UNIVERSITÉ CLAUDE BERNARD

Numéro attribué par la bibliothèque
230-2011

THÈSE

pour obtenir le grade de

Docteur de Université Claude Bernard

Spécialité : **Mathématiques Appliquées**

préparée au laboratoire : **Institut Camille Jordan - UMR 5208**

préparée dans le cadre de l'**École Doctorale InfoMaths**

présentée et soutenue publiquement par

Thomas Dufaud

le 25 Novembre 2011

Titre :

Contribution au développement du préconditionnement Aitken Schwarz Additif Restreint et son application aux systèmes linéaires issus de la différentiation automatique des solutions de Navier-Stokes dépendant des paramètres de la simulation.

English Title:

Contribution to the development of Aitken Restricted Additive Schwarz preconditioning and application to linear systems arising from automatic differentiation of compressible Navier-Stokes solution with respect to the simulation's parameters.

Jury

Rapporteurs :	M. Luc Giraud	-	INRIA-CERFACS
	M. Eric de Sturler	-	Virginia Tech
Président :	Mme. Jocelyne Erhel	-	INRIA
Examinateurs :	M. Stéphane Aubert	-	FLUOREM
Directeur de Thèse :	M. Damien Tromeur-Dervout	-	ICJ

Ce manuscrit est rédigé en anglais. Cependant, il comporte une version française de l'introduction et de la conclusion. Chaque chapitre débute par un résumé en français.

This manuscript is written in English. Nevertheless, a French version of the introduction and the conclusion is provided. Each chapter begins with a summary written in French.

Acknowledgments

Je souhaite remercier l'ensemble des membres du jury d'avoir accepté d'évaluer mon travail. Parmis eux, je remercie tout d'abord Eric de STURLER et Luc GIRAUD qui m'ont fait l'honneur d'être les rapporteurs de cette thèse. Je leur témoigne toute ma gratitude pour le temps passé à la revue de ce manuscrit, pour leurs critiques avisées et l'intérêt qu'ils ont porté à mon travail. Je souhaite également adresser mes sincères remerciements à Jocelyne ERHEL et Stéphane AUBERT pour leur présence au sein de ce jury.

Mes remerciements vont vers les membres du projet LIBRAERO financé par l'ANR. Je souhaite remercier l'entreprise FLUOREM en les personnes de Stéphane AUBERT et François PACULL pour avoir apporté les problèmatiques motivant les innovations présentes dans les travaux de cette thèse. Merci pour votre disponibilité et pour la qualité des échanges que nous avons pu avoir au cours de ces trois années de collaborations. Merci à Jocelyne ERHEL pour les échanges que nous avons pu avoir. Une pensée également vers Désiré NUENTSA-WAKAM qui tout comme moi a effectué son doctorat sur ce projet et qui a été confronté à ces problèmatiques.

Je souhaite exprimer toute ma gratitude envers Damien TROMEUR-DERVOUT, mon directeur de thèse, qui a su me guider et me conseiller. Vous m'avez laissé la liberté de proposer et d'exploiter des idées qui ont fini par porter leur fruit tout en encourageant le travail de production. J'ai beaucoup appris à travers nos échanges et à votre contact.

Merci à Laurent BERENGUER qui a su fournir un travail de qualité ayant permis de finaliser une partie du travail que j'avais entrepris en début de thèse.

Ces trois ans de thèse n'aurait pas été aussi agréables et fructueux sans la présence de plusieurs de mes collègues partageant les locaux du bâtiment ISTIL. Merci à Naïma, Olivier, Daniel, Farid, Toan, Jonathan, Anne et Bérangère que ce soit pour leurs discussions autour d'un café ou d'un repas, pour les rencontres sportives autour d'une table de billard, dans une piscine, ou une salle de badminton, pour les débats animés à la pause de midi, et enfin pour leur soutien et leurs encouragements à la fin de la thèse.

Dans les bons comme dans les mauvais moments j'ai pu compter sur le soutien de ma famille et de mes amis. Merci donc à mes parents, Hervé et Brigitte, ma sage-femme de soeur, Laetitia, et mon cuisto de frère, Nicolas qui ont toujours pensé à moi et m'ont soutenu. Merci à Charlotte pour m'avoir soutenu dans la vie et les études depuis mes premiers pas dans les grandes études. Un grand merci à Mathieu et Aurélien pour leur soutien indéfectible et les bons moments que l'on sait passer ensemble.

Je dédie cette thèse à Yves, mon grand-père, qui un jour m'a donné l'idée sans laquelle je n'aurais pas suivi ce parcourt scolaire et écrit ces pages.

Contents

Chapter 1

Introduction

1.1 French version - Version française

L'objectif principal de cette étude est la conception d'une méthode de préconditionnement novatrice permettant d'appréhender de grands systèmes linéaires provenant de différents problèmes de la physique, sans avoir connaissance des équations sous-jacentes et de la méthode de discrétisation employée. Bien que ceci soit ambitieux, il est possible de trouver un préconditionneur usuel et robuste et de l'améliorer selon des considérations algébriques. Cela permet d'avoir un préconditionneur robuste qui peut être utilisé par défaut dans un logiciel industriel. Différentes techniques existent déjà mais peuvent présenter des difficultés numériques sur des problèmes complexes venant, par exemple, de l'industrie du Calcul en Dynamiques des Fluides (CFD). Le sujet de recherche s'oriente vers une classe particulière de systèmes linéaires apparaissant dans des applications industrielles. Une étude préliminaire présente les principales difficultés et donne les idées générales pour envisager de nouvelles techniques. Une combinaison de différentes idées et méthodes mène au développement de la technique de préconditionnement d'Aitken Schwarz Additif Restreint. Le manuscrit présente les développements effectués sur ce préconditionneur, de la théorie à son application, avec un intérêt particulier quant à la réalisation d'un tel préconditionneur sur les architectures parallèles.

D'année en année, dans le domaine du Calcul en Dynamiques des Fluides (CFD), les entreprises sont confrontées à des problèmes de plus en plus complexes. Plus une simulation doit être proche de la réalité, plus le modèle doit être riche en terme de dimension, de nombre de variables à expliquer, ou du domaine géomètrique à décrire. La difficulté induite devrait dégrader la phase de résolution d'un code de CFD. Les logiciels doivent donc évoluer afin de produire des résultats toujours précis, aussi rapidement que possible. C'est dans ce contexte que cette étude est proposée. Plus précisément, notre but est de développer des méthodes permettant de traiter des systèmes linéaires apparaissant dans un logiciel développé par l'entreprise FLUOREM, qui travaille sur la paramétrisation d'écoulement à l'état d'équilibre. Dans un premier temps, ce code réalise une simulation RANS à l'équilibre sur une configuration de référence, en résolvant l'équation :

$$F(q(p_{ref}), p_{ref}) = 0$$

Où $q(p)$ est le champ de l'écoulement à exprimer, qui dépend des paramètres p. Alors, il calcule les dérivées $q^{(1)}$,$q^{(2)}$,...,$q^{(n)}$ conformément aux paramètres p. Les

dérivées du premier ordre s'écrivent :

$$\begin{aligned} \frac{\partial F}{\partial q}(q,p)\cdot q^{(1)}\cdot\Delta p &= -\frac{\partial F}{\partial p}(q,p)\cdot\Delta p \\ G(q,p)\cdot q^{(1)}\cdot\Delta p &= R(q,p,\Delta p) \end{aligned}$$

Et celles d'ordres élevés s'écrivent :

$$G\cdot q^{(n)}\cdot\Delta p = R^{(n-1)} - \sum_{i=1}^{i=n-1} C_{n-1}^{i} G^{(i)}\cdot q^{(n-i)}\cdot\Delta p$$

Cet opérateur G est la matrice du système linéaire à résoudre. Puis, les champs d'écoulement $q(p_{ref}+\Delta p)$ sont extrapolés en utilisant les dérivées données par le logiciel Turb'Opty©

$$q(p_{ref}+\Delta p) = q(p_{ref}) + q^{(1)}\cdot\Delta p + \cdots + \frac{q^{(n)}}{n!}\cdot\Delta p^{n} + O(\Delta p^{n+1})$$

Les équations de Navier-Stokes compressibles (RANS) conduisent à des systèmes linéaires avec des matrices réelles, carrées et indéfinies. Ces matrices G, générées par la différentiation automatique du processus de résolution de l'écoulement autour d'un état d'équilibre, correspondent à la Jacobienne de F par rapport aux variables d'écoulement conservatives des équations discrétisées (discrétisation Volumes-Finis). Le second membre, représente la dérivée des équations considérant un paramètre.

Nous proposons, ici, une présentation de trois cas de la collection de matrices du projet. La plupart de ces matrices sont disponibles sur internet, dans la collection de matrices creuses de (Davis & Hu 20YY)(`http://www.cise.ufl.edu/research/sparse/matrices`). Elles possèdent une structure bloc et ne sont pas symétriques. Dans la liste suivante, nous présentons un cas $1D$ dénommé CASE_001 DK01, un cas $2D$ dénommé CASE_004 GT01 et un cas $3D$ dénommé CASE_017 RM07. Le profil creux ainsi que les principales caractéristiques de ces matrices sont présentés dans la Figure 1.1 et dans la Table 1.1.

- *L'opérateur CASE_001 DK01 provient d'un cas turbulent $1D$. La solution du système discret est définie avec sept variables par noeud. La discrétisation $1D$ des équations aux dérivées partielles utilise un schéma à cinq points. La discrétisation est effectuée sur 129 noeuds espacés de manière non uniforme. La matrice est alors penta-diagonale par bloc et chaque bloc est de taille sept. Dès lors, le problème à résoudre est un système linéaire de 903 équations algébriques réelles. La matrice n'est pas symétrique. Dans cette matrice de test, chaque bloc diagonal est lié au bloc diagonal précédent et au bloc diagonal suivant. Ce qui correspond aux 14 lignes de la matrice précédant le bloc diagonal et aux 14 lignes de la matrice suivant le bloc diagonal, l'ordonnancement des noeuds étant cohérent avec la distribution spatiale $1D$ des noeuds. Concernant la physique du problème, l'écoulement stationnaire sur lequel la matrice repose est dominé par l'advection, caractérisée par un nombre de Mach autour de 0.3.*

- *L'opérateur CASE_ 004 GT01 provient d'un cas* $2D$ *non-visqueux dans le contexte d'une turbine en cascade linéaire. La solution du système discret est définie avec cinq variables par noeud. La discrétisation est effectuée sur 1596 noeuds, décrivant un canal inter-lame. La maille de calcul résultant du schéma convectif utilise neuf noeuds. Il y a donc neuf blocs d'éléments non-nuls dans la matrice pour les inconnues de l'écoulement associées à chaque noeud du maillage. La particularité de cette discrétisation, est que le domaine de calcul est périodique, introduisant dans la structure de la matrice des éléments non-nuls très éloignés de la diagonale principale. La matrice obtenue est de taille 7980 et n'est pas symétrique.*

- *Enfin une autre catégorie de problème réside dans les cas* $3D$ *provenant de la stratégie de ce type de CFD. Nous considérons ici le CASE_ 017 RM07 qui représente un cas* $3D$ *visqueux avec une turbulence dite "gelée" où la géométrie considérée est un compresseur de moteur jet. Le problème est discrétisé sur 54527 noeuds. Sept variables par noeud sont considérées. La matrice obtenue est de taille* 381689 *avec* 37464962 *éléments non-nuls. La matrice n'est pas symétrique.*

Les matrices sont totalement non-symétriques. En fait, pour toutes ces matrices, le nombre d'éléments non diagonaux et non nuls correspondant au nombre total d'éléments non diagonaux est égal à zéro.

Les profils creux des matrices de la collection de FLUOREM, nous poussent à choisir des méthodes de résolution itératives telles que les méthodes de Krylov. Les valeurs propres des cas $1D$ *et* $2D$*, tracées en Figure 1.2, présentent un spectre complexe étendu. Toutes les valeurs propres ont une partie réelle positive. Certaines valeurs propres sont très éloignées d'un groupe de valeurs propres près de l'origine du plan complexe. Une telle distribution du spectre est un problème pour les méthodes itératives de type Krylov. Une attention particulière doit être portée vers le choix d'un préconditionneur.*
Initialement, des tests ont été effectués en considérant la combinaison d'une méthode GMRES avec une factorisation Incomplète LU. Mais, ceci a révélé que cette stratégie impliquait un trop grand nombre de remplissage pour un résultat qui varie beaucoup trop d'un cas à l'autre.
La Figure 1.3 montre l'historique de convergence de la résolution des cas $1D$ *et* $2D$ *avec un GMRES complet, c'est à dire sans* restart*, et avec ou sans préconditionnement MILU, avec l'implémentation MATLAB. Le paramètre de tolérance de saut, identifié par la variable* drop tolerance*, de la méthode MILU est affecté de la valeur* 0.1*. Pour chaque cas, on peut observer que le nombre de vecteurs de Krylov à garder est élevé. Le préconditionnement MILU est efficace, dans une certaine mesure, sur le cas* $1D$ *mais conduit à détériorer la convergence du GMRES sur le cas* $2D$*.*

Ce problème de consommation d'espace mémoire pour ces préconditionneurs de type ILU peut être évité par le choix d'une méthode de préconditionnement issue de la décomposition de domaine. Les méthodes de décomposition de domaine de type

Schwarz donnent de bons préconditionneurs avec recouvrement. La décomposition de domaine peut être produite par un partitionneur tel que METIS ou sa version parallèle, PARMETIS (Karypis et al. 2003). L'inverse local peut être approximé par une technique de factorisation incomplète mais les mêmes comportements que pour la factorisation de l'opérateur complet peuvent être observés.

L'expérience montre que la factorisation Incomplète LU n'est pas un choix robuste puisque les usuels facteurs d'amélioration, tel que l'ordre de remplissage, n'améliorent pas le préconditionnement et peuvent même dégrader sa qualité. Considérant notre problème, il n'y a aucune raison pour que la méthode ILU procure une bonne approximation de l'opérateur. Cependant, la décomposition de domaine donne la possibilité de réaliser des factorisations complètes des opérateurs locaux moins coûteuses que pour le domaine complet. Donc, pour plus de robustesse, il serait bénéfique d'utiliser une factorisation complète des systèmes locaux quand cela est possible.

Cette présentation rapide nous amène à suivre une idée qui est fondée sur la résolution de ces systèmes linéaires par une méthode de type GMRES préconditionnée par une technique de décomposition de domaine de type Schwarz, avec une résolution "exacte" des systèmes locaux provenant de la décomposition de domaine. Mais, le fait d'utiliser une résolution directe "exacte" du système local, engendre une augmentation de la consommation de mémoire comparée à une méthode avec factorisation incomplète. Nous concentrons alors nos efforts vers une manière d'améliorer une technique de décomposition de domaine de type Schwarz qui peut être extensible pour les problèmes $3D$. La structure bloc des matrices, le fait que celles-ci ne proviennent pas d'équations aux dérivées partielles classiques, et l'observation de la part du partenaire industriel que ces matrices présentent souvent un comportement proche de celui des opérateurs hyperboliques discrets, ne nous permettent pas d'envisager l'utilisation des méthodes multi-grilles basées sur des aggrégations algébriques par exemple. Dès lors, il convient de considérer une méthode multi-niveaux incluant une représentation de l'interface.

La plupart des méthodes de préconditionnement émergeant des méthodes de décomposition de domaine appliquées aux problèmes elliptiques sont améliorées en utilisant une approximation de l'opérateur de Steklov-Poincaré défini sur l'interface (voir le chapitre 2). La récente technique d'Aitken-Schwarz permet l'accélération des méthodes de décomposition de domaine de type Schwarz. Bien que cette technique repose sur une amélioration de la convergence par une approximation de l'opérateur de Steklov-Poincaré, elle ne requiert pas le calcul de l'opérateur mais utilise ses effets sur la convergence à l'interface. Et elle n'utilise que les effets de l'opérateur de Steklov-Poincaré sur les éléments d'une base qui ne convergent pas aussi rapidement que souhaité. Ainsi, cette méthode donne la possibilité de représenter la solution sur l'interface dans un espace généré par un petit nombre de vecteurs de base comparé à la taille de l'interface. Donc, d'aucun peut considérer la méthode d'Aitken-Schwarz comme un bon choix pour résoudre les problèmes $3D$ avec des interfaces de grande taille.

La Table 1.2 illustre l'accroissement de l'interface générée par la décomposition de l'opérateur CASE_017 RM07 en utilisant le partitionneur PARMETIS avec un recouvrement égal à un. Ceci montre que les cas industriels que nous devons traiter peuvent avoir un nombre de dépendances de données entre les sous-domaines plus grand que la taille du problème global. Donc, l'accélération d'Aitken dans une base de petite taille, appropriée, devrait être un bon choix.

Cependant, il semble hasardeux d'utiliser une décomposition de domaine lorsque l'interface devient plus grande que le domaine. Généralement, dans la plupart des bibliothèques de calcul scientifique parallèle, l'implémentation d'une méthode de type Schwarz consiste à attribuer une résolution locale à un unique processeur. Pour les problèmes 3D que nous devons résoudre, cette stratégie de distribution ne sera pas probante. De fait, le nombre de partitions doit être petit comme le suggère la Table 1.2. Ainsi, la taille des systèmes locaux devient grande, conduisant à un accroissement de la mémoire requise par un processeur et une augmentation des temps de calcul. On peut espérer trouver une technique pour effectuer ces résolutions local en parallèle.

Nous concentrons notre recherche sur le développement d'un préconditionneur robuste avec de bonnes propriétés de convergence, pour résoudre de grands systèmes linéaires par une méthode itérative de type Krylov, sans connaissance des équations sous-jacentes et du schéma de discrétisation. Ce préconditionneur reposera sur une technique de décomposition de domaine à cause de la grande taille et de la complexité du problème que nous voulons résoudre. Les systèmes locaux doivent être résolus par une méthode de résolution directe ou une méthode itérative puisque nous ne connaissons pas de critères pour choisir les bons paramètres pour la factorisation incomplète. L'effet d'une application du préconditionneur doit être amélioré dans le but de réduire le nombre d'itérations de la méthode de krylov choisie pour résoudre le système global.

Nous proposons de présenter les développements de cette technique au travers de quatre chapitres. Dans le chapitre 2, nous exposons un état de l'art des techniques de préconditionnement fondées sur des méthodes de décomposition de domaine de type Schwarz et du complément de Schur. Après une présentation des méthodes de résolution itératives de type Krylov pour les matrices non symétriques, nous présentons la méthode de Schwarz utilisée comme méthode de résolution afin de montrer ses propriétés et différentes techniques pour améliorer sa convergence. Puis nous présenterons des méthodes de préconditionnement de type Schwarz et Schur. Plus précisément, l'objectif principal de cet état de l'art est de présenter les différentes approches et préconditionneurs développés par la communauté dans le but de produire un préconditionnement efficace en faisant une approximation de l'opérateur de Steklov-Poincaré.

La technique d'accélération d'Aitken, qui semble avoir de bonnes propriétés aux vues des problèmes des cas de FLUOREM, n'a jamais été employée comme technique de précondtionnement. Le chapitre 3 est dédié à la mise en oeuvre d'une écriture formelle du couplage entre la formule de l'accélération d'Aitken et le précondition-

neur Schwarz Additif Restreint. Après une présentation de l'accélération d'Aitken dans différentes bases, nous présentons comment écrire formellement un préconditionneur Aitken Schwarz Additif Restreint (ARAS) sous la forme d'un préconditionneur multi-niveaux. Une étude de convergence de cette nouvelle technique de préconditionnement est proposée afin d'enrichir la compréhension du mécanisme du préconditionnement ARAS. Les illustrations des propositions et du théorème provenant de l'étude de convergence sont effectuées sur des cas académiques $2D$*. Le chapitre conclut sur une étude du coût d'une telle méthode en terme de nombre d'opérations.*

La méthode que nous avons proposée est dédiée à la résolution de grands systèmes sans connaissance du maillage et des équations sous-jacentes. La méthode basée sur la méthode d'Aitken-Schwarz s'adapte parfaitement aux architectures parallèles. Le chapitre 4, énumère les différents points à prendre en compte dans la conception d'une méthode de résolution d'Aitken-Schwarz afin de concevoir un préconditionneur de type ARAS. Dès lors, deux implémentations sont présentées. Dans un premier temps, une méthode de résolution de type Aitken-Schwarz est conçue pour la résolution de grands systèmes linéaires issus de la discrétisation des équations de Darcy $3D$ *avec un champ de perméabilité distribué aléatoirement. Ce code de Calcul Haute Performance (HPC) est développé pour étudier la méthode avec différentes bases d'approximation lorsqu'elle est appliquée comme méthode de résolution. Dans un second temps, une implémentation de la technique de préconditionnement ARAS est proposée. La finalité de ce travail étant un transfert technologique vers les partenaires industriels, les développements du préconditionneur sont effectués dans l'environnement de la bibliothèque de calcul PETSc. L implémentation de ces deux codes diffère car nous ne bénéficions pas de la connaissance de la géomètrie des maillages pour distribuer les calculs dans les problèmes de FLUOREM (PETSc). Le partitionnement doit donc être effectué algébriquement. Des développements originaux sur la bibliothèque PETSc sont proposés dans le but de doter la bibliothèque d'une implémentation MPI à deux niveaux pour son préconditionneur de type Schwarz.*

Enfin, dans le chapitre 5, la technique de préconditionnement ARAS est testée sur des cas industriels de la collection de matrices de FLUOREM et sur les problèmes de Darcy $3D$*. Les premiers tests sont effectués sur des cas industriels* $2D$ *plus grands que ceux présentés dans cette introduction, afin de montrer les performances numériques, de consommations en mémoire, de temps de calcul, de la technique ARAS sur des systèmes linéaires d'envergure moyenne (de l'ordre de la centaine de milliers d'inconnues) pour les stratégies d'implémentation choisies. Fort de ces premières observations, on étudie l'extensibilité de la méthode aux problèmes* $3D$ *en effectuant un test de performance de la méthode d'Aitken-Schwarz avec une approche algébrique de l'accélération, sur les problèmes de Darcy* $3D$ *(de l'ordre de plusieurs millions à plusieurs centaines de millions d'inconnues). Les deux codes présentés sont impliqués dans cette étude permettant de mettre en évidence les écueils à éviter et les bénéfices de l'utilisation de ces techniques. Finalement, le préconditionneur ARAS est testé sur un cas* $3D$ *de CFD proposé par FLUOREM.*

Nous concluons cette étude par les perspectives de ce travail.

1.2 English version - Version anglaise

The main interest of the following study is the design of an innovative preconditioning technique which is able to deal with large sparse linear systems coming from different kind of physics, without knowledge of the underlying equations and the way they were discretised. Although it is very ambitious, it is possible to find a functional and robust preconditioning technique and to work on an algebraic enhancement of the chosen method. This allows us to have a robust preconditioner which can be used by default in industrial software. Different techniques already exist but can present numerical difficulties on complex problems coming from the Computational Fluid Dynamics (CFD) industry, for example. The research focuses on a particular class of linear systems appearing in an industrial application. A preliminary study points out the principal difficulties and gives global ideas to investigate new techniques. A combination of different ideas and methods leads to the development of the Aitken Restricted Additive Schwarz preconditioning technique. The manuscript focuses on the development of this preconditioner, from the theory to the application with special care to the design of such a method on parallel architectures.

Every year, in the domain of Computational Fluid Dynamics (CFD), industries have to deal with more complex problems. The more a simulation needs to reflect reality, the richer the model must be in terms of dimensions, number of variables to explain or domain geometry to describe. Resulting software has to evolve in order to still provide accurate results as fast as possible. The difficulty induced should degrade the solution phase of a CFD code. It is in this context that the following study is proposed. More precisely, our goal is to develop efficient methods to deal with linear systems appearing in software, developed by the company FLUOREM, which deals with steady flow parametrization. As a first step, the code performs a steady RANS simulation on a reference configuration solving the equation:

$$F(q(p_{ref}), p_{ref}) = 0 \tag{1.1}$$

where $q(p)$ is the flow field to find, which depends on the parameters p. Then it computes the derivatives $q^{(1)}$,$q^{(2)}$,...,$q^{(n)}$ with respect to the parameters p. The first-order derivatives are written as:

$$\begin{aligned} \frac{\partial F}{\partial q}(q,p)\cdot q^{(1)}\cdot \Delta p &= -\frac{\partial F}{\partial p}(q,p)\cdot \Delta p \\ G(q,p)\cdot q^{(1)}\cdot \Delta p &= R(q,p,\Delta p) \end{aligned} \tag{1.2}$$

And the high order derivatives write:

$$G\cdot q^{(n)}\cdot \Delta p = R^{(n-1)} - \sum_{i=1}^{i=n-1} C_{n-1}^{i} G^{(i)}\cdot q^{(n-i)}\cdot \Delta p \tag{1.3}$$

This operator G is the matrix involved in the linear system to solve. Then, flow fields $q(p_{ref} + \Delta p)$ are extrapolated using the derivatives given by the

Turb'Opty©software:

$$q(p_{ref} + \Delta p) = q(p_{ref}) + q^{(1)} \cdot \Delta p + \cdots + \frac{q^{(n)}}{n!} \cdot \Delta p^n + O(\Delta p^{n+1}) \tag{1.4}$$

The compressible Navier-Stokes equations (RANS) lead to linear systems with real, square and indefinite matrices. Those matrices G, generated through automatic differentiation of the flow solver around a steady state, correspond to the Jacobian with respect to the conservative fluid variables of the discretized governing equations (finite-volume discretization). The right hand side represents the derivative of the equations with respect to a parameter (of operation or shape).

We propose, here, an overview of three cases of the matrix collection of the project. Most of those matrices are available on the internet on the sparse matrix collection of (Davis & Hu 20YY) (`http://www.cise.ufl.edu/research/sparse/matrices`). They have a block structure and are not symmetric. We present in the following list, a $1D$ case denoted by CASE_001 DK01, a $2D$ case denoted by CASE_004 GT01 and a $3D$ case called CASE_017 RM07. The sparse profile and the main characteristics of those matrices are presented in the Figure 1.1 and in the Table 1.1.

- The CASE_001 DK01 operator comes from a $1D$ turbulent case. The solution of the discrete system is defined over seven variables per node. The $1D$ discretisation of the partial differential equations uses a five-point stencil. The discretisation is done among 129 nodes, non-uniformly spaced. Then, the matrix is block penta-diagonal and each block is of size seven. Then the problem to solve is a linear system of 903 real algebraic equations. The matrix is not symmetric. In this test matrix, each diagonal block is related to two up- and two down-stream neighbouring nodes, corresponding respectively to the 14 upper and 14 lower matrix rows, the node ordering being coherent with the 1D spatial node distribution. Concerning the physics, the stationary flow on which the matrix is based, is dominated by advection, characterized by a Mach number around 0.3.

- The CASE_004 GT01 operator comes from a $2D$ inviscid case in the context of a linear cascade turbine. The solution of the discrete system is defined over five variables per node. The discretisation is done among 1596 nodes, describing one inter-blade channel. The stencil involved by the convective scheme uses nine nodes. Thus, there are nine non-zero blocks for each node in the matrix. The peculiarity is that the computational domain is periodic, which introduces some non-zero elements far away from the diagonal. The resulting matrix is of size 7980 and not symmetric.

- Finally another issue is the $3D$ cases that result from the CFD strategy. We consider CASE_017 RM07 which represents a $3D$ viscous case with a "frozen" turbulence. Here, the geometry is a jet engine compressor. The problem is

discretized among 54527 nodes. Seven variables per node are considered. The resulting matrix is of size 381689 with 37464962 non-zeros. The matrix is not symmetric.

The matrices are all asymmetric. Indeed, for all these matrices, the number of matched off-diagonal non-zeros over the total number of off-diagonal entries is equal to zero.

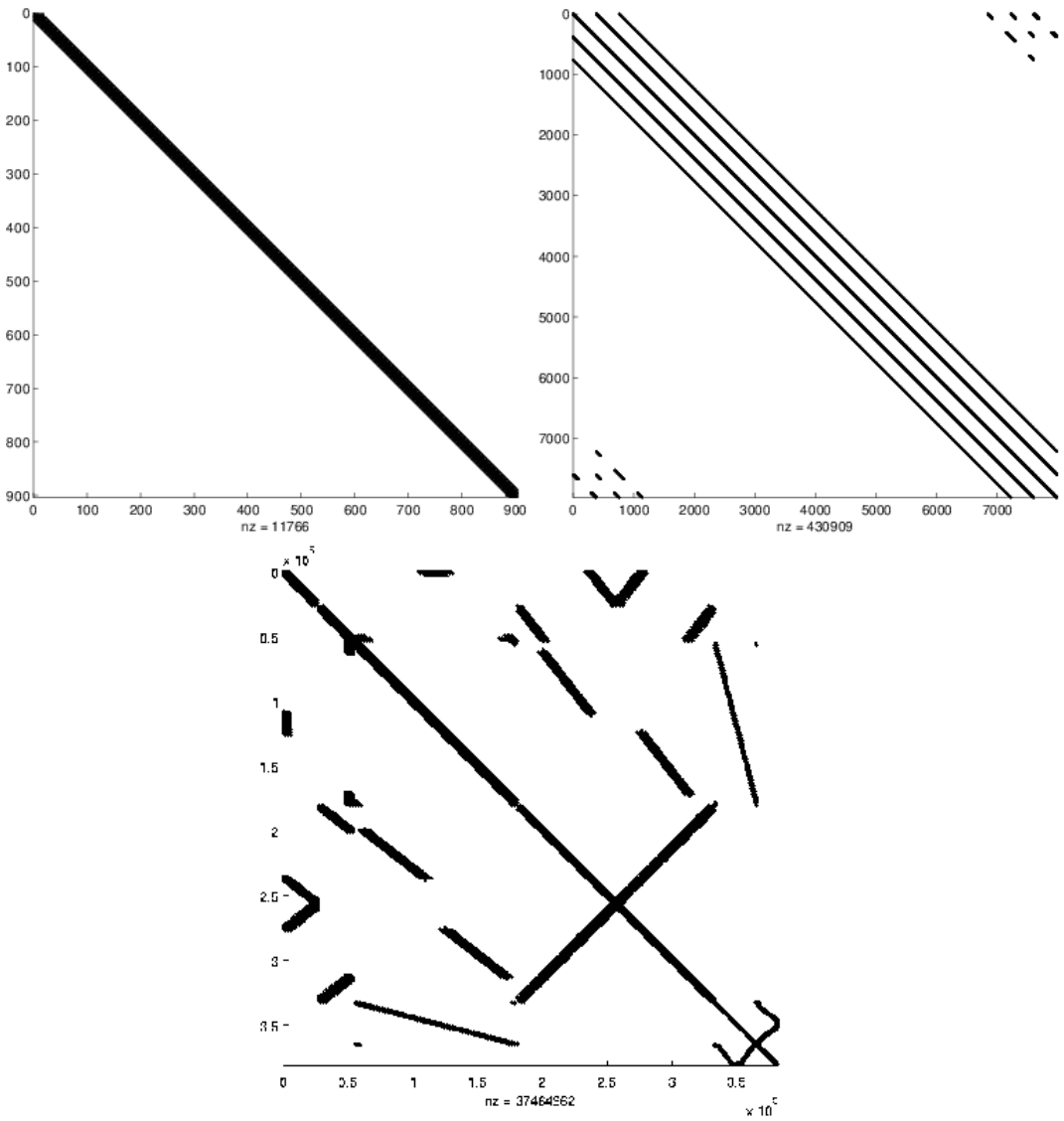

Figure 1.1: Matrix profile of (top left) CASE_001 DK01, (top right) CASE_004 GT01 and (bottom) CASE_017 RM07.

The sparsity pattern of the matrix from the FLUOREM collection spurs us on to

case	m	nnz	Symmetric	Positive Definite
CASE_001 DK01	903	11 766	NO	NO
CASE_004 GT01	7980	430 909	NO	NO
CASE_017 RM07	381 689	37 464 962	NO	NO

Table 1.1: Main characteristics of three operators of the FLUOREM collection sparse matrix collection. m is the global size of the linear system, nnz is the number of non-zeros element of the matrix.

choose iterative methods such as Krylov methods. The eigenvalues of the $1D$ case and the $2D$ case, plotted in Figure 1.2, present a large complex spectrum. All the eigenvalues have a positive real part. Some eigenvalues are highly distant to a group of eigenvalues close to the origin of the complex plane. Such a distribution of the spectrum is an issue for Krylov iterative methods. Therefore, special care has to be given to the choice of a preconditioner.

Initially, some tests were performed with the combination of a Generalized Minimal Residual Method (GMRES) and an Incomplete LU (ILU) factorization, but it appeared that this strategy implies too large of a filling number for a result which varies too much from one case to another.

Figure 1.3 shows the convergence history of the solution of the $1D$ case and the $2D$ case with a full GMRES with a MILU preconditioning or no preconditioning with the MATLAB implementation. The "drop tolerance" parameter of the MILU is set to 0.1. For each case, we can see that the number of Krylov vectors to keep is high. The MILU preconditioning is efficient for the $1D$ case but leads to a worse convergence for the $2D$ case.

The problem of memory consumption for the preconditioner can be avoided by choosing a preconditioning technique coming from the domain decomposition field. The Schwarz domain decomposition methods provide good overlapping preconditioners. The domain decomposition can be given by a partitioner such as METIS or its parallel version, PARMETIS (Karypis *et al.* 2003). The local inverse can be approximated by an incomplete factorization technique but the same behaviour as for the factorization of the complete operator should be observed.

The experience shows that the ILU factorization is not a robust choice since the traditional enhancement factors, such as the filling order, do not improve the preconditioning and can deteriorate it. Considering our problem there is no reason for ILU to provide a good approximation of the operator. Nevertheless, the domain decomposition gives us the possibility to perform cheaper complete factorization of the local systems than for a complete domain solution. Hence, for robustness, it should be better to use a complete factorization of the local systems when it is possible.

This quick overview leads us to follow an idea which is founded on the solution of those linear systems by a GMRES solver, preconditioned by a Schwarz domain

decomposition technique, with an "exact" solution on the local systems arising from the decomposition. However, if we use the exact solution of the local system, we would expect to increase the memory consumption compared to a method with incomplete factorization. We therefore focus on a way to enhance a Schwarz domain decomposition technique which can be scalable for $3D$ problems. The block structure of the matrix, the fact that it is not derived from classic PDEs, and the observation from the industrial partner that they often show some kind of hyperbolic behavior, do not allow us to use multigrid methods based on algebraic aggregations, for example. We should then consider a multilevel method involving a representation of the interface.

Most of the preconditioning techniques coming from domain decomposition methods applied to elliptic problems are enhanced using an approximation of the Steklov-Poincaré operator defined on the interface (see Chapter 2). Recently, the Aitken-Schwarz technique has enabled the acceleration of Schwarz domain decomposition methods. Although this technique is based on an enhancement of the convergence by an approximation of the Steklov-Poincaré operator, it does not require the computation of the operator but uses its effects on the convergence on the interface. Also, it uses only the effects of the Steklov-Poincaré operator on elements of a base which do not converge as fast as expected. This method allows us, then, to represent the interface's solution in a space span by a small number of base's vectors compared to the size of the interface. Hence, one can consider the Aitken-Schwarz method as a good choice to solve large $3D$ problems with large interface.

In Table 1.2, we illustrate the growth of the interface generated by the decomposition of the CASE_017 RM07 operator using a PARMETIS partitioning with an overlap set to one. It shows that the industrial cases we have to deal with can have a number of data dependencies between sub-domains greater than the size of the global domain itself. Hence, the Aitken acceleration performed in an appropriate small base should be a good choice.

p	n	n/m
3	54 243	0.1421
6	118 398	0.3102
12	208 341	0.5458
24	335 384	0.8787
36	448 707	1.1756
72	670 768	1.7574

Table 1.2: Size n of the interface generated by the PARMETIS partitioning into p partitions of CASE_017 RM07. n/m is the ratio between n and the global size m of the linear system.

Nevertheless, it seems hazardous to deal with domain decomposition when the interface appears to be larger than the domain. Generally, in most of the available parallel libraries, parallel implementation of Schwarz preconditioning technique

consists on computing each local solution on one process. For the $3D$ case we have, this strategy will not be efficient. Actually, the number of partitions needs to be low, according to the illustration of the Table 1.2. Then the size of the local systems to solve becomes large, leading to large memory and time consumption. One can expect to find a way to parallelize the local solutions.

We focus our research on the development of a robust preconditioner with good convergence properties, to solve large sparse linear systems with a Krylov type iterative method, without knowledge of the underlying equations and the discretization scheme. This preconditioner will be based on domain decomposition techniques due to the large size and complexity of the problem we want to solve. The local systems have to be solved with a direct solver or an iterative solver since we do not have any criteria for choosing the proper parameters to perform an incomplete factorization. The effect of an application of the preconditioner needs to be enhanced in order to reduce the number of iterations of the Krylov method chosen to solve the global linear system.

We propose to present the development of this preconditioning technique in four chapters. In Chapter 2, we expound the state of the art in preconditioning techniques based on Schwarz and Schur complement -type domain decomposition methods. After a presentation of Krylov-type iterative solvers for non-symmetric matrices, we focus on the Schwarz domain decomposition method used as a solver in order to exhibit its properties and different techniques used to enhanced its convergence. Then, an overview of the Schwarz and Schur preconditiong methods is given. More precisely, the principal interest is to present the different approaches and preconditioners developed by the community in order to produce efficient preconditioning, making an approximation of the Steklov-Poincaré operator.

The Aitken acceleration technique, which seems to have good properties considering the issues of the FLUOREM cases, has never been used as a preconditioning technique. Chapter 3 is dedicated to setting up a formal writing of the coupling between the Aitken acceleration formula and the Restricted Additive Schwarz preconditioner. After a presentation of the Aitken acceleration in different bases, we show how to formally write an Aitken Restricted Additive schwarz preconditioner as a multilevel preconditioner. A convergence study of this new preconditioning technique is proposed in order to improve the understanding of the mechanism of the ARAS preconditioning. Illustrations of the highlighted propositions and theorem are provided on $2D$ academic cases. The chapter ends with a study about the computing cost of such a method.

The method we proposed is dedicated to the solution of large systems without knowledge of the mesh and the underlying equations. The method based on the Aitken-Schwarz method perfectly fits parallel architecture. Chapter 4, at first, focuses on the different points to take into account to design an Aitken-Schwarz solver and, by consequence, to design an Aitken Restricted Additive Schwarz (ARAS) preconditioner. Then, two implementations are presented. On the one hand, an Aitken-Schwarz solver is designed to solve large sparse linear systems coming from

a discretization of the $3D$ Darcy equations with randomly distributed permeability fields. This High Performance Computing (HPC) code is developed in order to study the method with different bases when it is applied as a solver. On the second hand, an implementation of the ARAS preconditioning technique is proposed. Motivated by the fact that our work should be used by industrial partners, the developments of the ARAS preconditioner are written in the PETSc framework. The orientation of the implementation differs from the previous one because the decomposition is done algebraically without knowledge of any mesh. Original developments to PETSc are proposed in order to make the library support a two-level MPI implementation of its Schwarz preconditioners.

Finally, in Chapter 5, the ARAS preconditioning technique is tested on industrial cases coming from the FLUOREM test cases collection and on the $3D$ Darcy problems. The first tests are performed on $2D$ industrial cases larger that the one presented in this introduction. These tests are done in order to show the numerical performances of the ARAS technique on medium cases and in order to show the memory and time consumption coming from the choices in the implementation strategy. Considering that issues for $3D$ problems which are more complicated to deal with, we propose a performance test of the Aitken Schwarz method, with an algebraic approach of the acceleration on the $3D$ Darcy problem. The two codes presented before are involved in this study, highlighting the different issues to avoid and the benefits which can be done. In the final section, the preconditioner is tested on $3D$ CFD cases proposed by FLUOREM.

We conclude the study with a discussion on prospective works.

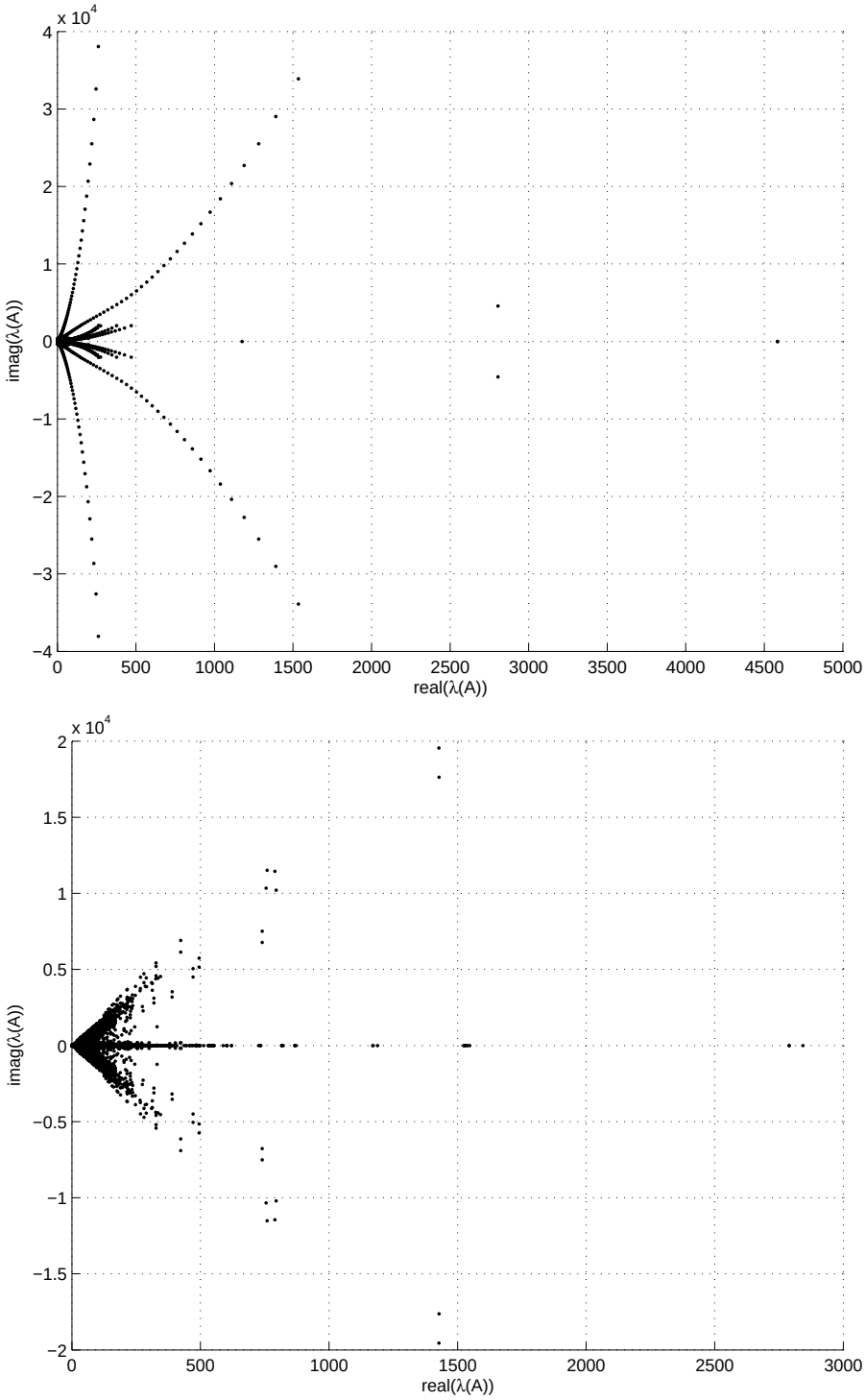

Figure 1.2: Eigenvalues of (top) CASE_001 and (bottom) CASE_004.

