still require point-to-point communications of the vector entries across the cores. Furthermore, the denser the SPAI matrix, the more communications are required for a sparse matrix-vector multiplication. Therefore, the application of the SPAI preconditioner becomes prohibitively expensive beyond a certain density threshold. We note that the block SOR and SPAI applications are nearly equivalent in this case.

Block Jacobi combined with ILU preconditioners requires more time than block SOR due to the additional construction cost of the local ILU factorization and the local GMRES solvers. Hence, an average GMRES step with block Jacobi-ILU(0) takes 70 ms which is more than three times more expensive than a local application of the SOR preconditioner. Finally, the ASM-ILU(0) preconditioning uses an overlapping domain decomposition contrary to the block Jacobi method. As a consequence, an additional communication step is added across the processors to transfer the coefficients located in the overlapping areas. The average time per linear iteration suffers from this cumbersome additional communication.

In Figure 5.22, the SPAI computational cost that takes place before the Krylov method was omitted to only compare the time spent throughout the GMRES process. In contrast, the SPAI computational time has been added in Figure 5.23 and divided by the total number of linear iterations to spread its cost over the successive linear iterations. The highest peaks correspond to a complete adaptive SPAI computation, using the adaptive algorithm from a diagonal sparsity pattern. In these steps, the average cost per linear iteration is dramatically worse than the cost of the domain decomposition preconditioners per linear iteration. On the other hand, intermediate spikes occur at Newton steps where the SPAI preconditioner is recomputed from the latest saved sparsity pattern. A fixed SPAI computation stands somewhere between a frozen SPAI and a full recomputation. In this instance, it is approximately as expensive as a block Jacobi-ILU(0) in terms of average cost per linear iteration.

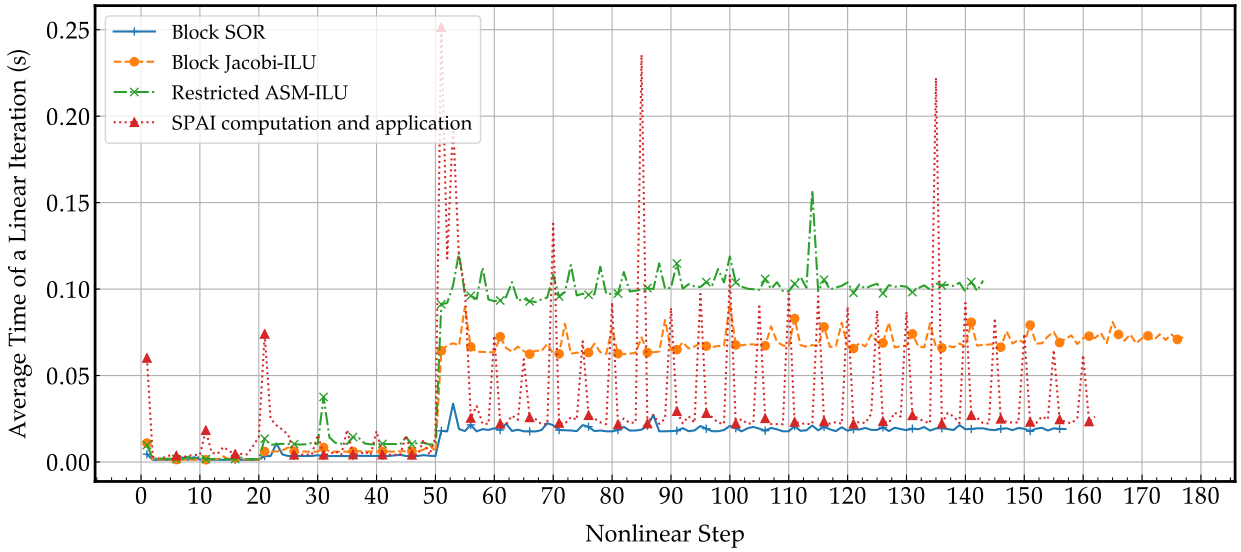


Figure 5.23: Average time of a linear iteration along the convergence of the ONERA M6 case with a matrix-free approach.

The preceding figures show the time performance normalized by the number of linear iterations. Therefore, they are not reflective of whether a preconditioning method entails several or few linear iterations to achieve convergence for the linear systems. For this purpose, the CPU time per nonlinear step is plotted in Figure 5.24.

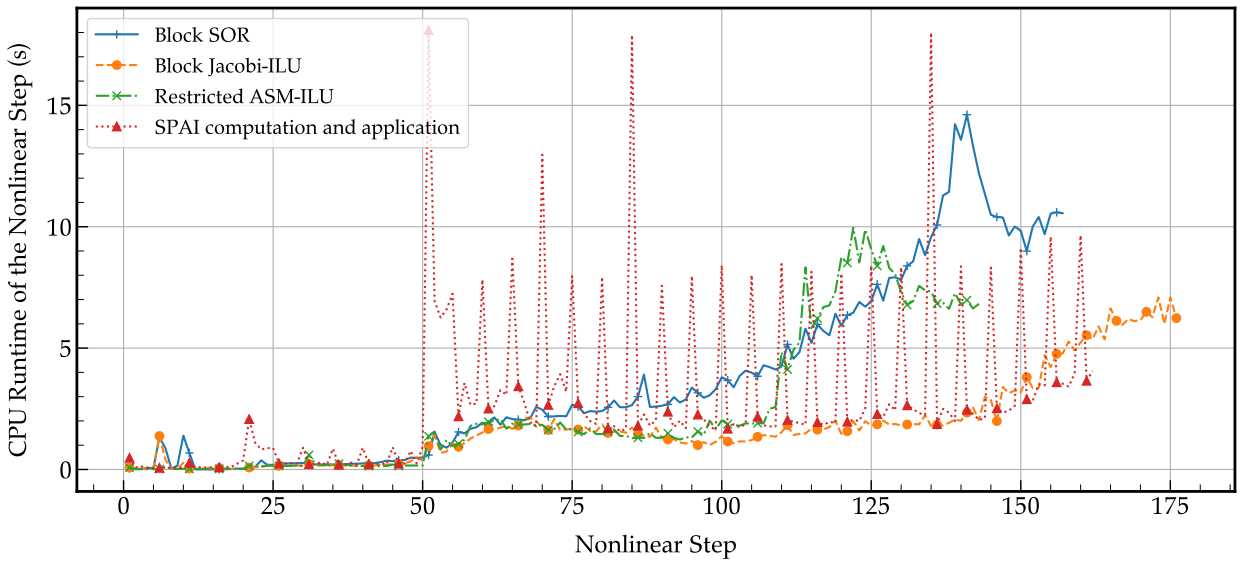


Figure 5.24: CPU time for the nonlinear steps for the ONERA M6 case with a matrix-free approach.