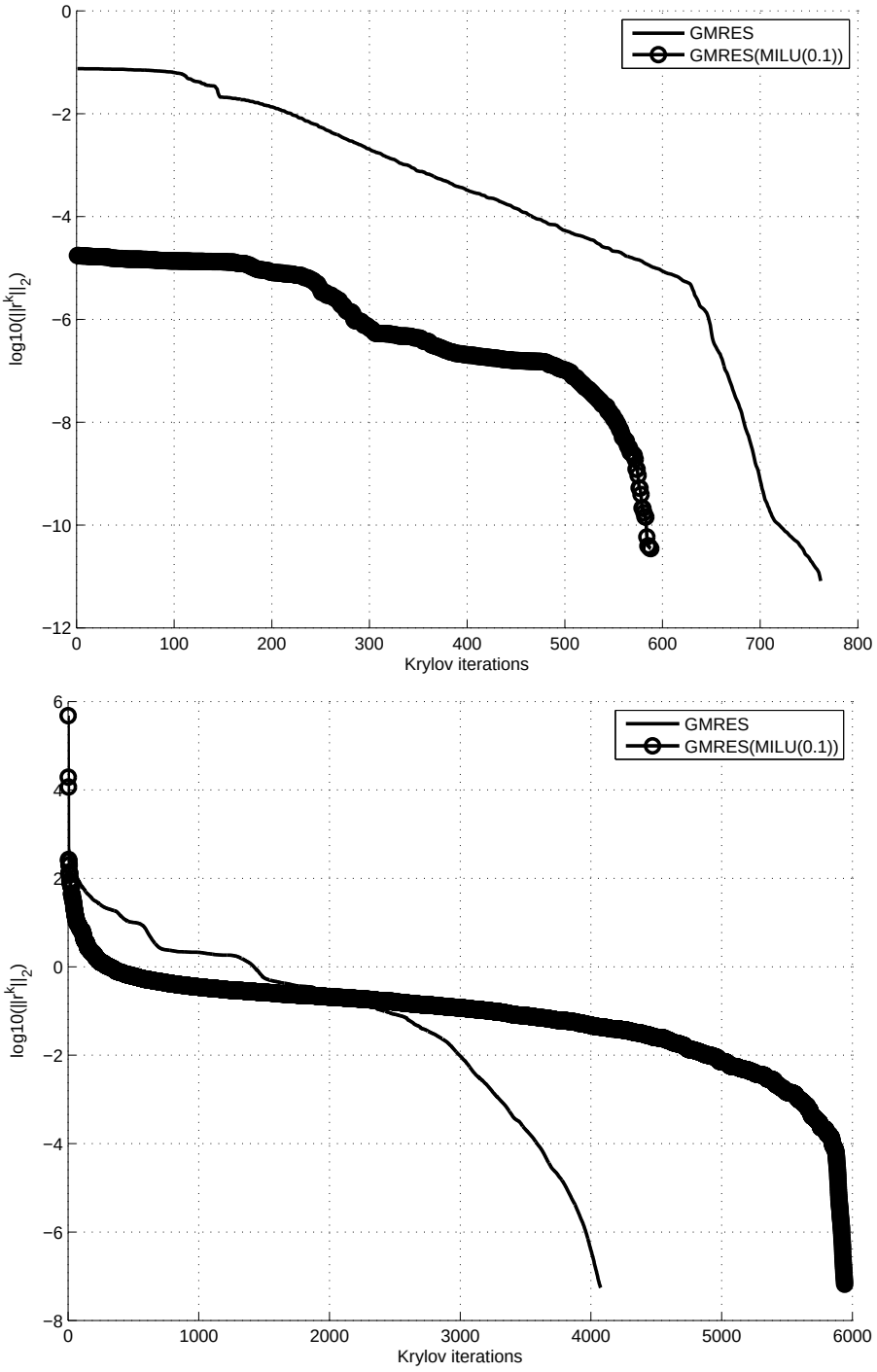

Figure 1.3: Solving (top) CASE_001 and (bottom) CASE_004 with a GMRES method with MILU or without preconditioning.

Chapter 2

State of the art of parallel solution and preconditioning methods for sparse linear systems

French summary - Résumé en français

État de l'art des méthodes de résolution parallèles et de préconditionnement pour les systèmes linéaires creux

L'objectif de ce chapitre est de donner un apperçu des techniques de préconditionnement reposant sur les méthodes de décomposition de domaine de type Schwarz et complément de Schur afin de situer les contributions de cette thèse. Cette présentation se veut non-exhaustive. Nous ne présenterons pas, notamment, les méthodes basées sur les préconditionneurs polynômiaux (voir (van Gijzen 1995)), les techniques de préconditionnement ILU (voir (Meijerink & van der Vorst 1981, Beauwens 2004, de Sturler 1995)), et les techniques de préconditionnement tirant avantage de la structure creuse et régulière de la matrice du problème discrétisé. On peut également mentionner les méthodes tenant compte de la séparabilité de l'opérateur tel que le préconditionnement ADI (Direction Alternée Implicite) (Starke 1994), ou le préconditionnement basé sur PSCR (Résolution Partielle de Réduction Cyclique) (Martikainen et al. 2002) qui ont prouvés leurs efficacités numérique et parallèle lors de leur application aux problèmes de Poisson.

Nous considérons, ici, le système linéaire :

$$Ax = f. \tag{2.1}$$

avec $A \in \mathbb{R}^{n\times n}$, $b, x \in \mathbb{R}^n$.
Dans une première section nous présentons les méthodes de résolution de système linéaire creux utilisées dans cette thèse. Elle consiste en deux classes de méthodes itératives : les méthodes de Krylov appliquées aux matrices non symétriques et non positives définies d'une part, et les méthodes de résolution de décomposition de domaine de type Schwarz d'autre part.

Une méthode de Krylov est initiée avec une condition initiale arbitraire $x_0 \in \mathbb{R}^n$*, et calcule le résidu* $r_0 = f - Ax_0$*. La méthode repose sur deux principes clefs : la minimisation d'une fonction* $F(x)$ *définie sur un sous-espace* $K_m(A, r_0) \stackrel{def}{=} \{r_0, Ar_0, \ldots, A^{m-1}r_0\} = \{v_0, v_1, \ldots, v_{m-1}\}$ *de* $\mathbb{R}^n$ *et une relation d'orthoganilité entre les directions de descentes* v_j*, selon un produit scalaire défini, qui génèrent le sous-espace* $K_m(A, r_0)$*. La conséquence de ces propriétés est la convergence théorique de la méthode de Krylov considérée en, au plus, n itérations, du fait de la propriété d'algèbre* $K_{n-1}(A, r_0) = \mathbb{R}^n$*. Cette propriété peut ne pas être vérifiée si la condition d'orthogonalité n'est pas numériquement satisfaite. Pour la plupart des problèmes, la convergence est atteinte en un nombre d'itérations inférieur à n, mais ce nombre d'itérations dépend de propriétés numériques telles que l'amplitude des valeurs propres de A et leur distribution.*
Comme les systèmes linéaires que nous proposons d'étudier ne présentent pas de matrice A symétrique ou positive définie, nous nous tournons vers les méthodes de Krylov de type GCR (Résidu Conjugué Généralisé) (Eisenstat et al. 1983) ou GMRES (Résidu Minimum Généralisé) (Saad & Schultz 1986). Les fonctionnelles à minimiser ainsi que la propriété d'orthoganilité de ces méthodes sont présentées dans la Table 2.1.
Les algorithmes de ces méthodes sont présentés. Une technique pour s'affranchir des rotations de Given est présentée dans (Saad 2003). Des versions dites flexible *sont introduites dans les travaux de (Saad 1993, Vuik 1995, de Sturler 1996, Frayssé et al. 2009, Giraud et al. 2010) pour la méthode GMRES.*
Concernant la convergence de ces méthodes, il apparaît que celle-ci dépend du nombre de conditionnement $\kappa(A) = ||A||\ ||A^{-1}||$*. Dans (Eisenstat et al. 1983) le taux de convergence de GCR est défini par l'équation* (2.3)*.*
La méthode GMRES est quand à elle mathématiquement équivalente à l'algorithme ORTHORES développé dans (Young & Jea 1980). Si A est positive définie, alors sont taux de convergence suit l'équation (2.3)*. Sinon, quand A est diagonalisable, i.e.* $A = X\Lambda X^{-1}$*, la convergence de la méthode GMRES dépend de la distribution des valeurs propres. et du nombre de conditionnement* $\kappa(X)$*. Le taux de convergence est défini par l'équation* (2.4)*.*
Il est à noter que lorsque plusieurs second membres sont à considérer pour un même opérateur, des techniques de projection sur l'espace généré par les directions de descente déjà calculé peuvent améliorer la convergence (Parks et al. 2006, Tromeur-Dervout & Vassilevski 2006).

Parmi les méthodes de décomposition de domaine appréhendées comme méthodes de résolution, nous nous intéressons aux méthodes de type Schwarz. Nous présentons tout d'abord la Méthode de Schwarz Alternée Généralisée (GSAM) présentée dans (Engquist & Zhao 1998) qui regroupe plusieurs techniques (Quarteroni & Valli 1999). L'algorithmique de la méthode est présentée sous deux déclinaisons, une version additive (Algorithme 3) et une version multiplicative (Algorithme 4). Puis, les différents choix de paramètres possibles conduisent

à l'identification d'une famille de méthodes de décomposition de domaine de Schwarz. Nous en identifions certaines dans la Table 2.2 issues des travaux de (St-Cyr et al. 2007, Marini & Quarteroni 1989, Lions 1988).
Puis nous présentons les propriétés de convergence de la méthode GSAM. Les travaux de (Engquist & Zhao 1998) montrent q'un choix approprié d'opérateurs Λ_i permet à la méthode de converger en deux itérations. Cette propriété est liée à la théorie de l'opérateur de correspondance Dirichlet-Neumann. Cet opérateur linéaire associe les conditions aux limites de Dirichlet, du problème défini par une Équations aux Dérivées Partielles (PDE) avec conditions aux limites homogènes, à ses conditions aux limites de Neumann (dérivée co-normal de la PDE). Ces opérateurs ne sont pas locaux à un sous-domaine, mais relient l'ensemble des sous-domaines. Dans la pratique on peut utiliser des approximations définies algébriquement de ces opérateurs (Chevalier & Nataf 1998, Gander et al. 2002, Gerardo-Giorda & Nataf 2005).
Dans la méthodologie de Aitken-Schwarz, présentée par (Garbey & Tromeur-Dervout 2001, Garbey & Tromeur-Dervout 2002), on considère seulement les propriétés de convergence de la méthode de Schwarz utilisée. Par conséquent, aucune approximation directe de l'opérateur Dirichlet-Neumann n'est effectuée. Lorsque les opérateurs locaux sont des opérateurs linéaires, on peut montrer que la méthode GSAM a une convergence dite purement linéaire (voir [(Engquist & Zhao 1998),p.354]).

Il a été évoqué que le taux de convergence d'une méthode de Krylov, telle que GCR ou GMRES (Eisenstat et al. 1983), lors de la résolution de système linéaire $Au = f, A = (a_{ij}) \in \mathbb{R}^{m\times m}, u \in \mathbb{R}^m, f \in \mathbb{R}^m$, dépend de la distribution des valeurs propres de la matrice considérée, comme le montrent les équations (2.3) *et* (2.4)*. Plus généralement, ce taux de convergence se dégrade avec l'augmentation du nombre de conditionnement de la matrice A. Les méthodes de décomposition de domaine, comme celle de type Schwarz, peuvent être utilisées également comme technique de préconditionnement d'une méthode de Krylov. Dans une seconde partie, nous présentons certaines de ces méthodes de préconditionnement.*

La décomposition de domaine est effectuée à partir du graphe d'adjacence $G = (W, E)$ de A où $W = \{1, 2, ..., m\}$ et $E = \{(i, j) : a_{ij} \neq 0\}$ sont les sommets et les arrêtes de G. À partir d'une partition sans recouvrement, $W = \cup_{i=1}^{p} W_{i,0}$ et $\delta \geq 0$, donnée, la partition avec recouvrement, $\{W_{i,\delta}\}$, est obtenue en définissant p partitions $W_{i,\delta} \supset W_{i,\delta-1}$ en incluant toutes les arrêtes voisines immédiates des arrêtes de la partition $W_{i,\delta-1}$. On construit un opérateur de restriction $R_{i,\delta} : W \to W_{i,\delta}$ définissant l'opérateur local $A_{i,\delta} = R_{i,\delta} A R_{i,\delta}^T, A_{i,\delta} \in \mathbb{R}^{m_{i,\delta}\times m_{i,\delta}}$ sur $W_{i,\delta}$.

Le préconditionneur AS s'écrit alors: $M_{AS,\delta}^{-1} = \sum_{i=1}^{p} R_{i,\delta}^T A_{i,\delta}^{-1} R_{i,\delta}$.

Avec la matrice de restriction $\tilde{R}_{i,\delta}$, sur un sous-domaine sans recouvrement $W_{i,0}$, introduite dans (Cai & Sarkis 1999) on peut écrire le préconditionneur Schwarz Ad-

dititif Restreint (RAS) $M^{-1}_{RAS,\delta} = \sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta}$ *(cf.* (2.22)*). Cette méthode converge plus rapidement que AS selon les travaux de (Efstathiou & Gander 2003) sur les problèmes de Poisson. On obtient un meilleur préconditionnement qui dépend du nombre de sous-domaines.*

Comme il a été présenté pour GSAM, RAS a une convergence/divergence purement linéaire lorsqu'elle est appliquée à des problèmes linéaires. Celle-ci peut alors être améliorée avec des conditions aux limites optimisées donnant la méthode ORAS de (St-Cyr et al. 2007).

La méthode de SchurRAS a été introduite par (Li & Saad 2006) et est basée sur la factorisation incomplète des opérateurs locaux de la méthode RAS.

La méthode de sous-structuration par patch, patch substructuring*, introduite par (Gander et al. 2007) consiste à introduire un recouvrement dans la technique du complément de Schur. Il est à noter que son équivalence avec les méthodes de Schwarz avec recouvrement a été démontrée.*

Enfin nous proposons d'étudier une forme de préconditionnement deux niveaux. Le complément de Schur est impliqué dans la conception des techniques SchurRAS et patch substructuring*. Ces techniques conservent la localité des données afin de réduire les communications globales entre les sous-domaines. Cependant, du fait de cette localité conservée, leur efficacité diminue lorsque le nombre de sous-domaines augmente, comme pour RAS. Un travail permettant de tenir compte de tous les sous-domaines dans le complément de Schur est la méthode de sous-structuration avec un préconditionneur adapté pour l'équation réduite (Bramble et al. 1986, Carvalho et al. 2000, Carvalho et al. 2001, Khoromskij & Wittum 2004). On décrit la méthode itérative de sous-structuration associée à un préconditionnement du problème sur l'interface du complément de Schur par une méthode de Schwarz Additif présentée par (Carvalho et al. 2000).*

Notre approche est dans la lignée de ce type de préconditionneur à deux niveaux travaillant sur l'interface. Mais nous choisissons de ne pas résoudre de problème sur l'interface en utilisant uniquement l'information sur le domaine complet avec une connaissance a-posteriori de l'opérateur de correspondance Dirichlet-Neumann qui repose sur la convergence/divergence purement linéaire de RAS pour définir l'espace grossier.

The objective of this chapter is to give an overview of preconditioning techniques based on domain decomposition methods of type Schwarz and Schur complement in order to highlight the contributions of this thesis. We do not intend to list all the methods, notably those based on polynomial preconditioning techniques (see (van Gijzen 1995)), or the ILU preconditioning technique (see (Meijerink & van der Vorst 1981, Beauwens 2004, de Sturler 1995)) and the preconditioning techniques that take advantage of the sparse and regular structure of the

matrix of the discretized problem. One can cite methods based on the separability of the operator such as ADI (Alternated Direction Implicit) preconditioning technique (Starke 1994) or the preconditioning techniques based on PSCR (Partial Solution of Cyclic Reduction) (Martikainen *et al.* 2002) that have proved their numerical and parallel efficiency when they are applied to Poisson's problems.

2.1 About the choice of a solution method

2.1.1 Krylov methods for non-symmetric and non-positive definite matrices

Let us first introduce the Krylov methods used in the thesis in order to solve:

$$Ax = f. \tag{2.2}$$

with $A \in \mathbb{R}^{n\times n}$, $b, x \in \mathbb{R}^n$.

A Krylov iterative method is initiated from an arbitrary initial condition $x_0 \in \mathbb{R}^n$, and then computes the residual $r_0 = f - Ax_0$. It can then be summarized with the two following features: the minimisation of a function $F(x)$ defined on a subspace $K_m(A, r_0) \stackrel{def}{=} \{r_0, Ar_0, \ldots, A^{m-1}r_0\} = \{v_0, v_1, \ldots, v_{m-1}\}$ of $\mathbb{R}^n$ and an orthogonality relation, with respect to a scalar product, between the directions of descent v_j that generate the subspace $K_m(A, r_0)$.

The consequence of these properties is the theoritical convergence of the Krylov method in at most n iterations, due to the algebraic property $K_{n-1}(A, r_0) = \mathbb{R}^n$. This property can fail if the orthogonality condition is not numerically satisfied. For most of problems, the convergence is achieved with a number of iterations less than n, but this number of iterations depends of the numerical properties such as the eigenvalues' amplitude and distribution of the matrix A.

2.1.1.1 Generalized Conjugate Residual and Generalized Minimal Residual methods

In the problems that we have to solve, the matrix A is not positive definite and not symmetric. These lacks of properties lead us to focus on GCR (Generalized Conjugate Residual) (Eisenstat *et al.* 1983) and GMRES (General Minimum Residual) (Saad & Schultz 1986) which are two Krylov methods. For GCR and GMRES, the functional to minimize and the orthogonality property that must be satisfied are shown in Table 2.1.

Krylov Method	Function to minimize	scalar product
GCR	$F(x_k) = (f - Ax_k, f - Ax_k)$	$(A., A.)$
GMRES	$F(x_k) = (f - Ax_k, f - Ax_k)$	$(.,.)$

Table 2.1: Function to minimize and scalar product for GCR and GMRES Krylov methods

The GCR algorithm is written as:

Algorithm 1 GCR (Saad 2003)

1: Compute $r_0 = b - Ax_0$. Set $v_0 = r_0$.
2: For $j = 0, 1, 2, \ldots,$ until convergence Do:
3: $\quad \alpha_i = \dfrac{(r_j, Av_j)}{(Av_j, Av_j)}$
4: $\quad x_{j+1} = x_j + \alpha_j v_j$
5: $\quad r_{j+1} = r_j - \alpha_j Av_j$
6: $\quad$ Compute $\beta_{ij} = -\dfrac{(Ar_{j+1}, Av_i)}{(Av_i, Av_i)}$, for $i = 0, 1, \ldots, j$
7: $\quad v_{j+1} = r_{j+1} + \sum_{j=0}^{j} \beta_{ij} v_i$
8: Enddo

The GMRES (Arnoldi version) algorithm is written as:

Algorithm 2 GMRES (Saad 2003)

1: Compute $r_0 = b - Ax_0$. $\beta = ||r_0||_2$, Set $v_1 = r_0/\beta$.
2: For j=1,...,m Do
3: Compute $\omega_j = Av_j$
4: For i=1,...,j Do
5: $h_{i,j} = (w_j, v_i)$
6: $w_j = w_j - h_{i,j} v_i$
7: Enddo i
8: $h_{j+1,j} = ||w_j||_2$, If $h_{j+1,j} = 0$ set m=j and exit
9: $v_{j+1} = w_j / h_{j+1,j}$
10: Enddo j
11: Form the matrix $V_m = [v_1, \ldots, v_m]$, set $e_1 = (1, 0, .., 0)^t \in \mathbb{R}^m$
12: Apply Given rotations to transform the Hessenberg matrix $\bar{H}_m = (h_{ij})_{1 \leq j \leq m+1, 1 \leq j \leq m}$ in an upper triangular matrix H_m.
13: Compute y_m the minimizer of $||\beta e_1 - H_m y||_2$ and $x_m = x_0 + V_m y_m$

The GMRES algorithm involves Given's rotations. They can be replaced by Householder transformations to transform the Hessenberg matrix $\bar{H}_m$ into an upper triangular matrix H_m (Saad 2003). To be complete, the flexible version of this algorithm has been introduced in (Saad 1993, Vuik 1995, de Sturler 1996, Frayssé *et al.* 2009, Giraud *et al.* 2010) for GMRES.

2.1.1.2 Convergence rates of the GCR and GMRES algorithms

The major drawback of the GCR and GMRES methods is their convergence rate that depends on the conditioning number $\kappa(A) = ||A||\ ||A^{-1}||$. In (Eisenstat *et al.* 1983)

the convergence rate of the GCR method was derived as:

$$||r_{i1}||_2 \leq [1 - \frac{\lambda_{min}(M)^2}{\lambda_{min}(M)\lambda_{max}(M)\rho(R)^2}]^{i/2}||r_0||_2 \leq [1 - \frac{1}{\kappa(M)}]^{i/2}||r_0||_2 \qquad (2.3)$$

The GMRES method is mathematically equivalent to the ORTHORES algorithm developed in (Young & Jea 1980). Its convergence rate follows the same formula as (2.3) if A is positive real. Otherwise when A is diagonalisable $A = X\Lambda X^{-1}$, the convergence of the GMRES method depends on the distribution of the eigenvalues and the condition number $\kappa(X)$ as follows. Let $\lambda_1, \ldots, \lambda_\mu$ be the eigenvalues of A with a non-positive real part, and let $\lambda_{\mu 1}, \ldots, \lambda_n$ be those with a positive real part belonging to the circle centred in $C > 0$ with a radius R such that $C > R$. Then the GMRES convergence rate can be written as:

$$||r_{i1}||_2 \leq \kappa(X) \left[\frac{D}{d}\right]^{\mu} \left[\frac{R}{C}\right]^{i-\mu} ||r_0||_2 \qquad (2.4)$$

with $D = \max_{i=1,\mu;j=\mu 1,n} |\lambda_i - \lambda_j|$ and $d = \min_{i=1,\mu} |\lambda_i|$. To be complete, notice that when several problems have to be solved with the same matrix and different right hand sides, projection techniques onto the space spanned by the directions of descent from previous computations can improve the convergence (Parks *et al.* 2006, Tromeur-Dervout & Vassilevski 2006). These projections use the orthogonality relation between the directions of descent to provide a good initial solution.

2.1.2 Schwarz domain decomposition methods

Let us first recall the Generalized Schwarz Alternating Method introduced by (Engquist & Zhao 1998) that gathers several Schwarz techniques (Quarteroni & Valli 1999).

2.1.2.1 The Generalized Schwarz Alternating Method

For sake of simplicity, let us consider the case where the whole domain Ω is decomposed into two sub-domains Ω_1 and Ω_2, with or without overlapping, defining two artificial boundaries Γ_1, Γ_2. Let $\Omega_{11} = \Omega_1 \backslash \Omega_2$, $\Omega_{22} = \Omega_2 \backslash \Omega_1$ if there is an overlap. Let $L(x)$ be the continuous operator associated with the discrete operator A. It can be written as:

Algorithm 3 GSAM: Additive version

1: DO until convergence

2: Solve

$$\begin{aligned} L(x)u_1^n(x) &= f(x),\ \forall x \in \Omega_1, && (2.5)\\ u_1^n(x) &= g(x),\ \forall x\ \in \partial\Omega_1\backslash\Gamma_1, && (2.6)\\ \Lambda_1 u_1^n &+ \lambda_1\frac{\partial u_1^n(x)}{\partial n_1} = \Lambda_1 u_2^{n-1} + \lambda_1\frac{\partial u_2^{n-1}(x)}{\partial n_1},\ \forall x \in \Gamma_1 && (2.7)\end{aligned}$$

AND

$$\begin{aligned} L(x)u_2^n(x) &= f(x),\ \forall x \in \Omega_2, && (2.8)\\ u_2^n(x) &= g(x),\ \forall x \in \partial\Omega_2\backslash\Gamma_2, && (2.9)\\ \Lambda_2 u_2^n &+ \lambda_2\frac{\partial u_2^n(x)}{\partial n_2} = \Lambda_2 u_1^{n-1} + \lambda_2\frac{\partial u_1^{n-1}(x)}{\partial n_2},\ \forall\ x \in \Gamma_2. && (2.10)\end{aligned}$$

3: Enddo

Algorithm 4 GSAM: Multiplicative version

1: DO until convergence

2: Solve

$$\begin{aligned} L(x)u_1^{2n+1}(x) &= f(x),\ \forall x \in \Omega_1, && (2.11)\\ u_1^{2n+1}(x) &= g(x),\ \forall x \in \partial\Omega_1\backslash\Gamma_1, && (2.12)\\ \Lambda_1 u_1^{2n+1} &+ \lambda_1\frac{\partial u_1^{2n+1}(x)}{\partial n_1} = \Lambda_1 u_2^{2n} + \lambda_1\frac{\partial u_2^{2n}(x)}{\partial n_1},\ \forall x \in \Gamma_1 && (2.13)\end{aligned}$$

3: Solve

$$\begin{aligned} L(x)u_2^{2n+2}(x) &= f(x),\ \forall x \in \Omega_2, && (2.14)\\ u_2^{2n+2}(x) &= g(x),\ \forall x \in \partial\Omega_2\backslash\Gamma_2, && (2.15)\\ \Lambda_2 u_2^{2n+2} &+ \lambda_2\frac{\partial u_2^{2n+2}(x)}{\partial n_2} = \Lambda_2 u_1^{2n+1} + \lambda_2\frac{\partial u_1^{2n+1}(x)}{\partial n_2},\ \forall x \in \Gamma_2 && (2.16)\end{aligned}$$

4: Enddo

where Λ_i are some operators and λ_i are constants.

According to the specific choice of the operators Λ_i and the values of scalars λ_i, we obtain the family of Schwarz domain decomposition techniques:

Overlap	Λ_1	Λ_2	λ_1	λ_2	Method
Yes	Id	Id	0	0	Schwarz
Yes	Id	Id	α	α	ORAS (St-Cyr *et al.* 2007)
No	Id	0	0	1	Neumann-Dirichlet (Marini & Quarteroni 1989)
No	Id	Id	1	1	Modified Schwarz (Lions 1988)

Table 2.2: Derived methods obtained from the specific choices of the operators Λ_i and the values of scalars λ_i in the GSAM.

If $\Lambda_1 = \Lambda_2 = I$ and $\lambda_1 = \lambda_2 = 0$ then the above multiplicative version is the classical Multiplicative Schwarz. If $\Lambda_1 = \Lambda_2 = constant > 0$ and $\lambda_1 = \lambda_2 = 1$ then it is the modified Schwarz proposed by Lions in (Lions 1990).

2.1.2.2 Convergence property of the GSAM

(Engquist & Zhao 1998) showed that with an appropriate choice of the operators Λ_i this domain decomposition method converges in two iterations. This property is related to the theory of the Dirichlet-to-Neumann map. This map represents the linear operator that associates the Dirichlet boundary condition of the homogeneous PDE problem to its Neumann boundary condition (co-normal derivative of the PDE).

They established the proposition that follows:

Proposition 2.1.1 *If Λ_1 (or Λ_2) is the Dirichlet to Neumann operator at the artificial boundary Γ_1 (or Γ_2) for the corresponding homogeneous PDE in Ω_2 (or Ω_1) with homogeneous boundary condition on $\partial\Omega_2 \cap \partial\Omega$ (or $\partial\Omega_1 \cap \partial\Omega$), then the Generalized Schwarz Alternating method converges in two steps.*

Proof *Let $e_i^n = u - u^n, i = 1, 2,$ be the error function. Then e_i^n satisfies the homogeneous equation associated with boundary conditions that follow:*

$$\begin{aligned} L(x)e_1^1(x) &= 0,\ \forall x \in \Omega_1, \\ e_1^1(x) &= 0,\ \forall x \in \partial\Omega_1 \backslash \Gamma_1, \\ \Lambda_1 e_1^1 &+ \lambda_1 \frac{\partial e_1^1(x)}{\partial n_1} = \Lambda_1 e_2^0 + \lambda_1 \frac{\partial e_2^0(x)}{\partial n_1},\ \forall x \in \Gamma_1 \end{aligned}$$

and

$$\begin{aligned} L(x)e_2^1(x) &= 0,\ \forall x \in \Omega_2, \\ e_2^1(x) &= 0,\ \forall x \in \partial\Omega_2 \backslash \Gamma_2, \\ \Lambda_2 e_2^1 &+ \lambda_2 \frac{\partial e_2^1(x)}{\partial n_2} = \Lambda_2 e_1^0 + \lambda_2 \frac{\partial e_1^0(x)}{\partial n_2},\ \forall x \in \Gamma_2. \end{aligned}$$

since Λ_1 (Λ_2) is the Dirichlet to Neumann operator at Γ_1 (Γ_2) in Ω_2 (Ω_1), we must have

$$\frac{\partial e_2^0}{\partial n_1} + \Lambda_1 e_2^0 = -\frac{\partial e_2^0}{\partial n_2} + \Lambda_1 e_2^0,\ (\frac{\partial e_1^0}{\partial n_2} + \Lambda_2 e_1^0 = -\frac{\partial e_1^0}{\partial n_1} + \Lambda_2 e_1^0) \qquad (2.17)$$

and we get $e_1^1 = 0$ *in* Ω_1 *(*$e_2^1 = 0$ *in* Ω_2*). Hence we get the exact solution in two steps* []

The GSAM method converges in two steps if the Dirichlet-Neumann operators Λ_i, $i = 1, 2$, are available. These operators are not local to a sub-domain but they link together all the sub-domains. In practice, some algebraically-defined approximations of these operators are used (see (Chevalier & Nataf 1998) (Gander *et al.* 2002) (Gerardo-Giorda & Nataf 2005)). In the Aitken-Schwarz methodology introduced by (Garbey & Tromeur-Dervout 2001, Garbey & Tromeur-Dervout 2002), only the convergence property of the Schwarz method is used. Consequently, no direct approximation of the Dirichlet-Neumann map is used, but an approximation of the operator of error linked to this Dirichlet-Neumann map is performed. Let us describe the first component of the Aitken-Schwarz methodology: the purely linear convergence for the Schwarz Alternating method when the local operators are linear operators.

The pure linear convergence of the GSAM can be established as follows. Consider the decomposition of domain Ω in $\Omega = \Omega_1 \cup \Omega_2$ and define $\bar{i} = mod(i, 2) + 1$ and

$$\Omega_{12} = \Omega_1 \bigcap \Omega_2,\ \Omega_{11} = \Omega_1 \backslash \Omega_{12},\ \Omega_{22} = \Omega_2 \backslash \Omega_{12}. \tag{2.18}$$

Define the projection operator

$$P_i : H^1(\Omega_i) \quad \rightarrow \quad H^1(\Omega_{ii})$$

to be the restriction from Ω_i to Ω_{ii}. Denote S_i (respectively S_{ii}) to be the Dirichlet to Neumann map operator in Ω_i (respectively Ω_{ii}) on Γ_i (respectively $\Gamma_{\bar{i}}$).
Let R_i be the restriction operator from $H^1(\Omega_i)$ to $H^{1/2}(\Gamma_i)$ and R_i^* be the right inverse operator of R_i, i.e, $R_i R_i^* = I$,

$$\forall g \in H^{1/2}(\Gamma_i),\ L(x)R_i^* g = 0, R_i^* g = g \text{ on } \Gamma_i, R_i^* g = 0 \text{ on } \partial\Omega_i \backslash \Gamma_i \tag{2.19}$$

let R_{ii} be the corresponding operator in Ω_{ii},

$$\forall g \in H^{1/2}(\Gamma_{\bar{i}}),\ L(x)R_{ii}^* g = 0, R_{ii}^* g = g \text{ on } \Gamma_{\bar{i}}, R_{ii}^* g = 0 \text{ on } \partial\Omega_{ii} \backslash \Gamma_{\bar{i}} \tag{2.20}$$

The following proposition (see [(Engquist & Zhao 1998),p.354]) establishes the pure linear convergence of the GSAM.

Proposition 2.1.2 *After one cycle of iteration on the error function* $e_i^n = u - u_i^n$ *in* Ω_i *for the GSAM, we have for* $\{i, j\} = \{1, 2\}, i \neq j$*:*

$$e_i^n = R_i^*(\Lambda_i + \lambda_i S_i)^{-1}(\Lambda_i - \lambda_i S_{jj})R_{jj}P_j R_j^*(\Lambda_j + \lambda_j S_j)^{-1}(\Lambda_j - \lambda_i S_{ii})R_{ii}P_i(e_i^{n-1}) \tag{2.21}$$

Proof *The error* e_1^{2n+1} *and* e_2^{2n+2} *satisfy:*

$$\begin{aligned}
L(x)e_1^{2n+1} &= 0, \forall x \in \Omega_1, \\
e_1^{2n+1} &= 0, \forall x \in \partial\Omega_1 \backslash \Gamma_1 \\
\Lambda_1 e_1^{2n+1} &+ \lambda_1 \frac{\partial e_1^{2n+1}}{\partial n_1} = \Lambda_1 e_2^{2n} + \lambda_1 \frac{\partial e_2^{2n}}{\partial n_1}, \forall x \in \Gamma_1 \\
L(x)e_2^{2n+2} &= 0, \forall x \in \Omega_2, \\
e_2^{2n+2} &= 0, \forall x \in \partial\Omega_2 \backslash \Gamma_2 \\
\Lambda_2 e_2^{2n+2} &+ \lambda_2 \frac{\partial e_2^{2n+2}}{\partial n_2} = \Lambda_2 e_1^{2n+1} + \lambda_2 \frac{\partial e_1^{2n+1}}{\partial n_2}, \forall x \in \Gamma_2
\end{aligned}$$

Thus the errors at interfaces write:

$$\begin{aligned}
(\Lambda_1 + \lambda_1 S_1) R_1 e_1^{2n+1} &= (\Lambda_1 + \lambda_1 S_{22}) R_{22} P_2 e_2^{2n} \\
(\Lambda_2 + \lambda_2 S_2) R_2 e_2^{2n+2} &= (\Lambda_2 + \lambda_2 S_{22}) R_{11} P_1 e_1^{2n+1}
\end{aligned}$$

Consequently

$$\begin{aligned}
e_2^{2n+2} &= R_2^*(\Lambda_2 + \lambda_2 S_2)^{-1}(\Lambda_2 + \lambda_2 S_{22}) R_{11} P_1 R_1^*(\Lambda_1 + \lambda_1 S_1)^{-1}\Lambda_1 + \lambda_1 S_{22}) R_{22} P_2 e_2^{2n} \\
e_1^{2n+1} &= R_1^*(\Lambda_1 + \lambda_1 S_1)^{-1}(\Lambda_1 + \lambda_1 S_{11}) R_{22} P_2 R_2^*(\Lambda_2 + \lambda_2 S_2)^{-1}\Lambda_2 + \lambda_2 S_{11}) R_{11} P_1 e_1^{2n-1}
\end{aligned}$$

[]

This proposition allows us to retrieve the following results:

1. For the classical multiplicative Schwarz ($\Lambda_1 = \Lambda_2 = 0, \lambda_1 = \lambda_2 = 1$) and $\Gamma_1 \neq \Gamma_2$, (2.21) is reduced to

$$e_i^n = R_i^* R_{jj} P_j R_j^* R_{ii} P_i (e_i^{n-1}).$$

2. For the non-overlapping Schwarz ($\Gamma_1 = \Gamma_2$, $P_i = Id$, $R_{ii} = R_i$), (2.21) is reduced to

$$e_i^n = R_i^*(\Lambda_i + \lambda_i S_i)^{-1}(\Lambda_i - \lambda_i S_j)(\Lambda_j + \lambda_j S_j)^{-1}(\Lambda_j - \lambda_j S_i) R_i (e_i^{n-1}).$$

2.2 Schwarz and Schur preconditioning methods

The convergence rate of a Krylov method such as GCR (Eisenstat *et al.* 1983) and GMRES (Eisenstat *et al.* 1983), to solve a linear system $Au = f$, $A = (a_{ij}) \in \mathbb{R}^{m \times m}, u \in \mathbb{R}^m, b \in \mathbb{R}^m$, depends on the matrix eigenvalues' distribution as shown by (2.3) and (2.4), and, generally speaking, decreases when the condition number $\kappa_2(A) = ||A||_2 ||A^{-1}||_2$ of the non-singular matrix A increases. This implies the need to reduce the scattering of the eigenvalues' distribution in the complex plane in order to improve the convergence rate. This is the goal of a preconditioning technique. The left-preconditioning techniques consist to solve $M^{-1}Au = M^{-1}f$ such that $\kappa_2(M^{-1}A) << \kappa_2(A)$.

In this work we focus on the Schwarz preconditioning techniques and the preconditioning techniques that are related to the Schur complement of the matrix A. In what follows, we give an overview of the state of the art in preconditioning techniques based on domain decompositions.

2.2.1 Restricted Additive Schwarz preconditioning

The preconditioning techniques that are based on domain decomposition of Schwarz's type have been widely developed this last decade and accelerated multiplicative Schwarz has been "*a consistently good performer*" (Cai *et al.* 1994). The first type of domain decomposition preconditioning to appear was domain decomposition based on substructuring technique (Bramble *et al.* 1986) followed by the Additive Schwarz (AS) preconditioning (Dryja & Widlund 1990, Gropp & Keyes 1992). It is built from the adjacency graph $G = (W, E)$ of A, where $W = \{1, 2, ..., m\}$ and $E = \{(i, j) : a_{ij} \neq 0\}$ are the edges and vertices of G. Starting with a non-overlapping partition $W = \cup_{i=1}^{p} W_{i,0}$ and $\delta \geq 0$ given, the overlapping partition $\{W_{i,\delta}\}$ is obtained defining p partitions $W_{i,\delta} \supset W_{i,\delta-1}$ by including all the immediately-neighbouring vertices of the vertices in the partition $W_{i,\delta-1}$. Then the restriction operator $R_{i,\delta} : W \to W_{i,\delta}$ defines the local operator $A_{i,\delta} = R_{i,\delta} A R_{i,\delta}^T$, $A_{i,\delta} \in \mathbb{R}^{m_{i,\delta} \times m_{i,\delta}}$ on $W_{i,\delta}$.

The AS preconditioning writes: $M_{AS,\delta}^{-1} = \sum_{i=1}^{p} R_{i,\delta}^T A_{i,\delta}^{-1} R_{i,\delta}$.

(Cai & Sarkis 1999) introduced the restriction matrix $\tilde{R}_{i,\delta}$ on a non-overlapping sub-domain $W_{i,0}$, and then derived the Restricted Additive Schwarz (RAS) iterative process as:

$$u^k = u^{k-1} + M_{RAS,\delta}^{-1} \left(f - A u^{k-1} \right), \text{ with } M_{RAS,\delta}^{-1} = \sum_{i=1}^{p} \tilde{R}_{i,\delta}^T A_{i,\delta}^{-1} R_{i,\delta} \tag{2.22}$$

They showed experimentally that the RAS exhibits a faster convergence than the AS, as demonstrated in (Efstathiou & Gander 2003) for the Poisson problem, leading to a better preconditioning that depends on the number of sub-domains. We note that (Cai *et al.* 2004) develop extensions of RAS for symmetric positive definite problems using the so-called harmonic overlaps (RASHO).

When it is applied to linear problems, the RAS has a purely linear rate of convergence/divergence that can be enhanced with optimized boundary conditions giving the ORAS method of (St-Cyr *et al.* 2007). In this case, the transmission condition in GSAM takes the form Λ_i to be the normal derivative (Neumann boundary condition). Then, an optimisation problem is done to minimize the amplification factor of the Schwarz method with this Robin coefficient in the Fourier space. The drawback of this method is that it can only be applied to separable operators, and needs a regular step size and periodic boundary conditions in the direction orthogonal to the interface to be mathematically valid. Nevertheless, if it is not the case, the parameters in the Robin conditions are set based on this postulate and applied in the current case.

2.2.2 SchurRAS preconditioning

The SchurRAS method has been introduced by (Li & Saad 2006) and is based on the ILU factorisation of the local operators $A_{i,\delta}$ present in the RAS method.

For convenience, a local reordering of the data is often exploited by splitting the local vector u_i into two parts: the subvector x_i of interior nodes followed by the subvector v_i of local interface nodes. The local right-hand side vector f_i is conformally split into the subvectors b_i and g_i, so that

$$A_{i,\delta} = \begin{pmatrix} B_{i,\delta} & E_{i,\delta} \\ F_{i,\delta} & C_{i,\delta} \end{pmatrix} \tag{2.23}$$

Assume that the ILU factorization of $B_{i,\delta}$ is available as

$$B_{i,\delta} \simeq L_{i,\delta}.U_{i\delta} \tag{2.24}$$

Then the local operator with overlap is approximated by the ILU block factorization:

$$\begin{aligned} A_{i,\delta} &\simeq \begin{pmatrix} L_{i,\delta} & 0 \\ F_{i,\delta}U_{i\delta}^{-1} & I \end{pmatrix} \begin{pmatrix} U_{i,\delta} & L_{i,\delta}^{-1}E_{i,\delta} \\ 0 & S_{i\delta} \end{pmatrix} \\ &= \begin{pmatrix} L_{i,\delta} & 0 \\ F_{i,\delta}U_{i\delta}^{-1} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & S_{i\delta} \end{pmatrix} \begin{pmatrix} U_{i,\delta} & L_{i,\delta}^{-1}E_{i,\delta} \\ 0 & I \end{pmatrix} \\ &\equiv \mathcal{L}_{i,\delta}\mathcal{D}_{i,\delta}\mathcal{U}_{i,\delta} \end{aligned} \tag{2.25}$$

$$M_{ILU-AS,\delta}^{-1} = \sum_{i=1}^{p} R_{i,\delta}^T \mathcal{U}_{i,\delta}^{-1} \mathcal{D}_{i,\delta}^{-1} \mathcal{L}_{i,\delta}^{-1} R_{i,\delta},$$

$$M_{ILU-AS,\delta}^{-1} = \sum_{i=1}^{p} (R_{i,\delta}^T \mathcal{U}_{i,\delta}^{-1} R_{i,\delta})(R_{i,\delta}^T \mathcal{D}_{i,\delta}^{-1} R_{i,\delta})(R_{i,\delta}^T \mathcal{L}_{i,\delta}^{-1} R_{i,\delta}). \tag{2.26}$$

The RAS preconditioning takes the form (with $\tilde{R}_{i,\delta}$ to be the restriction on the sub-domain without the overlap):

$$M_{ILU-RAS,\delta}^{-1} = \sum_{i=1}^{p} (\tilde{R}_{i,\delta}^T \mathcal{U}_{i,\delta}^{-1} R_{i,\delta})(\tilde{R}_{i,\delta}^T \mathcal{D}_{i,\delta}^{-1} R_{i,\delta})(\tilde{R}_{i,\delta}^T \mathcal{L}_{i,\delta}^{-1} R_{i,\delta}). \tag{2.27}$$

The SchurRAS preconditioner sees $\tilde{R}_{i,\delta}^T \mathcal{U}_{i,\delta}^{-1} R_{i,\delta}$ and $\tilde{R}_{i,\delta}^T \mathcal{L}_{i,\delta}^{-1} R_{i,\delta}$ as projection and restriction operators to finally obtain:

$$M_{SchurRAS,\delta}^{-1} = \sum_{i=1}^{p} (\tilde{R}_{i,\delta}^T U_{i,0}^{-1} \tilde{R}_{i,\delta})(\tilde{R}_{i,\delta}^T \mathcal{D}_{i,\delta}^{-1} R_{i,\delta})(\tilde{R}_{i,\delta}^T L_{i,0}^{-1} \tilde{R}_{i,\delta}). \tag{2.28}$$

Let us point out the advantages and disadvantages of this preconditioner formulation:

1. The prolongation and restriction operations do not need neighbouring communications, and the term $E_{i,j}$ linking the local sub-domain with others subdomains sharing the same interface variable are unchanged during the block Gaussian elimination. This preconditioning works only on the local interface variables.

2. The main drawback is to disconnect the effect of the not immediate neighbouring sub-domains to the Schur complement.

2.2.3 Patch Subtructuring method

(Gander *et al.* 2007) introduced the patch substructuring methods for which the equivalence with the overlapping Schwarz methods has been demonstrated. In this work the Dirichlet and Neumann boundary conditions present in the Schwarz alternated method have been replaced by Robin boundary conditions to enhance the convergence rate. The patch method consists in introducing an overlap in the Schur complement technique. It is written as follows:

$$\begin{pmatrix} A_{11} & A_{1\Gamma} & \\ A_{\Gamma 1} & A_{\Gamma\Gamma} & A_{\Gamma 2} \\ & A2\Gamma & A_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_\Gamma \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_\Gamma \\ f_2 \end{pmatrix} \tag{2.29}$$

Then by introducing the Lagrange multipliers λ_1 and λ_2 in order to satisfy the continuity constraints on the solutions of the sub-domains:

$$u_{\Gamma_1} - u_{\Gamma_2} = 0 \tag{2.30}$$

$$\lambda_1 + \lambda_2 - \Lambda_1 u_{\Gamma_1} - \Lambda_2 u_{\Gamma_2} = 0 \tag{2.31}$$

for any matrices Λ_1 and Λ_2 of size the matrix $A_{\Gamma\Gamma}$,

$$\begin{pmatrix} A_{11} & A_{1\Gamma} \\ A_{\Gamma 1} & A^1_{\Gamma\Gamma} + \Lambda_1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_{\Gamma_1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\Gamma_1} + \lambda_1 \end{pmatrix} \tag{2.32}$$

$$\begin{pmatrix} A^2_{\Gamma\Gamma} + \Lambda_2 & A_{\Gamma_2 2} \\ A_{2\Gamma_2} & A_{22} \end{pmatrix} \begin{pmatrix} u_{\Gamma_2} \\ u_2 \end{pmatrix} = \begin{pmatrix} f_{\Gamma_2} + \lambda_2 \\ f_2 \end{pmatrix} \tag{2.33}$$

This gives the algorithm:

$$\begin{pmatrix} A_{11} & A_{1\Gamma} \\ A_{\Gamma 1} & A^1_{\Gamma\Gamma} + \Lambda_1 \end{pmatrix} \begin{pmatrix} u^k_1 \\ u^k_{\Gamma_1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\Gamma_1} - A_{\Gamma_2 2} u_2^{k-1} + (\Lambda_1 - A^2_{\Gamma\Gamma}) u^{k-1}_{\Gamma_2} \end{pmatrix} \tag{2.34}$$

$$\begin{pmatrix} A^2_{\Gamma\Gamma} + \Lambda_2 & A_{\Gamma_2 2} \\ A_{2\Gamma_2} & A_{22} \end{pmatrix} \begin{pmatrix} u^k_{\Gamma_2} \\ u^k_2 \end{pmatrix} = \begin{pmatrix} f_\Gamma - A_{\Gamma_1\Gamma_1} u_1^{k-1} + (\Lambda_2 - A^1_{\Gamma\Gamma}) u^{k-1}_{\Gamma_1} \\ f_2 \end{pmatrix} \tag{2.35}$$

They showed that if $\Lambda_1 = A^2_{\Gamma\Gamma} - A_{\Gamma 2} A^{-1}_{22} A_{2\Gamma}$ and $\Lambda_2 = A^1_{\Gamma\Gamma} - A_{\Gamma 1} A^{-1}_{11} A_{1\Gamma}$, then the previous algorithm converges in two iterations. Notice that the result of Engquist and Zhao has been rediscovered at the discrete level given that $A^2_{\Gamma\Gamma} - A_{\Gamma 2} A^{-1}_{22} A_{2\Gamma}$ is

the Steklov-Poincaré matrix on the sub-domain Ω_1 corresponding to the Neumann to Dirichlet map. Then they introduce an overlap in the Schur complement method, the extended area corresponding to the patch substructuring. The original system is split, introducing an overlap in the decomposition of the domain giving the extended sub-domains to be $\bar{\Omega}_i = \Omega_i \cup \Gamma \cup P_i \cup \Gamma_i$.

$$\begin{pmatrix} A_{11} & A_{1\Gamma_2} & & & & & & \\ A_{\Gamma_2} & A_{\Gamma_2\Gamma_2} & A_{\Gamma_2 P_2} & & & & & \\ & AP_2\Gamma_2 & A_{P_2P_2} & A_{P_2\Gamma} & & & & \\ & & A_A\Gamma P_2 & A_{\Gamma\Gamma} & A_{\Gamma P_1} & & & \\ & & & A_{P_1 n\Gamma} & A_{P_1P_1} & A_{P_1\Gamma_1} & & \\ & & & & A_{\Gamma_1 P_1} & A_{\Gamma_1\Gamma_1} & A_{\Gamma_1} & \\ & & & & & A_{2\Gamma_1} & A_{22} & \end{pmatrix} \begin{pmatrix} u_1 \\ u_{\Gamma_2} \\ u_{P_2} \\ u_\Gamma \\ u_{P_1} \\ u_{\Gamma_1} \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\Gamma_2} \\ f_{P_2} \\ f_\Gamma \\ f_{P_1} \\ f_{\Gamma_1} \\ f_2 \end{pmatrix}$$

Then the Patch substructuring method gives:

$$\begin{pmatrix} A_{11} & A_{1\Gamma_2} & & \\ A_{\Gamma_2} & A_{\Gamma_2\Gamma_2} & A_{\Gamma_2 P_2} & \\ & AP_2\Gamma_2 & A_{P_2P_2} & A_{P_2\Gamma} \\ & & A_A\Gamma P_2 & \Lambda_1 + A^1_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} v_1^k \\ u_{\Gamma_2}^k \\ v_{P_2}^k \\ v_\Gamma^k \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\Gamma_2} \\ f_{P_2} \\ f_\Gamma - (A^2_{\Gamma\Gamma} - \Lambda_1)\omega_\Gamma^{k-1} + A_{\Gamma P_1}\omega_{P_1}^{k-1} \end{pmatrix} \tag{2.36}$$

$$\begin{pmatrix} \Lambda_2 + A^2_{\Gamma\Gamma} & A_{\Gamma P_1} & & \\ A_{P_1 n\Gamma} & A_{P_1P_1} & A_{P_1\Gamma_1} & \\ & A_{\Gamma_1 P_1} & A_{\Gamma_1\Gamma_1} & A_{\Gamma_1} \\ & & A_{2\Gamma_1} & A_{22} \end{pmatrix} \begin{pmatrix} u_\Gamma^k \\ u_{P_1}^k \\ u_{\Gamma_1}^k \\ u_2^k \end{pmatrix} = \begin{pmatrix} f_\Gamma - (A^1_{\Gamma\Gamma} - \Lambda_2)v_\Gamma^{k-1} + A_{\Gamma P_2}v_{P_2}^{k-1} \\ f_{P_1} \\ f_{\Gamma_1} \\ f_2 \end{pmatrix} \tag{2.37}$$

Let us describe the advantages and disadvantages of this method:

1. The advantage is to have "optimized boundary conditions" for the transmission condition, as it is equivalent to a classical Schwarz with Optimized Robin conditions.

2. As with the SchurRAS methods, the local Schur complement involves only the effect of the immediate neighboring sub-domains.

2.2.4 A two-level preconditioner defined on the interface for the Schur complement

As it has been seen, the Schur complement associated with the sub-domain's problem is involved in the design of the ShurRAS and the Patch substructuring methods. These techniques take care of the data locality in order to avoid global communications involving all sub-domains. Nevertheless, as in the RAS preconditioning technique, the main drawback of this locality is a decrease of the preconditioning efficiency with respect to the number of sub-domains. Another related work that takes care to involve all the sub-domains present in the Schur complement is the substructuring method with a suitable preconditioner for the reduced equation (Bramble *et al.* 1986) (Carvalho *et al.* 2000) (Carvalho *et al.* 2001) (Khoromskij & Wittum 2004).

Let us describe the iterative substructuring method and the preconditioning by the additive Schwarz preconditioner for the Schur complement reduced equation on the interface problem designed by (Carvalho *et al.* 2000).

Let Γ be the set of all the indices of the mesh points which belong to the interfaces between the sub-domains. Grouping together the unknowns associated to points of the mesh corresponding to Γ into the vector u_Γ and the ones corresponding to the other unknowns (corresponding to the points of mesh associated to the interior I of sub-domains) into the vector u_I, we get the reordered problem:

$$\begin{pmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_\Gamma \end{pmatrix} \begin{pmatrix} u_I \\ u_\Gamma \end{pmatrix} = \begin{pmatrix} f_I \\ f_\Gamma \end{pmatrix} \tag{2.38}$$

Eliminating the unknowns u_I from the second block row of (2.38) leads to the following reduced equation for u_Γ:

$$S u_\Gamma = f_\Gamma - A_{\Gamma I} A_{II}^{-1} f_I, \tag{2.39}$$

where

$$S^{(i)} = A_{\Gamma\Gamma} - A_{\Gamma I} A_{II}^{-1} A_{I\Gamma} \tag{2.40}$$

is the Schur complement of the matrix A_{II} in A. Let be $\Gamma_i = \partial\Omega_i \backslash \partial\Omega$. Let $R_{\Gamma_i} : \Gamma \to \Gamma_i$ be the canonical pointwise restriction which maps vectors on Γ into defined vectors on Γ_i, and let be $R_{\Gamma_i}^T : \Gamma_i \to \Gamma$ its transposed. The Schur complement matrix (2.40) can also be written as:

$$S = \sum_{i=1}^{p} R_{\Gamma_i}^T S^{(i)} R_{\Gamma_i} \tag{2.41}$$

where

$$S = A_{\Gamma_i\Gamma_i}^{(i)} - A_{\Gamma_i i} A_{ii}^{-1} A_{i\Gamma_i} \tag{2.42}$$

is referred to the local Schur complement associated with the sub-domain Ω_i. $S^{(i)}$ that involves the submatrices from the local matrix $A^{(i)}$ which is defined as

$$A^{(i)} = \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{\Gamma_i i} & A_{\Gamma_i \Gamma_i} \end{pmatrix} \tag{2.43}$$

Then they defined a BPS (Bramble Pasciak and Schatz (Bramble *et al.* 1986)) preconditioner which is based on the set V which gathers the cross points between sub-domains (*i.e.* points that belong to more than two sub-domains) and the sets E_i of interface points (without the cross points in V)

$$E_i = (\partial\Omega_j \cap \partial\Omega_l) - V \tag{2.44}$$

$$\Gamma = (\bigcup_{i=1}^{m} E_i) \cup V \tag{2.45}$$

The operator R_i defined the standard pointwise restriction of nodal values on E_i while operator R_V defined the canonical restriction on V. Then a coarse mesh is associated with the sub-domains and an interpolation operator R^T is defined. This operator corresponds to the linear interpolation between two adjacent cross points V_j V_i in order to define values on the edge E_i. This allows us to define A_H the Galerkin cross grid operator $A_H = RAR^T$. They deduced a very close variant of BPS preconditioner that can be written as:

$$M_{BPS} = \sum_{E_i} R_i^T S_{ii} R_i + R^T A_H^{-1} R \tag{2.46}$$

They defined a coarse-space operator

$$\Lambda_0 = R_0 S R_0^T \tag{2.47}$$

where $R_0 : U \to U_0$ is a restriction operator which maps full vector of U into vector in U_0 where U_0 is a q-dimensional subspace of U, the algebraical space of nodal vectors where the Schur complement matrix is defined.

$$M_{BPS} = \sum_{E_i} R_i^T \tilde{S}_{ii} R_i + R_0^T \Lambda_0^{-1} R_0 \tag{2.48}$$

where $\tilde{S}_{ii}$ is an approximation of S_{ii}. The definition of U_0 gives different preconditioners: vertex-based coarse space, sub-domain-based coarse space, or edge-based coarse space, depending on the set of points of the interface Γ that are involved. From the implementation practical point of view, the coarse matrix Λ_0 is constructed once and involves matrix vector products of the local Schur complement only.

1. The advantage of this method is defining the two-level preconditioner only on the interface Γ. It is intimately related to the Schur complement operator defined on the interface.

2. The drawback is having to define *a priori* the coarse space U_0 without any knowledge of the solution behavior. Consequently, it can be expensive in terms of the number of coarse space vectors, specifically for 3D problems where cross-points between sub-domains in 2D become cross-regions between sub-domains in 3D.

Our approach will follow the same spirit as this two-level preconditioning working only on the interface. Nevertheless, we still work on the system $Ax = b$ and not $Su_\Gamma = g_\Gamma$, and we use an *a posteriori* knowledge of the global Dirichlet-to-Neumann map that is based on the pure linear convergence/divergence of the RAS to define the coarse space (equivalent of the definition of U_0).

Chapter 3

Aitken-Schwarz preconditioning technique

French summary - Résumé en français

Technique de préconditionnement d'Aitken-Schwarz

Ce chapitre est dédié à la conception d'un nouveau préconditionneur consistant en l'amélioration de la technique de préconditionnement Schwarz Additif Restreint (RAS) par la technique d'accélération d'Aitken. On présente, tout d'abord, la technique d'accélération d'Aitken dans le cas vectoriel. Différentes méthodes pour effectuer l'accélération dans différentes bases d'approximation sont proposées. Dès lors qu'il est possible de construire algébriquement l'opérateur de transfert d'erreur sur les interfaces artrificielles d'une méthode de décomposition de domaine, il semble intéressant d'étudier la possibilité d'amélioration d'une technique de préconditionnement par Aitken. On propose donc une formulation d'un processus d'Aitken Schwarz Additif Restreint (ARAS) sous forme de processus itératif de Richardson. Le nouveau processus itératif obtenu est une méthode à deux niveaux qui s'applique d'une part sur le domaine entier par la méthode RAS, et d'autre part sur l'interface par la formule d'Aitken. Cette formulation offre un nouveau cadre d'étude pour les méthodes d'Aitken-Schwarz qui peut être exploité pour étudier sa convergence et la manière dont une telle technique peut être utilisée comme méthode de préconditionnement dans une approche formelle. Des tests de vérification sont proposés. Enfin, le coût de construction et d'application d'un préconditionneur ARAS en terme de nombre d'opérations est présenté.

La technique d'accélération d'Aitken de convergence de suite peut s'écrire dans le cas des suites vectorielles selon la définition suivante :

Définition 3.0.1 (version française de la définition 3.1.1) *Une suite vectorielle* $\left(\left(u_i^k\right)_{i=1,\dots,N} = u^k\right)_{k\in\mathbb{N}}$ *converge purement linéairement vers* $(\xi)_{i=1,\dots,N} = \xi$ *si*

$$u^{k+1} - \xi = P\left(u^k - \xi\right)$$

où $P \in \mathbb{R}^{n\times n}$ *est un opérateur de transfert d'erreur constant, indépendant de* k *et inversible.*

On suppose qu'il existe une norme $||.||$ telle que $||P|| < 1$. Alors P et ξ peuvent être déterminés à partir d'un certain nombre d'itérations $N+1$, en utilisant l'équation :

$$\left(u^{k+1} - u^k, ..., u^2 - u^1\right) = P\left(u^k - u^{k-1}, ..., u^1 - u^0\right)$$

Donc, si $\left(u^k - u^{k-1}, ..., u^1 - u^0\right)$ est inversible, P peut s'écrire :

$$P = \left(u^{k+1} - u^k, ..., u^2 - u^1\right)\left(u^k - u^{k-1}, ..., u^1 - u^0\right)^{-1}$$

Il est alors possible de calculer ξ avec la formule d'Aitken (3.11) *:*

$$\xi = (I - P)^{-1}\left(u^{n+1} - Pu^n\right)$$

L'algorithme 5 permet d'appliquer l'accélération d'Aitken dans l'espace physique (base canonique de $\mathbb{R}^n$). On note que cet algorithme nécessite $n+1$ itérés pour une suite de vecteurs de taille n. Il ne peut donc être utilisé que lorsque n est petit. De plus, la matrice $\left[E^{n-1}, \ldots, E^0\right]$ peut s'avérer être difficilement inversible.

L'objectif est d'effectuer l'accélération d'Aitken en limitant les calculs. Deux approches sont à prendre en considération :

(a) Troncature de l'opérateur dans une base complète construite analytiquement.

(b) Approximation de l'opérateur dans une base construite explicitement.

Dans le cas de la méthode (a), l'opérateur peut être construit analytiquement dans une nouvelle base en effectuant un changement de base. Nous pouvons mentionner par exemple, un changement de base par transformée de Fourier discrète sur une grille régulière, ou bien dans une base constituée des vecteurs propres provenant de la décomposition en valeurs propres d'un opérateur semi-discret. Dans ce cas, l'opérateur de transfert d'erreur $P = \mathbb{U}\hat{P}\mathbb{V}$, où $\mathbb{U}$ et $\mathbb{V}$ sont des opérateurs de changement de base, est considéré comme exact. Dès lors, en introduisant un opérateur Q de troncature de la base tel que $Q \in \mathbb{R}^{n\times n}$, $q_l = 1$ si $1 \leq l \leq q$ et $q_l = 0$ si $q < l$ on peut écrire la formule d'accélération d'Aitken sur les q modes les plus forts sous la forme :

$$\hat{\xi} = \left(I - Q\hat{P}\right)^{-1}\left(\hat{u}_{N+1} - Q\hat{P}\hat{u}_N\right)$$

Dans le cas de la méthode (b), une base d'approximation peut être calculée en considérant l'effet d'une itération d'un processus itératif de Schwarz sur différentes solutions sur l'interface. Dans ce cas, il n'est pas possible d'écrire analytiquement l'opérateur de transfert d'erreur mais on obtient une approximation de cet opérateur. Ces techniques sont dites explicites car elles nécessitent le calcul de solutions de Schwarz sur l'interface. On présente trois types d'approximations : l'une avec Transformée de Fourier Discrete Non Uniforme (NUDFT) (Algorithme 6), une autre approximation dans une base orthogonale venant d'une approximation algébrique grossière arbitraire de l'interface (Algorithme 7), et enfin, une approximation venant

de la Décomposition en Valeurs Singulières des solutions de Schwarz sur l'interface (Algorithmes 8 et 9). Généralement, on écrira la formule d'Aitken pour les méthodes de classes (b), en considérant un opérateur de changement de base $\mathbb{U} \in \mathbb{R}^{n\times q}$*, telle que :*

$$\tilde{\xi} = \left(I - \tilde{P}\right)^{-1}\left(\tilde{u}_{N+1} - \tilde{P}\tilde{u}_N\right)$$

et,

$$\xi = \mathbb{U}\tilde{\xi}$$

Tandis que la méthode (a) demande une discrétisation régulière et des hypothèses fortes sur la discrétisation de l'opérateur A, la méthode (b) est algébrique. De plus, la méthode (a) semble être plus sensible aux perturbations que la méthode (b).

Nous nous intéressons maintenant à l'intégration de l'accélération d'Aitken dans un processus itératif RAS écrit sous la forme d'un processus de Richardson afin de pouvoir formuler un préconditionneur à deux niveaux Aitken-Schwarz.

On pose $\Gamma_i = (I_{m_{i,\delta}} - R_{i,\delta}^T)W_{i,\delta}$ *comme étant l'interface associée à* $W_{i,\delta}$ *et* $\Gamma = \cup_{i=1}^{p}\Gamma_i$ *comme étant l'interface globale. La solution* $u \in \mathbb{R}^m$ *restreinte à l'interface* Γ *s'écrit* $u_{|\Gamma} \in \mathbb{R}^n$*. On note* $e_{|\Gamma}^k = u_{|\Gamma}^k - u_{|\Gamma}^\infty$ *l'erreur d'une itération d'un processus itératif RAS restreinte à l'interface* Γ*.*

On introduit un opérateur de restriction $R_\Gamma \in \mathbb{R}^{n\times m}$ *restreignant* W *à l'interface artificelle globale* Γ*, avec* $R_\Gamma R_\Gamma^T = I_n$*.*

Le processus RAS à une convergence/divergence purement linéaire. Le processus itératif Aitken Schwarz Additif Restreint (ARAS) doit générer une suite de solutions sur l'interface Γ*, et accélérer la convergence du processus de Schwarz à partir de cette suite. Alors, la solution accélérée sur l'interface remplace la dernière solution. Ceci peut être écrit en combinant un processus RAS eq.*(3.23a) *avec un processus d'Aitken dans* $\mathbb{R}^{m\times m}$ *eq.*(3.23b) *et en ôtant la solution de Schwarz eq.*(3.23c)*. On peut alors écrire l'approximation* u^* *de la solution* u *suivante :*

$$u^* = u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1}) \tag{3.23a}$$

$$+ R_\Gamma^T\left(I_n - P\right)^{-1}\left(u_{|\Gamma}^k - Pu_{|\Gamma}^{k-1}\right) \tag{3.23b}$$

$$- R_\Gamma^T I_n R_\Gamma\left(u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1})\right) \tag{3.23c}$$

Cette formulation conduit à l'expression d'une solution itérée u^* *:*

$$u^* = u^{k-1} + \left(I_m + R_\Gamma^T\left(\left(I_n - P\right)^{-1} - I_n\right)R_\Gamma\right)M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)$$

Cette solution itérée peut être vue comme une solution accélérée d'un processus itératif RAS. En nous inspirant de la méthode de Stephensen (Stoer & Bulirsch 2002), on construit une nouvelle suite d'itérés à partir des solutions accélérées par la méthode d'Aitken. On considère alors u^* *comme étant un nouvel itéré* u^k *et on écrit le processus itératif ARAS :*

$$u^k = u^{k-1} + \left(I_m + R_\Gamma^T\left(\left(I_n - P\right)^{-1} - I_n\right)R_\Gamma\right)M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)$$

Et donc le préconditionneur ARAS :

$$M^{-1}_{ARAS,\delta} = \left(I_m + R_\Gamma^T \left((I_n - P)^{-1} - I_n\right) R_\Gamma\right) \sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta}$$

On remarque que si P est connu exactement, le processus ARAS converge en deux itérations vers la solution u en partant de la solution initiale $u^0 = 0$. On peut alors écrire la propriété 3.2.1 et en déduire une approximation de A^{-1} à partir des deux premières itérations du processus ARAS. Cette démarche mène à l'écriture d'une forme multiplicative de ARAS, ARAS2 :

$$u^{k+1} = u^{k-1} + \left(2M^{-1}_{ARAS,\delta} - M^{-1}_{ARAS,\delta} A M^{-1}_{ARAS,\delta}\right)\left(f - Au^{k-1}\right)$$

On en déduit le préconditionneur ARAS2 :

$$M^{-1}_{ARAS2,\delta} = 2M^{-1}_{ARAS,\delta} - M^{-1}_{ARAS,\delta} A M^{-1}_{ARAS,\delta} \tag{3.2}$$

Où, si P est exact, $M^{-1}_{ARAS2,\delta} = A^{-1}$.

Comme nous l'avons vu précédemment, il nous faut approximer P avec la méthode (a) ou (b) afin de pouvoir effectuer une accélération efficace et peu coûteuse. En notant $\mathbb{U}_q \in \mathbb{R}^{n\times q}$la matrice de changement de base, contenant $q \leq n$ vecteurs orthogonaux, on écrit la technique ARAS(q) sous la forme :

$$M^{-1}_{ARAS(q),\delta} = \left(I_m + R_\Gamma^T \mathbb{U}_q \left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} - I_q\right) \mathbb{U}_q^T R_\Gamma\right) \sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta}$$

et ARAS2(q),

$$M^{-1}_{ARAS2(q),\delta} = 2M^{-1}_{ARAS(q),\delta} - M^{-1}_{ARAS(q),\delta} A M^{-1}_{ARAS(q),\delta}$$

Il est à noter que la technique de préconditionnement ARAS(q) a été présentée dans (Dufaud & Tromeur-Dervout 2010b, Dufaud & Tromeur-Dervout 20YYa).

Ces formulations sous forme de processus de Richardson, cadre usuel d'écriture des préconditionneurs de ce type, permettent d'envisager des études de convergence afin d'analyser le mécanisme du préconditionnement ARAS(q). On notera :

$$T_* = (I - M_*^{-1} A)$$

Tout processus de Richardson peut être écrit sous la forme :

$$u^k = T_* u^{k-1} + c, \text{ where } c \in \mathbb{R}^n \text{is constant}$$

On remarque également que $T_{ARAS2} = T^2_{ARAS}$.

Dans le cas idéal où P est connu exactement, T_{ARAS} est nilpotente. Ceci conduit à la proposition 3.3.1 qui donne que le rayon spectral , noté ρ, des processus itératifs ARAS et ARAS2 sont égaux à zéro.

$$\rho\left(T_{ARAS2}\right) = \rho\left(T_{ARAS}\right) = 0$$

Mais comme nous l'avons précisé, nous ne caculons pas de manière exacte P. Ainsi les rayons spectraux de $T_{ARAS(q)}$ et $T_{ARAS2(q)}$ sont différents de zéro.

Nous proposons une étude en ne considérant que des opérateurs elliptiques discrets. Dans ce cas on montre la proposition 3.3.2, qui donne le rayon de convergence de T_{RAS} en fonction des valeurs propres λ de l'opérateur de transfert d'erreur P sur l'interface :

$$\rho(T_{RAS}) = \max\left\{|\lambda| \,:\, \lambda \in \lambda(P)\right\}$$

Dans le cas d'opérateurs elliptiques séparables, on établit un théorème donnant le taux de convergence d'un processus itératif ARAS dans le cas des méthodes d'approximation (b).

Théorème 3.0.2 (version française du théorème 3.3.3) *Soit A un opérateur discret d'un problème elliptique sur le domaine Ω. On considère un processus itératif RAS tel que $T_{RAS} = I - M_{RAS}^{-1}A$, défini sur p domaines. L'opérateur de transfert d'erreur P sur l'interface Γ est diagonalisable. Si P est diagonalisable, sa décomposition en valeurs propres conduit à $P = \mathbb{U}\hat{\hat{P}}\mathbb{U}^{-1}$ où pour $i \in [\![1, n]\!]$, $\hat{\hat{P}} = diag(\lambda_i)$. L'erreur sur l'interface dans l'espace d'approximation issu de la décomposition en valeurs propre des P suit la relation $\hat{\hat{e}}^{k+1}_{|\Gamma} = \hat{\hat{P}}\hat{\hat{e}}^{k}_{|\Gamma}$. Chaque mode converge linéairement et indépendamment des autres, d'où $\hat{\hat{e}}^{k+1}_{|\Gamma,i} = \lambda_i \hat{\hat{e}}^{k}_{|\Gamma,i}$. On pose l'opérateur de troncature $Q_\lambda \in \mathbb{R}^{n\times n}$, $q_l = 1$ si $1 \leq l \leq q$ et $q_l = 0$ if $q < l$. Une approximation grossière de $\hat{\hat{P}}$ peut être faite en choisissant un ensemble de q modes forts avec $\tilde{P} = Q_\lambda \hat{\hat{P}}$. Le préconditionneur ARAS(q) sécrit alors*

$$M^{-1}_{ARAS(q),\delta} = \left(I_m + R_\Gamma^T \mathbb{U}\left(\left(I_n - \tilde{P}\right)^{-1} - I_n\right)\mathbb{U}^{-1}R_\Gamma\right)\sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta}$$

Le rayon spectral de $T_{ARAS(q)}$ est :

$$\rho(T_{ARAS(q)}) = \rho(\bar{Q}_\lambda \hat{\hat{P}}) = \lambda_{q+1} < \min\{|\lambda| : \lambda \in \lambda(Q_\lambda \hat{\hat{P}})\}$$

Ce théorème et son application sont vérifiés dans le cas des équations de Poisson $2D$. Ils permettent de comprendre l'idée d'approximation de l'opérateur P dans une base expliquant les modes de convergence les plus forts.

Différents tests numériques ont été effectués dans des cas $1D$ et $2D$ des équations de Poisson et Helmholtz. Notamment l'illustration du comportement de la méthode dans le cas de partitionnement algébrique (type METIS) où l'opérateur P n'est plus diagonalisable.

Enfin, les coûts, en terme de calcul, de construction et d'application du préconditionneur ont été évalués. Le nombre d'opérations lors de l'application de ARAS(q) est du même ordre que celui de RAS lorsque q est suffisamment petit, quelque soit le type de base d'approximation choisi. En revanche, le coût de construction du préconditionneur est proportionnel au nombre de vecteurs de base choisi. Dans le cas des méthodes d'approximation (b) avec SVD, le coût est de l'ordre de deux fois q applications de RAS sur le problème global dans le cas de l'algorithme le plus robuste, l'algorithme sans inversion 9. On remarque que pour l'algorithme 8 avec inversion, plus sensible aux perturbations, le coût est divisé par deux, c'est à dire qu'il est de l'ordre de q applications de RAS sur le problème global. Les réductions de calculs par parallélisation dépendent de la parallélisation de RAS.

This chapter is dedicated to the conception of a new preconditioner consisting in the enhancement of the Restricted Additive Schwarz preconditioner by the Aitken acceleration technique. We first present the Aitken acceleration technique in the vectorial case. Then we propose different methods to perform the acceleration in different bases. The corresponding part will focus on the way to build an approximation space to perform an Aitken acceleration. Since it is possible to algebraically build the error transfer operator on artificial interfaces of a domain decomposition method, it seems interesting to investigate the enhancement of a preconditioning technique by Aitken. We therefore propose a Richardson form of the Aitken Restricted Additive Schwarz iterative process. The new iterative process is a two-level method which works on the entire domain through the RAS process and works on the interface through the Aitken formula. This offers a new framework for the Aitken-Schwarz method which can be exploited to study its convergence and how the Aitken-Schwarz method can be used as a preconditioning technique in a formal approach. Then we propose some validation tests to illustrate the method and its convergence. Finally, we present the computational cost of applying an ARAS type preconditioner.

3.1 Aitken acceleration technique for iterative process converging linearly

In this section, we present the basics of the Aitken acceleration technique. One applies this technique, first known as the Δ^2 method, to accelerate the convergence of a sequence of scalars. We present how to apply it in the vectorial case and then write it for any iterative process which converges linearly. The theoretical cost of building the Aitken acceleration can be large and the matrix involved in its computing can be close to singular. A way to guarantee the capability to apply an Aitken acceleration in the vectorial case consists in computing the acceleration in an approximation space. We propose different ways to perform the acceleration in such a space.

3.1.1 Convergence acceleration by the Aitken's technique

Let $(u_n)_{n\in\mathbb{N}}$ be a sequence with a purely linear convergence or divergence toward ξ.

$$\lim_{k->\infty} u_k = \xi \tag{3.3}$$

Assuming that the sequence (u_n) converges toward ξ as a geometric sequence with a factor θ, $|\theta| < 1$:

$$u_{k+1} - \xi = \theta(u_k - \xi) \tag{3.4}$$

Then we can also write,

$$u_{k+2} - \xi = \theta(u_{k+1} - \xi) \tag{3.5}$$

θ can be obtained by subtracting equations (3.5) and (3.4):

$$\theta = \frac{u_{k+2} - u_{k+1}}{u_{k+1} - u_k} \tag{3.6}$$

And by substitution of the equation (3.6) into the equation (3.4), since $\theta \neq 1$,

$$\xi = \frac{u_k u_{k+2} - u_{k+1}^2}{u_{k+2} - 2u_{k+1} + u_k} \tag{3.7}$$

Then the limit of the sequence has been calculated. A similar analysis can be done in the vectorial case. The following definition illustrates the vectorial case.

Definition 3.1.1 *Let* $\left(\left(u_i^k\right)_{i=1,...,N} = u^k\right)_{k\in\mathbb{N}}$ *be a vectorial sequence converging toward* $(\xi)_{i=1,...,N} = \xi$ *purely linearly if*

$$u^{k+1} - \xi = P\left(u^k - \xi\right) \tag{3.8}$$

where $P \in \mathbb{R}^{n\times n}$ *is a constant error transfer operator independent of* k *and non-singular.*

We assume that there exists a norm $||.||$ such that $||P|| < 1$. Then P and ξ can be determined from a certain number of iterations $N+1$, using the equations:

$$\left(u^{N+1} - u^N, ..., u^2 - u^1\right) = P\left(u^N - u^{N-1}, ..., u^1 - u^0\right) \tag{3.9}$$

So, if $\left(u^N - u^{N-1}, ..., u^1 - u^0\right)$ is non-singular P can be written as :

$$P = \left(u^{N+1} - u^N, ..., u^2 - u^1\right)\left(u^N - u^{N-1}, ..., u^1 - u^0\right)^{-1} \tag{3.10}$$

Then, it is possible to compute ξ.

Proposition 3.1.2 *If* $||P|| < 1$ *then* $I - P$ *is non singular.*
Proof *We assume* $I - P$ *to be singular. Then there exists at least one non-zero vector* u *for which* $(I - P)\,u = 0$. *In this case,* $||u|| = ||Pu||$ *infers* $||P|| \geq 1$, *in contradiction with the initial assumption.* []

Hence, since $||P|| < 1$ we can derive ξ as

$$\xi = (I - P)^{-1} \left(u^{n+1} - Pu^n\right) \tag{3.11}$$

We want to directly apply the Aitken acceleration in the vectorial case, *i.e.*, to construct the matrix P or an approximation, and to apply the Aitken acceleration in the matrix form. Then, the following algorithm of the vectorial Aitken acceleration, based on the sequence of vectors written in their original canonical base of $\mathbb{R}^n$ named physical space, can be derived:

Algorithm 5 Vectorial Aitken acceleration in the physical space

Require: $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}^n$ an iterative method having a pure linear convergence

Require: $(u^k)_{1 \leq k \leq n+1}$, $n+1$ successive iterates of $\mathcal{G}$ starting from an arbitrary initial guess u^0

1: Form $E^k = u^{k+1} - u^k$, $0 \leq k \leq n$
2: **if** $\left[E^{n-1}, \ldots, E^0\right]$ is invertible **then**
3: $P = \left[E^n, \ldots, E^1\right] \left[E^{n-1}, \ldots, E^0\right]^{-1}$
4: $u^\infty = (I_n - P)^{-1}(u^{n+1} - Pu^n)$

This algorithm is limited to sequence of small size vectors because it needs $n+1$ iterations, where n is the vector size. The main difficulty is inverting the matrix $\left[E^{n-1}, \ldots, E^0\right]$, which can be close to singular.

The objective is then to accelerate the pure linear convergence of the sequence of vectors, saving as much as computation as possible.

We propose here to write the error of the iterative process, which leads to the sequence of vectors $(u_k)_{k \in \mathbb{N}}$, in a base where the modes of the error can be uncoupled or weakly coupled (eigenvalues, Fourier), or where the vectorial sequence can be compressed (SVD).

3.1.2 Approximation when modes are uncoupled or weakly coupled

The Aitken's technique was used to accelerate sequences of Schwarz domain decomposition method solutions. It has been shown that the so-called interface's solution which are the data dependencies between domains, converge linearly. Different approaches of writing a base to represent the error on the interface were proposed for elliptic problems (Garbey 2005) for the past ten years. We recall some basics of those techniques based on eigenvalue decomposition of a semi-discretized error transfer operator and multilevel techniques with FFT. We can group those approaches in the family of methods dedicated to Schwarz processes for which the modes of the error are uncoupled or weakly coupled. At the end of the subsection, we present the technique with NUDFT (Frullone & Tromeur-Dervout 2006), which is also based on a Fourier transform, but can provide acceleration for non-regular grids and possibly strongly-coupled modes.

3.1.2.1 Using Eigenvalue decomposition of P

In (Garbey 2005), the eigen pair is computed on the trace transfer operator P which links the Schwarz iterate errors on the interface. Then three algorithms are proposed:

(a) a bandwidth approximation of P: a block diagonal approximation of P is performed and then the eigen pair of this matrix block is searched,

(b) a coarse grid approximation of P: it consists in defining a coarse grid and having restriction $T_{h/H}$ and projection $T_{H/h}$ to exchange between the fine grid and the coarse grid and to define the coarse approximation of P as $T_{h/H}P_HT_{H/h}$,

(c) a compact representation of the interface: a least square approximation of the Schwarz iterates solution is done on a regular mesh where a sine or cosine expansion is performed, depending on the boundary condition.

Some accelerations have been obtained on finite volume with approach (b), while better results come from (c) with very few modes in the expansion even if the convergence seems to retrieve the original rate of convergence after the first acceleration. Strategy (a) works quite well with a band approximation of P of size $2Z+1$ with $Z = \{1, 2, 3\}$ on a regular grid with a convection-diffusion operator close to the Laplacian for which the P matrix is diagonal. Nevertheless, the eigenvalue problem is very sensitive to perturbations of the matrix entries. Consequently, not taking into account a part of the P matrix can greatly deteriorate the computed eigen base to represent the property of pure linearity of the sequence of iterate solutions.

We would like to write the Aitken formula with an exact P which can be written in the space spanned by the eigenvectors. In some cases, it is possible to diagonalize P. Then we denote by $\hat{\hat{P}}$ the error transfer operator in the space generated by all the eigenvectors. Since the convergence of each mode is independent, the matrix $\hat{\hat{P}}$ is diagonal. Considering $\mathbb{U} \in \mathbb{R}^{n\times n}$, the transfer matrix into the eigenvalues' space, we write for convenience the eigenvalue decomposition of $P \in \mathbb{R}^{n\times n}$ as,

$$P = \mathbb{U}\hat{\hat{P}}\mathbb{U}^{-1} \tag{3.12}$$

Then the acceleration is written as,

$$\hat{\hat{\xi}} = \left(I - \hat{\hat{P}}\right)^{-1}\left(\hat{\hat{u}}_{N+1} - \hat{\hat{P}}\hat{\hat{u}}_N\right) \tag{3.13}$$

The acceleration, in this space, can be done truncating $\hat{\hat{P}}$, which is diagonal, and performing the acceleration only on the q strongest values of the spectrum. Let $Q_\lambda \in \mathbb{R}^{n\times n}$, $q_l = 1$ if $1 \leq l \leq q$ and $q_l = 0$ if $q < l$. Then the acceleration with a matrix P approximated in the eigenvalue space can be written as,

$$\hat{\hat{\xi}} = \left(I - Q_\lambda\hat{\hat{P}}\right)^{-1}\left(\hat{\hat{u}}_{N+1} - Q_\lambda\hat{\hat{P}}\hat{\hat{u}}_N\right) \tag{3.14}$$

Then, only the strongest modes are accelerated.

3.1.2.2 Using Discrete Fourier Transform or Non Uniform Discrete Fourier Transform

Another way to build an approximation space is to consider the Fourier transform of a solution on the artificial interface. This technique was the first method used on the Laplacian operator on a regular grid, but it should not work on non-regular grids. Therefore, different solutions have been proposed. The BVDT approach (Boursier *et al.* 2005) consists in building a second grid which is regular and represents the interface by projection. The projection was defined by a Chebychev polynomial and then the transform was performed for the regular grid. However this method did not provide good results. Hence, another approach based on Non-Uniform DFT, that takes into account coupling between modes in P, has been introduced (Frullone & Tromeur-Dervout 2006).

In certain cases, the matrix $\hat{P}$ can be built analytically in the Fourier base (see subsection 3.3.4). The approach with the NUDFT set up the basics of an explicit building of the error transfer operator in a given base. In the following, we describe the technique while using the Fourier base.

Let $\hat{u} \in \mathbb{C}^n$ be the Fourier transform of $u \in \mathbb{R}^n$. Then, we can write the Aitken Formula in the Fourier space:

$$\hat{u}^{l+3} - \hat{u}^{l+2} = \hat{P}(\hat{u}^{l+2} - \hat{u}^{l+1}) \tag{3.15}$$

This matrix $\hat{P}$ has the same size as the matrix P. Nevertheless, we have more flexibility to define some consistent approximation of this matrix, since we have access to an *a posteriori* estimate based on the module value of the Fourier coefficients.

The explicit building of $\hat{P}$ consists to compute how the base functions Φ_{jk} are modified by the Schwarz iteration. Figure 3.1 describes the steps for constructing the matrix $\hat{P}$. Step (a) starts from the base function Φ_{jk} and gets its value on the interface in the physical space. Then step (b) performs a complete Schwarz iteration with zero local right hand sides and homogeneous boundary conditions on the others artificial interfaces. Step (c) decomposes the trace solution on the interface in the base Φ. Thus, we obtain column k of the matrix $\hat{P}$.

The full computation of $\hat{P}$ can be done in parallel, but it needs as much local domain solution as the number of interface points (*i.e.* the size of the matrix $\hat{P}$).

Its adaptive computation is required to save computing. The Fourier modes' convergence gives a tool to select the Fourier modes that slow the convergence and that have to be accelerated. We compute the sequence of the error and study its convergence in the Fourier space, mode by mode. Then we can select the modes which slow the convergence and perform the acceleration on a small set of vectors. As for the method with the eigenvalues, we introduce a matrix $Q_{\mathcal{F}} \in \mathbb{R}^{n \times n}$, $q_l = 1$ if $1 \leq l \leq q$ and $q_l = 0$ if $q < l$. Then the acceleration with a matrix P approximated in the Fourier space can be written as,

$$\hat{\xi} = \left(I - Q_{\mathcal{F}}\hat{P}\right)^{-1} \left(\hat{u}_{N+1} - Q_{\mathcal{F}}\hat{P}\hat{u}_N\right) \tag{3.16}$$

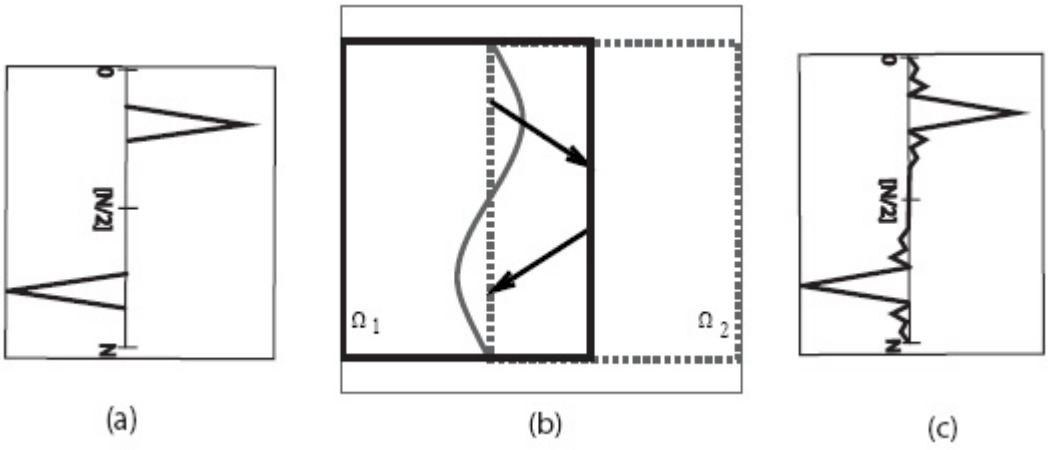

Figure 3.1: Steps to build the $\hat{P}$ matrix

We then consider the following algorithm to build $\hat{P}$ with respect to the modes that have not converged, and manage the Aitken acceleration adaptively (*i.e.* implicitly building $Q_{\mathcal{F}}$).