At the beginning of the convergence on the fine mesh, the SPAI preconditioner is recomputed at each nonlinear step. Then, since SPAI requires more linear iterations than the block Jacobi-ILU(0) and ASM preconditioner, the SPAI cost per Newton step is mostly higher even though the average cost of SPAI per linear iteration is cheaper. Some factors that may hinder the SPAI application advantage over the ASM and block Jacobi-ILU(0) preconditioners are:

- If the SPAI preconditioner requires too many linear iterations, the average time per Newton step can become worse than that of the domain decomposition preconditioners. This explains for instance why the SPAI line is above the block Jacobi-ILU(0) line in the previous Figure 5.24, even though the SPAI application cost for a single linear iteration is cheaper (cf. Figure 5.22).

- The cost of the SPAI computation is too expensive and does not reduce efficiently the required number of linear iterations. This occurs at the beginning of the convergence on the fine mesh where the SPAI computation undermines the cheapness of the SPAI application. It also explains why the SPAI is quickly overtaken by the block Jacobi-SOR preconditioned solver. Yet, the SPAI preconditioner is able to be

competitive near the end of the convergence, once the SPAI preconditioner is less frequently updated.

- Within the matrix-free approach, the SPAI matrix is a sparse approximation of the inverse of the first-order Jacobian matrix, different from the high-order Jacobian. Because of this lack of consistency, a more accurate SPAI matrix closer to the first-order Jacobian inverse is not necessarily a better approximation of the inverse of the high-order Jacobian. Therefore, below a certain SPAI tolerance ϵ, the entries added to the SPAI matrix not only increase the cost of applying the matrix within a Krylov method, but also no longer provide a reduction in the number of linear iterations. In the same way, it may be possible to freeze the SPAI preconditioner for longer periods without seeing the number of linear iterations increase (cf. Figure 5.25).

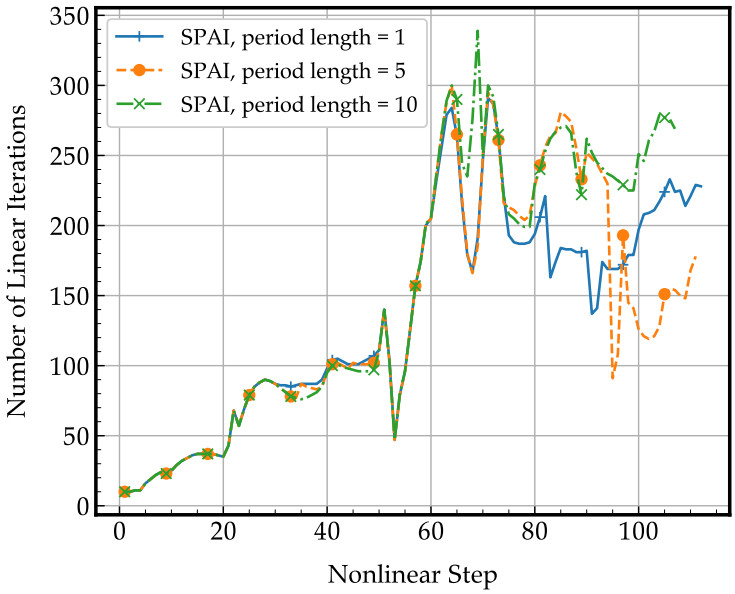
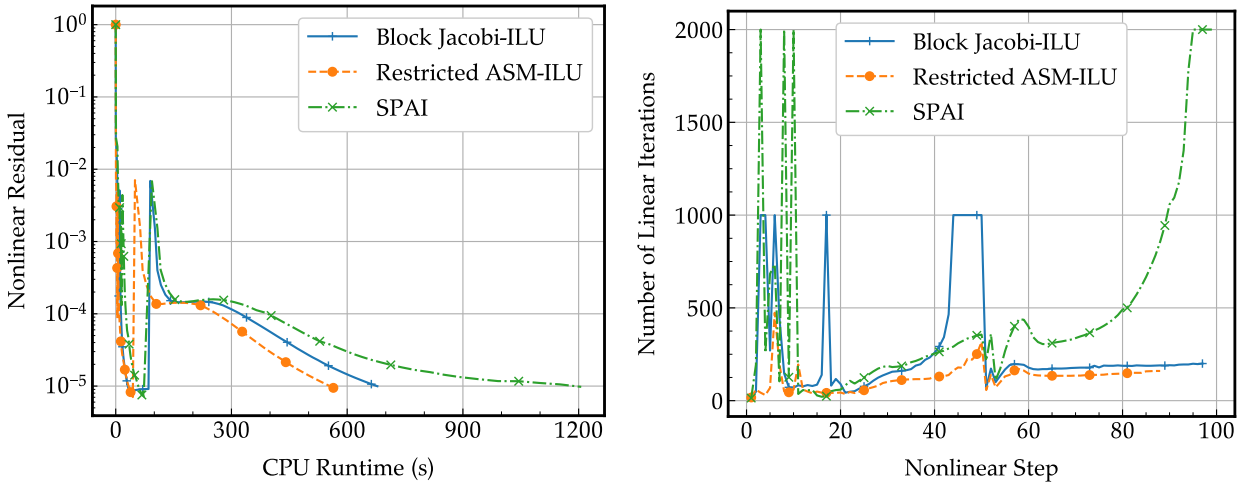


Figure 5.25: Number of linear iterations along the convergence for different period lengths in the SPAI framework for the ONERA M6 case.

Furthermore, we could not lower the linear tolerance τ_{lin} to less than 10^{-1}, otherwise SPAI was not robust enough to guarantee the required convergence of the linear systems, even with very tight tolerances ϵ. To this extent, the same experiment is conducted for a relative linear tolerance $\tau_{\text{lin}} = 10^{-2}$. Hence, more GMRES iterations are required to converge than with the preceding tolerance $\tau_{\text{lin}} = 10^{-1}$. The SPAI preconditioner was

able to outperform the domain decomposition preconditioners for tighter tolerances with a first-order Jacobian as the operator (cf. Figure 5.17). In the matrix-free approach, linear systems are more demanding and the SPAI preconditioner is inefficient in its reduction of the number of linear iterations (cf. Figure 5.26b), yielding a very slow convergence.