We write the algorithm for a problem with $2D$ interfaces below.

Algorithm 6 Aitken acceleration in the Fourier base

Require: $\mathcal{G} : \mathbb{R}^{qN^2} \to \mathbb{R}^{qN^2}$ the Schwarz B.C. update of domain's interfaces, let $u^m = \mathcal{G}(u^{m-1}) = (u_1^m, u_2^m, \ldots, u_q^m)$ be the m^{th} iterate on these q interfaces $u_i^m \in \mathbb{R}^{N^2}$

1: For $\left[u^{m+1}, u^m\right]$ write each interface i in its Fourier base Φ^i,
$\Phi^i = \left[\phi_{kl}^i(x_\gamma, y_\beta)\right]_{1\leq k,l\leq N}$ producing $\left[\hat{u}^{m+1}, \hat{u}^m\right]$ where $x_\gamma = \frac{\gamma 2\pi}{2N-1}$ and $y_\beta = \frac{\beta 2\pi}{2N-1}$

2: Set $(k_{i,max}, l_{i,max}) = \max_{1\leq(k,l)\leq N} \left\{(k,l) \backslash |\hat{u}_{i,kl}^{m+1} - \hat{u}_{i,kl}^m| > tol\right\}$,

3: Build the $p \stackrel{def}{=} \sum_{1\leq i\leq q} l_{i,max} k_{i,max}$ right hand side b_{kl}^i with ϕ_{kl}^i on the interface i and completed by zeros elsewhere.

4: Perform a complete Schwarz iteration, write the results in Φ, keeping only the selected modes to obtain $\hat{P} \in \mathbb{R}^{p\times p}$

5: Compute u^{m+2} and write it in Φ: $u^{m+2} \to \hat{u}^{m+2}$, keep only the selected modes: $\hat{u}^{m+2} \to \hat{\hat{u}}^{m+2}$

6: $\hat{\hat{u}}^\infty = (I_p - \hat{P})^{-1} \left(\hat{\hat{u}}^{m+2} - \hat{P}\, \hat{\hat{u}}^{m+1}\right)$ {Aitken Formula}

7: Complete $\hat{\hat{u}}^\infty$ with the non accelerated modes $\hat{u}^{m+2}$ and perform the inverse transformation $\hat{u}^\infty \to u^\infty$

A uniform DFT can be done only when the discretization is regular and when the interface is described in only one direction. Hence, this method strongly depends on the mesh and depends on the domain decomposition, since the domain needs to be split in only one direction. However, the DFT enables us to write an exact error transfer operator in the Fourier space. An approximation of the operator consists, in this case of a truncation given by the operator $Q_{\mathcal{F}}$.

As an enhancement of this technique, the use of the NUDFT enables us to have a non regular grid and an interface which can be described in more than one direction. This method works when modes are strongly coupled. However, the error transfer operator is built in an approximate space leading to an approximation of this operator.

Moreover, the techniques with Fourier transforms need a sufficient number of points to approximate the interface in order to be able to express the solution in the base.

3.1.3 Orthogonal "base" arising from an arbitrary coarse algebraic approximation of the interface.

One choice consists in having a coarse representation of the interface's solution $u \in \mathbb{R}^n$ from an algebraical point of view. Nevertheless, it is not possible to take a subset of q vectors of the canonical base of $\mathbb{R}^n$, since if some components of u are not reachable by the "base" $\mathbb{U}_q$, then the approximation $||u - \mathbb{U}_q(\mathbb{U}_q^T u)||$ will be very bad. This leads us to define $\mathbb{U}_q$ as a set of orthogonal vectors where each component comes from a random process so that each vector can contribute to a part of the searched solution at the interface. We split q such as $q = \sum_{i=1}^{p} q_i$ and we associate q_i random vectors to the interface Γ_i, $1 \leq i \leq p$. Then these q_i vectors are orthogonalized to form q_i columns of the "base" $\mathbb{U}_q$.

This strategy is hazardous but can be a simple way to improve the convergence of a Schwarz process without knowledge of the problem and the mesh.

The orthogonal "base" $\mathbb{U}_q$ is obtained applying the same principle as illustrated in Figure 3.1, leading to Algorithm 7.

Algorithm 7 Vectorial Aitken acceleration in an arbitrarily-built space without inversion

Require: $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}^n$ an iterative method having a pure linear convergence

1: Compute q random vectors $v_i \in \mathbb{R}^n$ following a uniform law on $[0, 1]$
2: Orthogonalize those q vectors to form $\mathbb{U}_q \in \mathbb{R}^{n \times q}$
3: Apply **one** iteration of $\mathcal{G}$ on the homogeneous problem, $\mathbb{U}_q \to W = \mathcal{G}(\mathbb{U}_q)$
4: Set $\hat{P} = \mathbb{U}_q^T W$
5: $\hat{\xi} = (I_q - \hat{P})^{-1} (\hat{u}^1 - \hat{P}\, \hat{u}^0)$ {Aitken Formula}
6: $\xi = \mathbb{U}_q \hat{\xi}$

The drawback of this method is that it is not possible to control the quality of the base to perform the acceleration. A more controllable method will be preferred. In the following subsection, we propose a different starting point to build the base. The main idea will be that we can compress the vectorial sequence using a Singular Value Decomposition. Since the $\mathbb{U}_q$ matrix is built, P is built the same way.

3.1.4 Approximation compressing the vectorial sequence

A totally algebraic method based on the Singular Value Decomposition of the Schwarz solutions on the interface has been proposed for cases where the modes of the error could be strongly coupled (Tromeur-Dervout 2009). This method offers the possibility for the Aitken-Schwarz method to be used on a large class of problems without mesh consideration.

This section focuses on the definition of the orthogonal "base" $\mathbb{U}_q$. In the following, we use the term "base" to denote a spanning vectors set that defines the approximation space. The key point of these preconditioners' efficiency is the choice of this orthogonal "base" $\mathbb{U}_q$. It must be sufficiently rich to numerically represent the solution at the interface, but it has to be not too large for the computational efficiency.

As we pointed out in Subsection 3.1.1, the main difficulty is inverting the matrix $\left[E^{n-1},\ldots,E^{0}\right]$, which can be close to singular. In a computation, most of the time is consumed solving noise that does not actually contribute to the solution. The singular value decomposition offers a tool to concentrate the effort only on the main parts of the solution.

3.1.4.1 The singular value decomposition

A singular-value decomposition (SVD) of a real $n \times m$ $(n > m)$ matrix A is its factorization into the product of three matrices:

$$A = \mathbb{U}\Sigma\mathbb{V}^*, \tag{3.17}$$

where $\mathbb{U} = [U_1,\ldots,U_m]$ is a $n \times m$ matrix with orthonormal columns, Σ is a $n \times m$ non-negative diagonal matrix with $\Sigma_{ii} = \sigma_i$, $1 \leq i \leq m$ and the $m \times m$ matrix $\mathbb{V} = [V_1,\ldots,V_m]$ is orthogonal. The left $\mathbb{U}$ and right $\mathbb{V}$ singular vectors are the eigenvectors of AA^* and A^*A respectively. It readily follows that $Av_i = \sigma_i u_i$, $1 \leq i \leq m$

Below, we recall some properties of SVD. Assume that the values $\sigma_i, 1 \leq i \leq m$ are ordered in decreasing order and there exists an r such that $\sigma_r > 0$ while $\sigma_r + 1 = 0$. Then A can be decomposed in a dyadic decomposition:

$$A = \sigma_1 U_1 V_1^* + \sigma_2 U_2 V_2^* + \ldots + \sigma_r U_r V_r^*. \tag{3.18}$$

This means that SVD provides a way to find optimal lower dimensional approximations of a given series of data. More precisely, it produces an orthonormal base for representing the data series in a certain least squares optimal sense. This can be summarized by the theorem of Schmidt-Eckart-Young-Mirsky:

Theorem 3.1.3 *A non-unique minimizer X_* of the problem $\min_{X,rankX=k} ||A - X||_2 = \sigma_{k+1}(A)$, provided that $\sigma_k > \sigma_{k+1}$, is obtained by truncating the dyadic decomposition of (3.18) to contain the first k terms: $X_* = \sigma_1 U_1 V_1^* + \sigma_2 U_2 V_2^* + \ldots + \sigma_k U_k V_k^*$*

The SVD of a matrix is well-conditioned with respect to perturbations of its entries. Consider the matrix $A, B \in \mathbb{R}^n$, the Fan inequalities write $\sigma_{r+s+1}(A+B) \leq \sigma_{r+1}(A) + \sigma_{s+1}(B)$ with $r, s \geq 0$, $r+s+1 \leq n$. Considering the perturbation matrix E such that $||E|| = O(\varepsilon)$, then $|\sigma_i(A+E) - \sigma_i(A)| \leq \sigma_1(E) = ||E||_2$, $\forall i$. This property does not hold for eigenvalues decomposition where small perturbations in the matrix entries can cause a large change in the eigenvalues.

This property allows us to search the acceleration of the convergence of the sequence of vectors in the base linked to its SVD.

Proposition 3.1.4 *Let $(u^k)_{1\leq k\leq q}$ be q successive iterates satisfying the pure linear convergence property: $u^k - u^\infty = P(u^{k-1} - u^\infty)$. Then there exists an orthogonal base $\mathbb{U}_q = \left[U^1, U^2, \ldots, U^q\right]$ of a subspace of $\mathbb{R}^n$ such that*

- *$\alpha_l^k = \sigma_l V_{kl}^*$ with $(\sigma_l)_{l\in\mathbb{N}}$ decreasing and $|V_{kl}^*| \leq 1 \Rightarrow \forall l \in \{1, ..., q\}$, $|\alpha_l^k| \leq |\sigma_l|$*
- *$u^k = \sum_{l=1}^q \alpha_l^k U^l, \forall k \in \{1, ..., q\}$*

One can write:

$$(\alpha_1^{k+1} - \alpha_1^k, \ldots, \alpha_q^{k+1} - \alpha_q^k)^T = \hat{P}(\alpha_1^k - \alpha_1^{k-1}, \ldots, \alpha_q^k - \alpha_q^{k-1})^T \tag{3.19}$$

where $\hat{P} \stackrel{def}{=} \mathbb{U}_q^* P \mathbb{U}_q$. Moreover $(\alpha_1^\infty, \ldots, \alpha_q^\infty)^T$ obtained by the acceleration process represents the projection of the limit of the sequence of vectors in the space generated by $\mathbb{U}_q$.

Proof *By theorem 3.1.3 there exists a SVD decomposition of $\left[u^1, \ldots, u^q\right] = \mathbb{U}_q \Sigma \mathbb{V}^*$ and we can identify α_l^k as $\sigma_l V_{kl}^*$. The orthonormal property of $\mathbb{V}$ associated to the decrease of σ_l with increasing l leads to have α_l^k bounded by $|\sigma_l|$: $\forall l \in \{1, ..., q\}$, $|\alpha_l^k| \leq |\sigma_l|$.*
Taking the pure linear convergence of u^k in the matrix form, and applying $\mathbb{U}_q$ leads to:

$$\mathbb{U}_q^*(u^k - u^\infty) = \mathbb{U}_q^* P \mathbb{U}_q \mathbb{U}_q^*(u^{k-1} - u^\infty) \tag{3.20}$$

$$(\alpha_1^k - \gamma_1^\infty, \ldots, \alpha_q^k - \gamma_q^\infty)^T = \hat{P}(\alpha_1^{k-1} - \gamma_1^\infty, \ldots, \alpha_q^{k-1} - \gamma_q^\infty)^T \tag{3.21}$$

where $(\gamma_j^\infty)_{1\leq j\leq q}$ represents the projection of u^∞ on the span $\{U_1, \ldots, U_q\}$. []

We can then derive Algorithm 8. This algorithm is similar to Algorithm 5 since the error transfer operator is defined using the errors of the linear iterative process in a space arising from the Singular Values Decomposition of $q+2$ successive iterates. Therefore, the third step of Algorithm 5 is equivalent to the sixth step of Algorithm 8.

Proposition 3.1.5 *Algorithm 8 converges to the limit u^∞.*
Proof *As the sequence of vector u^k converges to a limit u^∞ then we can write*

Algorithm 8 Vectorial Aitken acceleration in the SVD space with inversion

Require: $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}^n$ an iterative method having a pure linear convergence

Require: $(u^k)_{1\leq k\leq q+2}$, $q+2$ successive iterates of $\mathcal{G}$ starting from an arbitrary initial guess u^0

1: Form the SVD decomposition of $Y = \left[u^{q+2}, \ldots, u^1\right] = \mathbb{U}SV^T$
2: Set the index l such that $l = \max_{1\leq i\leq m+1}\{S(i,i) > tol\}$, {ex.:$tol = 10^{-12}$.}
3: Set $\hat{Y}_{1:l,1:l+2} = S_{1:l,1:l}V^T_{1:l,q-l:q+2}$
4: Set $\hat{E}_{1:l,1:l+1} = \hat{Y}_{1:l,2:l+2} - \hat{Y}_{1:l,1:l+1}$
5: **if** $\hat{E}_{1:l,1:l}$ is non singular **then**
6: $\quad \hat{P} = \hat{E}_{1:l,2:l+1}\hat{E}^{-1}_{1:l,1:l}$
7: $\quad \hat{y}^{\infty}_{1:l,1} = (I_l - \hat{P})^{-1}\,(\hat{Y}_{1:l,l+1} - \hat{P}\hat{Y}_{1:l,l})$ {Aitken Formula}
8: $\quad u^{\infty} = \mathbb{U}_{:,1:l}\,\hat{y}^{\infty}_{1:l,1}$

$\Xi = \left[u^1, \ldots, u^q\right] = [u^{\infty}, \ldots, u^{\infty}] + E$ *where* E *is an* $n \times q$ *matrix with decreasing coefficients with respect to the columns. The SVD of* $\Xi^{\infty} = [u^{\infty}, \ldots, u^{\infty}]$ *leads to have* $U^1 = u^{\infty}$ *and* $\sigma_i(\Xi^{\infty}) = 0$, $i \geq 2$. *The fan inequalities lead to* $\sigma_i(\Xi) \leq \sigma_1(E) = ||E||_2, i \geq 2$. *Consequently, Algorithm 8 decreases the number of non zero singular values at each loop iteration.* []

In Algorithm 8, building P requires the inversion of the matrix $\hat{E}^{-1}_{1:l,1:l}$, which can contain very small singular values even if we selected those greater than a certain tolerance. These singular values can deteriorate the ability of P to accelerate the convergence. If this is the case, we can proceed inverting this matrix with its SVD, replacing by zeros the singular values less than a tolerance instead of inverting them (see numerical recipes (Flannery *et al.* 2007)). A more robust algorithm can be obtained without inverting $\hat{E}^{-1}_{1:l,1:l}$. It consists in building P by applying the iterative method $\mathcal{G}$ to the selected columns of $\mathbb{U}_q$ that appears in Algorithm 8. Then $\hat{P} = \mathbb{U}^*_{1:n,1:l}\mathcal{G}(\mathbb{U}_{1:n,1:l})$ as done in Algorithm 9.

Algorithm 9 Vectorial Aitken acceleration in the SVD space without inversion

Require: $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}^n$ an iterative method having a pure linear convergence

Require: $(u^k)_{1\leq k\leq q+2}$, $q+2$ successive iterates of $\mathcal{G}$ starting from an arbitrary initial guess u^0

1: Form the SVD decomposition of $Y = \left[u^{q+2}, \ldots, u^1\right] = \mathbb{U}SV^T$
2: Set the index l such that $l = \max_{1\leq i\leq q+1}\{S(i,i) > tol\}$, {ex.:$tol = 10^{-12}$.}
3: Apply **one** iteration of $\mathcal{G}$ on the homogeneous problem with $l+2$ initial guesses $\mathbb{U}_{:,1:l} \to W_{:,1:l} = \mathcal{G}(\mathbb{U}_{:,1:l})$
4: Set $\hat{P} = \mathbb{U}^T_{:,1:l}W_{:,1:l}$
5: Set $\hat{Y}_{1:l,1:2} = S_{1:l,1:l}V^T_{1:l,q+1:q+2}$
6: $\hat{y}^{\infty}_{1:l,1} = (I_l - \hat{P})^{-1}\,(\hat{Y}_{1:l,2} - \hat{P}\,\hat{Y}_{1:l,1})$ {Aitken Formula}
7: $u^{\infty} = \mathbb{U}_{:,1:l}\,\hat{y}^{\infty}_{1:l,1}$

Several techniques to compute the SVD have been pro-

posed (Golub & Kahan 1965) (Dongara 1983) (Eisenstat & Ming 1995) (Brent & Luk 1985) (Jessup & Sorensen 1994) (Demmel & Kahan 1990). Two main approaches exist. One consists in two phases: the first phase transforms the matrix into a bidiagonal matrix and then the second phase computes the singular value of this bidiagonal matrix (Golub & Kahan 1965). In (Eisenstat & Ming 1995) an algorithm using a divide-and-conquer procedure based on a rank one modification of a bidiagonal matrix is proposed. A parallel divide-and-conquer algorithm was done in (Jessup & Sorensen 1994) and quasi-optimal implementation using cyclic reduction was performed in (Bar-On & Leoncini 2002). The second approach consists in using the Jacobi method which is based on transformation with Given's rotations to compute the eigenvalues of a symmetric matrix. In (Drmač & Veselić 2008a) (Drmač & Veselić 2008b), authors enhanced the Jacobi Algorithm to compute the SVD of a matrix with comparable computational efficiency as the bidiagonalisation approach with more accuracy. The Bidiagonal Divide and Conquer (BDC) algorithm computes all the singular values of an $N \times N$ matrix in $O(N^2)$ time and all the singular values and singular vectors in $O(N^3)$ time. By using the fast multipole method in (Carrier *et al.* 1988), BDC can be accelerated to compute all the singular values in $O(Nlog_2N)$ time and all the singular values and singular vectors in $O(N^2)$. Nevertheless, in our case, we do not need to compute all the singular values and corresponding singular vectors, but only those corresponding to the singular values larger than a fixed tolerance.

3.1.5 Categorisation of ways to approximate the error transfer operator

This section exhibits two different approaches in the process of approximating the error transfer operator P:

(a) Truncation of the operator in a complete base built analytically

(b) Approximation of the operator in a "base" built explicitly

For method (a), the operator can be analytically computed in a new base operating a base transfer. For example, a Discrete Fourier base transfer on a regular grid or a space spaned by the eigenvalues vector derived from the eigenvalue decomposition of a semi-discretized operator. In this case, the error transfer operator is considered as exact, since it is obtained from a base transfer. Then an approximation of P consists in a truncation of the operator in the new base. For method (b), an approximation base can be computed considering the effect of a Schwarz iteration on different solutions on the interface. In this case, it is not possible to analytically write the error transfer operator, but we obtain an approximation of the error transfer operation. Those techniques are called explicit because they need the computation of Schwarz solutions on the interface. Three kinds of approximations were presented: an approximation with NUDFT, an approximation in an orthogonal base arising from an arbitrary coarse algebraic approximation of the interface, and finally an approxima-

tion coming from the Singular Values Decomposition of Schwarz solutions on the interface.

While method (a) requires a regular discretization or strong assumption on the discretized operator A, method (b) is algebraic. Moreover, method (a) seems to be more sensitive to perturbations than method (b).

3.2 Formulation of the Aitken Restricted Additive Schwarz preconditioner and its composite forms

In this section we study the integration of the Aitken acceleration into a Richardson process in order to formulate a preconditioning technique based on Aitken. More precisely, we propose enhancing the RAS preconditioning technique, presented in Subsection 2.2.1, by the Aitken acceleration. We first present the mechanism of the method and develop the equation to extract a corresponding Richardson process. Then we point out that the method in its simple form does not exhibit the complete acceleration after one application, and we need an update, as when the method is used as solver. The result is a multiplicative preconditioner based on the Aitken RAS preconditioner. Finally we present those preconditioners in their approximated form in order to save computing. Note that the ARAS preconditioning technique has been presented in (Dufaud & Tromeur-Dervout 2010b) and in a submitted proceeding (Dufaud & Tromeur-Dervout 20YYa).

3.2.1 The Aitken Restricted Additive Schwarz preconditioner: ARAS

Let $\Gamma_i = (I_{m_{i,\delta}} - R_{i,\delta}R_{i,\delta}^T)W_{i,\delta}$ be the interface associated to $W_{i,\delta}$ and $\Gamma = \cup_{i=1}^{p}\Gamma_i$ be the global interface. Then $u_{|\Gamma} \in \mathbb{R}^n$ is the restriction of the solution $u \in \mathbb{R}^m$ on the Γ interface and $e^k_{|\Gamma} = u^k_{|\Gamma} - u^\infty_{|\Gamma}$ is the error of an iteration of a RAS iterative process (Subsection 2.2.1, equation (3.22)) at the interface Γ.

$$u^k = u^{k-1} + M^{-1}_{RAS,\delta}(f - Au^{k-1}) \tag{3.22}$$

In Subsection 2.1.2.2, we wrote that the Schwarz iterative method has a pure linear convergence. This property enables us to use the Aitken's technique presented in section 3.1. The previously mentioned $\mathcal{G}$ iterative process is replaced by a RAS iterative process. All the approximation techniques of the error transfer operator matrix P presented in Section 3.1 and referring to (Garbey 2005, Garbey & Tromeur-Dervout 2002, Barberou *et al.* 2003, Baranger *et al.* 2008, Frullone & Tromeur-Dervout 2006, Tromeur-Dervout 2009) can be applied.

Using the linear convergence property of the RAS method, we would like to write a preconditioner which includes the Aitken acceleration process. We introduce a restriction operator $R_\Gamma \in \mathbb{R}^{n\times m}$ from W to the global artificial interface Γ, with $R_\Gamma R_\Gamma^T = I_n$. The Aitken Restricted Additive Schwarz (ARAS) must generate

a sequence of solutions on the interface Γ, and accelerate the convergence of the Schwarz process from this original sequence. Then the accelerated solution on the interface replaces the last one. This could be written combining an AS or RAS process (Eq.(3.23a)) with the Aitken process written in $\mathbb{R}^{m\times m}$ (Eq.(3.23b)) and subtracting the Schwarz solution which is not extrapolated on Γ (Eq.(3.23c)). We can write the following approximation u^* of the solution u:

$$u^* = u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1}) \tag{3.23a}$$
$$+ R_\Gamma^T \left(I_n - P\right)^{-1} \left(u_{|\Gamma}^k - P u_{|\Gamma}^{k-1}\right) \tag{3.23b}$$
$$- R_\Gamma^T I_n R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1})\right) \tag{3.23c}$$

We would like to write u^* as an iterated solution derived from an iterative process of the form $u^* = u^{k-1} + M_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right)$, where $M_{ARAS,\delta}^{-1}$ is the Aitken-RAS preconditioner.

First of all, we write an expression of eq.(3.23b) depending on eq.(3.22) and which only involves the iterated solution $u^{k-1} \in \mathbb{R}^m$, as follows:

$$\begin{aligned}
eq.(3.23b) \;:=\; & R_\Gamma^T \left(I_n - P\right)^{-1} \left(u_{|\Gamma}^k - P u_{|\Gamma}^{k-1}\right) R_\Gamma \\
=\; & R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma \left(R_\Gamma^T I_n R_\Gamma u^k - R_\Gamma^T P R_\Gamma u^{k-1}\right) \\
\downarrow\; & \text{ with eq.(3.22)} \\
=\; & R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma \left(R_\Gamma^T I_n R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right)\right. \\
& \left. - R_\Gamma^T P R_\Gamma u^{k-1}\right) \\
=\; & R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma R_\Gamma^T I_n R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right) \\
& - R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma R_\Gamma^T P R_\Gamma u^{k-1} \\
=\; & R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right) \\
& - R_\Gamma^T \left(I_n - P\right)^{-1} P R_\Gamma u^{k-1}
\end{aligned}$$

Then, we re-write Eq.(3.23) with this new expression of Eq.(3.23b) as follows:

$$\begin{aligned}
u^* &= u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1}) \\
&\quad + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right) \\
&\quad - R_\Gamma^T \left(I_n - P\right)^{-1} P R_\Gamma u^{k-1} - R_\Gamma^T I_n R_\Gamma \left(u^{k-1} + M_{RAS,\delta}^{-1}(f - Au^{k-1})\right) \\
&\downarrow \quad \text{factorizing by } \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right) \\
&= \left(I_m - R_\Gamma^T I_n R_\Gamma + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma\right) \left(u^{k-1} + M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right) \\
&\quad - R_\Gamma^T \left(I_n - P\right)^{-1} P R_\Gamma u^{k-1} \\
&\downarrow \quad \text{isolating } u^{k-1} \text{ from } M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\
&= u^{k-1} + \left(-R_\Gamma^T I_n R_\Gamma + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma - R_\Gamma^T \left(I_n - P\right)^{-1} P R_\Gamma\right) u^{k-1} \\
&\quad + \left(I_m - R_\Gamma^T I_n R_\Gamma + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)
\end{aligned}$$

One can simplify $E = \left(R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma - R_\Gamma^T \left(I_n - P\right)^{-1} P R_\Gamma\right)$ as follows:

$$\begin{aligned}
E &= R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma \left(R_\Gamma^T I_n R_\Gamma - R_\Gamma^T P R_\Gamma\right) \\
&= R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma R_\Gamma^T \left(I_n - P\right) R_\Gamma \\
&= R_\Gamma^T \left(I_n - P\right)^{-1} \left(I_n - P\right) R_\Gamma \\
&= R_\Gamma^T I_n R_\Gamma
\end{aligned}$$

And then writes,

$$\begin{aligned}
u^* &= u^{k-1} + \left(-R_\Gamma^T I_n R_\Gamma + R_\Gamma^T I_n R_\Gamma\right) u^{k-1} \\
&\quad + \left(I_m - R_\Gamma^T I_n R_\Gamma + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\
&= u^{k-1} + \left(I_m - R_\Gamma^T I_n R_\Gamma + R_\Gamma^T \left(I_n - P\right)^{-1} R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\
&= u^{k-1} + \left(I_m + R_\Gamma^T \left(\left(I_n - P\right)^{-1} - I_n\right) R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)
\end{aligned}$$

Hence the formulation Eq.(3.23) leads to an expression of an iterated solution u^*:

$$u^* = u^{k-1} + \left(I_m + R_\Gamma^T \left(\left(I_n - P\right)^{-1} - I_n\right) R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right)$$

This iterated solution u^* can be seen as an accelerated solution of the RAS iterative process. Drawing our inspiration from the Stephensen's method (Stoer & Bulirsch 2002), we build a new sequence of iterates from the solutions accelerated by the Aitken acceleration method. Then, one considers u^* as a new u^k and writes the following ARAS iterative process:

$$u^k = u^{k-1} + \left(I_m + R_\Gamma^T \left(\left(I_n - P\right)^{-1} - I_n\right) R_\Gamma\right) M_{RAS,\delta}^{-1}\left(f - Au^{k-1}\right) \qquad (3.24)$$

Then we defined the ARAS preconditioner as

$$M_{ARAS,\delta}^{-1} = \left(I_m + R_\Gamma^T \left((I_n - P)^{-1} - I_n\right) R_\Gamma\right) \sum_{i=1}^{p} \tilde{R}_{i,\delta}^T A_{i,\delta}^{-1} R_{i,\delta} \tag{3.25}$$

Remark 1 The ARAS preconditioner can be considered as a two-level additive preconditioner. The preconditioner consists in computing a solution on an entire domain applying the RAS preconditioner and add components computed only on the interface Γ.

3.2.2 Composite Multiplicative form of ARAS: ARAS2

If P is known exactly, the ARAS process written in the equation (3.24) needs two steps to converge to the solution u with an initial guess $u^0 = 0$. Then we have:

Proposition 3.2.1 *If* P *is known exactly then we have* $A^{-1} = \left(2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1} A M_{ARAS,\delta}^{-1}\right)$ *that leads* $\left(I - M_{ARAS,\delta}^{-1} A\right)$ *to be a nilpotent matrix of degree 2.*

Proof *We consider the postulate: "If* P *is known exactly, the ARAS process written in eq.*(3.24) *needs two steps to converge to the solution with an initial guess* $u^0 = 0$ *".*

We write the two first iterations of the ARAS process for any initial guess $u^0 \in \mathbb{R}^m$ *and for all* $f \in \mathbb{R}^m$*:*

$$u^1 = u^0 + M_{ARAS,\delta}^{-1}\left(f - Au^0\right)$$

And the second iterations leads to:

$$\begin{aligned} u^2 &= u^1 + M_{ARAS,\delta}^{-1}\left(f - Au^1\right) \\ &= u^0 + M_{ARAS,\delta}^{-1}\left(f - Au^0\right) + M_{ARAS,\delta}^{-1}\left(f - A\left(u^0 + M_{ARAS,\delta}^{-1}\left(f - Au^0\right)\right)\right) \end{aligned}$$

Let $u^0 = 0$*, then,*

$$\begin{aligned} u^2 &= M_{ARAS,\delta}^{-1} f + M_{ARAS,\delta}^{-1}\left(f - A\left(M_{ARAS,\delta}^{-1} f\right)\right) \\ &= \left(2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1} A M_{ARAS,\delta}^{-1}\right) f \\ &= u \end{aligned}$$

Since this expression is true for all $f \in \mathbb{R}^m$ *we can write:*

$$A^{-1} = 2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1} A M_{ARAS,\delta}^{-1}$$

Now we can write:

$$\begin{aligned} u &= \left(2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1}AM_{ARAS,\delta}^{-1}\right) f \\ &= \left(I_m + I_m - M_{ARAS,\delta}^{-1}A\right) M_{ARAS,\delta}^{-1} f \\ &= M_{ARAS,\delta}^{-1} f + \left(I_m - M_{ARAS,\delta}^{-1}A\right) M_{ARAS,\delta}^{-1} f \\ &\downarrow \quad \textit{with } Au = f \\ &= M_{ARAS,\delta}^{-1} Au + \left(I_m - M_{ARAS,\delta}^{-1}A\right) M_{ARAS,\delta}^{-1} Au \end{aligned}$$

Thus,

$$\left(I_m - M_{ARAS,\delta}^{-1}A\right) u = \left(I_m - M_{ARAS,\delta}^{-1}A\right) M_{ARAS,\delta}^{-1} Au$$

Which is equivalent to

$$0 = \left(I_m - M_{ARAS,\delta}^{-1}A\right)^2 u, \; \forall u \in \mathbb{R}^m$$

Hence $\left(I_m - M_{ARAS,\delta}^{-1}A\right)$ *is nilpotent of degree* 2. []

The previous proposition leads to an approximation of A^{-1} written from the 2 first iterations of the ARAS iterative process (3.24). Those 2 iterations compute the Schwarz solutions sequence on the interface needed in order to accelerate the Schwarz method by the Aitken acceleration. We now write 2 iterations of the ARAS iterative process (3.24) for any initial guess and for all $u^{k-1} \in \mathbb{R}^m$.

$$\begin{aligned} u^{k+1} &= u^{k-1} + M_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\ &\quad + M_{ARAS,\delta}^{-1}\left(f - A\left(u^{k-1} + M_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right)\right)\right) \\ &= u^{k-1} + M_{ARAS,\delta}^{-1} f - M_{ARAS,\delta}^{-1} Au^{k-1} \\ &\quad + M_{ARAS,\delta}^{-1} f - M_{ARAS,\delta}^{-1} Au^{k-1} - M_{ARAS,\delta}^{-1} AM_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\ &= u^{k-1} + 2M_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right) - M_{ARAS,\delta}^{-1} AM_{ARAS,\delta}^{-1}\left(f - Au^{k-1}\right) \\ &= u^{k-1} + \left(2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1} AM_{ARAS,\delta}^{-1}\right)\left(f - Au^{k-1}\right) \end{aligned}$$

Then we defined the ARAS2 preconditioner as

$$M_{ARAS2,\delta}^{-1} = 2M_{ARAS,\delta}^{-1} - M_{ARAS,\delta}^{-1} AM_{ARAS,\delta}^{-1} \tag{3.26}$$

Remark 2 According to the linear algebra literature about preconditioning technique (Beauwens 2004), the ARAS2 preconditioner can be considered as a composite multilevel preconditioner. Actually, the ARAS2 preconditioner is a multiplicative form of ARAS which is itself an additive preconditioner adding an operation on the entire domain with RAS and an operation on a coarse interface with the Aitken formula.

3.2.3 Approximated form of ARAS and ARAS2

As the previous subsection suggests, since P is known exactly there is no need to use ARAS as a preconditioning technique. Nevertheless, when P is approximated, the Aitken acceleration of the convergence depends on the local domain solving accuracy, and the cost of the building of an exact P depends on the size n. This is why P is numerically approximated by $P_{\mathbb{U}_q}$ as in (Tromeur-Dervout 2009), defining $q \leq n$ orthogonal vectors $\mathbb{U}_q \in R^{n\times q}$, such as $q = \sum_{i=1}^{p} q_i$ with $q_i = q_j$, $1 \leq i,j \leq p$. q_i of these vectors being associated to Γ_i, $1 \leq i \leq p$, and having non-zero components only on Γ_i. Then it makes sense to use it as a preconditioning technique. ARAS(q) denotes the approximated form of ARAS considering q orthogonal vectors of a base $\mathbb{U}_q$:

$$M^{-1}_{ARAS(q),\delta} = \left(I_m + R_\Gamma^T \mathbb{U}_q \left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} - I_q\right) \mathbb{U}_q^T R_\Gamma\right) \sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta} \quad (3.27)$$

and ARAS2(q),

$$M^{-1}_{ARAS2(q),\delta} = 2M^{-1}_{ARAS(q),\delta} - M^{-1}_{ARAS(q),\delta} A M^{-1}_{ARAS(q),\delta} \quad (3.28)$$

3.3 Convergence of ARAS and ARAS2 and their approximated form

As an enhancement of the RAS preconditioning technique, ARAS and ARAS2 should have a better convergence rate than the RAS technique. We formulate the convergence rate of a RAS technique considering the linear convergence of the Restricted Additive Schwarz method and extend this formulation to the Aitken's technique. Then we propose a relation between the spectral radius of those methods.

In the following we note

$$T_* = (I - M_*^{-1} A) \quad (3.29)$$

Any Richardson's process can be written as

$$u^k = T_* u^{k-1} + c, \text{ where } c \in \mathbb{R}^n \text{is constant} \quad (3.30)$$

Remark 3 As the ARAS2 iterative process correspond to 2 *iterations of the ARAS process, we notice that* $T_{ARAS2} = T^2_{ARAS}$.

3.3.1 Ideal case

When building the preconditioner ARAS we exhibit the fact that T_{ARAS} is nilpotent when the error's transfer operator on the interface Γ, P is considered exact. This property gives the following proposition:

Proposition 3.3.1 *If P is known exactly then,*

$$\rho\left(T_{ARAS2}\right) = \rho\left(T_{ARAS}\right) = 0 \tag{3.31}$$

Proof *If P is known exactly then Proposition 3.2.1 is verified and T_{ARAS} and T_{ARAS2} are nilpotent. The spectral radius of a nilpotent matrix is equal to* 0. []

Remark 4 Obviously, $\rho\left(T_{ARAS2}\right) = \rho\left(T_{ARAS}\right) < \rho\left(T_{RAS}\right)$

But the matrix P is often numerically computed and then $\rho\left(T_{ARAS}\right)$ is no longer equal to 0. The value of $\rho\left(T_{ARAS}\right)$ depends on the accuracy of the local domain solutions and when P is written in another space, depends on the quality of this space. In the following we propose a framework to study the convergence of $T_{ARAS(q)}$ and $T_{ARAS2(q)}$. The goal is to provide key elements to understand the influence of approximating P in an orthogonal base on the preconditioner.

3.3.2 Convergence of RAS for an elliptic operator

In this subsection we express the convergence rate of a RAS iterative process considering its convergence on the artificial interfaces in Proposition 3.3.2. Since we can link the convergence of RAS on the entire domain to the convergence on the interface, it becomes possible to study the effect of modifying the error transfer operator P.

Proposition 3.3.2 *Let A be a discretized operator of an elliptic problem on a domain Ω. Let us consider a RAS iterative process such that $T_{RAS} = I - M_{RAS}^{-1}A$ defined on p domains. The data dependencies between domains is located on an artificial interface Γ. Then there exists an error's transfer operator on the interface Γ, P such as there exists a norm $||.||$ for which $||P|| < 1$. The convergence rate of T_{RAS} is*

$$\rho(T_{RAS}) = \max\left\{|\lambda| \,:\, \lambda \in \lambda(P)\right\} \tag{3.32}$$

Proof *In the case of elliptic problems the maximum principle is observed. Then, for the Schwarz method the error is maximal on artificial interfaces. We write the error of a Schwarz process starting from the definition of the RAS iterative method:*

$$u^{k+1} = T_{RAS}u^k + M_{RAS}^{-1}b \tag{3.33}$$

The convergence of such a process is given by (Axelsson 1996) (Ciarlet 1994):

$$e^k = T_{RAS}^k e^0 \tag{3.34}$$

On the interface, one can write:

$$e^k_{|\Gamma} = P^k e^0_{|\Gamma} \tag{3.35}$$

The error is maximal on the interface thus,

$$||e^k||_\infty = ||e^k_{|\Gamma}||_\infty \tag{3.36}$$

Equations 3.34, 3.35 and 3.36 lead to

$$||T^k_{RAS} e^0||_\infty = ||P^k e^0_{|\Gamma}||_\infty \tag{3.37}$$

Then we can write,

$$\begin{aligned} \sup_{||e^0||_\infty=1} \left(||T^k_{RAS} e^0||_\infty \right) &= \sup_{||e^0_{|\Gamma}||_\infty=1} \left(||P^k e^0_{|\Gamma}||_\infty \right) && (3.38) \\ &= ||P^k||_\infty = ||T^k_{RAS}||_\infty && (3.39) \end{aligned}$$

Hence,

$$\lim_{k->\infty} ||P^k||_\infty^{\frac{1}{k}} = \rho(P) = \rho(T_{RAS}) \tag{3.40}$$

□

3.3.3 Convergence of ARAS and ARAS2 in their approximated form

Here we focus on elliptic and separable operators. We propose a theorem giving the convergence rate of an ARAS iterative process when the error transfer operator can be exactly computed in a space spanned by the eigenvectors of P and then truncated to provide an approximation of the error transfer operator in the physical space.

Theorem 3.3.3 *Let A be a discretized operator of an elliptic problem on a domain Ω. Let us consider a RAS iterative process such as $T_{RAS} = I - M^{-1}_{RAS} A$ defined on p domains. Let the error transfer operator P on an interface Γ be diagonalisable. If P is diagonalisable, its decomposition in eigenvalues leads to $P = \mathbb{U}\hat{\hat{P}}\mathbb{U}^{-1}$ where for $i \in [\![1, n]\!]$, $\hat{\hat{P}} = diag(\lambda_i)$. The error on the interface Γ in the approximation space follows $\hat{e}^{k+1}_{|\Gamma} = \hat{\hat{P}}\hat{e}^k_{|\Gamma}$. Each mode converges linearly and independently from the others following $\hat{e}^{k+1}_{|\Gamma,i} = \lambda_i \hat{e}^k_{|\Gamma,i}$. Let $Q_\lambda \in \mathbb{R}^{n\times n}$ be a diagonal matrix such that $q_l = 1$ if $1 \leq l \leq q$ and $q_l = 0$ if $q < l$. Let $\bar{Q}_\lambda = I_n - Q_\lambda$. A coarse approximation of $\hat{\hat{P}}$ can be made choosing a set of q strong modes as $\tilde{P} = Q_\lambda \hat{\hat{P}}$. Writing the preconditioner as:*

$$M^{-1}_{ARAS(q),\delta} = \left(I_m + R^T_\Gamma \mathbb{U} \left(\left(I_n - \tilde{P} \right)^{-1} - I_n \right) \mathbb{U}^{-1} R_\Gamma \right) \sum_{i=1}^{p} \tilde{R}^T_{i,\delta} A^{-1}_{i,\delta} R_{i,\delta} \tag{3.41}$$

The spectral radius of $T_{ARAS(q)}$ is :

$$\rho(T_{ARAS(q)}) = \rho(\bar{Q}_\lambda \hat{\hat{P}}) = \lambda_{q+1} < \min\{|\lambda| : \lambda \in \lambda(Q_\lambda \hat{\hat{P}})\} \tag{3.42}$$

Proof *We consider the assumptions of the theorem and the formula of the approximated preconditioner 3.41. Equation* (3.23) *can be written for ARAS(q) as:*

$$\begin{aligned} u^* &= T_{RAS}u^{k-1} + M_{RAS}^{-1}b \\ &\quad + R_\Gamma^T \left(I_n - P\right)^{-1} \left(u_{|\Gamma}^k - P u_{|\Gamma}^{k-1}\right) R_\Gamma \\ &\quad - R_\Gamma^T I_n R_\Gamma \left(T_{RAS}u^{k-1} + M_{RAS}^{-1}b\right) \end{aligned}$$

We consider that on the interface:

$$\left(T_{RAS}u^{k-1} + M_{RAS}^{-1}b\right)_{|\Gamma} = P u_{|\Gamma}^{k-1} + c \tag{3.43}$$

With $c \in \mathbb{R}^n$, a constant vector independent of $u_{|\Gamma}$.

Extracting the interface's solution in the approximation space,

$$\hat{u}_{|\Gamma}^* = \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} + \hat{c} + \left(I_n - \tilde{P}\right)^{-1} \left(\hat{u}_{|\Gamma}^k - \tilde{P}\hat{u}_{|\Gamma}^{k-1}\right) - Q_\lambda \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} - Q_\lambda \hat{c}$$

Then,

$$\hat{u}_{|\Gamma}^k = \bar{Q}_\lambda \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} + Q_\lambda \hat{u}_{|\Gamma}^\infty + \bar{Q}_\lambda \hat{c} \tag{3.44}$$

The error on the interface is

$$\hat{u}_{|\Gamma}^\infty - \hat{u}_{|\Gamma}^k = \hat{u}_{|\Gamma}^\infty - \bar{Q}_\lambda \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} - Q_\lambda \hat{u}_{|\Gamma}^\infty - \bar{Q}_\lambda \hat{c}$$

Thus,

$$\hat{e}_{|\Gamma}^k = \bar{Q}_\lambda \left(\hat{u}_{|\Gamma}^\infty - \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} - \hat{c}\right) \tag{3.45}$$

Regarding Equation (3.43), $\hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} + \hat{c} = \hat{u}_{|\Gamma}^k$, *and then,*

$$\hat{u}_{|\Gamma}^\infty - \hat{\hat{P}}\hat{u}_{|\Gamma}^{k-1} - \hat{c} = \hat{e}_\Gamma^k = \hat{\hat{P}}\hat{e}_\Gamma^k \tag{3.46}$$

Hence we write,

$$\hat{e}_{|\Gamma}^k = \bar{Q}_\lambda \hat{\hat{P}}\hat{e}_{|\Gamma}^{k-1} \tag{3.47}$$

We showed that the ARAS iterative process has an error transfer operator, $\bar{Q}_\lambda \hat{\hat{P}}$, equal to the part of the error transfer operator $\hat{\hat{P}}$ that we did not compute. We note that $||\bar{Q}_\lambda \hat{\hat{P}}|| \leq ||\hat{\hat{P}}|| < 1$.