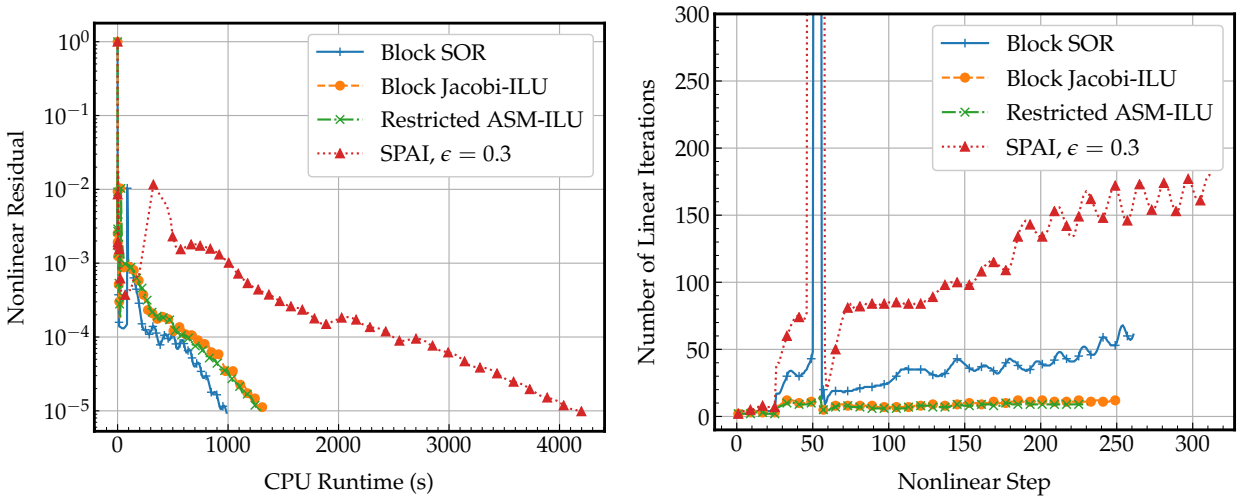


(a) Convergence history for $\tau_{\text{lin}} = 10^{-2}$.

(b) Number of linear iterations per nonlinear step.

Figure 5.26: Results for the ONERA M6 case solved with a matrix-free approach for $\tau_{\text{lin}} = 10^{-2}$.

Finally, the CRM case is solved with the matrix-free Jacobian method on 160 cores. We note that this mesh is less partitioned than that of the ONERA M6 wing case. Therefore, the domain decomposition preconditioning methods yield fast convergences compared to the SPAI preconditioner. The convergences are shown in Figure 5.27a. In this case, block Jacobi and ASM methods require far fewer linear iterations than the SPAI method (cf. Figure 5.27b). The study of the preconditioning application in the GMRES methods proves once again the low cost of the SPAI application compared to the block Jacobi and ASM methods. In Figure 5.28, the average cost of a linear iteration preconditioned with SPAI is approximately the same as that of block SOR. However, the partitioning into a few blocks encourages the use of domain decomposition preconditioners that entail fewer linear iterations. In this case, SPAI preconditioning does not compete with block Jacobi and ASM preconditioners (cf. Figure 5.29).



(a) Convergence of the nonlinear residual as a function of the CPU time.

(b) Number of linear iterations per nonlinear step.

Figure 5.27: Results for the CRM case solved with a matrix-free approach for $\tau_{\text{lin}} = 10^{-1}$.

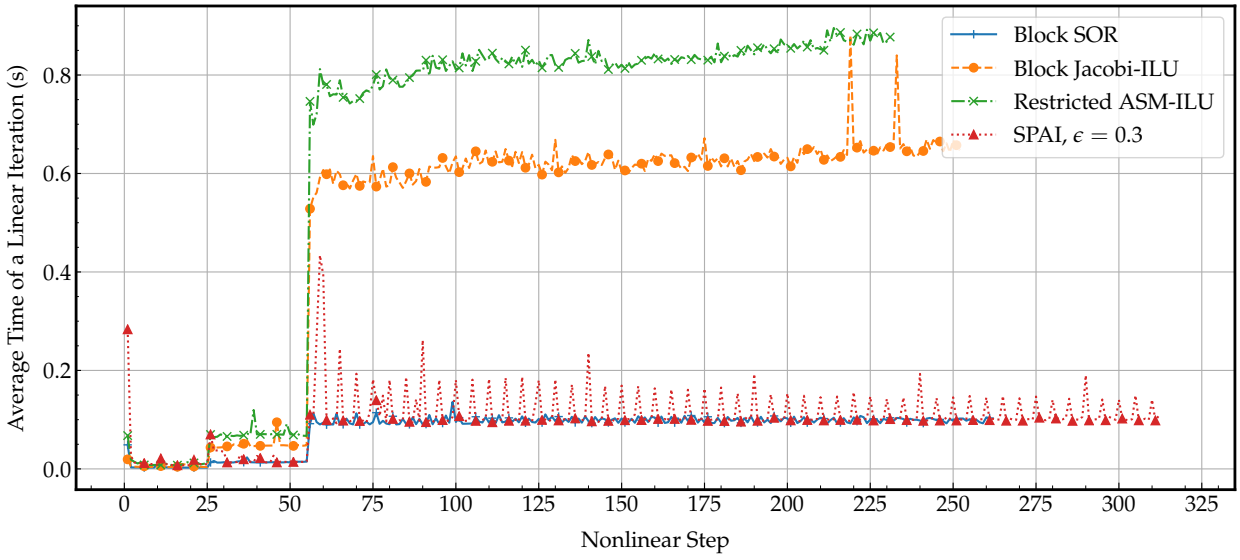


Figure 5.28: Average cost of a linear iteration along the convergence for the CRM case for different preconditioners.

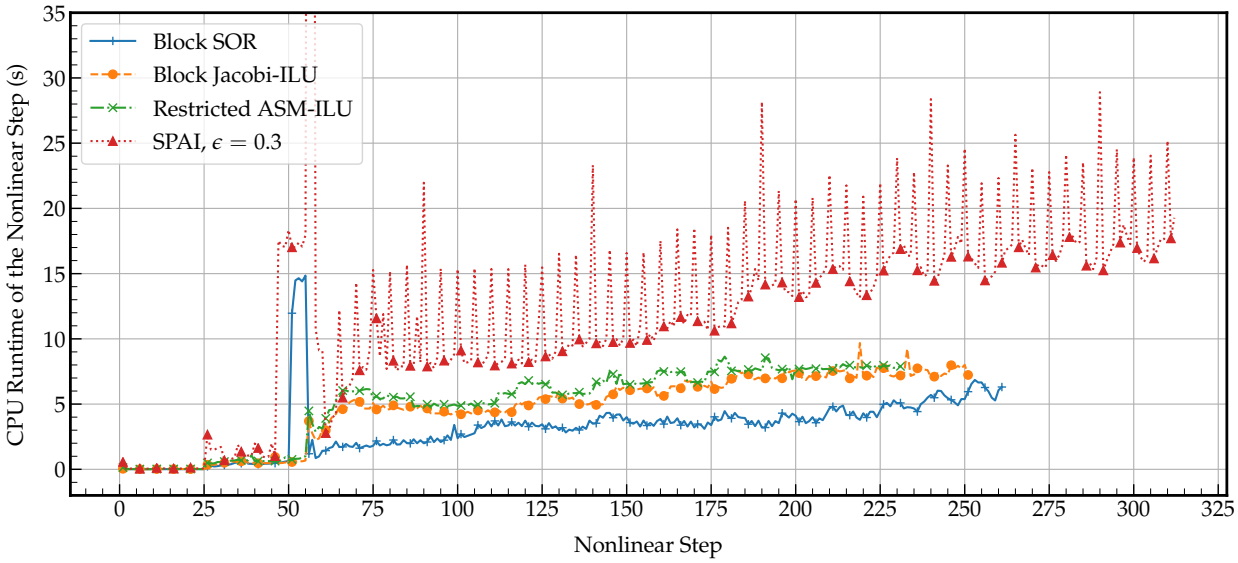


Figure 5.29: CPU timings to complete the nonlinear steps along the convergence for the CRM test case with a matrix-free approach.

Chapter 6

Conclusion and Future Work

6.1 Concluding Remarks

The revolution in computer architectures has profoundly reshaped the way algorithms are designed to make efficient use of available resources. It is becoming commonplace to favor the parallel efficiency of an algorithm at the expense of its robustness to accelerate convergence. This paradigm is the driving force behind decomposition methods to adapt sequential preconditioning methods on large-scale parallel computers. In accordance with previous studies on SPAI preconditioning, it has been observed that SPAI is not only more computationally expensive but also less robust than the implicit preconditioning techniques, such as SOR and ILU preconditioners, on weak parallel simulations. This observation is not an obstacle to the use of SPAI, since its main advantage comes from its high parallelism, its low cost of application and its invariance with respect to the number of cores.

In Section 5.1, we investigated the performance of the SPAI preconditioner in solving highly partitioned linear systems. We have shown that the SPAI method is capable of converging problems where domain decomposition preconditioning fails. The principal weakness of domain decomposition preconditioners is their sensitivity to the number of cores. They progressively lacked robustness and stalled in the convergence as the number of cells per block declined. Highly partitioned domains and deep linear convergences are typically the conditions for which SPAI preconditioning proved to be more robust. Later on, its integration in a finite volume solver was studied under highly parallel conditions. For this purpose, we introduced a SPAI preconditioning framework to spread the cost of the SPAI computation along the convergence. However, the partitioning was

not sufficient to prevent convergence with domain decomposition preconditioning for the cases treated with FANSC in Section 5.3. As a matter of fact, the increase in the problem size is usually accompanied with an increase in computational resources. This allows domain decomposition preconditioners to remain competitive. Furthermore, quasi-Newton methods require only an approximate solution of the linear systems. Thus, a convergence of a few orders of magnitude for the intermediate linear systems is often sufficient in the test cases to ensure the global convergence of the nonlinear residual r_{nl}. Consequently, a sound parallel preconditioner to achieve deep convergence is not necessarily relevant for Newton-Krylov flow solvers.

The difficulty in finding an appropriate *a priori* SPAI pattern encourages the use of the adaptive algorithm. Nevertheless, determining appropriate SPAI parameters which at least ensure convergence remains a challenging task. The SPAI preconditioning is not suitable for use as a black box. Besides, it should be noted that a user usually does not have time to adjust and test different preconditioning parameters. On the one hand, SPAI preconditioning yielded the best results in the case of deep convergence of the linear systems, with low-order Jacobian matrices and highly partitioned domains. However, the deep convergence of the linearized systems was not necessary because of the inaccurate Jacobian operator. With a high-order matrix-free Jacobian, the linear systems were more demanding to solve. The SPAI method did not appear to be robust: the linear tolerance τ_{lin} could not be tightened too much. Otherwise, the maximum number of linear iterations allowed was always reached with a SPAI preconditioner. The lack of consistency between the low-order Jacobian matrix used to construct the SPAI preconditioner and the high-order left-hand side operator could be the reason of this inefficiency. These conclusions lead us to believe that SPAI preconditioning would be most suitable for problems that meet the following conditions:

- The left-hand side operator used in the Newton-Krylov method is the same matrix used to construct the SPAI preconditioner. In this manner, the SPAI preconditioner becomes more effective in reducing the number of linear iterations in the linear solvers.

- The left-hand side operator is an exact, or at least accurate, linearization of the residual **R**. Hence, deep convergences of the linear solvers would be effective in reducing the overall number of nonlinear iterations. Therefore, we could take advantage of the low application cost of the SPAI preconditioner and amortize the computational cost of SPAI with a greater number of linear iterations.

6.2 Future Work

In closing, there are some interesting aspects that could help improve the efficiency of SPAI preconditioning:

- In this work, the SPAI implementation assigns an MPI task to each core. Following the hybrid memory architecture presented in Subsection 1.2, it could be advantageous to use the OpenMP interface between cores belonging to the same node. In this way, some core-to-core communications could be replaced with data retrieval.

- Secondly, the SPAI framework could be further improved with an automatic adjustment of the framework parameters, in line with the convergence of the linearized systems in Newton's method.

- Lastly, we have only considered the use of SPAI preconditioning for multi-CPU architectures. An implementation of SPAI on GPUs in [153] showed a substantial acceleration in its computation to precondition a Bi-CGSTAB method. It would be interesting to study SPAI on GPUs, as the use of GPU-accelerated parallel architectures becomes an active research area in CFD [154, 155].

Bibliography

[1] R. Schaller. "Moore's law: past, present and future". *IEEE Spectrum* 34.6 (1997), pp. 52–59.

[2] D. Etiemble. "45-year CPU evolution: one law and two equations" (2018).

[3] V. Eijkhout. *Introduction to High Performance Scientific Computing*. 2013.

[4] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, and J. Wawrzynek. "A view of the parallel computing landscape". *Communications of the ACM* 52.10 (2009), pp. 56–67.

[5] R. Duncan. "A survey of parallel computer architectures". *Computer* 23.2 (1990), pp. 5–16.

[6] F. Magoules, F.-X. Roux, and G. Houzeaux. *Parallel Scientific Computing*. Computer Engineering Series. Wiley-ISTE, 2015.

[7] L. Dagum and R. Menon. "OpenMP: an industry standard API for shared-memory programming". *IEEE Computational Science and Engineering* 5.1 (1998), pp. 46–55.

[8] D. W. Walker. "The design of a standard message passing interface for distributed memory concurrent computers". *Parallel Computing*. Message Passing Interfaces 20.4 (1994), pp. 657–673.

[9] Á. Fernández-González, R. Rosillo, J. Á. Miguel-Dávila, and V. Matellán. "Historical review and future challenges in Supercomputing and Networks of Scientific Communication". *The Journal of Supercomputing* 71.12 (2015), pp. 4476–4503.

[10] A. Jameson. "Computational Aerodynamics for Aircraft Design". *Science* 245.4916 (1989), pp. 361–371.