A is a discretized operator of an elliptic problem. We can therefore apply Proposition 3.3.2 and write:

$$\rho(T_{ARAS(q)}) = \rho(\bar{Q}_\lambda \hat{\hat{P}}) \tag{3.48}$$

We have then proven Equation 3.42. []

Remark 5 For a separable operator, the error transfer operator on an interface between two domains is diagonalisable (Garbey 2005). Then, the error transfer operator P on an interface Γ is diagonalisable for a global operator which is separable and for which the interfaces of all the domains are parallel to one discretization direction.

3.3.4 Convergence study in the case of the $2D$ Poisson's equation

We consider a simple case of an elliptic problem on a rectangle defined in Equation (3.49). The problem is discretized by $2D$ finite differences and decomposed into two domains, Ω_1 and Ω_2, of the same size.

$$\left\{ \begin{array}{rcl} -\triangle u & = & f, \text{ in } \Omega = [0,1] \times [0,\pi] \\ u & = & 0, \text{ on } \partial\Omega \end{array} \right. \tag{3.49}$$

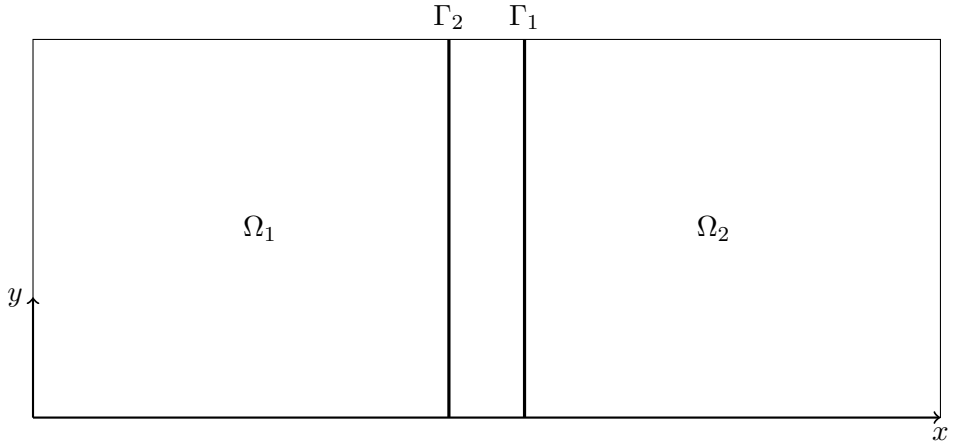

Figure 3.2: $2D$ domain decomposition for Poisson's equation

In the Fourier base, for a separable operator with two artificial interfaces, the acceleration is written with $\hat{P} = \left(\begin{smallmatrix} 0 & \hat{P}_{\Gamma_2} \\ \hat{P}_{\Gamma_1} & 0 \end{smallmatrix} \right)$ such as

$$\begin{pmatrix} \hat{e}^k_{\Gamma_1} \\ \hat{e}^k_{\Gamma_2} \end{pmatrix} = \begin{pmatrix} 0 & \hat{P}_{\Gamma_2} \\ \hat{P}_{\Gamma_1} & 0 \end{pmatrix} \begin{pmatrix} \hat{e}^{k-1}_{\Gamma_1} \\ \hat{e}^{k-1}_{\Gamma_2} \end{pmatrix} \tag{3.50}$$

We study the case of a regular grid and write the semi-discretized $2D$ Poisson operator on $[0,\Gamma_1]\times[0,\pi]\cup[\Gamma_2,1]\times[0,\pi]$, $\Gamma_1 > \Gamma_2$. We consider N_x+1 points in the direction x. The overlap in terms of number of step size in direction x is denoted by δ. And we denote by λ_l the eigenvalues of the discretized operator $-\frac{\partial^2}{\partial y^2}$ considering a second order finite difference scheme. We find the coefficient of the matrices $\hat{P}_{\Gamma_i}$ solving the equations:

$$\left\{ \begin{array}{rcl} -\frac{\hat{u}^1_{i+1,l} - 2\hat{u}^1_{i,l} + \hat{u}^1_{i-1,l}}{h_x^2} + \lambda_l \hat{u}^1_{i,l} & = & 0 \ , \ i \in [\![1, N_x - 1]\!] \\ \hat{u}^1_{0,l} & = & 0 \\ \hat{u}^1_{N_x,l} & = & 1 \end{array} \right. \tag{3.51}$$

$$\left\{ \begin{array}{rcl} -\frac{\hat{u}^2_{i+1,l} - 2\hat{u}^2_{i,l} + \hat{u}^2_{i-1,l}}{h_x^2} + \lambda_l \hat{u}^2_{i,l} & = & 0 \ , \ i \in [\![1, N_x - 1]\!] \\ \hat{u}^2_{0,l} & = & 1 \\ \hat{u}^2_{N_x,l} & = & 0 \end{array} \right. \tag{3.52}$$

The roots of those equations are such as,

$$r^1 = \frac{2+\lambda_l h_x^2 + \sqrt{\lambda_l^2 h_x^4 + 4\lambda_l h_x^2}}{2} \tag{3.53}$$

$$r^2 = \frac{2+\lambda_l h_x^2 - \sqrt{\lambda_l^2 h_x^4 + 4\lambda_l h_x^2}}{2} \tag{3.54}$$

Then for the first domain Ω_1 the solutions have the form:

$$\hat{u}^1_{l,j} = \frac{r_1^j - r_2^j}{r_1^{N_x} - r_2^{N_x}} \tag{3.55}$$

Thus,

$$\hat{u}^1_{N_x-\delta,l} = \frac{r_1^{N_x-\delta} - r_2^{N_x-\delta}}{r_1^{N_x} - r_2^{N_x}} \tag{3.56}$$

$\hat{P}_{\Gamma_1}$ appears to be diagonal and its diagonal coefficients δ_{l,Γ_1} can be analytically derived such as:

$$\delta_{l,\Gamma_1} = \hat{u}^1_{N_x-\delta,l} * \hat{u}^2_{\delta,l} \tag{3.57}$$

$$= \left(\frac{r_1^{N_x-\delta} - r_2^{N_x-\delta}}{r_1^{N_x} - r_2^{N_x}}\right)\left(\frac{-r_2^{N_x} r_1^{\delta} + r_1^{N_x} r_2^{\delta}}{r_1^{N_x} - r_2^{N_x}}\right) \tag{3.58}$$

Exactly the same development can be done to find δ_{l,Γ_2}.

Remark 6 If the two domains have the same size, then $\delta_{l,\Gamma_1} = \delta_{l,\Gamma_2}$.

The $\hat{P}$ matrix has the form:

$$\hat{P} = \begin{pmatrix} 0 & & & \delta_{1,\Gamma_2} & & \\ & \ddots & & & \ddots & \\ & & 0 & & & \delta_{n,\Gamma_2} \\ \delta_{1,\Gamma_1} & & & 0 & & \\ & \ddots & & & \ddots & \\ & & \delta_{n,\Gamma_1} & & & 0 \end{pmatrix}, \text{ with } 1 > \delta_{1,\Gamma_i} \geq ... \geq \delta_{n,\Gamma_i}. \tag{3.59}$$

For the sake of simplicity we consider that the domain Ω_1 and Ω_2 have the same size. We note $\delta_l = \delta_{l,\Gamma_1} = \delta_{l,\Gamma_2}$. Then we can calculate the determinant of $(\hat{P} - \lambda I_n)$:

$$det(\hat{P} - \lambda I_n) = \begin{vmatrix} -\lambda & & & \delta_1 & & \\ & \ddots & & & \ddots & \\ & & -\lambda & & & \delta_n \\ \delta_1 & & & -\lambda & & \\ & \ddots & & & \ddots & \\ & & \delta_n & & & -\lambda \end{vmatrix}$$

We replace the first column by its sum with the column (n+1). Then, we subtract the first row to the new row (n+1). We exhibit a pivot equal to $(\delta_l - \lambda)$ in the first entry of the array. We develop from the first column. A new pivot equal to $-(\delta_l + \lambda)$ appears in the entry (n,n). We develop by the n^{th} column and obtain the following:

$$\begin{aligned} det(\hat{P} - \lambda I_n) &= (\delta_1 - \lambda)(\delta_1 + \lambda) \begin{vmatrix} -\lambda & & \delta_2 & & \\ & \ddots & & \ddots & \\ & & -\lambda & & \delta_n \\ \delta_2 & & & -\lambda & \\ & \ddots & & & \ddots \\ & & \delta_n & & -\lambda \end{vmatrix} \\ &= \prod_{l=1}^{n} (\delta_l - \lambda)(\delta_l + \lambda) \end{aligned}$$

The result is obtained recalling the procedure. Hence the spectrum is $\{\delta_1, ..., \delta_n, -\delta_1, ..., -\delta_n\}$.

We showed that the error transfer operator is diagonalisable for this problem. P can be written in a base $\mathbb{U}$ of eigenvectors as follows:

$$P = \mathbb{U}\hat{\hat{P}}\mathbb{U}^{-1}, \text{ with } \hat{\hat{P}} = diag(\delta_1, ..., \delta_n, -\delta_1, ..., -\delta_n) \tag{3.60}$$

Then we can estimate the convergence rate of the RAS, ARAS(q) and ARAS2(q) applying Theorem 3.3.3.

$$\rho(T_{RAS}) = \delta_1 \tag{3.61}$$

$$\rho(T_{ARAS(q)}) = \delta_{q+1} \tag{3.62}$$

$$\rho(T_{ARAS2(q)}) = \delta_{q+1}^2 \tag{3.63}$$

Because the eigenvalues and the values of $\hat{P}_{\Gamma_i}$ are equal we can verify a correspondence between the approximation by truncation in the eigenvectors space and the Fourier space. Selecting the first Fourier mode corresponds to selecting the highest eigenvalues. Let us introduce the transfer matrix C_{Γ_i} from the real space to the Fourier space and the transfer matrix D_{Γ_i} from the Fourier space to the real space:

$$\begin{array}{rcl} C_{\Gamma_i} : \mathbb{R}^n \longrightarrow \mathbb{C}^n & \text{and} & D_{\Gamma_i} : \mathbb{C}^n \longrightarrow \mathbb{R}^n \\ e_{|\Gamma_i} \longmapsto \hat{e}_{|\Gamma_i} & & \hat{e}_{|\Gamma_i} \longmapsto e_{|\Gamma_i} \end{array}$$

Then we write

$$P = D\hat{P}C = \begin{pmatrix} D_{\Gamma_1} & 0 \\ 0 & D_{\Gamma_2} \end{pmatrix} \begin{pmatrix} 0 & \hat{P}_{\Gamma_2} \\ \hat{P}_{\Gamma_1} & 0 \end{pmatrix} \begin{pmatrix} C_{\Gamma_1} & 0 \\ 0 & C_{\Gamma_2} \end{pmatrix} \tag{3.64}$$

The approximation is done by applying the operator

$$Q_{\mathcal{F}} = \begin{pmatrix} Q_{\Gamma_1,\mathcal{F}} & 0 \\ 0 & Q_{\Gamma_2,\mathcal{F}} \end{pmatrix}$$

where $Q_{\Gamma_1,\mathcal{F}} = diag(q_l)$, $q_l = 1$ if $1 \leq l \leq q$ and $q_l = 0$ if $q < l$. Then we write the preconditioner

$$M^{-1}_{ARAS(q)} = (I + R^T_\Gamma D(I_n - Q_\mathcal{F}\hat{P})^{-1} - I_n)CR_\Gamma)M^{-1}_{RAS} \tag{3.65}$$

As previously we introduce a matrix $\bar{Q}_\mathcal{F} = (I - Q_\mathcal{F})$.

We can then follow the demonstration done in the proof of theorem 3.3.3 writing the error on the interface in the Fourier space as

$$\hat{u}^\infty_{|\Gamma} - \hat{u}^k_{|\Gamma} = \hat{u}^\infty_{|\Gamma} - \bar{Q}_\mathcal{F}\hat{P}\hat{u}^{k-1}_{|\Gamma} - Q_\mathcal{F}\hat{u}^\infty_{|\Gamma} - \bar{Q}_\mathcal{F}\hat{c} \tag{3.66}$$

Thus,

$$\hat{e}^k_{|\Gamma} = \bar{Q}_\mathcal{F}\left(\hat{u}^\infty_{|\Gamma} - \hat{P}\hat{u}^{k-1}_{|\Gamma} - \hat{c}\right) \tag{3.67}$$

Regarding equation (3.43), $\hat{P}\hat{u}^{k-1}_{|\Gamma} + \hat{c} = \hat{u}^k_{|\Gamma}$, and then,

$$\hat{e}^k_{|\Gamma} = \bar{Q}_\mathcal{F}\hat{e}^k_{|\Gamma} \tag{3.68}$$

As $\hat{e}^k_{|\Gamma} = \hat{P}\hat{e}^{k-1}_{|\Gamma}$ we write

$$\hat{e}^k_{|\Gamma} = \bar{Q}_\mathcal{F}\hat{P}\hat{e}^{k-1}_{|\Gamma} \tag{3.69}$$

We showed that the ARAS iterative process has an error transfer operator, $\bar{Q}_\mathcal{F}\hat{P}$, equal to the part of the error transfer operator $\hat{P}$ that we did not compute. We note that $||\bar{Q}_\lambda\hat{P}|| \leq ||\hat{P}|| < 1$.

We can apply Proposition 3.3.2 and write:

$$\rho(T_{ARAS(q)}) = \rho(\bar{Q}_\mathcal{F}\hat{P}) \tag{3.70}$$

The conclusion becomes the same as applying Theorem 3.3.3.

We pointed out here the way the approximation of the error transfer operator affects the convergence of Schwarz iterative processes in the case of a separable operator for a two domain decomposition. This enables us to understand the philosophy of approximating the matrix P in different spaces and links the works done in (Garbey 2005, Baranger *et al.* 2008, Garbey & Tromeur-Dervout 2002, Frullone & Tromeur-Dervout 2006).

3.4 Notable properties

This section focuses on some remarks which can be pointed out due to the preceding propositions and remarks. For example, the idea of multilevel preconditioning of ARAS coming from the fact that T_{ARAS2} is nilpotent of degree 1 authorised us to think about other combination when ARAS(q) is considered. The other issue is the way it is possible to approximate $(I - P)^{-1}$. When the matrix to invert is sparse or small the LU factorization can be used, but many possibilities can be considered.

3.4.1 ARAS2 recursivity

The convergence study showed that the enhancement of a linear process such RAS by Aitken leads to a linear process ARAS. Then we pointed out that T_{ARAS2} is nilpotent of degree 1 given that $M^{-1}_{ARAS2(q)}$ can be a good approximation of A^{-1}. But the quality of this approximation depends on how rich is the base we choose to describe the solution on the artificial interface Γ. In most cases the preconditioner $M^{-1}_{ARAS2(q)}$ should not lead to have $T_{ARAS2(q)}$ nilpotent. The ARAS2(q) iterative process is a new linear process. Considering the linear convergence of the ARAS2(q) iterative process we assume that it is possible to apply the Aitken acceleration to the ARAS2(q) iterative method. It offers the possibility to choose small base to describe the solution's interface and compute a new one which can be small too, to perform the acceleration.

3.4.2 Approximation of $(I - P)^{-1}$

After computing P in a chosen base the last numerical issue is the application of $(I - P)^{-1}$ to a vector $x \in \mathbb{R}^{n\times n}$. P should be dense if the operator is non separable or if the operator is non-elliptic. The choice of a solver will depend on the size of P in its approximated form and will depend of the memory available on the computer. Usually it will be possible to compute a LU factorization of P and compute a triangular solution. If it is not possible for memory reasons, we may consider the Krylov methods, but it seems to be complicated to find a good preconditioner for the system involving P. Otherwise, one can remark that if $||P|| < 1$ then we can write the following serie's development:

$$(I - P)^{-1} = \sum_{k=0}^{+\infty} P^k \tag{3.71}$$

Then it is possible to compute the $y = (I - P)^{-1}x$ following Algorithm 10.

Algorithm 10 Computing $y = (I - P)^{-1}x$ with serie's development

Require: $P \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$
1: **if** $||P|| < 1$ **then**
2: Initialise $k = 0$, $y^0 = x$, $y^1 = x + Py^0$
3: **while** $||y^{k+1} - y^k|| > \varepsilon$ **do**
4: $y^k = x + Py^{k-1}$
5: $k = k + 1$

Remark 7 There exists, at least, a norm $||.||$ and $\varepsilon > 0$ such as,

$$\rho(P) \leq ||P|| \leq \rho(P) + \varepsilon \tag{3.72}$$

Then we can approach numerically $||P||$ by the largest eigenvalue of P:

$$||P|| \simeq |\lambda(P)|_{max} \tag{3.73}$$

3.5 Numerical tests

In this section, we provide numerical validation of the ARAS method in terms of numerical efficiency. The general scheme of this section is such that we first describe a simple test case and then solve the linear system proposed with a Matlab code. For the first case we validate the method on a simple $1D$ problem. We study the convergence of an additive Schwarz domain decomposition and analytically build the matrix P. In the second case we recall the equations studied in section 3.3.4 and write the acceleration in the Fourier base. We numerically check the application of Theorem 3.3.3 and moreover, we compute the acceleration in the space arising from the SVD decomposition of the Schwarz solution on the interface. Eventually, we perform some tests on a $2D$ Helmholtz problem with different partitioners and different bases.
Part of those observations was presented in (Dufaud & Tromeur-Dervout 2010b, Dufaud & Tromeur-Dervout 2011).

3.5.1 $1D$ theoretical study on Poisson equation

The goal of this section is to validate the ARAS method on a simple case where the ARAS preconditioner can be written analytically.

3.5.1.1 $1D$ theoretical framework

We consider the $1D$ Poisson equation discretized by second order finite differences with m points the x direction. The domain Ω is split into N sub-domains Ω_i, $i \in [\![0, N-1]\!]$, with an overlap of R points. Each sub-domain Ω_i has n_i points including non-local dependencies. Domains with only local dependencies are denoted by D_i. Each D_i has $n_i - 2$ points.
A domain split into N sub-domains gives $N-1$ groups of 2 interfaces denoted by $I_p = \Gamma_{l_p}, \Gamma_{r_p}$, $p \in [\![1, N-1]\!]$.

For each iteration k of an Additive Schwarz method, the local problems i have a general form as follows:

$$\forall i \in [\![0, N-1]\!], \begin{cases} u_i^{j-1(k)} - 2u_i^{j(k)} + u_i^{j+1(k)} = hb_i^j \ , \ j \in [\![1, n_i - 1]\!] \\ \text{if } i \neq 0 \ , \ u_i^{0(k)} = {u_{i-1}}^{(k-1)}(\Gamma_{l_i}) = u_{i-1}^{n_{i-1}-(R+1)^{(k-1)}} \\ \text{if } i = 0 \ , \ u_i^{0(k)} = u(\Gamma_L) \\ \text{if } i \neq N-1 \ , \ u_i^{n_i(k)} = {u_{i+1}}^{k-1}(\Gamma_{r_{i+1}}) = u_{i+1}^{R+1^{(k-1)}} \\ \text{if } i = N-1 \ , \ u_i^{n_i(k)} = u(\Gamma_R) \end{cases}$$

We would like to write the relation between two consecutive errors of the AS method on the interface. First we write the error system on each sub-domain. Then we find the expression of the Schwarz error on each point of the discretization. Finally, we write the Aitken's formula.

The difference between 2 consecutive iterations of (AS) gives the local equations on the error:

$$\forall i \in [\![0, N-1]\!], \begin{cases} e_i^{j-1(k)} - 2e_i^{j(k)} + e_i^{j+1(k)} = 0 \ , \ j \in [\![1, n_i - 1]\!] \\ \qquad \text{if } i \neq 0 \ , \ e_i^{0(k)} = e_{i-1}^{(k-1)}(\Gamma_{l_i}) = e_{i-1}^{n_{i-1}-(R+1)^{(k-1)}} \\ \qquad \text{if } i = 0 \ , \ e_i^{0(k)} = 0 \\ \text{if } i \neq N-1 \ , \ e_i^{n_i(k)} = e_{i+1}^{k-1}(\Gamma_{r_{i+1}}) = e_{i+1}^{R+1^{(k-1)}} \\ \text{if } i = N-1 \ , \ e_i^{n_i(k)} = 0 \end{cases}$$

The equation $e_i^{j-1(k)} - 2e_i^{j(k)} + e_i^{j+1(k)} = 0$ leads to an expression of the error $e_i^{j(k)}$ such as,

$$e_i^{j(k)} = a * j + b \ , \ j \in [\![1, n_i - 1]\!] \ , \ a, b \in \mathbb{R} \tag{3.74}$$

Let us find a and b.

- For $j = 1$, we write : $e_i^{0(k)} - 2e_i^{1(k)} + e_i^{2(k)} = 0$
  hence, if $i \neq 0$,

$$\begin{aligned} e_{i-1}^{(k-1)}(\Gamma_{l_i}) &= 2e_i^{1(k)} - e_i^{2(k)} \\ &= 2 * (a + b) - (2 * a + b) \\ &= b \end{aligned}$$

  else if $i = 0$, $b = 0$

- For $j = n_i - 1$, we write : $e_i^{n_i-2(k)} - 2e_i^{n_i-1(k)} + e_i^{n_i(k)} = 0$
  Then,

$$\begin{aligned} e_{i+1}^{(k-1)}(\Gamma_{r_{i+1}}) &= 2e_i^{n_i-1(k)} - e_i^{n_i-2(k)} \\ &= 2 * ((n_i - 1) * a + b) - (n_i - 2) * a - b \\ &= n_i * a + b \end{aligned}$$

  if $i \neq N - 1$,

$$a = \frac{e_{i+1}^{(k-1)}(\Gamma_{r_{i+1}}) - e_{i-1}^{(k-1)}(\Gamma_{l_i})}{n_i}$$

  else if $i = N - 1$, $a = \frac{-e_{i-1}^{(k-1)}(\Gamma_{l_i})}{n_i}$

The Schwarz error on each sub-domain D_i can be written as follows:

$$\forall j \in [\![1, n_i{-}1]\!], \begin{cases} i \in [\![1, N-2]\!] \ , \ e_i^{j(k)} = \dfrac{e_{i+1}^{(k-1)}(\Gamma_{r_{i+1}}) - e_{i-1}^{(k-1)}(\Gamma_{l_i})}{n_i} * j + e_{i-1}^{(k-1)}(\Gamma_{l_i}) \\ i = 0 \ , \ e_0^{j(k)} = \dfrac{e_1^{(k-1)}(\Gamma_{r_1})}{n_0} * j \\ i = N-1 \ , \ e_{N-1}^{j\ (k)} = \dfrac{-e_{N-2}^{(k-1)}(\Gamma_{l_{N-1}})}{n_{N-1}} * j + e_{N-2}^{(k-1)}(\Gamma_{l_{N-1}}) \end{cases} \tag{3.75}$$

The error at the iteration k writes,

$$e_{|\Gamma}^{(k)} = u_{|\Gamma}^{(k)} - u_{|\Gamma}^{\infty}$$

We would like to find $P \in \mathbb{R}^{nbinterf \times nbinterf}$ such as $e_{|\Gamma}^{k} = P e_{|\Gamma}^{k-1}$.
The previous results on error gives the coefficient of the matrix P.

Let us build P on the decomposition given in 3.3. In this example, $N = 4$ and $R = 2$. The number of interfaces is $nbinterf = 6$. Note that,

$$e_{|\Gamma} = (\quad e_0(\Gamma_{l_1}) \quad e_1(\Gamma_{r_1}) \quad e_1(\Gamma_{l_2}) \quad e_2(\Gamma_{r_2}) \quad e_2(\Gamma_{l_3}) \quad e_3(\Gamma_{r_3}) \quad)^T$$

For each coefficient we apply the equation (3.75) such as,

- $e_{|\Gamma}(1)$

$$\begin{aligned} e_0^{(k)}(\Gamma_{l_1}) &= e_0^{n_0-(\delta+1)^{(k)}} \\ &= \frac{n_0 - (\delta+1)}{n_0} e_1^{(k-1)}(\Gamma_{r_1}) \end{aligned}$$

- $e_{|\Gamma}(2)$

$$\begin{aligned} e_1^{(k)}(\Gamma_{r_1}) &= e_1^{\delta+1^{(k)}} \\ &= \frac{n_1 - (\delta+1)}{n_1} e_0^{(k-1)}(\Gamma_{l_1}) + \frac{\delta+1}{n_1} e_2^{(k-1)}(\Gamma_{r_2}) \end{aligned}$$

- $e_{|\Gamma}(3)$

$$\begin{aligned} e_1^{(k)}(\Gamma_{l_2}) &= e_1^{n_1-(\delta+1)^{(k)}} \\ &= \frac{n_1 - (\delta+1)}{n_1} e_2^{(k-1)}(\Gamma_{r_2}) + \frac{\delta+1}{n_1} e_0^{(k-1)}(\Gamma_{l_1}) \end{aligned}$$

- $e_{|\Gamma}(4)$

$$\begin{aligned} e_2^{(k)}(\Gamma_{r_2}) &= e_2^{\delta+1^{(k)}} \\ &= \frac{n_2 - (\delta+1)}{n_2} e_1^{(k-1)}(\Gamma_{l_2}) + \frac{\delta+1}{n_2} e_3^{(k-1)}(\Gamma_{r_3}) \end{aligned}$$

- $e_{|\Gamma}(5)$

$$\begin{aligned} e_2^{(k)}(\Gamma_{l_3}) &= e_2^{n_2-(\delta+1)^{(k)}} \\ &= \frac{\delta+1}{n_2} e_1^{(k-1)}(\Gamma_{l_2}) + \frac{n_2-(\delta+1)}{n_2} e_3^{(k-1)}(\Gamma_{r_3}) \end{aligned}$$

- $e_{|\Gamma}(6)$

$$\begin{aligned} e_3^{(k)}(\Gamma_{l_3}) &= e_3^{\delta+1^{(k)}} \\ &= \frac{n_3-(\delta+1)}{n_3} e_2^{(k-1)}(\Gamma_{l_3}) \end{aligned}$$

Hence the Aitken's formula is :

$$\begin{pmatrix} e_0(\Gamma_{l_1}) \\ e_1(\Gamma_{r_1}) \\ e_1(\Gamma_{l_2}) \\ e_2(\Gamma_{r_2}) \\ e_2(\Gamma_{l_3}) \\ e_3(\Gamma_{r_3}) \end{pmatrix}^{(k)} = \underbrace{\begin{pmatrix} 0 & \frac{n_0-(\delta+1)}{n_0} & 0 & 0 & 0 & 0 \\ \frac{n_1-(\delta+1)}{n_1} & 0 & 0 & \frac{\delta+1}{n_1} & 0 & 0 \\ \frac{\delta+1}{n_1} & 0 & 0 & \frac{n_1-(\delta+1)}{n_1} & 0 & 0 \\ 0 & 0 & \frac{n_2-(\delta+1)}{n_2} & 0 & 0 & \frac{\delta+1}{n_2} \\ 0 & 0 & \frac{\delta+1}{n_2} & 0 & 0 & \frac{n_2-(\delta+1)}{n_2} \\ 0 & 0 & 0 & 0 & \frac{n_3-(\delta+1)}{n_3} & 0 \end{pmatrix}}_{P} \begin{pmatrix} e_0(\Gamma_{l_1} \\ e_1(\Gamma_{r_1}) \\ e_1(\Gamma_{l_2}) \\ e_2(\Gamma_{r_2}) \\ e_2(\Gamma_{l_3}) \\ e_3(\Gamma_{r_3}) \end{pmatrix}^{(k-1)} \quad (3.76)$$

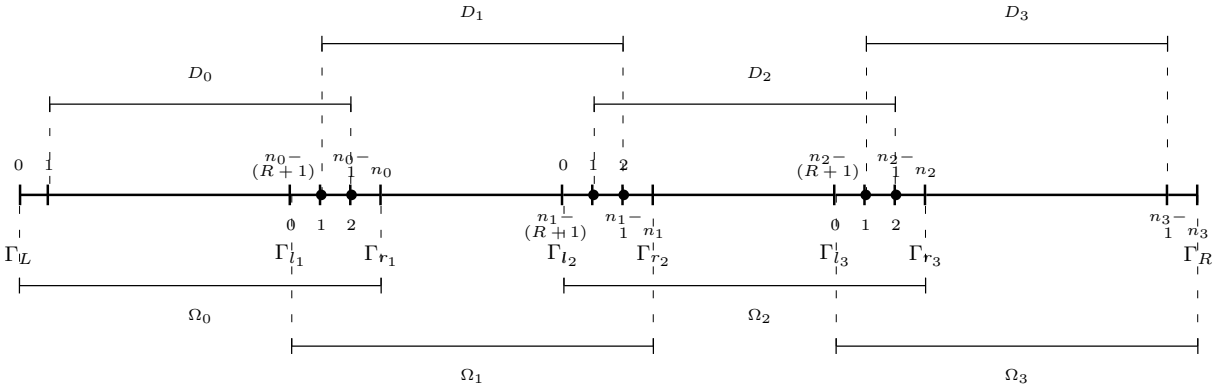

Figure 3.3: Domain decomposition of the 1D Poisson problem with 4 subdomains and an overlap $\delta = 2$

3.5.1.2 Property checking

We decompose the $1D$ domain into 4 domains. We set up the domain grid with 502 points per domain including the boundary points. The overlap between domains is equal to 2.

We build the matrix P applying formula 3.76. Hence the Aitken's formula is :

$$P = \begin{pmatrix} 0 & 0.9940 & 0 & 0 & 0 & 0 \\ 0.9940 & 0 & 0 & 0.0060 & 0 & 0 \\ 0.0060 & 0 & 0 & 0.9940 & 0 & 0 \\ 0 & 0 & 0.9940 & 0 & 0 & 0.0060 \\ 0 & 0 & 0.0060 & 0 & 0 & 0.9940 \\ 0 & 0 & 0 & 0 & 0.9940 & 0 \end{pmatrix} \tag{3.77}$$

The eigenvalues of P are $\lambda \in \text{sp} = \{0.9982,\ 0.9940,\ 0.9898,\ -0.9982,\ -0.9898,\ -0.9940\}$. The spectral radius of P is $\rho(P) = 0.9982$. Then we compute the RAS, ARAS and ARAS2 preconditioners and compute for each preconditioner of type $*$ the spectral radius of T_* and the conditioning number in the 2-norm of $M_*^{-1}A$. Figure 3.4 shows the structure of the preconditioner RAS and the preconditioner ARAS. We remark that the enhancement by Aitken results in the appearance of lines in the preconditioner linking the domain and acting on the interface. We solve the problem with the Richardson iterative process as solver and as preconditioner of a GCR for a tolerance $\varepsilon = 1e-9$. Those results are presented in Table 3.1.

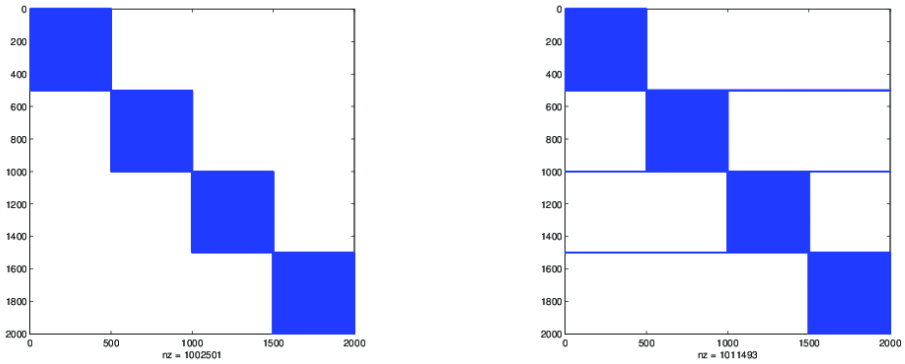

Figure 3.4: Struture of preconditioners (left) M_{RAS}^{-1}, (right) M_{ARAS}^{-1}

prec. $*$	$\rho(T_*)$	$\kappa(M_*^{-1}A)$	It. Rich.	It. GCR
RAS	0.9982	1.3579 e+05	-	6
ARAS	3.5679 e-05	2.5050 e+02	2	2
ARAS2	1.2847 e-09	1.0000	1	1

Table 3.1: Numerical performance of RAS ARAS and ARAS2 on the $1D$ Poisson problem.

We first observe that $\rho(T_{RAS}) = \rho(P)$ as Proposition 3.3.2 suggested. The convergence of the RAS iterative is too slow and it is not possible to compute

the solution in a reasonable time. Nevertheless, the RAS preconditioner reduces the conditioning number of the system by two orders and the GCR converges quite well. The ARAS and ARAS2 preconditioners have been estimated by analytic analysis. Then we refer to Proposition 3.3.1 and should have a spectral radius of T_{ARAS} and T_{ARAS2} equal to 0. In Table 3.1 the results are, at least, five orders smaller than the smallest eigenvalue of P. With those estimations, the ARAS2 preconditioner leads to $\kappa(M_*^{-1}A) = 1.0000$ for $\rho(T_{ARAS2}) = 1.2847e-09$. Then we assume that the spectral radius computed for T_{ARAS} and T_{ARAS2} can be considered close to 0 and assume that T_{ARAS} is nilpotent. Although it is not 0 for numerical reasons, we observe two important points:

- $\rho(T_{ARAS2}) < \rho(T_{ARAS})$
- As T_{ARAS} is nilpotent of degree 2, the iterative processes of ARAS and ARAS2 converge respectively into 2 and 1 iterations for a tolerance $\varepsilon = 1e-9$ equal to $\rho(ARAS2)$.

3.5.2 $2D$ theoretical study

The goal of this section is to validate the ARAS method on a simple case where the ARAS preconditioner can be written analytically and where we can apply Theorem 3.3.3. We consider the $2D$ problem decomposed in 2 domains presented in subsection 3.3.4. The grid size is about 32×32. This subsection provides the theoretical framework we implement in Matlab. Here we point out some technical aspect that it is important to consider in order to perform the acceleration in the Fourier base and then we verify the results given previously.

3.5.2.1 Technical aspect of writing the acceleration in a Fourier base

We build $\hat{P}$ as it is written in equation (3.59). We set up the transfer matrices C_{Γ_i} and D_{Γ_i}.

The most important numerical problem in the Fourier base transfer are the Gibbs phenomena. In order to reduce those effects, one idea is to symmetrize the signal on the interface and perform a Fourier transform over $[\![0, 2\pi]\!]$. Let us consider $n_i + 1 = 32$ points on Γ_i including the boundary points. The application of the acceleration is performed on the internal points. The transfer matrices are defined following Algorithm 11.

Algorithm 11 Computing a symmetrized FFT using fft and iift of Matlab (FFTW)

1: Compute $C_{sym,\Gamma_i} = fft(I_{2*n-2})$ and $D_{sym,\Gamma_i} = ifft(I_{2*n-2})$
2: Extract the part of each matrix transfer which has an effect on the true interface non symmetrized in $C_{\Gamma_i} \in \mathbb{C}^{n_i+1\times 2(n_i+1)-2}$ and $D_{\Gamma_i} \in \mathbb{C}^{n_i+1\times n_i+1}$such as, $C_{\Gamma_i} = C_{sym,\Gamma_i}(1:(n_i+1), 1:(2(n_i+1)-2))$ and $D_{\Gamma_i} = 2.0 * D_{sym,\Gamma_i}(1:(n_i+1), 1:(n_i+1))$
3: Build $D \in \mathbb{C}^{2(n+1)\times 2(n+1)}$ and $C \in \mathbb{C}^{2(n+1)\times 2(2(n_i+1)-2)}$ such as in(3.64)

3.5.2.2 Property checking

We build the matrix $\hat{P} \in \mathbb{C}^{30\times 30}$. Only the internal points are taken, leading to 30 modes which can be accelerated. Those modes decrease from 0.8106 to 0.1531. We decide to compute the entire $\hat{P}$ and a truncated one of size $q = 15$, $Q_{\mathcal{F}}\hat{P}$. Figure 3.5 shows the coefficient of the matrix computed. The goal here is to retrieve the convergence rate given by the application of theorem 3.3.3 in equation (3.61). The convergence rate of a ARAS(q) type preconditioner is related to the coefficients of $\hat{P}$ denoted by δ_l.

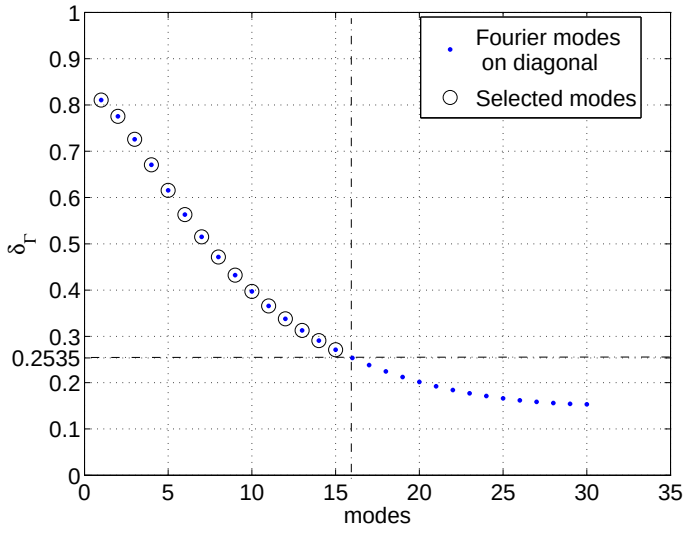

Figure 3.5: Diagonal coefficients of $\hat{P}$ and $Q_{\mathcal{F}}\hat{P}$ corresponding to the modes to accelerate.

For $q = 15$ we note that,

$$\delta_1 = 0.8106 \tag{3.78}$$
$$\delta_{q+1} = 0.2535 \tag{3.79}$$
$$\delta_{q+1}^2 = 0.0643 \tag{3.80}$$

Then we compute the RAS, ARAS(q) and ARAS2(q) preconditioners and compute for each preconditioner of type $*$ the spectral radius of T_* and the conditioning number in the 2 norm of $M_*^{-1}A$. The convergence for each appropriate stopping criteria is $1.0e-10$.

Looking at Table 3.2 concerning the convergence rate of RAS, ARAS(15) and ARAS2(15), the study done in subsection 3.3.4 is verified.

This numerical test shows that even if the conditioning numbers for ARAS(15) and ARAS(30) are equal, the convergence rate of both the method are different. This can be explained by the fact that for $q = 30$ all the modes are selected. The

prec. $*$	$\rho(T_*)$	$\kappa(M_*^{-1}A)$	It. Rich.	It. GCR
RAS	0.8106	30.0083	96	18
ARAS(q=15)	0.2535	5.2358	14	7
ARAS2(q=15)	0.0643	1.1451	7	5
ARAS(q=30)	3.6980 e-07	5.2358	2	2
ARAS2(q=30)	1.4319 e-13	1.0000	1	1

Table 3.2: Numerical performance of RAS, ARAS and ARAS2 on the $2D$ Poisson problem.

theory tells that if P is fully computed then T_{ARAS} is nilpotent of degree 2 and its spectral radius is equal to 0. Hence the quality of the acceleration here has to be measured with both the conditioning number of M_*^{-1} and the spectral radius of T_*.

Moreover the number of iterations of the iterative process ARAS(q=15) is twice the number of iterations of the iterative process ARAS2(q=15).

Finally, for $q = 30$ the same behaviours as for the $1D$ case of the preconditioner are observed.

3.5.3 Observing the influence of the partitioning and the approximation space on a $2D$ Helmholtz problem

Here, we focus on the influence of the partitioning chosen to set up the domain decomposition method. We also focus on the influence of the choice of a base to approximate the Aitken acceleration. When the mesh is known it is possible to partition the operator following a geometric partitioning. One point is to see what can happen if we partition the operator with a graph partitioning approach such as METIS. Another point is to see how the choice of a base influence the performance and the cost of the ARAS type preconditioner.

Let us consider the 2D Helmholtz problem $(-\omega - \triangle)u = f$ in $\Omega = [0,1]^2$, $u = 0$ on $\partial\Omega$. The problem is discretized by second order finite differences with m points in each direction x and y giving a space step $h = \frac{1}{m-1}$. The set value $\omega = 0.98\frac{4}{h^2}(1 - cos(\pi h))$ is close to the minimum eigenvalue of the discrete $-\triangle$ operator in order to have an ill-conditioned discrete problem with $\kappa_\infty(A) = 1.7918\,E + 07$ for $m = 164$.

First we solve the problem with a physical band partitioning, and then we solve it with a METIS partitioning using eight sub-domains. Figure 3.6 illustrates the physical band and the Metis partitioning using eight sub-domains. In the physical partitioning, borders are smooth, contrary to the METIS partitioning which creates corners and irregular borders. The corners give cross points which deteriorate the convergence of the Schwarz method.

For each partitioning, we build the ARAS2 preconditioner in two different bases:

- an orthogonal base arising from the application of the preconditioner RAS on a sequence of random vectors (see subsection 3.1.3).

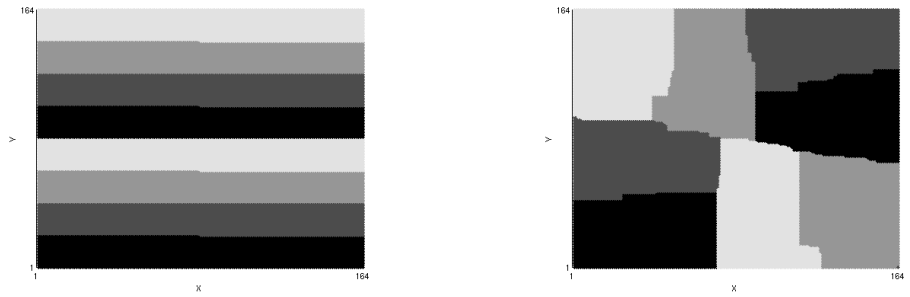

Figure 3.6: Partitioning into 8 parts on a 2D Helmholtz problem of size 164×164, (left) Physical Band partitioning, (right) METIS partitioning.

- a base built from the successive Schwarz solution on the interface and passed in its SVD base (see subsection 3.1.4).

Remark 8 In the following, we denote by ARAS2($r=\frac{n}{q}$) the preconditioner approximated in the "random" base. Because the number of vectors can be high for this kind of base, we choose to express the reduction number r in parentheses instead of q, the number of column of $\mathbb{U}_q$, but the formula is still:

$$M^{-1}_{ARAS(r=\frac{n}{q}),\delta} = \left(I_m + R_\Gamma^T \mathbb{U}_q \left(\left(I_q - \tilde{P}_{\mathbb{U}_q} \right)^{-1} - I_q \right) \mathbb{U}_q^{-1} R_\Gamma \right) \sum_{i=1}^{p} \tilde{R}_{i,\delta}^T A_{i,\delta}^{-1} R_{i,\delta}$$

Figure 3.7 presents the Richardson with ARAS2 and the ARAS2 preconditioned GCR Krylov method for the physical band partitioning using a random base or a SVD base while Figure 3.8 uses the METIS partitioning. These results were obtained with a sequential Matlab code able to run small academic problems. The Krylov method used is the gradient conjugate residual GCR while the LU factorisation is used to solve the sub-domain's problems. These results exhibit:

- Richardson processes in Figure 3.7 converge while in Figure 3.8 only the RAS iterative process converges and the ARAS2 process diverges. Consequently the physical partitioning enables good convergence of the RAS which can be accelerated with the approximation of P by $P_{\mathbb{U}_q}$. The METIS partitioning slows the convergence of the RAS due to the cross points.Then, using a domain decomposition method as a solver with an algebraic partitioning can produce bad results when it is accelerated by Aitken. Let us notice that the full P makes the RAS process converge in one iteration.