[11] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Elsevier, 2015.

[12] A. Pueyo. "An efficient Newton-Krylov method for the Euler and Navier-Stokes equations." PhD Thesis. University of Toronto, 1998.

[13] G. Birkhoff, R. S. Varga, and D. Young. "Alternating Direction Implicit Methods". *Advances in Computers*. Ed. by F. L. Alt and M. Rubinoff. Vol. 3. Elsevier, 1962, pp. 189–273.

[14] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. "Inexact Newton Methods". *SIAM Journal on Numerical Analysis* 19.2 (1982), pp. 400–408.

[15] D. E. Keyes. "Aerodynamic applications of Newton- Krylov-Schwarz solvers". *Fourteenth International Conference on Numerical Methods in Fluid Dynamics*. Ed. by S. M. Deshpande, S. S. Desai, and R. Narasimha. Springer Berlin Heidelberg, 1995, pp. 1–20.

[16] X.-C. Cai, D. E. Keyes, and V. Venkatakrishnan. *Newton-Krylov-Schwarz: An Implicit Solver for CFD*. Tech. rep. Hampton, VA, USA: Institute for Computer Applications in Science and Engineering, 1995.

[17] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems". *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[18] 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 (1986), pp. 856–869.

[19] H. A. 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 (1992), pp. 631–644.

[20] P. Birken, J. Duintjer Tebbens, A. Meister, and M. Tůma. "Preconditioner Updates Applied to CFD Model Problems". *Applied Numerical Mathematics* 58.11 (2008), pp. 1628–1641.

[21] M. Benzi. "Preconditioning Techniques for Large Linear Systems: A Survey". *Journal of Computational Physics* 182.2 (2002), pp. 418–477.

[22] Y. Saad. "Iterative methods for linear systems of equations: A brief historical journey" (2019).

[23] Å. Björck. *Numerical Methods in Matrix Computations*. Vol. 59. Springer, 2015.

[24] G. H. Golub and C. F. V. Loan. *Matrix Computations*. JHU Press, 2013.

[25] V. Puzyrev, S. Koric, and S. Wilkin. "Evaluation of Parallel Direct Sparse Linear Solvers in Electromagnetic Geophysical Problems". *Computers & Geosciences* 89 (2016), pp. 79–87.

[26] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. "Hybrid scheduling for the parallel solution of linear systems". *Parallel Computing* 32.2 (2006), pp. 136–156.

[27] A. Kuzmin, M. Luisier, and O. Schenk. "Fast Methods for Computing Selected Elements of the Green's Function in Massively Parallel Nanoelectronic Device Simulations". *Euro-Par 2013 Parallel Processing*. Ed. by F. Wolf, B. Mohr, and D. an Mey. Lecture Notes in Computer Science. Springer, 2013, pp. 533–544.

[28] M. Blome, H. R. Maurer, and K. Schmidt. "Advances in three-dimensional geoelectric forward solver techniques". *Geophysical Journal International* 176.3 (2009), pp. 740–752.

[29] R. Streich. "3D finite-difference frequency-domain modeling of controlled-source electromagnetic data: Direct solution and optimization for high accuracy". *GEOPHYSICS* 74.5 (2009), F95–F105.

[30] S. Operto, J. Virieux, P. Amestoy, J.-Y. L'Excellent, L. Giraud, and H. B. H. Ali. "3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study". *GEOPHYSICS* 72.5 (2007), SM195–SM211.

[31] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. "A survey of direct methods for sparse linear systems". *Acta Numerica* 25 (2016), pp. 383–566.

[32] 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*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[33] D. M. Young. "A historical overview of iterative methods". *Computer Physics Communications* 53.1 (1989), pp. 1–17.

[34] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2003.

[35] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. 2nd ed. Texts in Applied Mathematics. Springer-Verlag, 2007.

[36] J. Liesen and Z. Strakos. *Krylov Subspace Methods: Principles and Analysis*. Publication Title: Krylov Subspace Methods. Oxford University Press, 2012.

[37] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[38] W. Ford. *Numerical Linear Algebra with Applications*. Elsevier, 2015.

[39] D. P. Arbenz. "Lecture Notes on Solving Large Scale Eigenvalue Problems". Lecture notes. ETH, Zurich, Switzerland, 2012.

[40] Y. Saad and M. H. Schultz. "Conjugate gradient-like algorithms for solving nonsymmetric linear systems". *Mathematics of Computation* 44.170 (1985), pp. 417–424.

[41] H. F. Walker. "Implementations of the GMRES method". *Computer Physics Communications* 53.1 (1989), pp. 311–320.

[42] V. Frayssé, L. Giraud, S. Gratton, and J. Langou. "A Set of GMRES Routines for Real and Complex Arithmetics" (1997).

[43] J. Gatsis. "Preconditioning Techniques for a Newton-Krylov Algorithm for the Compressible Navier-Stokes Equations". PhD Thesis. University of Toronto, 2014.

[44] W. Joubert. "On the convergence behavior of the restarted GMRES algorithm for solving nonsymmetric linear systems". *Numerical Linear Algebra with Applications* 1.5 (1994), pp. 427–447.

[45] M. Embree. "The Tortoise and the Hare Restart GMRES". *SIAM Review* 45.2 (2003), pp. 259–266.

[46] M. Rehman, C. Vuik, and G. Segal. *Solution of the incompressible Navier Stokes equations with preconditioned Krylov subspace methods*. Delft University of Technology, 2006.

[47] P. Chin and P. Forsyth. "A comparison of GMRES and CGSTAB accelerations for incompressible Navier-Stokes problems". *Journal of Computational and Applied Mathematics* 46.3 (1993), pp. 415–426.

[48] A. M. Bruaset. *A Survey of Preconditioned Iterative Methods*. CRC Press, 1995.

[49] M. Ferronato. "Preconditioning for Sparse Linear Systems at the Dawn of the 21st Century: History, Current Developments, and Future Perspectives". *ISRN Applied Mathematics* 2012 (2012), pp. 1–49.

[50] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. San Diego, CA, USA: Academic Press, 2001.

[51] D. Hysom and A. Pothen. "Level-based incomplete LU factorization: Graph model and algorithms". *Lawrence Livermore National Labs, Tech. Rep* (2002).

[52] M. Sedlacek. "Sparse Approximate Inverses for Preconditioning, Smoothing, and Regularization". PhD Thesis. Munich: Technische Universität München, 2012.

[53] Y. Saad and J. Zhang. "BILUTM: A Domain-Based Multilevel Block ILUT Preconditioner for General Sparse Matrices". *SIAM Journal on Matrix Analysis and Applications* 21.1 (1999), pp. 279–299.

[54] Y. Saad. "ILUT: A dual threshold incomplete LU factorization". *Numerical Linear Algebra with Applications* 1.4 (1994), pp. 387–402.

[55] M. Bollhöfer. "A robust ILU with pivoting based on monitoring the growth of the inverse factors". *Linear Algebra and its Applications* 338.1-3 (2001), pp. 201–218.

[56] J. Mayer. "A multilevel Crout ILU preconditioner with pivoting and row permutation". *Numerical Linear Algebra with Applications* 14.10 (2007), pp. 771–789.

[57] E. Chow and Y. Saad. "Experimental study of ILU preconditioners for indefinite matrices". *Journal of Computational and Applied Mathematics* 86.2 (1997), pp. 387–414.

[58] W. D. Gropp and D. E. Keyes. "Domain decomposition methods in computational fluid dynamics". *International Journal for Numerical Methods in Fluids* 14.2 (1992), pp. 147–165.

[59] F. Chalot, G. Chevalier, Q. V. Dinh, and L. Giraud. "Some Investigations of Domain Decomposition Techniques in Parallel CFD". *Euro-Par'99 Parallel Processing*. Ed. by P. Amestoy, P. Berger, M. Daydé, D. Ruiz, I. Duff, V. Frayssé, and L. Giraud. Vol. 1685. Lecture Notes in Computer Science. Springer, 1999, pp. 595–602.

[60] M. J. Gander. "Schwarz Methods over the Course of Time". *Electronic Transactions on Numerical Analysis* 31 (2008), pp. 228–255.

[61] K. Nakajima and H. Okuda. "Parallel Iterative Solvers with Localized ILU Preconditioning for Unstructured Grids on Workstation Clusters". *International Journal of Computational Fluid Dynamics* 12.3-4 (1999), pp. 315–322.

[62] E. Efstathiou. "Study of the Schwarz algorithms: Understanding restricted additive Schwarz". PhD Thesis. McGill University, 2002.

[63] X.-C. Cai and M. Sarkis. "A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems". *SIAM Journal on Scientific Computing* 21.2 (1999). Publisher: Society for Industrial and Applied Mathematics, pp. 792–797.

[64] E. Efstathiou and M. J. Gander. "Why Restricted Additive Schwarz Converges Faster than Additive Schwarz". *BIT Numerical Mathematics* 43.5 (2003), pp. 945–959.

[65] X.-C. Cai, W. D. Gropp, D. E. Keyes, R. G. Melvin, and D. P. Young. "Parallel Newton–Krylov–Schwarz Algorithms for the Transonic Full Potential Equation". *SIAM Journal on Scientific Computing* 19.1 (1998), pp. 246–265.

[66] W. Gropp, D. E. Keyes, L. C. McInnes, and M. D. Tidriri. "Globalized Newton-Krylov-Schwarz Algorithms and Software for Parallel Implicit CFD". *The International Journal of High Performance Computing Applications* 14.2 (2000), pp. 102–136.

[67] J. E. Hicken and D. W. Zingg. "Parallel Newton-Krylov Solver for the Euler equations Discretized Using Simultaneous Approximation Terms". *AIAA Journal* 46.11 (2008), pp. 2773–2786.

[68] V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2015.

[69] M. Sala. "Domain decomposition preconditioners: theoretical properties, application to the compressible Euler equations, parallel aspects" (2003). PhD Thesis. EPFL, p. 201.

[70] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. "Approximating the inverse of a matrix for use in iterative algorithms on vector processors". *Computing* 22.3 (1979), pp. 257–268.

[71] I. Ammara and C. Masson. "Development of a fully coupled control-volume finite element method for the incompressible Navier–Stokes equations". *International Journal for Numerical Methods in Fluids* 44.6 (2004), pp. 621–644.

[72] P. R. McHugh and D. A. Knoll. "Fully Coupled Finite Volume Solutions of the Incompressible Navier–Stokes and Energy Equations Using an Inexact Newton Method". *International Journal for Numerical Methods in Fluids* 19.5 (1994), pp. 439–455.

[73] L. Wigton, N. Yu, and D. Young. "GMRES acceleration of computational fluid dynamics codes". *7th Computational Physics Conference*. Cincinnati, OH, USA: American Institute of Aeronautics and Astronautics, 1985.

[74] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2003.

[75] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. "Globalization Techniques for Newton–Krylov Methods and Applications to the Fully Coupled Solution of the Navier–Stokes Equations". *SIAM Review* 48.4 (2006), pp. 700–721.

[76] S. C. Eisenstat and H. F. Walker. "Choosing the Forcing Terms in an Inexact Newton Method". *SIAM Journal on Scientific Computing* 17.1 (1996), pp. 16–32.

[77] D. A. Knoll and D. E. Keyes. "Jacobian-free Newton–Krylov methods: a survey of approaches and applications". *Journal of Computational Physics* 193.2 (2004), pp. 357–397.

[78] L. Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives". *Pacific Journal of Mathematics* 16.1 (1966), pp. 1–3.

[79] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[80] J. Ortega and W. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Elsevier, 1970.

[81] D. J. Higham. "Trust Region Algorithms and Timestep Selection". *SIAM Journal on Numerical Analysis* 37.1 (1999), pp. 194–210.

[82] T. S. Coffey, C. T. Kelley, and D. E. Keyes. "Pseudotransient Continuation and Differential-Algebraic Equations". *SIAM Journal on Scientific Computing* 25.2 (2003), pp. 553–569.

[83] D. E. Keyes, L. McInnes, and M. Tidriri. "Parallel Implicit PDE Computations: Algorithms and Software" (1997).

[84] O. Reynolds. "IV. On the dynamical theory of incompressible viscous fluids and the determination of the criterion". *Philosophical Transactions of the Royal Society of London. (A.)* 186 (1895), pp. 123–164.

[85] Z. J. Zhai, Z. Zhang, W. Zhang, and Q. Y. Chen. "Evaluation of Various Turbulence Models in Predicting Airflow and Turbulence in Enclosed Environments by CFD: Part 1—Summary of Prevalent Turbulence Models". *HVAC&R Research* 13.6 (2007), pp. 853–870.

[86] P. Spalart and S. Allmaras. "A one-equation turbulence model for aerodynamic flows". *30th Aerospace Sciences Meeting and Exhibit*. Reno, NV, USA: American Institute of Aeronautics and Astronautics, 1992.

[87] A. Bouchard. "Wall Distance Evaluation Via Eikonal Solver for RANS Applications". MA thesis. École Polytechnique de Montréal, 2017.

[88] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*. Vol. 113. Fluid Mechanics and Its Applications. Cham, Switzerland: Springer International Publishing, 2016.

[89] A. Jameson, W. Schmidt, and E. Turkel. "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes". *14th Fluid and Plasma Dynamics Conference*. Palo Alto, CA, USA: American Institute of Aeronautics and Astronautics, 1981.