- Nevertheless, the Aitken-RAS used as a preconditioner is very efficient even on the METIS partitioning with cross points where the Aitken-RAS as a Richardson process diverges. This makes the Aitken-RAS a robust algebraic precon-

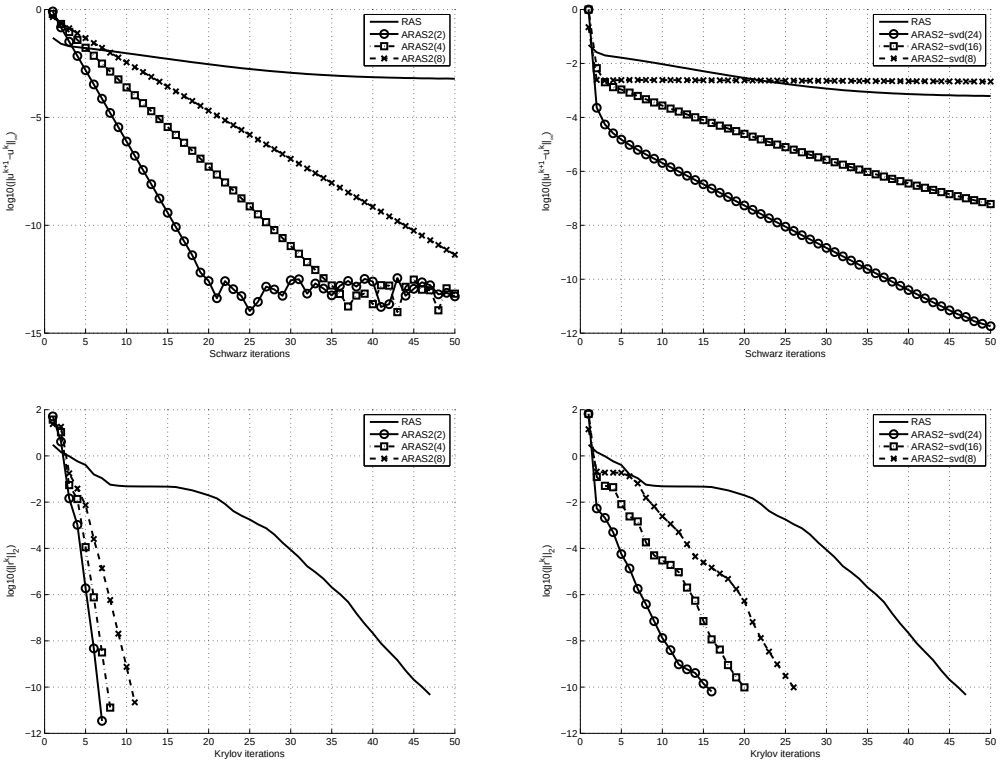

Figure 3.7: Solving 2D Helmholtz equation on a 164×164 Cartesian grid, physical band partitioning, $p = 8$,(left) ARAS2(r $= \frac{n}{q}$) is built with a Random base, (right) ARAS2(q) is built with a SVD base, (top) Convergence of Iterative Schwarz Process, (bottom) convergence of GCR method preconditioned by RAS and ARAS2.

ditioner. We must notice that the effect of the preconditioning with a METIS partitioning is less efficient than the one with the physical partitioning.

- The better the base $\mathbb{U}_q$ is able to represent the interface solution, the better the preconditioner is for the random base and the SVD base.

Let us observe the difference between the two choices of base to compute the acceleration. On the one hand, the choice of an orthogonal base arising from the application of the RAS preconditioner on a sequence of random vectors presents good advantages for a preconditioner. With this approach, the preconditioner can be used for different right-hand sides. However, the number of vectors necessary to describe the interface can be close to the size of the global interface, increasing the cost of the preconditioner. On the other hand, it is possible to build the acceleration for many iterations of the Additive Schwarz process, computing the SVD of the interface solutions. Then the acceleration process is problem-dependent, but experience shows that a small number of iterations can enable a good approximation of $P_{\mathbb{U}_q}$. For a

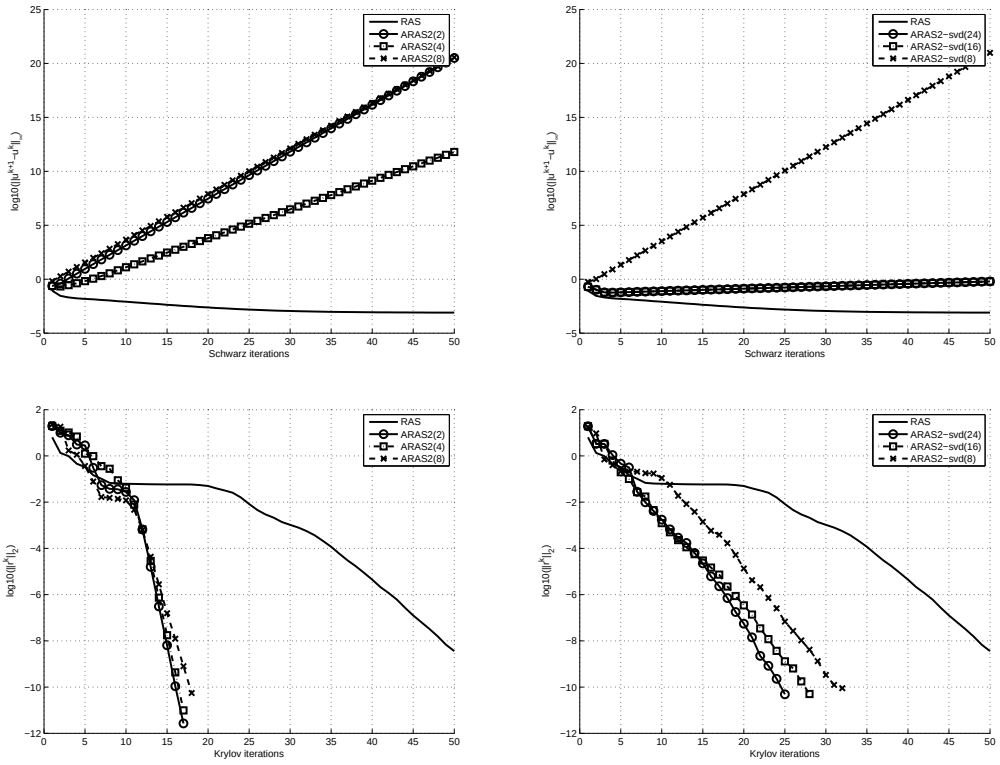

Figure 3.8: Solving 2D Helmholtz equation on a 164×164 Cartesian grid, METIS partitioning, $p = 8$,(left) ARAS2 is built with a Random base, (right) ARAS2 is built with a SVD base, (top) Convergence of Iterative Schwarz Process, (bottom) convergence of GCR method preconditioned by RAS and ARAS2.

physical partitioning, we can evaluate the cost of each preconditioner. For each sub-domain, the artificial interface is of size 164 for the uppermost or lowermost sub-domain, and $164 * 2 = 328$ for internal sub-domains. Hence, for $p = 8$ the global interface has a size of $164 * 9 = 1476$. For $r = 1$ the base is complete and $P_{\mathbb{U}_q}$ is exact. There is no need to use ARAS as a preconditioning technique. For all r the size of Γ is $\frac{1476}{r}$. Then the number of $M_{RAS}^{-1}x = y$ products to build M_{ARAS}^{-1} is $3 * \frac{1476}{r}$. While the number of $M_{RAS}^{-1}x = y$ products to build M_{ARAS}^{-1} for the base arising from SVD only depends on the number of Richardson iterations. On the left of Figure 3.7, we observe that for $r = 8$, $n = 185$ and the number of products $M_{RAS}^{-1}x = y$ is 555, the convergence of GCR is reached in 11 iterations. For 24 iterations of Schwarz, we build a matrix $P_{\mathbb{U}_q}$ of size 24, which is around eight times smaller than with the previous base. The number of matrix products is 48, 12 times smaller than with the previous base. The number of GCR iterations is 15. Eventually, the cost of a good independent preconditioner is excessive compared to the one with the SVD.

Now, we focus on the base arising from a SVD. We compute all the singular

values corresponding to the number of interface points and select 24 singular values from this set of n values. For 8 partitions with a manual partitioning, $n = 2296$ and with a METIS partitioning we obtain 1295 interfaces points. We saw that the Aitken-RAS technique used as a Richardson iterative process diverges for the METIS partitioning. We study the spectrum of a preconditioner in the two cases and compare it to the spectrum of the RAS preconditioning method. For convenience we consider only a set of the 40 largest eigenvalues.

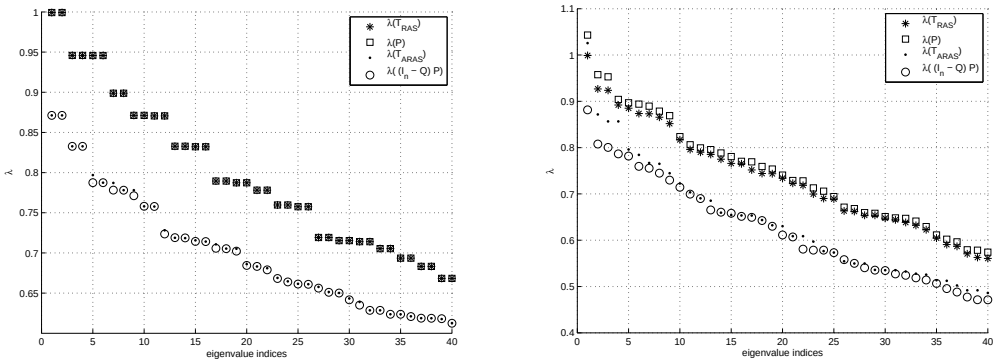

Figure 3.9: Eigenvalues of $T_{ARAS}(q = 24)$ compared to eigenvalues of P_{U_q} for a 164×164 Cartesian grid, $p = 8$,(left) Band partitioning, ARAS is built with a SVD base computed with 24 singular vectors chosen over 2296, (right) METIS partitioning, ARAS is built with a SVD base computed with 24 singular vectors chosen over 1295.

The predicted values of the spectrum of the error transfer operator on the interface gives a good approximation of the spectrum of the iterative process in the case of the band partitioning. Otherwise, for the METIS partitioning, it appears that the spectrum gives a good idea of the spectrum of the iterative method but differs for the first Eigenvalues. Then it appears that the first eigenvalue of T_{ARAS} is greather than 1 instead of the value computed for $\bar{Q}P$, which is less than 1. It should explain why the iterative process diverges.

This empirical analysis shows the influence of a partitioning technique on the Schwarz preconditioner. It is important to know that if the user has the entire knowledge of the linear system he solves then he should provide a physical partitioning which can have smooth boundaries. Otherwise, if the sub-problems are non-singular then the Schwarz method used as a preconditioning technique is efficient but can present a lack of speed in the convergence. The second point is the choice of a good base. The two bases are efficient, but the one arising from the SVD presents the best choice for time computing considerations.

3.6 Computing cost modelling

We want to evaluate the cost of building and applying an ARAS type preconditioner in terms of arithmetic complexity. We denote by $\mathcal{AC}(*)$ the arithmetic complexity of an operation $*$. Let considers a matrix $A \in \mathbb{R}^{m\times m}$. This matrix is decomposed into p sub-matrices $A_i \in \mathbb{R}^{m_i\times m_i}$. The decomposition leads to have an interface Γ of size n. The coarse interface is of size q. We denote by $x_\alpha \in \mathbb{R}^\alpha$ where $\alpha \in \mathbb{N}$ should be any of m, m_i, n or q.

3.6.1 Arithmetic complexity of applying an ARAS type preconditioner

Let considers the operation:

$$M_{RAS}^{-1}x_m = \sum_{i=1}^{p} \tilde{R}_i^T A_i^{-1} x_{m_i} = y$$

The cost of such an operation mostly consists in the p operations $A_i^{-1}x_{m_i}$ which depends on the cost of a chosen method to inverse A_i such as a Krylov methods or a LU factorization or maybe an incomplete LU factorization.

Then the cost of applying a RAS preconditioner is written as:

$$\mathcal{AC}\left(M_{RAS}^{-1}x_m\right) = p \times \mathcal{AC}\left(A_i^{-1}x_{m_i}\right) \tag{3.81}$$

We know consider the operation:

$$M_{ARAS(q)}^{-1}x_m = \left(I_m + R_\Gamma^T \mathbb{U}_q \left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} - I_q\right) \mathbb{U}_q^T R_\Gamma\right) \sum_{i=1}^{p} \tilde{R}_i^T A_i^{-1} R_i x_m$$

The cost of such an operation consists in one application of a RAS preconditioner, the base transfer operated by $\mathbb{U}_{||}$ and solving $\left(I_q - P_{\mathbb{U}_q}\right) y_q = x_q$. Others or summing operations: 1 sum between 2 vectors of size n and, one subtract between two vector of size q.

Then the cost of applying a RAS preconditioner is written as:

$$\begin{aligned}\mathcal{AC}\left(M_{ARAS}^{-1}x_m\right) &= p \times \mathcal{AC}\left(A_i^{-1}x_{m_i}\right) + \mathcal{AC}\left(\mathbb{U}_q^T x_m\right) \\ &\quad + \mathcal{AC}\left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} x_q\right) + \mathcal{AC}\left(\mathbb{U}_q x_q\right) \\ &\quad + \mathcal{AC}\left(x_q - x_q\right) + \mathcal{AC}\left(x_n + x_n\right)\end{aligned}$$

We note that:

- An addition between 2 vectors of size n consists in n operations.
- A subtraction between 2 vectors of size n consists in n operations.
- A scalar product between 2 vectors of size n consists in n product and n sum.

- A multiplication between a matrix with n lines and m columns and a vector with m lines consists in n scalar products of vectors of size m.

Then we write,

$$\begin{aligned}
\mathcal{AC}\left(x_n + x_n\right) &= n \\
\mathcal{AC}\left(x_q - x_q\right) &= q \\
\mathcal{AC}\left(\mathbb{U}_q x_q\right) &= m \times 2 \times q \\
\mathcal{AC}\left(\mathbb{U}_q^T x_m\right) &= q \times 2 \times m
\end{aligned}$$

And,

$$\begin{aligned}
\mathcal{AC}\left(M_{ARAS}^{-1} x_m\right) &= p \times \mathcal{AC}\left(A_i^{-1} x_{m_i}\right) && (3.82)\\
&\quad + \mathcal{AC}\left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} x_q\right) && (3.83)\\
&\quad + 4 \times q \times m + n + q && (3.84)
\end{aligned}$$

Remark 9 In most cases the coarsening is such that $q \ll m$. Then the cost $\mathcal{AC}\left(\left(I_q - P_{\mathbb{U}_q}\right)^{-1} x_q\right)$ should be very small compared to the cost $p \times \mathcal{AC}\left(A_i^{-1} x_{m_i}\right)$. If it is the case

$$\mathcal{AC}\left(M_{ARAS(q)}^{-1} x_m\right) = \mathcal{AC}\left(M_{RAS}^{-1} x_m\right) + \mathcal{O}(m) \tag{3.85}$$

This means that the cost of one application of the ARAS preconditioner is close to the cost of one application of the RAS preconditioner when $q \ll m$.

We now estimate the cost of applying an ARAS2 preconditioner:

$$M_{ARAS2(q)}^{-1} x_m = 2M_{ARAS(q)}^{-1} - M_{ARAS(q)}^{-1} A M_{ARAS(q)}^{-1} x_m \tag{3.86}$$

It consists in 2 applications of ARAS(q) and one matrix vector product on the entire domain. We note that the matrix A is sparse. So, denoting by $nnz(A)$ the number of non zeros elements of A we write,

$$\mathcal{AC}(A x_m) = 2 \times nnz(A)$$

Hence,

$$\mathcal{AC}\left(M_{ARAS2(q)}^{-1} x_m\right) = 2 \times \mathcal{AC}\left(M_{ARAS(q)}^{-1} x_m\right) + 2 \times nnz(A) + 2\mathcal{O}(m) \tag{3.87}$$

Remark 10 For $q \ll m$,

$$\mathcal{AC}\left(M_{ARAS2(q)}^{-1} x_m\right) = 2 \times \mathcal{AC}\left(M_{RAS}^{-1} x_m\right) + \mathcal{O}(nnz(A)) + \mathcal{O}(m)$$

3.6.2 Arithmetic complexity of building the coarse space $\mathbb{U}_q$ and $P_{\mathbb{U}_q}$

We focus here on the cost to build a base arising from the SVD of the Schwarz solutions on the interface. We refer to Algorithm 9 which proposes a robust way to implement the Aitken acceleration without inversion.

We compute $q+2$ iterations of a RAS iterative process. It consists in applying the preconditioner on a vector x_m and summing the result with another vector of the same size. We can write

$$\mathcal{AC}(T_{RAS}x_m) = \mathcal{AC}(M_{RAS}^{-1}x_m) + \mathcal{O}(m) \tag{3.88}$$

Then we perform a SVD over a set of $q+2$ vectors of size n, $X_{q+2} \in \mathbb{R}^{n\times(q+2)}$.

$$\mathcal{AC}(\text{building } \mathbb{U}_q) \leq (q+2) \times \mathcal{AC}(T_{RAS}x_m) + \mathcal{AC}(SVD(X_{q+2})) \tag{3.89}$$

After this, we apply one iteration of the Schwarz iterative process on at most the q first left singular vectors to build $P_{\mathbb{U}_{\text{ll}}}$. Thus,

$$\mathcal{AC}(\text{building } \mathbb{U}_q \text{ and } P_{\mathbb{U}_q}) \leq (q+2)\times\mathcal{AC}(T_{RAS}x_m)+\mathcal{AC}(SVD(X_{q+2})+q\times\mathcal{AC}(M_{RAS}^{-1}x_m) \tag{3.90}$$

Hence,

$$\mathcal{AC}(\text{building } \mathbb{U}_q \text{ and } P_{\mathbb{U}_q}) \leq 2(q+1) \times \mathcal{AC}(M_{RAS}^{-1}x_m) + \mathcal{AC}(SVD(X_{q+2})) + \mathcal{O}(m) \tag{3.91}$$

The cost of building the coarse space and the error transfer operator depends on the number q of vectors needed. Then the ARAS(q) preconditioner will be a good choice compared to RAS if q is sufficiently small compared to the number of application of the preconditioner involved in the Krylov iterative method.

Remark 11 This computation, following the robust algorithm 9, is nearly two times costly than Algorithm 8 with inversion. In fact, the building of $P_{\mathbb{U}_q}$ with Algorithm 8 consists in inverting an error matrix of size q and multiplying it, on the left, by another matrix of size q. For simplicity we consider those operations of order $\mathcal{O}(m)$.

$$\mathcal{AC}(\textit{building } \mathbb{U}_q \textit{ and } P_{\mathbb{U}_q}) \leq (q+2) \times \mathcal{AC}(M_{RAS}^{-1}x_m) + \mathcal{AC}(SVD(X_{q+2})) + \mathcal{O}(m) \tag{3.92}$$

In order to save computing, Algorithm 8 with inversion is the best choice.

3.6.3 Parallelization

It is important to note that the Restricted Additive Schwarz process is fully parallel, in the sense that the inverse of A_i can be computed independently by every single process i handling a domain i. Then for p processes, we can re-write the formula (3.81):

$$\mathcal{AC}\left(M_{RAS}^{-1}x_m\right) = \mathcal{AC}\left(A_i^{-1}x_{m_i}\right) \tag{3.93}$$

The parallelism leads to have a reduction of the matrix vector product. Then we write, for p processes:

$$\begin{aligned} \mathcal{AC}\left(\mathbb{U}_q x_q\right) &= m_i \times 2 \times q \\ \mathcal{AC}\left(\mathbb{U}_q^T x_m\right) &= q \times 2 \times m_i \\ \mathcal{AC}(A x_m) &= 2 \times nnz_i(A) \end{aligned}$$

Chapter 4

Parallel implementation of Aitken-Schwarz method as solver and preconditioner

French summary - Résumé en français

Implémentation parallèle de la méthode d'Aitken-Schwarz utilisée comme méthode de résolution ou de préconditionnement

Ce chapitre est dédié à l'implémentation parallèle des méthodes d'Aitken-Schwarz présentées dans le chapitre 3. On suppose que les méthodes de décomposition de domaine sont en général bien adaptées à l'utilisation de machines de calcul parallèle étant donnée leur disposition naturelle à distribuer un problème en plusieurs sous-problèmes qui peuvent être résolus indépendamment les uns des autres.
Dans une première partie on s'attache à présenter les différents points de la méthode qui peuvent être parallélisés. Une première étude est effectuée sur la méthode de Schwarz. En partant d'un découpage du problème sur une grille cartésienne on propose une distribution parallèle et naturelle du problème sur les processus et on étend cette approche à une distribution parallèle sur deux niveaux de parallélisation MPI. La distribution consiste à définir quels processus, ou groupes de processus, gèrent quelles inconnues du système. Ces inconnues peuvent faire partie du domaine, de la zone de recouvrement, ou de l'interface artificielle. À partir de la distribution choisie, plusieurs stratégies d'échange de données entre les sous-domaines sont envisagées. Cette étude permet d'appréhender une approche plus générale de la parallélisation basée sur une représentation algébrique des sous-problèmes par les sous-matrices. Les recouvrements et les interfaces sont définis par les dépendances de données entre les sous-matrices venant du partitionnement de la matrice globale. Concernant la méthode de Aitken, la parallélisation dépend de la technique d'approximation choisie.

Après avoir établi les concepts de base de parallélisation de la méthode, nous proposons deux implémentations distinctes.

La première est une implémentation de la méthode de résolution de Schwarz multiplicatif accélérée par la technique d'Aitken. Cette implémentation a fait

l'objet des travaux (Dufaud & Tromeur-Dervout 2010a, Berenguer et al. 20YYa, Berenguer et al. 20YYb). Le code, écrit en FORTRAN et MPI, est conçu pour la résolution des équations de Darcy $3D$ dans les milieux poreux fortement hétérogènes sur des machines massivement parallèles, distribuées. La distribution du problème sur les processeurs sur deux réseaux MPI suit le découpage physique du maillage cartésien selon une direction principale. Les sous-problèmes sont résolus par une méthode multigrille algébrique. Les communications s'effectuent processeur à processeur.
La seconde implémentation est celle de ARAS(q) dans l'environnement de la bibliothèque de calcul PETSc. Cette implémentation fait l'objet d'un proceeding *(Dufaud & Tromeur-Dervout 2011) et d'un papier soumis (Dufaud & Tromeur-Dervout 20YYb). Le code a été écrit dans le but de pouvoir traiter un grand nombre de systèmes linéaires creux et plus précisément ceux issus des problèmes de CFD provenant de la différentiation automatique. Ainsi, la distribution du problème est fonction du partitionnement initial de la matrice globale. Les interfaces artificielles et zones de recouvrement sont déterminées à partir de ces partitions. L'implémentation de RAS de PETSc a été réécrite afin de faciliter l'accès aux données des interfaces. L'ensemble des routines permettant l'utilisation du préconditionneur ARAS(q) dans PETSc a été programmé en utilisant l'interface PC_SHELL permettant à un utilisateur d'intégrer son propre préconditionneur. Mais la bibliothèque PETSc, aujourd'hui, ne propose pas d'implémentation MPI multiniveaux pour les méthodes de résolution et de préconditionnement. Le travail effectué propose une version à deux niveaux MPI de la méthode RAS. Le module implémentant l'accélération d'Aitken ne nécessite pas une réécriture lors du passage à l'implémentation deux niveaux MPI et peut donc être indifféremment utilisé avec un préconditionneur RAS sur un ou deux niveaux MPI.*
Les algorithmes 6, 7, 8, 9 sont ceux utilisés pour effectuer l'accélération d'Aitken. Leur implémentation ajoute un caractère adaptatif à la construction et à l'application de l'accélération permettant au code de gérer la quantité de mémoire à utiliser et de n'effectuer que le nombre d'opérations nécessaire à une accélération efficace.

Ce chapitre permet d'envisager l'expérimentation de la méthode d'Aitken-Schwarz en tant que méthode de préconditionnement ou de résolution pour de très grands systèmes linéaires creux. Dans la continuité de ce chapitre, le chapitre 5 montre les performances des codes et propose une vue d'ensemble des possibilités qu'ils offrent.

This chapter is dedicated to the parallel implementation of the Aitken-Schwarz methods presented in chapter 3. We assume that domain decomposition methods, in general, present good properties to be used on parallel machines due to their natural method of splitting a problem into several sub-problems which can be solved independently.

Considering this characteristic, we present the main points of the method which

can be parallelized. A first study is done on the Schwarz method. Starting from the splitting of problem on a Cartesian grid we propose a natural and parallel distribution of the problem among processes and extend this approach to a parallel distribution on a two-level MPI network. This kind of strategy has been studied in (Haidar 2008), for example. The distribution consists in defining which processes or sub-groups of processes compute which unknowns. Those unknowns can be part of the domain, the overlap or the interface. Then we propose different strategies to compute data exchanges between sub-domains. Those considerations enable us to describe a general approach of the parallelization based on the algebraic representations of sub-problems into sub-matrices. The interfaces and overlap are determined by the data dependencies coming from the global matrix of the global linear system.

A second study is done on the parallelization of the Aitken's acceleration. Regarding the kind of approximation space chosen, different points can be parallelized.

After dealing with the general aspects of the parallelization we present two implementations. The first is an implementation of the Multiplicative Schwarz algorithm accelerated by the Aitken method used as solver. The resulting code was designed for solving 3D Darcy equations in porous media on massively parallel distributed machines. The second concerns the integration of the Aitken Restricted Additive Schwarz preconditioner, and its variants, in the PETSc framework. This code has been designed for general linear systems and more precisely for CFD problem arising from the automatic differentiation. The work which has been done on this code enables the user to use PETSc on a two-level MPI network.

4.1 Parallelization of the Schwarz method

The domain decomposition of the problem leads to several sub-domains describing the same type of equations on a smaller problem. When the problem is discretized on a regular cartesian grid a natural decomposition can be done. Each sub-domain arising from this decomposition can be treated by a single process held by a processor. Then a process needs to be able to solve a linear system and sends and receives data involved in the Schwarz method between sub-domains. Furthermore, each process can be done in parallel, distributing the data of a sub-domain among a sub-group of processor. Finally we show a generalization of the parallelization of a Schwarz algorithm.

4.1.1 A natural parallelization coming from the domain decomposition

In order to illustrate the strategy for parallelizing a Schwarz method, we consider a partial differential equation discretized on a regular step size cartesian grid. For a clear visualisation of all the parallelization possibilities we consider, here, a $3D$ domain represented by a parallelepiped with the leading dimension on Z. Splitting the domain into p sub-domains can be done by slicing the parallelepiped in the leading dimension into p slabs. Each slab can be held by a single process, and more

precisely by a single processor. Then, the sub-problems can be solved the same way as the global problem using a finite volume scheme. The boundary conditions in X and Y directions do not change. But a new boundary condition on an artificial interface needs to be defined. In this study we consider overlapping methods. The boundary condition on artificial interfaces are Dirichlet boundary condition. Each processor can solve its own linear system and sends or receives interface solutions of other processors.

Figure 4.1 shows the strategy for splitting into two sub-domains. The entire domain Ω is sliced into the Z direction. Both the slabs are hold by a single processor 0 or 1. Each processor has the interface's solution to send and receive.

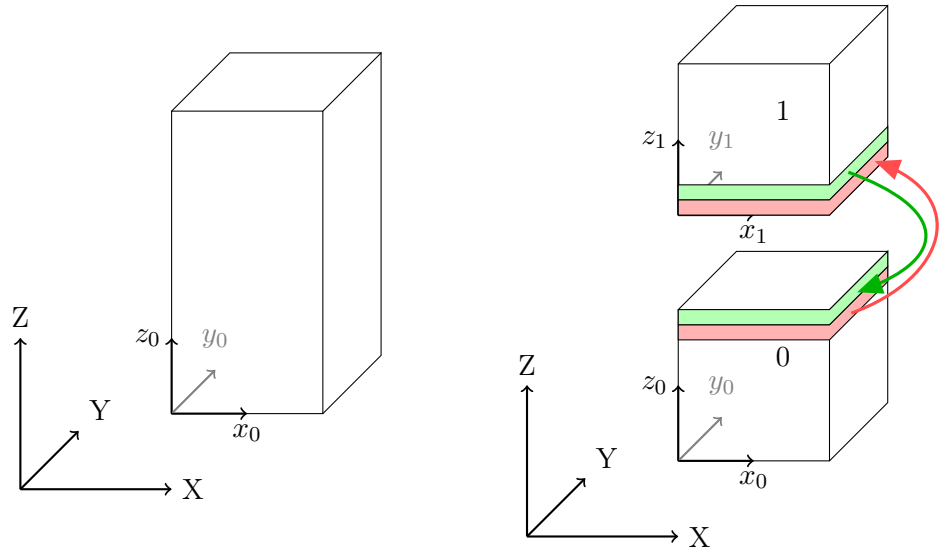

Figure 4.1: (left) Entire computational domain. (right) Z direction splitting in two sub-domains over two processors and exchange of faces between processors.

4.1.2 Two-level parallelization

The same problem is considered. The two-level parallelization is performed by creating a 6D cartesian topology under the Message Passing Interface (MPI). The first level consists of splitting the domain, whatever the number of directions, into sub-domains called macro-domains. The second level consists of splitting a macro-domain, whatever the number of directions, into sub-domains. Let us consider the basic strategy on one level illustrated in Figure 4.1. We would like to compute the solution among 16 processors. Figure 4.2 shows a possible distribution of two sub-domains among 16 processors. We spread the sub-domain previously held by the processor 0 on the sub-group of processors $\{0, 1, 2, 3\}$. We denote this sub-domain by $M0$, the acronym for the macro-domain 0. And now $M0$ contains sub-domains hold by each processor. The same strategy is used for $M1$.

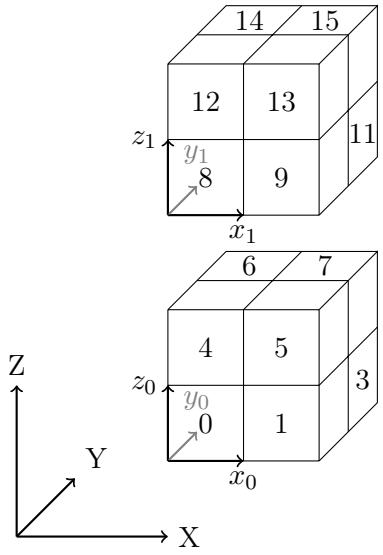

Figure 4.2: 6D MPI cartesian topology for a domain sliced in two sub-domains, in the Z direction, among 16 processors.

4.1.3 Communication strategies

In the case of a one-level MPI distribution the processor holds the interface's solution on its own. But in the case of a two-level MPI strategy, the Schwarz exchanges involve sending the face of a macro-domain which is distributed on several sub-domains. Here two issues can be considered. On one hand, the entire message to pass from a macro-domain to another can be collected on a single process. This possibility enables the developer to use sequential code to treat the interface solution and minimize the number of exchanges. However the size of the data transfer depends on the size of a macro-domain face. Note that some parallel libraries collect the data on a single processor and enforce the user to choose this implementation. On the other hand, each process should conserve its part of the solution's interface and should communicate only with the one which needs the information. The number of exchanges and the time spent for the communications increase with the number of sub-processors on a face of a macro-domain. But the size of the data transfer is smaller than a global send. Both strategies are presented in Figure 4.3. When it is possible, the developer should choose the fully distributed strategy without reducing operations.

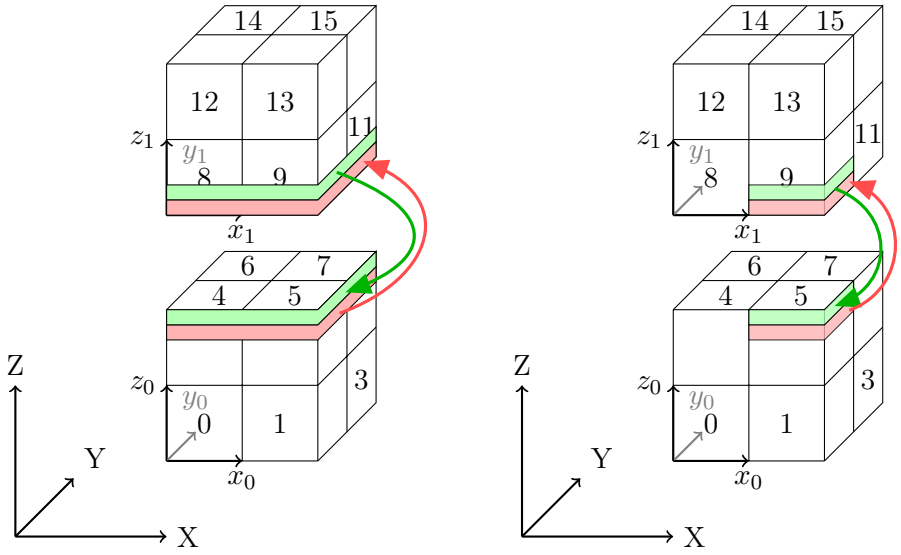

Figure 4.3: 6D MPI cartesian topology for a domain sliced in two sub-domains among 16 processors, (left) interface's solution collected on one processor, (right) interface's solution transfer from a local processor of a macro-domain to a local processor of another macro-domain.

4.1.4 Generalization - Matrix representation

In this subsection we show the link between the geometrical considerations shown in subsections 4.1.1 and 4.1.2 and a domain decomposition of a matrix.

For simplicity, we now consider the matrix representation of a partial differential equation in $1D$ discretized on a regular step size cartesian grid. The domain is split in only one direction as before. Indices of the discretization are sorted. The pattern of such a matrix is three diagonal. Figure 4.4 shows the analogy between the computational domain decomposition and the splitting of the corresponding matrix. With no overlap, the domain decomposition leads to two separated domains. The interface's points, represented by $\times$, are the artificial boundaries of sub-domains. The matrix approach consists of keeping the rows corresponding to the computational sub-domain. Then the data dependencies are marked by a $\times$ on each local row which has data dependencies with the other domain. The data dependencies of local matrices corresponds to the index of the artificial interface. While increasing the overlap of the domain decomposition, the previous process is repeated. The previous data dependencies are introduced in the sub-domain and are called overlap, increasing the size of either the sub-matrices or sub-domains. Then the data dependencies are pointed out by a $\times$ on each local row which has data dependencies with the other domain. It is possible to determine the overlap and interfaces index algebraically by searching the data dependencies between sub-matrices.

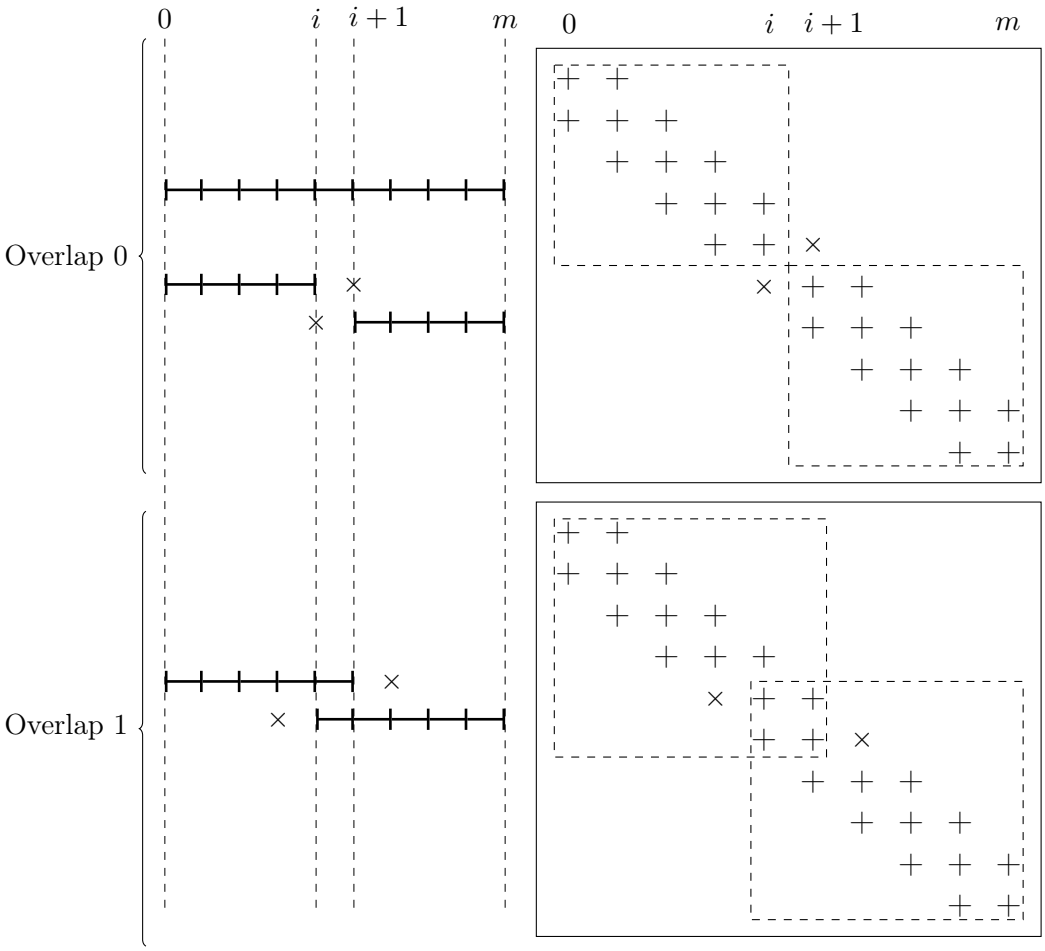

Figure 4.4: Analogy between domain decomposition and sub-matrix dependencies, (left) $1D$ domain decomposition, (right) corresponding matrix splitting.

Then, without any knowledge of both the mesh and the equation, and starting from any partitioning of a matrix, it is possible to design a Schwarz method. This generalization enables the user to set the overlap without knowledge of a regular pattern structure in the matrix.

4.2 Parallelization of the Aitken's acceleration

The parallelization of the Aitken's acceleration could be done for the application of the acceleration or for the building of the base. The parallelization of the building phase depends on the kind of approximation spaces the user chooses. For each choice, parallel algorithms could already exist. For example, if the Fourier base is chosen then the FFT can be done in parallel. Moreover, the projection in the approximation spaces consists of a Matrix-Matrix or Matrix-Vector product which

can be done in parallel. Those aspects will be discuss in sections 4.3 and 4.4.

The application of the acceleration consists in inverting $(I - P)^{-1}$. This matrix can be dense. We will apply LU factorization if the system is small and not necessary in parallel. If the size of this system is quite important we will prefer Krylov methods which scale better than a LU factorization.

Nevertheless, these considerations depend on the choice of the communication strategy presented in subsection 4.1.3. The choice will depend on the data location.

4.3 Code design for massively parallel distributed machines - Application to Darcy equations

In this section, we present an implementation and its optimizations to solve Darcy flow equations with strong variation in the permeability field using an Aitken-Schwarz method as solver. The goal of this code is to provide a robust and scalable tool for solving this equation on large $3D$ computational domains. The resulting number of unknowns goes from several million to hundreds of millions. Then the code must be deployed on massively parallel machine. We first present the equation. Then we proposed a general environment and framework for writing the code. When the framework is defined we propose an implementation of the Schwarz algorithm. Finally the Schwarz algorithm, which has a slow convergence for this case, must be accelerated by the Aitken's acceleration. We discuss two implementations of this acceleration. Results will be provide in chapter 5.
Note that these developments have been presented in (Dufaud & Tromeur-Dervout 2010a, Berenguer *et al.* 20YYa, Berenguer *et al.* 20YYb) which presents large scale computing using the resulting code.

4.3.1 The Darcy flow equation with strong variation in the permeability field

On the macroscopic scale, a porous medium can be described by a model where the solid and the fluid occupy the entire volume. The medium is regarded as a homogeneous domain and modelled as a continuum where a representative volume element is larger than the average pore size but much smaller than the length scale of the system. For this model of saturated flow in homogeneous porous media, the balance of momentum is given by Darcy's law:

$$\left\{ \begin{array}{lcl} u + K\nabla p & = & 0, \text{ in } \Omega \\ \nabla . u & = & 0, \text{ in } \Omega \end{array} \right. \tag{4.1}$$

$$p = g \text{ on } \partial\Omega. \tag{4.2}$$

where u and p are the fluid velocity and hydrostatic pressure, μ is the fluid dynamic viscosity, and K is the permeability of the porous medium. The last quantity depends on the structure of the solid matrix. We will assume that the medium is

isotropic, $K = kI$, where k is an averaged quantity. Since in the Darcy's equations viscous stresses on the fluid are neglected and only the damping force of the porous medium is considered, it is valid for small permeabilities.

In the case of a $3D$ medium, the computational domain is a regular grid on which a random hydraulic conductivity field k is generated. This random hydraulic conductivity field K follows a stationary log-normal probability distribution $Y = ln(k)$, which is defined by a mean m_Y and a covariance function $C_Y(x, y, z) = \sigma_Y exp(-[(\frac{x}{\lambda_x})^2 + (\frac{y}{\lambda_y})^2 + (\frac{z}{\lambda_z})^2]^{\frac{1}{2}})$ where σ_Y is the variance of the log hydraulic conductivity and λ_x, λ_y and λ_z are the directional correlation length scales in each direction. To generate the random hydraulic field, a spectral simulation based on the FFT method (Fast Fourier Transform method) is used. For the sake of simplicity, we take the same value λ for λ_x, λ_y and λ_z.

The physical modelling of the heterogeneous media leads to several types of difficulties even for the linear Darcy equation. The treatment of various scales leads to solving sparse linear systems of very big size with bad condition number due to the heterogeneous permeability K.

The σ and the λ parameters play on the stiffness of the linear system to be solved. The range of σ^2 is usually from two to six and λ goes from two to ten. σ plays on the amplitude of the permeability K, for $\sigma^2 = 4$ the K varies in mean from 10^{-4} to 10^4. λ plays on the length scale for the change of K, the smaller λ is the greater the probability is that K varies strongly from cell to cell.

4.3.2 Coding environment

Solving very large linear system involves large memory needs. Even if one machine could have enough RAM, a single processor could not access all the memory in a reasonable time. Parallelization is then necessary. Most of the biggest machines over the world use architecture with distributed memory. The current architectures have shared memory at a low level, generally on a processor, or possibly on a node. When the application needs several nodes, it is better to use a distributed approach. Expert developers try to exploit all kinds of hardware of such parallel machines. We choose to implement a fully distributed code to ensure portability. The protocol for communication between processes is MPI (MPI 2009).

We need a language improved for scientific computing. Most of the legacy codes are written in FORTRAN and are optimized. We naturally choose the FORTRAN 90 language to develop our code. It proposes dynamic management of memory and easy declaration of mathematical entities. Moreover, the FORTRAN 90 and following generations proposed organisation by modules. A module presents an external code with its own data and routines. This organisation enables the developer to work on his own module and easily share his work.

4.3.3 Domain decomposition on two-level MPI

We expect to be able to solve this $3D$ problem on regular cartesian grid of size $512 \times 512 \times 4096$. This represents more than one billion unknowns. For this kind of problem we must use a domain decomposition method in order to reduce the size and complexity of a solution. In order to optimize the load balance, we choose a two-level MPI implementation such as in subsection 4.1.2. We must give communicators to MPI and create a network to define our data locations. Then we can build the sub-systems without creating the entire system.