[90] R. Swanson, R. Radespiel, and E. Turkel. "Comparison of several dissipation algorithms for central difference schemes". *13th Computational Fluid Dynamics Conference*. Snowmass Village, CO, USA: American Institute of Aeronautics and Astronautics, 1997.

[91] T. Coffey, R. J. McMullan, C. T. Kelley, and D. S. McRae. "Globally convergent algorithms for nonsmooth nonlinear equations in computational fluid dynamics". *Journal of Computational and Applied Mathematics*. Proceedings of the International Conference on Recent Advances in Computational Mathematics 152.1 (2003), pp. 69–81.

[92] P. Deuflhard. "Adaptive Pseudo-transient Continuation for Nonlinear Steady State Problems" (2002).

[93] C. T. Kelley and D. E. Keyes. "Convergence Analysis of Pseudo-Transient Continuation". *SIAM Journal on Numerical Analysis* 35.2 (1998), pp. 508–523.

[94] C.-H. Tai, J.-H. Sheu, and P.-Y. Tzeng. "Improvement of explicit multistage schemes for central spatial discretization". *AIAA Journal* 34.1 (1996), pp. 185–188.

[95] N. Y. Gnedin, V. A. Semenov, and A. V. Kravtsov. "Enforcing the Courant-Friedrichs-Lewy Condition in Explicitly Conservative Local Time Stepping Schemes". *Journal of Computational Physics* 359 (2018), pp. 93–105.

[96] P. Eliasson, P. Weinerfelt, and J. Nordström. "Application of a Line-Implicit Scheme on Stretched Unstructured Grids". 2009.

[97] T. Pulliam. "Time accuracy and the use of implicit methods". *11th Computational Fluid Dynamics Conference*. Orlando, FL, USA: American Institute of Aeronautics and Astronautics, 1993.

[98] D. E. Keyes and V. Venkatakrishnan. "Newton-Krylov-Schwarz Methods: Interfacing Sparse Linear Solvers with Nonlinear Applications". *Journal of Applied Mathematics and Mechanics* 76 (1996), pp. 147–150.

[99] J. S. Cagnone, K. Sermeus, S. K. Nadarajah, and E. Laurendeau. "Implicit multigrid schemes for challenging aerodynamic simulations on block-structured grids". *Computers & Fluids* 44.1 (2011), pp. 314–327.

[100] T. H. Pulliam and J. L. Steger. "Implicit Finite-Difference Simulations of Three-Dimensional Compressible Flow". *AIAA Journal* 18.2 (1980), pp. 159–167.

[101] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996.

[102] J. Hales, S. Novascone, R. Williamson, D. Gaston, and M. Tonks. "Solving Nonlinear Solid Mechanics Problems with the Jacobian-Free Newton Krylov Method". *Computer Modeling in Engineering and Sciences* 84.2 (2012), pp. 123–154.

[103] P. Orkwis and K. Vanden. "On the accuracy of numerical versus analytical Jacobians". *32nd Aerospace Sciences Meeting and Exhibit*. Reno, NV, USA: American Institute of Aeronautics and Astronautics, 1994.

[104] H.-B. An, J. Wen, and T. Feng. "On finite difference approximation of a matrix-vector product in the Jacobian-free Newton–Krylov method". *Journal of Computational and Applied Mathematics* 236.6 (2011), pp. 1399–1409.

[105] J. K. Cullum and M. Tůma. "Matrix-free preconditioning using partial matrix estimation". *BIT Numerical Mathematics* 46.4 (2006), pp. 711–729.

[106] J. M. Donato. *A comparison of iterative methods for a model coupled system of elliptic equations*. Tech. rep. ORNL/TM-12447. Oak Ridge National Lab., TN, USA, 1993.

[107] F. Pacull, S. Aubert, and M. Buisson. "A Study of ILU Factorization for Schwarz Preconditioners with Application to Computational Fluid Dynamics". Stirlingshire, UK: Civil-Comp Press, 2011.

[108] K. Mohamed, K. Sermeus, E. Laurendeau, and S. Nadarajah. "Implementation and Validation of Detached Eddy Simulation in the Bombardier Navier-Stokes Flow Solver" (2009).

[109] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.11. Argonne National Laboratory, 2019.

[110] A. Rebaine, F. Fortin, and A. Benmeddour. "Parallelization of a finite volume CFD code". *Workshops on Mobile and Wireless Networking/High Performance Scientific, Engineering Computing/Network Design and Architecture/Optical Networks Control and Management/Ad Hoc and Sensor Networks/Compile and Run Time Techniques for Parallel Computing ICPP 2004*. Montreal, QC, Canada: IEEE, 2004, pp. 207–213.

[111] K. Sermeus, E. Laurendeau, and F. Parpia. "Parallelization and Performance Optimization of Bombardier Multiblock Structured Navier-Stokes Solver on IBM Eserver Cluster 1600". *45th AIAA Aerospace Sciences Meeting and Exhibit*. Reno, NV, USA: American Institute of Aeronautics and Astronautics, 2007.

[112] E. de Sturler. "Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer". *Applied Numerical Mathematics*. Special Issue on Massively Parallel Computing and Applications 19.1 (1995), pp. 129–146.

[113] G. Alléon, M. Benzi, and L. Giraud. "Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics". *Numerical Algorithms* 16.1 (1997), pp. 1–15.

[114] B. Carpentieri, I. S. Duff, L. Giraud, and M. M. m. Made. "Sparse symmetric preconditioners for dense linear systems in electromagnetism". *Numerical Linear Algebra with Applications* 11.8-9 (2004), pp. 753–771.

[115] M. Bhuruth, M. K. Jain, and A. Gopaul. "Preconditioned iterative methods for the nine-point approximation to the convection–diffusion equation". *Journal of Computational and Applied Mathematics* 138.1 (2002), pp. 73–92.

[116] G. Larrazábal, J. M. Cela, G. An, and L. Abal. "Study Of SPAI Preconditioners For Convective Problems" (1999).

[117] M. Benzi and M. Tůma. "A comparative study of sparse approximate inverse preconditioners". *Applied Numerical Mathematics* 30.2-3 (1999), pp. 305–340.

[118] M. J. Grote and T. Huckle. "Parallel Preconditioning with Sparse Approximate Inverses". *SIAM Journal on Scientific Computing* 18.3 (1997), pp. 838–853.

[119] P. O. Frederickson. *Fast approximate inversion of large sparse linear systems*. Lakehead University, Department of Mathematical Sciences, 1975.

[120] M. Benson, J. Krettmann, and M. Wright. "Parallel algorithms for the solution of certain large sparse linear systems". *International Journal of Computer Mathematics* 16.4 (1984), pp. 245–260.

[121] J. D. F. Cosgrove, J. C. Díaz, and A. Griewank. "Approximate inverse preconditionings for sparse linear systems". *International Journal of Computer Mathematics* 44.1-4 (1992), pp. 91–110.

[122] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. "Optimization of sparse matrix-vector multiplication on emerging multicore platforms". *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. 2007, pp. 1–12.

[123] E. Chow. "A Priori Sparsity Patterns for Parallel Sparse Approximate Inverse Preconditioners". *SIAM Journal on Scientific Computing* 21.5 (2000), pp. 1804–1822.

[124] K. Wang, S. Kim, and J. Zhang. "A Comparative Study on Dynamic and Static Sparsity Patterns in Parallel Sparse Approximate Inverse Preconditioning". *Journal of Mathematical Modelling and Algorithms* 2.3 (2003), pp. 203–215.

[125] S. T. Barnard, L. M. Bernardo, and H. D. Simon. "An MPI Implementation of the SPAI Preconditioner on the T3E:" *The International Journal of High Performance Computing Applications* (1999).

[126] N. I. M. Gould and J. A. Scott. "Sparse Approximate-Inverse Preconditioners Using Norm-Minimization Techniques". *SIAM Journal on Scientific Computing* 19.2 (1998), pp. 605–625.

[127] S. T. Barnard and M. J. Grote. "A Block Version of the SPAI Preconditioner." *PPSC*. 1999.

[128] S. T. Barnard, R. L. Clay, and M. K. Chancellor. "A portable MPI implementation of the SPAI preconditioner in ISIS++". Minneapolis, MN, USA, 1997.

[129] Y. Saad. *SPARSKIT: A basic tool kit for sparse matrix computations*. Tech. rep. 1990.

[130] N. Goharian, A. Jain, and Q. Sun. "Comparative Analysis of Sparse Matrix Algorithms for Information Retrieval". *Journal of Systemics, Cybernetics and Informatics* 1 (2003).

[131] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson. "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks". *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures - SPAA '09*. Calgary, AB, Canada: ACM Press, 2009, p. 233.

[132] T. Huckle, A. Kallischko, and M. Sedlacek. *MSPAI - TUM*. URL: https://www5.in.tum.de/wiki/index.php/MSPAI (visited on 06/12/2020).

[133] S. Booth and E. Mourao. "Single sided MPI implementations for SUN MPI". *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. 2000, pp. 2–2.

[134] *MPI Forum*. URL: https://www.mpi-forum.org (visited on 07/01/2020).

[135] T. A. Davis and Y. Hu. "The university of Florida sparse matrix collection". *ACM Transactions on Mathematical Software* 38.1 (2011), pp. 1–25.

[136] V. Deshpande, M. J. Grote, P. Messmer, and W. Sawyer. "Parallel implementation of a sparse approximate inverse preconditioner". *Parallel Algorithms for Irregularly Structured Problems*. Lecture Notes in Computer Science. Springer, 1996, pp. 63–74.

[137] R. Nabben. "Decay Rates of the Inverse of Nonsymmetric Tridiagonal and Band Matrices". *SIAM Journal on Matrix Analysis and Applications* 20.3 (1999), pp. 820–837.

[138] W. Briggs, V. Henson, and S. McCormick. *A Multigrid Tutorial, Second Edition*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2000.

[139] X.-C. Cai, W. D. Gropp, D. E. Keyes, and M. D. Tidriri. "Newton-Krylov-Schwarz Methods in CFD". *Numerical methods for the Navier-Stokes equations: Proceedings of the International Workshop Held at Heidelberg, October 25–28, 1993*. Notes on Numerical Fluid Mechanics (NNFM). Wiesbaden, 1994, pp. 17–30.

[140] *Cedar - Compute Canada Doc*. URL: https://docs.computecanada.ca/wiki/Cedar (visited on 07/06/2020).

[141] *Béluga - Compute Canada Doc*. URL: https://docs.computecanada.ca/wiki/B%C3%A9luga/en (visited on 06/21/2020).

[142] A. Kashi. *FVENS, Finite volume Euler/Navier-Stokes solver*. URL: https://github.com/Slaedr/FVENS (visited on 06/25/2020).

[143] J. Rivera Jose, B. Dansberry, R. Bennett, M. Durham, and W. Silva. "NACA 0012 benchmark model experimental flutter results with unsteadypressure distributions". *33rd Structures, Structural Dynamics and Materials Conference*. Structures, Structural Dynamics, and Materials and Co-located Conferences. American Institute of Aeronautics and Astronautics, 1992.

[144] P. L. Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". *Journal of Computational Physics* 43.2 (1981), pp. 357–372.

[145] C. Ansorge. *Analyses of Turbulence in the Neutrally and Stably Stratified Planetary Boundary Layer*. Springer, 2016.

[146] F. Magoulès, F.-X. Roux, and G. Houzeaux. "Computer Architectures". *Parallel Scientific Computing*. John Wiley & Sons, Ltd, 2015.

[147] J. Erhel. "A parallel GMRES version for general sparse matrices". *ETNA. Electronic Transactions on Numerical Analysis* 3 (1995), pp. 160–176.

[148] R. D. da Cunha and T. Hopkins. "A parallel implementation of the restarted GMRES iterative algorithm for nonsymmetric systems of linear equations". *Advances in Computational Mathematics* 2.3 (1994), pp. 261–277.

[149] Y. Saad. *Krylov Subspace Methods in Distributed Computing Environments*. Army High Performance Computing Research Center, 1992.

[150] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode. "Predicting the energy and power consumption of strong and weak scaling HPC applications". *Supercomputing Frontiers and Innovations* 1.2 (2014), pp. 20–41.

[151] B. Eisfeld. "ONERA M6 wing". *FLOMANIA — A European Initiative on Flow Physics Modelling*. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer, 2006, pp. 219–224.

[152] J. Vassberg, M. Dehaan, M. Rivers, and R. Wahls. "Development of a Common Research Model for Applied CFD Validation Studies". *26th AIAA Applied Aerodynamics Conference*. Honolulu, Hawaii, USA: American Institute of Aeronautics and Astronautics, 2008.

[153] M. Lukash, K. Rupp, and S. Selberherr. "Sparse Approximate Inverse Preconditioners for Iterative Solvers on GPUs". *Proceedings of the 2012 Symposium on High Performance Computing*. HPC '12. Orlando, Florida. San Diego, CA, USA: Society for Computer Simulation International, 2012.

[154] X. Liu, Z. Zhong, and K. Xu. "A hybrid solution method for CFD applications on GPU-accelerated hybrid HPC platforms". *Future Generation Computer Systems* 56 (2016), pp. 759–765.

[155] Y. Xiang, B. Yu, Q. Yuan, and D. Sun. "GPU Acceleration of CFD Algorithm: HSMAC and SIMPLE". *Procedia Computer Science*. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland 108 (2017), pp. 1982–1989.