4.3.3.1 $6D$ MPI network

We create a network which is going to fit the domain decomposition such as in Figure 4.2. In this figure, coordinates of the first level are in the referential (Z, Y, X) and for each macro-domain i on the second level, we denote by (z_i, y_i, x_i) the macro-local referential. In terms of MPI network, coordinates are replaced by process numbering. Then, on the first level, each macro-domain seen as a process, has coordinates on the first level MPI grid in the referential (M_z, M_y, M_x). On the second level, the processes are the processors of a macro-domain and are pointed out by their coordinates in a local referential corresponding to the numbering of processors in each direction: (P_z, P_y, P_x).

Then the position of a processor in the MPI network can be given in the $6D$ cartesian topology in the referential $(M_z, M_y, M_x, P_z, P_y, P_x)$.

To define such a network it is necessary to provide the number of processes needed in each direction. Those numbers can be contained in an array of integers of size six. Then, from an initial MPI communicator we define two communicators which describe the network. One of them organizes the processes in a global $6D$ cartesian grid and will be denoted by comm6D. The second one represents the $3D$ network on a macro-domain and will be denoted by commMacro. Those steps are summarized in Algorithm 12.
For convenience we define the notation we will use for defining the $6D$ topology:

- *nbprocdir*, an array of integers of size six which contains the number of processes in each direction.
- $comm6D$, the communicator on the entire network.
- $commMacro$, the communicator on the current macro-domain.
- *rank*, the rank of a processor on $comm6D$.
- *subrank*, the rank of a processor on $commMacro$.
- *coords*, an array of integers of size six containing the coordinates of the current processor.
- *procvois*, an array of integers of size 12 containing the numerous of the neighbours processes in each direction.

Algorithm 12 Define a $6D$ cartesian topology and its communicators

Require: Number of processes in each direction and an initial communicator, generally MPI_COMM_WORLD.

1: Create a communicator for a $6D$ cartesian topology using mpi_cart_create()
2: Collect the cartesian coordinates of the current processes using mpi_cart_create() in an array of integers of size six, *coords*.
3: Define neighbours of current processes in the $6D$ topology and write it in an array of integers of size 12, *procvois*.
4: Define a $3D$ sub-communicator denoted by *commMacro*, for macro-domain by coloration using mpi_comm_split().

Example 1 Let us take the splitting illustrated in Figure 4.2. The number of processes in each direction is given by $nbprocdir = [2, 1, 1, 2, 2, 2]$. *The following Table 4.1 shows the values for each variables we need to define the topology.*

rank	*subrank*	*coords*	*procvois*
0	0	$[0,0,0,0,0,0]$	$[-1,-1,-1,-1,-1,-1,-1,4,-1,2,-1,1]$
1	1	$[0,0,0,0,0,1]$	$[-1,-1,-1,-1,-1,-1,-1,5,-1,3,0,-1]$
2	2	$[0,0,0,0,1,0]$	$[-1,-1,-1,-1,-1,-1,-1,6,0,-1,-1,3]$
3	3	$[0,0,0,0,1,1]$	$[-1,-1,-1,-1,-1,-1,-1,7,1,-1,2,-1]$
4	4	$[0,0,0,1,0,0]$	$[-1,1,-1,-1,-1,-1,0,-1,-1,6,-1,5]$
5	5	$[0,0,0,1,0,1]$	$[-1,1,-1,-1,-1,-1,1,-1,-1,7,4,-1]$
6	6	$[0,0,0,1,1,0]$	$[-1,1,-1,-1,-1,-1,2,-1,4,-1,-1,7]$
7	7	$[0,0,0,1,1,1]$	$[-1,1,-1,-1,-1,-1,3,-1,5,-1,6,-1]$
8	0	$[1,0,0,0,0,0]$	$[-1,-1,-1,-1,-1,-1,-1,4,-1,2,-1,1]$
9	1	$[1,0,0,0,0,1]$	$[-1,-1,-1,-1,-1,-1,-1,5,-1,3,0,-1]$
10	2	$[1,0,0,0,1,0]$	$[-1,-1,-1,-1,-1,-1,-1,6,0,-1,-1,3]$
11	3	$[1,0,0,0,1,1]$	$[-1,-1,-1,-1,-1,-1,-1,7,1,-1,2,-1]$
12	4	$[1,0,0,1,0,0]$	$[-1,1,-1,-1,-1,-1,0,-1,-1,6,-1,5]$
13	5	$[1,0,0,1,0,1]$	$[-1,1,-1,-1,-1,-1,1,-1,-1,7,4,-1]$
14	6	$[1,0,0,1,1,0]$	$[-1,1,-1,-1,-1,-1,2,-1,4,-1,-1,7]$
15	7	$[1,0,0,1,1,1]$	$[-1,1,-1,-1,-1,-1,3,-1,5,-1,6,-1]$

Table 4.1: Variables values defining the $6D$ cartesian topology for $nbprocdir = [2, 1, 1, 2, 2, 2]$.

4.3.3.2 Creation of macro-domains grid

Since the network is set, the computational grid can be defined. In order to control the load balance, we wish to define the grid size defining the local grid size owned by a processor. The number of points in each local direction is given and is equal for each processor. For this overlapping domain decomposition we set an overlap of computational nodes or edges of a graph partitioning of the mesh. This overlap must be greater than 0.

For convenience we define the notation we will use for defining the computational grid dimension:

- nz, number of local grid points minus 1 in z direction.
- ny, number of local grid points minus 1 in y direction.
- nx, number of local grid points minus 1 in x direction.
- k, z coordinate of a grid point on a processor.
- j, y coordinate of a grid point on a processor.
- i, x coordinate of a grid point on a processor.
- *overlap*, the number of grid points which are overlapped between two domains in the Z direction.
- N, number of unknowns considerate in a macro-domain,
  $N = (nx+1)*(ny+1)*(nz+1)*nbprocdir(6)*nbprocdir(5)*nbprocdir(4)$.

The numeration of unknowns for each grid point is given by a ($subrank$,k,j,i) numeration. The correspondence with the global grid is found considering the topology described before.

4.3.3.3 Creation of sub-systems

The domain splitting among the processors is now defined. The next step is to define a linear system on a macro-domain. Each entries of the operator are computed and stored locally in a (i,j,Aij) format. The Dirichlet boundary conditions are incorporated in the matrix. Then the matrix is no longer symmetric.

4.3.4 Red-Black Multiplicative Schwarz Algorithm

Here we choose a Schwarz algorithm and plan to accelerate it by the Aitken's acceleration. Even if local domains are reduced, the size of the interface stays large. The Additive Schwarz algorithm is more suitable for parallelism than Multiplicative Schwarz because at each iteration every local problem is solved independently which is not the case for Multiplicative Schwarz, but the size of the interface may be a problem for the Aitken's acceleration. Nevertheless, the Multiplicative Schwarz algorithm has better convergence than the Additive Schwarz. For our problem we choose to slice the domain in one direction and then reduce the number of dependencies between sub-domains. Then a red-black approach can be used.

The domain Ω is split in the z direction into overlapping macro-domains Ω_i. The domain is discretized with regular step sizes in each direction. The Schwarz algorithm consists in updating the macro-domain boundary conditions until convergence with taking values in the neighbours macro-domains.

Defining A_i the discrete operator of the Darcy equation on the macro-domain Ω_i that takes into account the Dirichlet boundary conditions on the two artificial interfaces $\Gamma_{i,l}$ and $\Gamma_{i,r}$, then the multiplicative Schwarz algorithm writes:

$$\left\{\begin{array}{l} A_{2i+1}p_{2i+1}^{2n+1} = f_{2i+1}, \\ p_{2i+1|\Gamma_{2i+1,l}}^{2n+1} = p_{2i|\Gamma_{2i+1,1}}^{2n} \\ p_{2i+1|\Gamma_{2i+1,r}}^{2n+1} = p_{2i+2|\Gamma_{2i+1,r}}^{2n} \end{array}\right. \qquad \left\{\begin{array}{l} A_{2i}p_{2i}^{2n+2} = f_{2i}, \\ p_{2i|\Gamma_{2i,l}}^{2n+2} = p_{2i-1|\Gamma_{2i,1}}^{2n+1} \\ p_{2i|\Gamma_{2i,r}}^{2n+2} = p_{2i+1|\Gamma_{2i,r}}^{2n+1} \end{array}\right. \tag{4.3}$$

4.3.4.1 Choice of a solver

For reasonable sizes a direct method can be used. When the size of matrices increases, the factorization time and memory space start to be too expensive. At the beginning, we start to wrap a direct solver called MUMPS (Amestoy *et al.* 2001, Amestoy *et al.* 2006). We presented first results of our code on small problems in (Dufaud & Tromeur-Dervout 2010a). Unfortunately our expectations are greater than the possibility given by direct solvers. Indeed the maximum global grid size we reached were $128\times128\times128$. It is known that the computing cost of a LU factorization depends on the maximum bandwidth of the sparse profile.

The community uses for this kind of linear systems Algebraic Multigrid Methods. We wrapped the AGMG solver developed by Yvan Notay (Notay 2010). This code is written in FORTRAN. In order to use it we convert the (i,j,Aij) storage format into a CRCC storage format.

4.3.4.2 Exchanges between macro-domain

We use here the strategy involving only local communication without gathering data presented in 4.1.3.

4.3.5 Aitken's acceleration implementations

We present the implementation of the Aitken's acceleration in the case of this Red-Black Multiplicative Schwarz algorithm. The goal of this work is to save computational time by adaptively building the approximation space of the acceleration. On one hand, we present the algorithm to adaptively choose which modes of the Fourier base to accelerate. The case of Dirichlet boundary conditions on all the faces of a macro-domain is studied. On the second hand, we introduce the adaptive selection of singular values and present the possibility of computing those values incrementally.

4.3.5.1 Aitken's acceleration for Multiplicative Schwarz algorithm

The convergence of this two level domain decomposition is purely linear. Therefore the convergence of the solution can be accelerated by the Aitken's formula written in equation (3.11) at the artificial interfaces.

Once the converged solution is obtained at artificial interfaces, one local solution gives the solution on the macro-domain.

We note that for a Red-Black Multiplicative Schwarz algorithm it is possible to accelerate interfaces at a half iteration. Then it consists in computing acceleration only on even or odd interfaces. So, the acceleration will be computed by every other macro-domains. We choose here to accelerate only interfaces of odd macro-domains. This choice is illustrated on Figure 4.5.

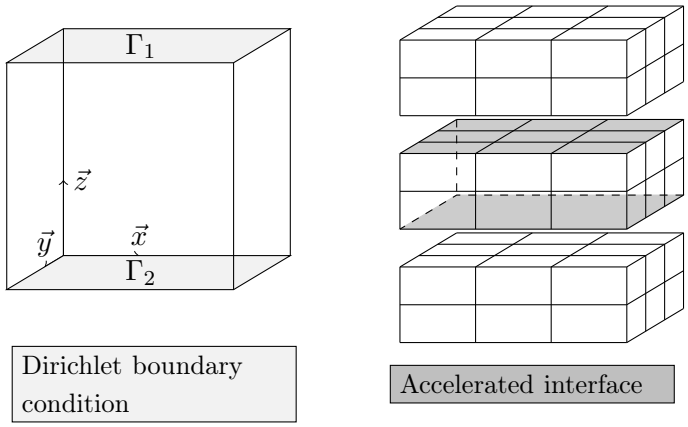

Figure 4.5: Location of the interface's solution to accelerate on every other macro-domain.

4.3.5.2 Adaptive building of the Fourier base

The full computation of $\hat{\mathbb{P}}$ can be done in parallel, but it needs as much local macro-domain solution as the number of points of the interface (i.e the size of the matrix $\hat{\mathbb{P}}$). Its adaptive computation is required to save computing. The Fourier mode convergence gives a tool to select the Fourier modes that slow the convergence and have to be accelerated. Then we consider Algorithm 6 to build the $\hat{\mathbb{P}}$ with respect to the modes that have not converged, and manage the Aitken's acceleration adaptively.

The error written in Fourier base at the interface i is considered. This error can be represented in a $2D$ space. Then, there is different possibilities for chosen the mode to accelerate. Algorithm 6 suggests covering a frequency domain taking all the mode contained in a rectangle delimited by $(1, 1)$, on the top left, and by $(k_{i,max}, l_{i,max})$ on the bottom right. This selection covering a delimited zone ensures that we consider the strongest interaction between modes. Nevertheless, this needs to be extended to the entire interface taking the largest zone covering the interface by projection. Then we consider $(k_{max}, l_{max}) = max_i \{(k_{i,max}, l_{i,max})\}$. The increasing of the number of modes to accelerate is illustrated in Figure 4.6.

After pointing out the rectangle zone of a face to accelerate, a treatment is done to take into account the previous acceleration, which means adding only the mode which has not been accelerated previously to the list of modes to accelerate. Then we do not compute the previous modes again.

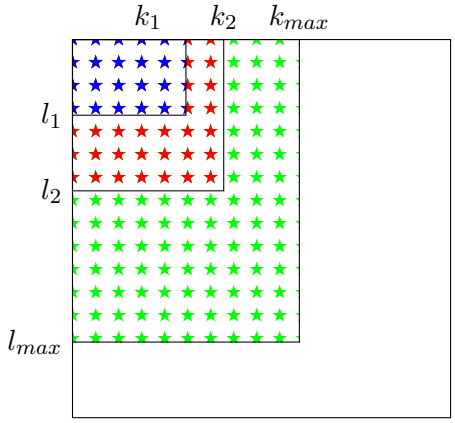

Figure 4.6: Mode selection on one interface in three steps. The first acceleration is performed on the smaller space and then grows.

4.3.5.3 Adaptive building of the base arising from the Singular Value Decomposition of interface's solutions

We choose Algorithm 9 without inversion for robustness. The adaptive algorithm consists of "throwing" the base after an acceleration and apply again the algorithm with a new sequence of Schwarz interface solutions.

Figure 4.7 illustrates the selection of the singular values. The SVD is computed from a set of Schwarz interface's solutions. The selection consists in cutting to the bend of the curve represented by the plot of the singular values.

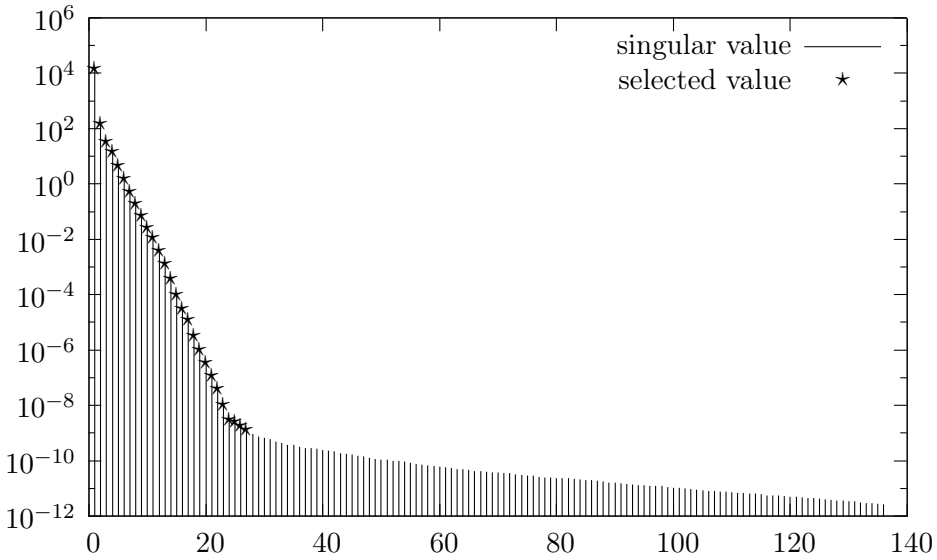

Figure 4.7: Selection of singular values from a set of size 135.

One issue of the algorithm is the choice of the number of Schwarz iterations to perform. If the number is too small the acceleration would not be efficient. If the number is too large, the approximation space could contain noise. The code should be able to adaptively choose the number of Schwarz solutions to compute. The idea is to compute an incremental SVD enabling the automatic choice of the number of Schwarz iterations to compute. So we consider the adaptive algorithm 8 or 9.

4.4 Design of the ARAS preconditioner in PETSc - Application to CFD problem arising from the automatic differentiation

In the context of solving sparse linear systems arising from the automatic differentiation in CFD, we do not always have access to the mesh and we do not know how the matrices are obtained. These constraints lead engineers to look for a black box solver sufficiently robust and cheap in terms of time consumption and memory allocation. A Krylov method preconditioned by a RAS preconditioner seems to be a robust answer. We propose to provide the ARAS2 preconditioner, which we suppose to be cheaper and with a better scalability than RAS, in a largely distributed library used by researchers and engineers.
Parts of the following work has been presented in (Dufaud & Tromeur-Dervout 2011) and is the object of a submitted paper (Dufaud & Tromeur-Dervout 20YYb).

4.4.1 PETSc framework and issues

We chose the Portable, Extensible Toolkit for Scientific Computation (Balay *et al.* 2009) library which is written in parallel (Balay *et al.* 1997), actively supported and used by a large community of researchers and engineers. We chose the C language implementation. An effort is made to write a code using only the data structures of the library in order to facilitate its development and future support.

From the PETSc website (Balay *et al.* 2009), we obtain the descrition of the principal data structures we use:

- Vec objects are used to store the field variables in PDE-based (or other) simulations.
- Mat objects are used to store Jacobians and other sparse matrices in PDE-based (or other) simulations.
- IS objects are used to index into vectors and matrices and to setup vector scatters.

- VecScatter objects are used to manage communication of data between vectors in parallel. It manages both scatters and gathers
- KSP, interface for Krylov subspace iterative methods.
- PC, interface for preconditioners. A PC_* type corresponds to the preconditioner *.

We develop our parallel implementation of the ARAS2 preconditioning technique with the PETSc user's provided interface for preconditioners (Balay *et al.* 2008). This interface provides the PC_SHELL interface which enables the developer to implement his own PC. Then, the PC_SHELL is provided to the KSP solver of Krylov type. PETSc provides easy way to select the local solver between direct or iterative class methods depending on the local matrix properties. These local solvers can also be sequential or parallel. It is important to remark that the current version of the Schwarz preconditioner implemented in PETSc does not support a two-level MPI network.

Although, PETSc provides a RAS preconditioner (PC_ASM) based on a context for data exchange, this is not adapted for the implementation of ARAS2 which needs to modify the value of the RAS iterate onto the sub-domain's interface. This is why we had to define our own PC context for data exchange.

In the following we describe the implementation of the main feature of the ARAS2, the PC_SHELL, the local solution, the context of data communication and finally the building of the $\mathbb{U}_q$ and $P_{\mathbb{U}_q}$. Then we discuss the limits of such an implementation and extend the code to a two-level MPI version.

4.4.2 Using PC_SHELL interface provided by PETSc

One can provide one's own preconditioner for a KSP method using the PC_SHELL interface. The user needs to define a context for his preconditioner and then write the corresponding functions to CREATE, SETUP, APPLY and DESTROY the preconditioner object. This particularity of the PETSc library is actually interesting for testing optimisation techniques. The idea is to build a preconditioner the classical way and use it in the setup of the optimized version. Such a trick is done integrating the classical PC in the PC_SHELL context.

For instance, the ARAS preconditioner is obtained by combining the RAS preconditioner with the Aitken acceleration. The RAS application is provided by the corresponding PC_Apply call and the Acceleration is performed in the PC_Apply, call of the PC_SHELL. The integration of the Aitken Schwarz preconditioner in PETSc is illustrated in Figure 4.8.

Although the plugging in of optimizers such as a RAS preconditioner is enabled, the PETSc implementation of ASM presents a lack of needed data to implement the Aitken's acceleration. Actually, in order to build the Aitken RAS preconditioner, we need knowledge of both the overlap index set and the interface index set. Unfortunately, in the current version, the PC_ASM type does not provide the overlap

Solver	Facto.	Iter.	$\|\|r^k\|\|_2$	tot. cpu time	Glo. Mem.
GMRES(ASM)	lu	61	5.1284485e-13	202.056s	5655m
GMRES(ASM)	ilu(3)	193	2.3678877e-12	13.391s	1116m
GMRES(RAS)	lu	61	5.1284485e-13	201.829s	5664m
GMRES(RAS)	ilu(3)	193	2.3678877e-12	13.540s	1135m

Table 4.2: Code performances for solving problem FR02 with $p = 4$, overlap 2, with Facto. the factorization method used for local solution, Iter. the number of GMRES iterations, $||r^k||_2$ the residual of the GMRES method, tot. cpu time the total elapsed time for reading, partitioning, solving and post treating the linear system, and Glo. Mem. the global memory used by the program.

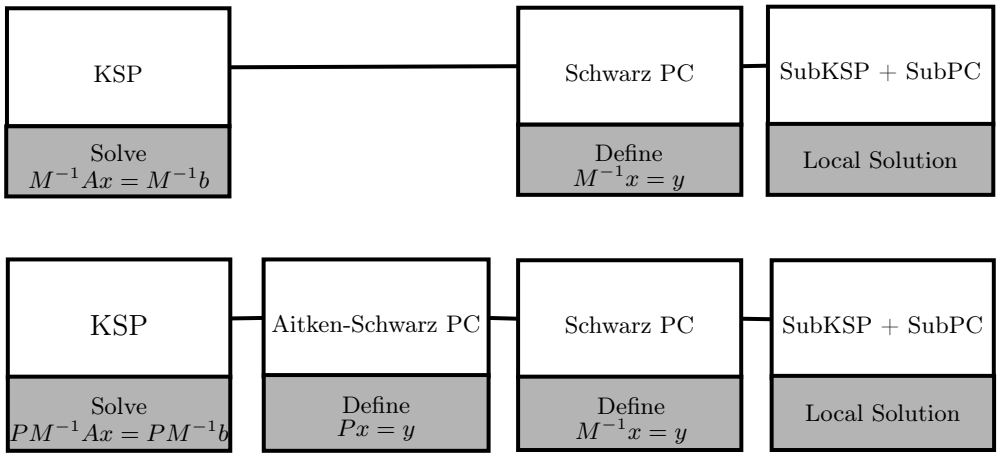

Figure 4.8: General implementation for solving a preconditioned linear system with PETSc : (top) KSP solver preconditioned by a Schwarz method, (bottom) KSP solver preconditioned by an Aitken-Schwarz method.

index set since the data locations are sufficient to determine if the solution is on the local domain or part of the neighbourhood.

Thus an implementation in the current version needs a re-writing of the Restricted Additive Schwarz preconditioner including the overlap index set. Furthermore, this lack gives us the opportunity to have total control of our preconditioner because the data access is simplified. The proposed implementation stays close (see Table 4.2) to the one of PETSc in terms of number of operations. Only the communications given by the VecScatter class are different.

4.4.3 Overview of the implementation of the RAS PC

The PC_ASM implementation does not give a clear access to the data contained in the overlap and in the interface. We make an effort to clearly separate those entities by creating independent index sets of type IS denoted by:

- is_rest, index set of the non-overlapping partition.
- $is\_overlap$, index set of overlapping data.
- is_interf, index set of data dependencies between sub-domains.
- $is\_lessinterf$, union of is_rest and $is\_overlap$.
- is, index set containing locally the data managed by a processor including the restricted partition, the overlap and the data dependencies.

Those index sets are created following the formal Algorithm 13 which is designed to suit the PETSc library. The local problem is defined on the index set $is\_lessinterf$.

Algorithm 13 Define index sets formally and with PETSc denomination in the set up phase of RAS PC

Require: A matrix A defined on PETSC_COMM_WORLD and its index set is, let $is(0)$ be the index set of the partition without overlap, $ov = 0$, $is\_temp = is(0)$

1: $is(0) \rightarrow is(1)$
Increase overlap of partitions of A defined by is and obtain a new is
2: $is\_overlap = is(1) \setminus is\_temp$
Compute difference between is and is_temp to obtain $is\_overlap$
3: $is\_interf = is\_overlap$
Copy $is\_overlap$ into is_interf
4: $is\_temp = is(1) \cup is\_temp$
Expand is with is_temp to define is_temp again
5: **for** $ov = 1$ **to** $overlap$ **do**
6: $is(ov) \rightarrow is(ov + 1)$
Increase overlap of partitions of A defined by is and obtain a new is
7: $is\_overlap = is\_interf \cup is\_overlap$
Expand is_interf with $is\_overlap$ to obtain the new $is\_overlap$
8: $is\_interf = is(ov + 1) \setminus is\_temp$
Compute the difference between is and is_temp to obtain is_interf
9: $is\_interf = is(ov + 1) \cup is\_temp$
Expand is with is_temp to define is_temp again
10: $is\_rest = is(ov + 1) \setminus (is\_interf \cup is\_overlap)$
Expand is_interf with $is\_overlap$ and subtract the result to is to obtain is_rest
11: $is\_lessinterf = is\_rest \cup is\_overlap$
Expand is_rest with $is\_overlap$ to obtain $is\_lessinterf$

VecScatter contexts are defined to make communications between domains. Since the index sets are clearly separated, we can simply define the correspondence between data on the entire domain and data located on a processor and part of the

non-overlapping partition, the overlap or the interface.

The local solution is performed using the KSP class. The local solver can be set as PREONLY for direct local solutions calling a complete or incomplete LU factorization. If the local problem has a too large band profile, then the user can set an iterative solver. The local solution is a sequential task because the current implementation does not include a multilevel parallelization as in the code written for solving $3D$ Darcy equations in section 4.3.

4.4.4 Aitken's acceleration on artificial boundary conditions

The data are distributed over the domain following the partitioning. Data on each processor can be part of the local domain, or be part of the overlap or the interface. The boundary conditions are easily updated while all the local solutions are done. In fact, the application of the Restricted Additive Schwarz preconditioner on a distributed vector makes this update transparent for the user. But the Aitken process is applied only on the interface without the overlap. This interface is distributed over the processors and then the interface problem is parallelized.
Computing an Aitken-Schwarz preconditioner consists in computing a process on an orthogonal base to create the Aitken acceleration matrix in the corresponding space and then writing the application of the acceleration and the substitution to the classical Restricted Additive Schwarz preconditioner.

4.4.4.1 Building $P_{\mathbb{U}_q}$ in $\mathbb{U}_q$

First we need to define an appropriate orthogonal base. Here we detail the approach used with PETSc for two different bases introduced in the preceding section. Algorithm 14 presents the computation of the random orthogonal base with SPRNG (Mascagni & Srinivasan 2000) arising from a coarse algebraic approximation of the interface. On the other hand, Algorithm 15 presents the computation of the orthogonal base arising from SVD of the interface's solutions of the Richardson process using the SLEPc (Hernandez *et al.* 2005) package.

Algorithm 14 Orthogonal Random Base

1: Generate random vectors on the global interface Γ with the SPRNG library.
2: Add the right hand side to this array of vectors.
3: Apply 2 times M_{RAS}^{-1}.
4: Orthogonalize the resulting vectors using the Classical Gram-Schmidt Algorithm.

Algorithm 15 SVD Base

1: Compute q iterations of Schwarz viewed as a Richardson process.
2: Extract the interface of each iteration.
3: Compute the SVD of those interfaces with SLEPc.

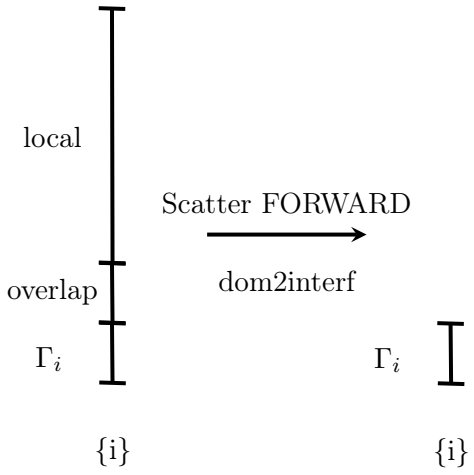

Figure 4.9: Scheme of the dom2interf scatter context computing $R_\Gamma x$

Once the user provides the orthogonal base $\mathbb{U}_q$, from one of the two algorithms, he should compute one iteration of the Richardson process on this base to obtain $P_{\mathbb{U}_q}$. Note that the local solution computed in the application of M_{RAS}^{-1} to a vector x can have a relaxed tolerance compared to the tolerance set for the local solution involved in the preconditioning.
This can be done only if the code contains a scatter context which represents the R_Γ operator, since the Richardson process is done on all the domain. We consider that the interface's data are the non-local dependencies corresponding to the column indices which are different from the local row indices. Figure 4.9 represents for a local domain i the domain-to-interface scattering. Doing this we compute $P_{\mathbb{U}_q} = \mathbb{U}_q^T P \mathbb{U}_q$.

4.4.4.2 Accelerating the Restricted Additive Schwarz Preconditioner

As we can see in Figure 4.8, the new preconditioner is built with the Restricted Additive Schwarz Preconditioner. Since the user provides the base $\mathbb{U}_q$ and the matrix $P_{\mathbb{U}_q}$, the main aspect of the implementation of ARAS is the application of M_{ARAS}^{-1} (see equation (3.25)) to a vector x, presented in Algorithm 16.

Algorithm 16 Computing $M_{ARAS}^{-1}x = y$

Require: pc_ras, the Restricted Additive Schwarz preconditioner context with an associated apply routine PC_APPLY, and an input vector x.

1: Apply pc_ras to x to obtain y : $y = M_{RAS}^{-1}x$
2: Scatter the interface with the dom2interf scatter context define in 4.9 : $y_\Gamma = R_\Gamma y$
3: Project the interface in the chosen base : $y_{\Gamma\mathbb{U}_q} = \mathbb{U}_q^T y_\Gamma$
4: Solve $(I_q - P_{\mathbb{U}_q})z = y_{\Gamma\mathbb{U}_q}$
5: Compute $y_{\Gamma\mathbb{U}_q} = z - y_{\Gamma\mathbb{U}_q}$
6: Project $y_{\Gamma\mathbb{U}_q}$ in the real space on the interface : $y_\Gamma = \mathbb{U}_q y_{\Gamma\mathbb{U}_q}$
7: Make a dom2interf scatter reverse with additive operation : $y = y + R_\Gamma^T y_{\Gamma\mathbb{U}_q}$

4.4.5 A two-level MPI implementation for Schwarz preconditioner in PETSc

At the time I worked on the implementation of the ARAS preconditioners in PETSc, the developer team of the library had not written their library to be performed on several MPI levels. We motivated our choice to write Schwarz algorithms in a two-level MPI network. The motivation has not changed for this code and even if there exists current work on a two-level MPI possibility in PETSc, we propose a methodology and an implementation concerning this feature. Concerning the Schwarz preconditioner, we need to define the sub-system and associate a parallel KSP solver preconditioned by a parallel PC and the communications through the MPI levels. We propose a general method describing the different add-ons.

4.4.5.1 Data distribution

We refer to the matrix representation of a domain decomposition in subsection 4.1.4. The sub-matrices are now distributed on processors. The first step is to split the processors into sub-groups containing the same number of processors and create the sub-communicator following Algorithm 17. Then we need to define parallel index sets, vectors, and matrices on a sub-communicator.

Algorithm 17 Define a sub-communicator for PETSc

Require: Number of processes treated by PETSc an initial communicator, generally PETSC_COMM_WORLD.

1: Define a color for each processor of a subgroup, each group has the same number of processors
2: Define a sub-communicator denoted by subcomm, for sub-domain by coloration using mpi_comm_split().

The problem must be partitioned in a number of partitions smaller than the number of processors involved. The matrix is viewed as a matrix on all the processors on the communicator PETSC_COMM_WORLD. The union of index sets of each

processor of a sub-group of processor gives the index set of the partition of the subgroup. Figure 4.10 shows the distribution of the vertices of a graph partitioning into two parts of a matrix A among two processors and among eight processors.

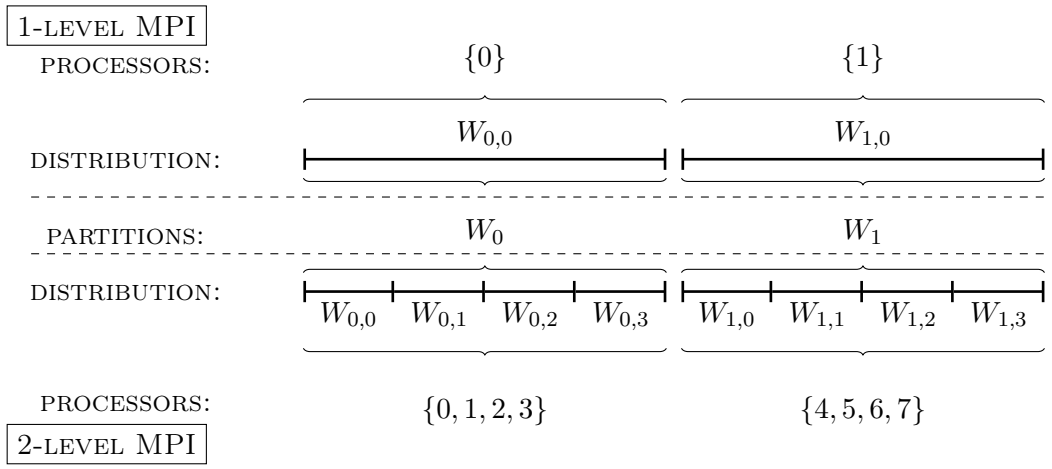

Figure 4.10: Data distribution after partitioning in 2 parts among subgroups of processors for one-level (two processors) or two-level MPI (two sub-groups of four processors)

The SetUp phase of the preconditioner involves the call of a PETSc function to increase the overlap between partitions. We use this function considering that the union of the resulting local index sets of a sub-domain is the index set of the overlapping partition. Those new index sets of type IS are denoted by:

- $subis\_rest$, index set of the non-overlapping partition on *subcomm*
- $subis\_overlap$, index set of overlapping data on *subcomm*
- $subis\_interf$, index set of data dependencies between sub-domains on *subcomm*
- $subis\_lessinterf$, union of $subis\_rest$ and $subis\_overlap$
- *subis*, index set containing on *subcomm* the data managed by a processor including the restricted partition, the overlap and the data dependencies

The building of index sets for the two-level MPI implementation can be summarized as in Algorithm 18. Note that this algorithm refers to Algorithm 13.

Then the sub-system on the sub-communicator *subcomm* can be computed using the index set $subis\_lessinterf$. The local solution can be performed on *subcomm* using a parallel KSP preconditioned by a parallel PC.

Algorithm 18 Define sub-index sets

Require: A matrix A defined on PETSC_COMM_WORLD and its index set is, representing a partitioning on p partition less than the number of processors.

1: Apply Algorithm 13
2: Create index sets corresponding to is_rest, $is\_overlap$, is_interf, $is\_lessinterf$, on $subcomm$ containing double values due to internal overlap in a sub-domain
3: Remove double values and create the corresponding index sets with prefix sub: $subis\_rest$, $subis\_overlap$, $subis\_interf$, $subis\_lessinterf$

4.4.5.2 Data communication

Nevertheless, the communication between the entire domain and the sub-domain need special care. In reality it is not possible to scatter data between entities defined on different groups of processors. The implementation of the VecScatter routines enable the user to scatter data between two vectors defined:

- both on the same communicator
- one on a communicator and the other on PETSC_COMM_SELF
- both on PETSC_COMM_SELF

We propose to provide a scatter between a communicator *comm* and a communicator *subcomm* computing two scatters. The first scatter enables the communication between *comm* and PETSC_COMM_SELF while the second one enables the communication between *subcomm* and PETSC_COMM_SELF. Figure 4.11 shows the scatter process for a graph partitioning into two parts of a matrix A among 8 processors.

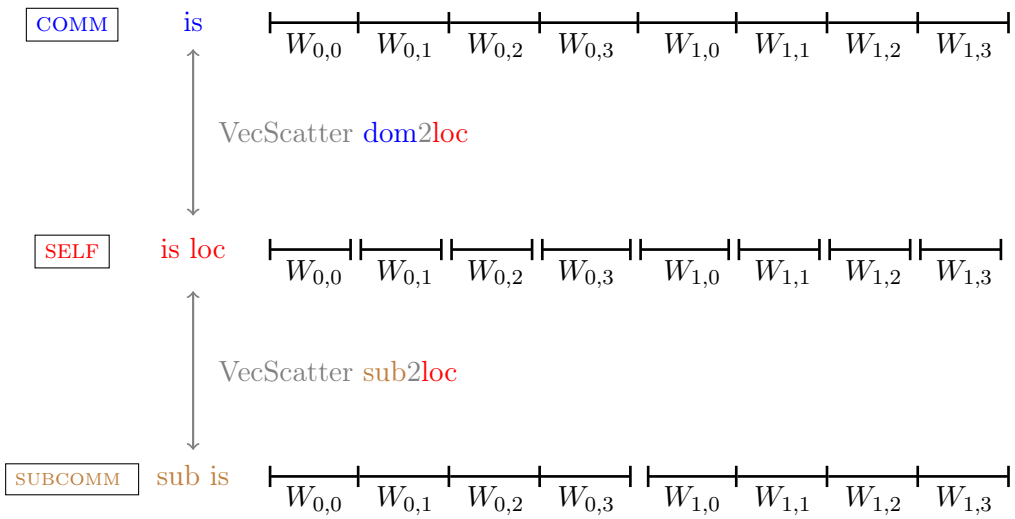

Figure 4.11: VecScatter : domain to local and sub-domain to local

4.5 Concluding remarks

The methodology presented in this chapter enables us to experiments the Aitken-Schwarz method as solver and preconditioner for very large sparse linear systems. A general way to massively parallelize a Schwarz method has been proposed. The method which consists of making a two-level grid parallelization is naturally designed for a general cartesian grid and can be extrapolated in a totally algebraic approach. The work which has been done proposes extending of the PETSc implementation of the Schwarz preconditioner on one-level to a two-level MPI implementation. An original adaptive and incremental algorithm for computing the Aitken's acceleration has been proposed and written based on the algorithms presented in section 3.1 of chapter 3. These adaptive accelerations enable the program to adaptively save computing and memory achieving good performances. Then research applications involving the solution of billions of unknowns can consider this kind of method. Moreover, in the PETSc implementation, it enables a user to enhance a decomposition method with a totally algebraic and mesh-free method. This point presents serious advantages for industrial applications. As a continuity of this part, chapter 5 shows performances of the codes and also provides an overview of the possibilities they offer.

Chapter 5

Numerical and parallel performances

French summary - Résumé en français

Performances numériques et parallèles

La motivation principale de notre recherche est de produire une méthode de préconditionnement robuste et performante dédiée à la résolution de systèmes linéaires industriels, présentés dans le chapitre introductif 1, et proposés par une entreprise spécialiste de la CFD, FLUOREM. L'objectif de ce chapitre est de mettre en évidence les performances numériques du préconditionneur ARAS (voir chapitre 3) sur les cas industriels et sur de grands systèmes linéaires creux. Mais les performances numériques ne sont pas des critères suffisant pour évaluer une méthode dédiée au calcul parallèle ainsi qu'aux applications industrielles. Chaque résultat, ou observation, doit être confronté à son coût de calcul. On présente ici, des tests avec les deux codes de développement présentés dans le chapitre 4. En plus d'observer les performances numériques, on compare différentes stratégies de parallélisation incluant le choix de l'algorithme d'accélération d'Aitken, le choix de la méthode de résolution locale et le choix entre une stratégie à un niveau ou deux niveaux MPI.

Deux machines sont considérées. La machine A est une SGI ALTIX ICE disponible au CINES à Montpellier en France. Elle comprend 23040 coeurs répartis sur 2880 noeuds. Chaque noeud comprend deux processeurs Intel Quad-Core E5472 ou X5560. Dans nos travaux, cette machine est principalement utilisée pour le calcul intensif et, ici, pour la résolution des équations de Darcy $3D$ *avec le code FORTRAN décrit dans la partie 4.3. La machine B est une SGI Xeon Xe3400. Elle est constituée de 192 coeurs qui peuvent être* hyper-threaded *pour atteindre le nombre de 384 processus indépendant. Ces coeurs sont répartis sur 16 noeuds. Chaque noeud est constitué de deux processeurs Intel Xeon.*

On débute l'étude avec les cas de CFD $2D$*. La stratégie générale choisie et développée précédemment pour résoudre ce type de système linéaire est testée avec différents nombres de partitions et l'efficacité du préconditionnement est évaluée selon le nombre de partitions employées. Le temps passé dans chaque phase du processus de résolution est controlé.*

Le nombre d'itérations de la méthode itérative globale employée (GMRES) diminue

fortement. Malgré d'excellentes propriétés numériques de ARAS(q) sur ce type de problème, le temps total pour chaque exécution avec ARAS(q) est toujours supérieur au temps total de calcul avec RAS. En observant chaque phase, on s'aperçoit que, comme vu dans l'étude des coûts de calcul du chapitre 3, la consommation de temps s'effectue en majorité dans la phase de construction du préconditionneur avec l'algorithme 9. Le temps de résolution, quant à lui, diminue proportionnellement à la réduction du nombre d'itérations. Ceci atteste donc du fait que l'application de ARAS(q) est en terme de coût, du même ordre que l'application de RAS. Pour les tests sur les problèmes $3D$ *nous testerons différentes techniques pour réduire le coût de construction de ARAS(q).*

Lors d'une seconde campagne de tests, les bonnes propriétés de parallélisation de la méthode de Aitken-Schwarz et son extensibilité sur de grands systèmes linéaires issus de la discrétisation des équations de Darcy $3D$ *avec un champ de perméabilité aléatoire variant fortement, sont évaluées. Ces tests nous permettent de comparer les propriétés numériques de Aitken-Schwarz en méthode de résolution et de ARAS(q) sur ces problèmes* $3D$*.*
L'extensibilité de la méthode Aitken-Schwarz est illustrée avec le code FORTRAN sur la machine A pour un nombre de processus allant jusqu'à 2048 pour la résolution de problème de plusieurs centaines de millions d'inconnues.
De même que pour les problèmes $2D$ *industriels, l'extensibilité numérique de ARAS(q) pour un nombre q petit, est observée. Les mêmes symptômes concernant les temps de résolution avec RAS ou ARAS(q) sont observés. Deux approches sont considérées pour réduire les temps de construction de ARAS(q). L'une consiste à utiliser l'algorithme 8. Cette technique a été utilisée dans le code FORTRAN et a montré l'efficacité attendue. Au final, un gain systèmatique de 30% du temps total d'exécution à été observé sur la machine A. L'autre technique consiste, dans le cas où les résolutions locales sont effectuées par une méthode itérative, à utiliser une tolérance plus grande sur la méthode de résolution locale lors de la phase de construction de la base. Les gains sont présentés.*

Pour clore ce chapitre on effectue des tests sur un problème industriel de CFD $3D$*. Ces tests montrent les difficultés engendrées par le fait que la taille de l'interface artificielle devienne plus grande que la taille du problème global. L'extensibilité numérique de ARAS(q) observée pour un faible nombre de vecteurs de base q de la taille de l'interface, répond à nos attentes qui sont de produire un préconditionneur à deux niveaux, agissant sur l'interface, efficace pour les problèmes* $3D$*. Mais la forte divergence du processus ARAS(q) diminue les performances numériques de ARAS2(q). La SVD des traces de RAS sur l'interface, employée pour effectuer l'accélération, ne permet pas de représenter un nombre suffisant de modes. Une possibilité serait de normaliser les vecteurs solutions sur l'interface avant d'effectuer la SVD. En faisant cela on force les modes les plus forts à être du même ordre, et on donne plus de poids à d'autre modes qui, avant étaient vus comme du bruit.*

The principal interest of our research is to provide a robust and efficient method for solving industrial linear systems presented in chapter 1 and proposed by a CFD company, FLUOREM. The objective of this chapter is to highlight the numerical performances of the ARAS preconditioner (see chapter 3) on the industrial cases and on large sparse linear systems. But the numerical performances are not sufficient criteria to evaluate a method dedicated to parallel computation and to industrial software. Each results, or observations, need to be compared to the computational costs involved in the solution of the linear system. The tests are performed with the two development codes discussed in chapter 4. Then, in addition to the observations of the numerical performances, we propose to discuss different strategies of parallelization including the choice of algorithm to perform the Aitken acceleration, the choice of the local solver and the choice between a one-level or two-level MPI implementation.

Two machines are considered in the following studies. Machine A is the SGI ALTIX ICE machine available at CINES in Montpelier, France. It consists in 23040 cores spread over 2880 nodes. Each node consists in two processors Intel Quad-Core E5472 or X5560. This machine is essentially used, in our work, for intensive computation and here for solving the $3D$ Darcy equations with the FORTRAN code described in section 4.3. Machine B is an SGI Xeon Xe3400. It consists in 192 cores which can be hyper-threaded to reach 384 independent processes. Those cores are spread among 16 nodes. Each node consists in two Intel Xeon processors.

A first study on $2D$ CFD cases is proposed. The general strategy chosen to solve this kind of linear system is tested with different numbers of partitions and the efficiency of the preconditioner is discussed regarding to the number of partitions chosen. A study of the time spent in each phase of the solution is proposed. Then we illustrate good parallelism of the Aitken-Schwraz method and its scalability on large linear systems coming from the $3D$ Darcy equations with strong variability of the permeability field. Those tests allow us to compare the numerical properties of an Aitken-Schwarz solver and an ARAS preconditioning technique on those $3D$ problems. Finally we run the PETSc implementation of ARAS on a $3D$ industrial linear system. Issues concerning the size of the interface compared to the size of the global operator are discussed.

5.1 $2D$ CFD test cases

In this section we study the behaviour of the Restricted Additive Schwarz preconditioner and its enhancement by Aitken on $2D$ industrial CFD cases. We selected two cases of the same size from the FLUOREM test cases. Table 5.1 presents the main characteristics of the matrices, while Figure 5.1 shows the sparse profile of the matrices. The notable difference is the difference between the number of non-zero elements in each matrix. It is because, in the first one, the CASE_005 FR02, the turbulence is frozen whereas it is taken into account in the CASE_006 PR02. For

each case we run the code written in the PETSc framework on machine B, on several partitions and set up a table with information about time consumption and convergence for each run. For each run, the solver is a GMRES method left preconditioned by a RAS or ARAS(q) preconditioner. Each sub-system is solved by a direct method (LU). After presenting the behaviour of RAS on the linear system, we select two situations, one where the RAS solver converges and one where it diverges. Then we present the characteristics of the Aitken Acceleration in each case and discuss the numerical performances. From a practical point of view, we need to evaluate the performance of the code, looking at the key phases of the solution of the cases: the preconditioner computation and the solution time. A study will be presented taking into account the parameters used.

case ID	*order*	*dim*	*nn*	*nnz*
FR02	161 070	$2D$	23 010	5 066 996
PR02	161 070	$2D$	23 010	8 185 136

Table 5.1: Main features of the linear systems with *order* the size of the matrix with real coefficients, *dim* the dimension of the problem, *nn* is the number of mesh nodes, *nnz* is the number of non-zero elements in the matrix

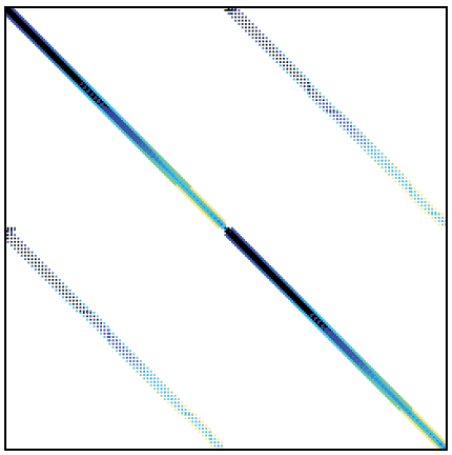

Figure 5.1: Profile of the matrice CASE_005 FR02 and CASE_006 PR02.

5.1.1 CASE_005 FR02

5.1.1.1 About RAS on CASE_005 FR02

Figure 5.2 presents the numerical behaviour of the RAS used as a solver (top) or as a preconditioner (bottom). The domain decomposition is performed by partitioning the operator with PARMETIS with $p \in \{2,\ 4,\ 6,\ 8,\ 12,\ 24\}$ partitions and an algebraic overlap equal to two.

As a Richardson process, the RAS method with overlap equal to two, converges linearly only for $p = 2$. Otherwise it diverges but the divergence is still linear. Nevertheless, an unexpected phenomena appears. Although increasing the number of partitions usually deteriorates the convergence of the RAS method, at least for a regular splitting, the divergence can be stronger here for a small number of partition than for a large number. Actually, the divergence is stronger for eight and twelve partitions than for twenty four partitions. And it is also stronger for four partitions than for six. Those variations in the convergence behaviour of the RAS preconditioner, should be the consequence of partitioning from the graph theory with a method such as METIS. There is no possibility to ensure that the partition we produced leads to have a good domain decomposition for the Schwarz algorithm.

Figure 5.3 presents the same tests with an overlap increased to four. The RAS iterative process converges for $p \in \{2,\ 4,\ 6\}$ and the process with six partitions converges faster that the one with four partitions.

As a preconditioner of a GMRES method, the RAS technique is efficient, even if the corresponding iterative process diverges. Meaning that, since the local operators are non-singular, any partitioning leads to an efficient preconditioner. Obviously, the number of partitions affects the convergence rate of the GMRES preconditioned by RAS. But the linear increasing of the number of partitions does not linearly increase the number of iterations of GMRES. More precisely, we observe that the gap between the number of iterations for $p = 2$ and $p = 4$ is larger than the one between $p = 4$ and $p = 6$. For this case four groups show up: $p = 2$, $p \in \{4,\ 6\}$, $p \in \{8,\ 12\}$, $p = 24$, for an overlap equal to two or four.

In the following, we choose two situations and we apply the ARAS preconditioner. First we study a convergent situation choosing $p = 2$ and an overlap equal to two. After we propose the same analyse but on a divergent case, choosing $p = 8$ with the same overlap.

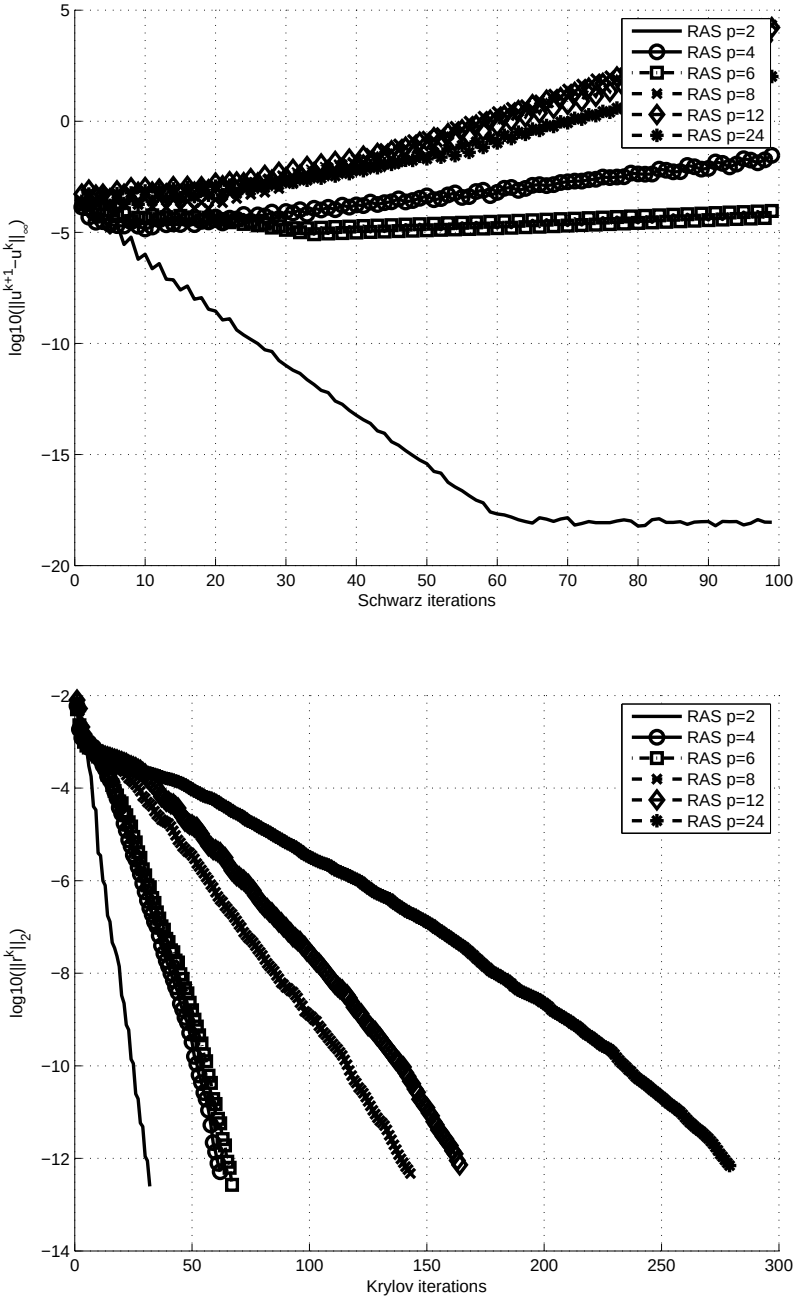

Figure 5.2: Solving the CASE_005 FR02 with RAS on p partitions (top) used as a solver (bottom) or a preconditioner for GMRES with overlap 2.

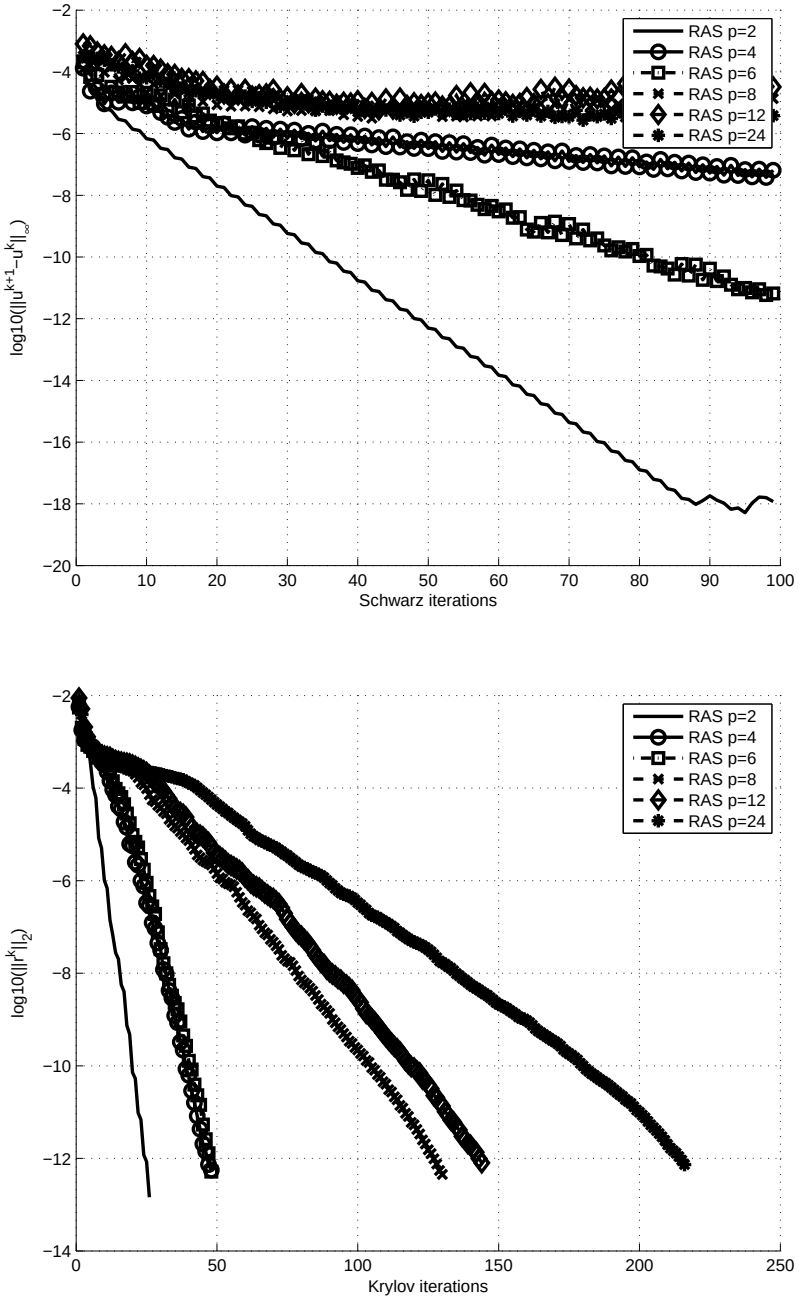

Figure 5.3: Solving the CASE_005 FR02 with RAS on p partitions (top) used as a solver (bottom) or a preconditioner for GMRES with overlap 4.

5.1.1.2 ARAS for a convergent case on CASE_005 FR02

We consider a set of options for which the RAS iterative process converges. Let us take $p = 2$ and an overlap equal to two. The iterative process needs 40 iterations to converge with a tolerance $\varepsilon = 10^{-10}$ whereas the GMRES(RAS) needs 25 iterations to converge.
We apply Algorithm 9 to build both the base $\mathbb{U}_q$ and the error transfer operator on the coarse interface, $P_{\mathbb{U}_q}$, with three different values of $q \in \{10,\ 20,\ 40\}$.

Figure 5.4 shows the q largest singular values computed with SLEPSc from the q interface solutions of the RAS iterative process. From one computation to another, nearly the same first values are computed. We observe an interlacing of the singular values. This interlacing is characterized by the fact that the values are bounded by the first (the largest) value computed, which is nearly the same from one computation to another, and then decrease. The Singlular values spectrum varies from four orders, seven orders and thirteen orders. Due to the large variation of the spectrum while computing 40 singular values, we expect to have a complete acceleration for ARAS(40). There is no bend. All the singular values can be considered as significant.

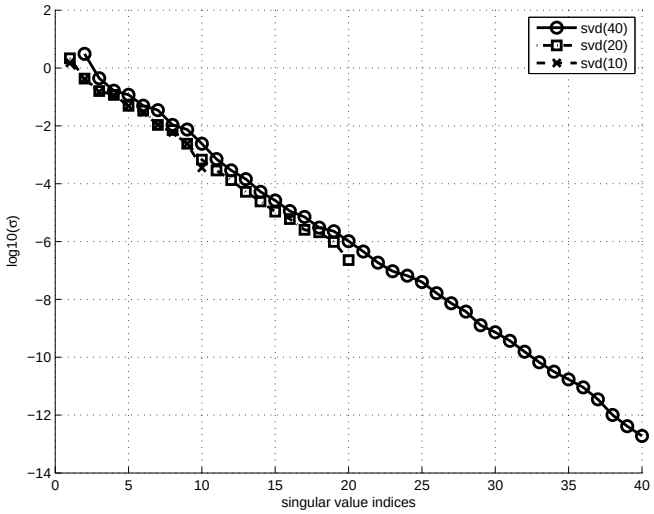

Figure 5.4: Singular values of the set of q interface's solutions of a RAS iterative process, for $p = 2$ and overlap 2.

Figure 5.5 shows the convergence history of both the Richardson process and the GMRES method with the RAS and ARAS(q) preconditioner. The ARAS(q) iterative processes converge faster than the RAS as expected. This enhancement is characterized by a starting error which is less than the initial one with RAS and then the linear convergence is retrieved. For the RAS iterative process the initial

error is around 10^{-4}. For $q = 10$ this error is close to 10^{-7}, gaining three orders of convergence, and for $q = 20$ the initial error is close to 10^{-10}, gaining six orders. For $q = 40$ we are close to a complete acceleration as expected since the error starts under 10^{-16} and stagnates to 10^{-18} after only four iterations.

The convergence of the GMRES preconditioned by ARAS(q) presents the same characteristics as for the RAS iterative process: the principal effect of the acceleration occurs at the beginning and after, the convergence behaviour of the Krylov iterative method is retrieved. We remark that the GMRES(ARAS(q)) needs one preliminary iteration before the convergence is effective. This can be explained by the fact that an Aitken's acceleration needs, at least, two initial guesses to be applied.

Figure 5.6 shows the convergence curves of both the Richardson process and the GMRES method with the RAS2 and ARAS2(q) preconditioners (multiplicative forms of RAS and ARAS(q)). The RAS2 iterative process converges two times as fast as the RAS. This behaviour was expected due to the fact than one iteration of RAS2 is equivalent to two iterations of RAS. We notice that every multiplicative process has an initial error which corresponds to the second error of the regular iterative processes.
For the GMRES, the number of iterations to reach the convergence is also divided by two. The acceleration occurs at the first iteration. We also remark that, after the acceleration, the convergence behaviour of the classical method is retrieved as for the regular form.

Figure 5.7 shows the eigenvalues of the three approximations of the error transfer operator we computed.
The convergence of each ARAS(q) process can be evaluated by studying the spectrum of the computed error transfer operator. Under certain assumptions, according to section 3.3, the efficiency of the acceleration can be evaluated by observing the eigenvalue spectrum of the matrix $P_{\mathbb{U}_q}$. If the eigenvalues of the approximated error transfer operator match the largest eigenvalues of P, then the spectral radius of the Aitken RAS iterative process is bounded by the smallest eigenvalue of $P_{\mathbb{U}_q}$ (see section 3.3).
Here, the assumptions cannot be applied. But, regarding the results presented in Figure 5.6, we assume that the operator obtained for $q = 40$ is a good and reliable approximation of the exact error transfer operator. Then we compare its eigenvalues to those of the operators computed with $q = 10$ and $q = 20$.
For this case all the eigenvalues have a module less than one, characterizing the convergence of the RAS iterative process on the interface. We observe that the largest real part and the largest imaginary part are first computed. The more values you compute, the more groups of eigenvalues far from zero are computed. We also observe that the eigenvalues of each operator do not exactly match, but extreme zones are located.

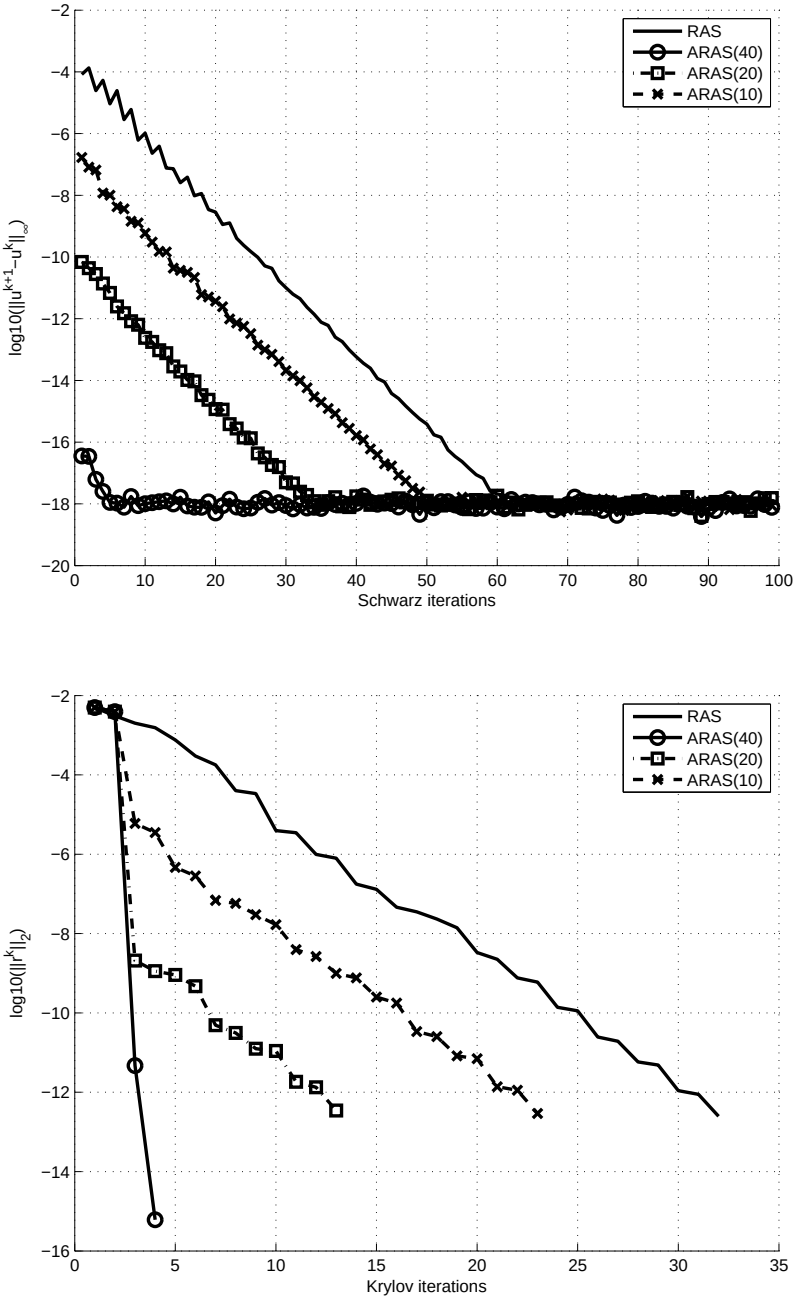

Figure 5.5: Solving the CASE_005 FR02 with RAS enhanced by Aitken on 2 partitions with overlap 2 in its regular form, (top) used as a solver (bottom) or a preconditioner for GMRES.

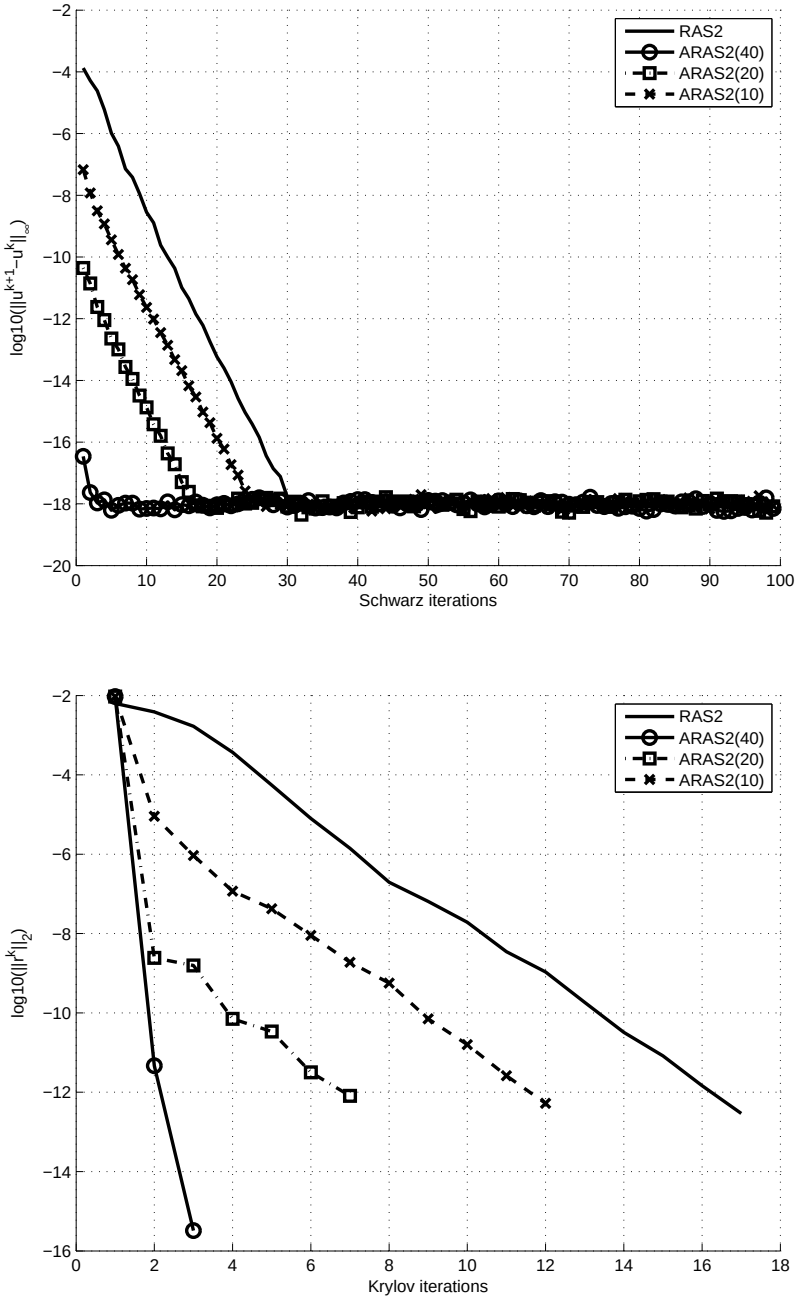

Figure 5.6: Solving the CASE_005 FR02 with RAS enhanced by Aitken on 2 partitions with overlap 2 in its multiplicative form, (top) used as a solver (bottom) or a preconditioner for GMRES.

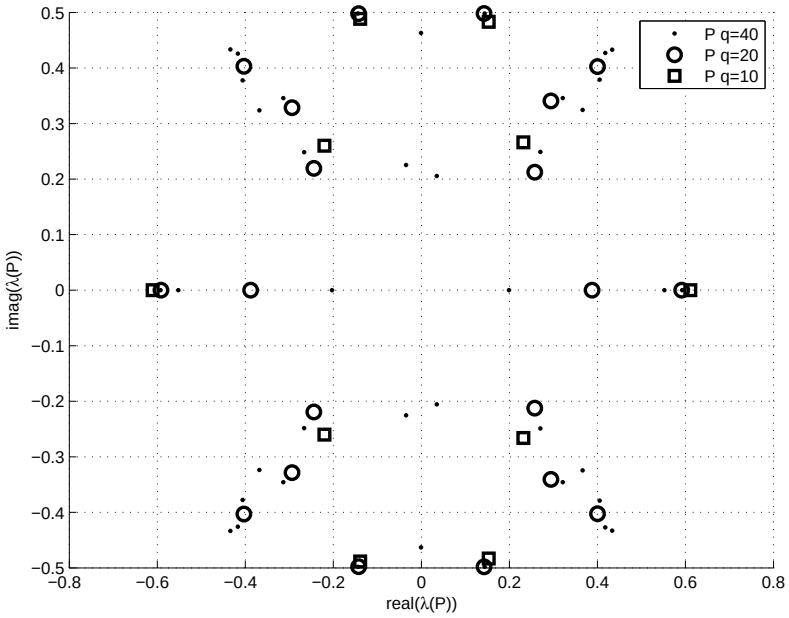

Figure 5.7: Eigenvalues of P in the $\mathbb{U}_q$ base computed from q iterations of RAS with $p = 2$ and overlap 2.

5.1.1.3 ARAS for a divergent case on CASE_005 FR02

We consider a set of options for which the RAS iterative process diverges. Let us take $p = 8$ and an overlap equal to two. The iterative process first stagnates with an error around 10^{-4}, and after twenty iterations starts to diverge. It goes from 10^{-4} to 10^4 in nearly eighty iterations. When used as a preconditioner, the RAS method makes the GMRES converges in 140 iterations.
We apply Algorithm 9 to build both the base $\mathbb{U}_q$ and the error transfer operator on the coarse interface $P_{\mathbb{U}_q}$ with four different values of $q \in \{30,\ 60,\ 90,\ 120\}$.

Figure 5.8 shows the q largest singular values computed with SLEPSc from the q interface solutions of the RAS iterative process. The more singular values we compute, the more we exhibit a bend. The bend indicates that the set of data we study can be explained by the largest singular values before and on the bend. The others can be used to explain more accurately the data set but can also add noise. The more singular values we take, the more information we explain since the order of magnitude increases. The spectrum computed with 60 vectors seems to begin to explain sufficiently the solution on the interface contrary to the singular values computed from 30 vectors which has a poor magnitude and no sign of bend.
However, the interlacing between singular values is not observed. In fact, the largest singular values of each snapshot do not have the same magnitude. The magnitude

is amplified while increasing the number q of interface solutions on which the SVD is performed. Then the criterion given in Algorithm for selecting the number of vectors to keep in order to build the error transfer operator in the base arising from the SVD of the interface's solutions should not be a sufficient criterion, since the difference of magnitude between values will be too high.

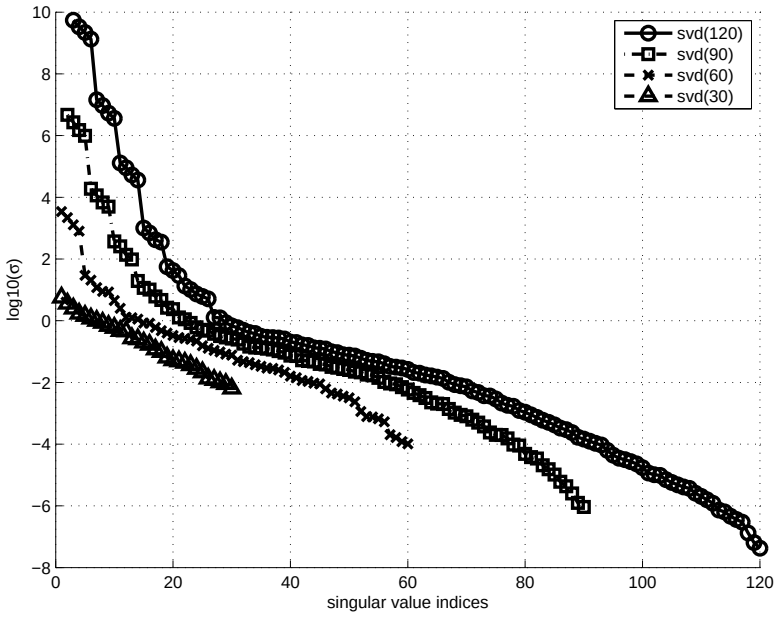

Figure 5.8: Singular values of the set of q interface's solutions of a RAS iterative process, for $p = 8$ and overlap 2.

Figure 5.10 shows the convergence history of both the Richardson process and the GMRES method with the RAS and ARAS(q) preconditioner.
The top of Figure 5.10 shows that the ARAS(q) iterative processes diverge. The acceleration does not annihilate the divergence. For $q \in \{90,\ 120\}$, the divergence is amplified. For $q \in \{30,\ 60\}$, the divergence is diminished, and the reduction of the divergence is more significant for $q = 60$. The first iterations seem to converge linearly instead of stagnating, and after nine iterations start to diverge. The stronger divergence may lead to too large an amount of noise resulting from the SVD.
The bottom of Figure 5.10 shows that the convergence of the GMRES preconditioned by ARAS(q) is better when q increases. All the ARAS(q) preconditioners are more efficient than the RAS preconditioner, even for a small number of singular values computed.

Figure 5.11 shows the convergence curves of both the Richardson process and

the GMRES method with the RAS2 and ARAS2(q) preconditioners (multiplicative form of RAS and ARAS(q)). Each multiplicative form diverges more strongly than the regular form. This behaviour was expected due to the fact that one iteration of RAS2 is equivalent to two iterations of RAS.
For the GMRES, the number of iterations to reach the convergence is less than that for preconditioning with the regular form, but it is not divided by two. Then, considering that each iteration costs two iterations of RAS, the use of ARAS2 is not really interesting.

Figure 5.9 shows the eigenvalues of the four approximations of the error transfer operator we computed.
We observe that the more singular values we keep, the more extreme the zone we describe becomes. The real part is less than one, but the imaginary part can be greater than one. This result was expected since the RAS iterative process diverges.

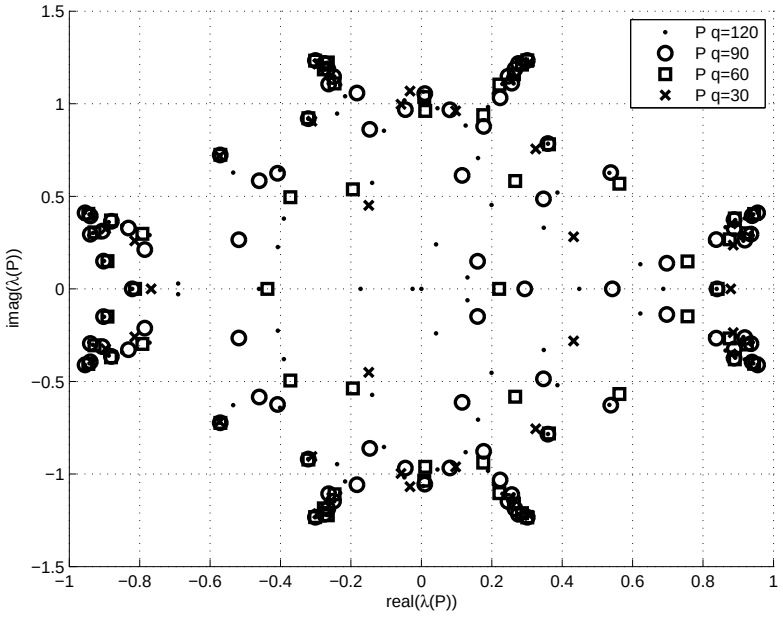

Figure 5.9: Eigenvalues of P in the $\mathbb{U}_q$ base computed from q iterations of RAS with $p = 8$ and overlap 2.

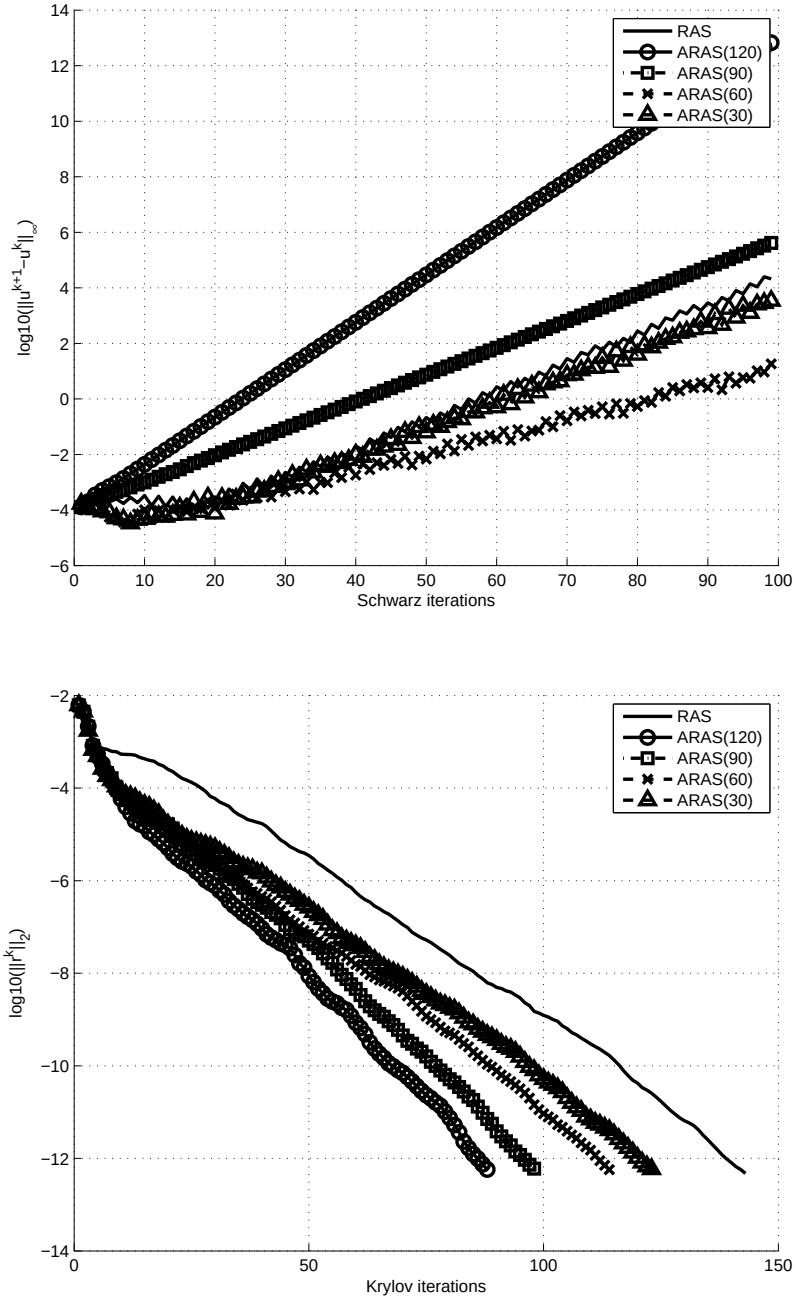

Figure 5.10: Solving the CASE_005 FR02 with RAS enhanced by Aitken on 8 partitions with overlap 2 in its regular form, (top) used as a solver (bottom) or a preconditioner for GMRES.

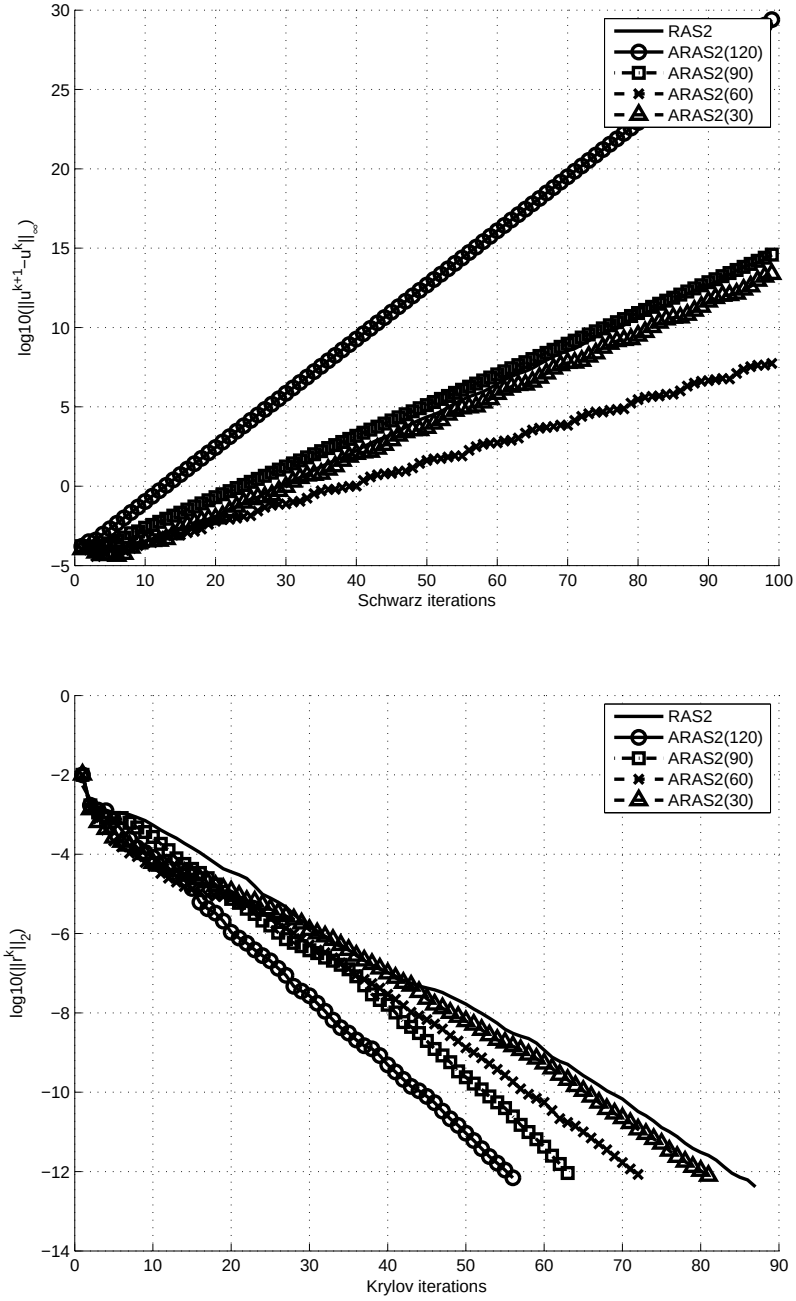

Figure 5.11: Solving the CASE_005 FR02 with RAS enhanced by Aitken on 8 partitions with overlap 2 in its multiplicative form, (top) used as a solver (bottom) or a preconditioner for GMRES.

5.1.2 CASE_006 PR02

This case is similar to the previous test. The pattern of the matrix CASE_006 PR02 is the same as the pattern of the matrix CASE_005 FR02 plot in Figure 5.1. The difference between the two cases is that here the turbulence is not frozen. In a sense, this case can be viewed as a numerically more difficult case than the previous one but with the same characteristics in terms of computational issue (storage, partitioning).

5.1.2.1 About RAS on CASE_006 PR02

Figure 5.12 presents the numerical behaviour of the RAS used as a solver (top) or as a preconditioner (bottom). The domain decomposition is performed by partitioning the operator with PARMETIS with $p \in \{2,\ 4,\ 6,\ 8,\ 12,\ 24\}$ partitions and and algebraic overlap equal to two.
As a Richardson process, the RAS method with overlap equal to two converges linearly only for $p = 2$. For $p \in \{4,\ 6\}$, it converges and, after 35 iterations, starts to diverge, otherwise it diverges. However, the divergence is still linear and we can see a pattern which is repeated linearly. Nevertheless, the same phenomena as for the CASE_005 FR02 appears. Although increasing the number of partitions usually deteriorates the convergence of the RAS method, at least for a regular splitting, the divergence can be stronger here for a small number of partitions than for a large number. The divergence is stronger for eight and twelve partitions than for twenty four partitions, and it is also stronger for four partitions than for six. The strength of the divergence for one number of partitions compared to another can be ordered the same way as for the previous case without turbulence.
This should be the consequence of partitioning from the graph theory with a method such as METIS. There is no way to ensure that the partition we produced, leads to a good domain decomposition for the Schwarz algorithm.

Figure 5.13 presents the same tests with an overlap increased to four. The RAS iterative process converges for $p \in \{2,\ 4,\ 6\}$ and the process with six partitions converges faster that the one with four partitions after 50 iterations.

The influence of the overlap is easier to point out for this case with turbulence. It is clear that, even if the RAS with six partitions converges faster than the method with four partitions, the convergence rate of the method with four partitions for the 30 first iterations is better than the one with six partitions.

As a preconditioner of a GMRES method, the preconditioner is efficient even if the corresponding iterative process diverges. This means that, since the local operators are non-singular, any partitioning leads to an efficient preconditioner. Obviously, the number of partitions affects the convergence rate of the GMRES preconditioned by RAS. Moreover, the linear increasing of the number of partitions does not linearly increase the number of iteration of GMRES. More precisely, we observe that the gap between the number of iterations for $p = 2$ and $p = 4$ is larger

than the one between $p = 4$ and $p = 6$. For this case four groups show up: $p = 2$, $p \in \{4,\ 6\}$, $p \in \{8,\ 12\}$, and $p = 24$, for an overlap equal to two or four. Finally, we nearly made the same observations as for the CASE_005 FR02 concerning the convergence for each partitioning. We will make the same study we have done before applying the ARAS preconditioning techniques with two or eight partitions.

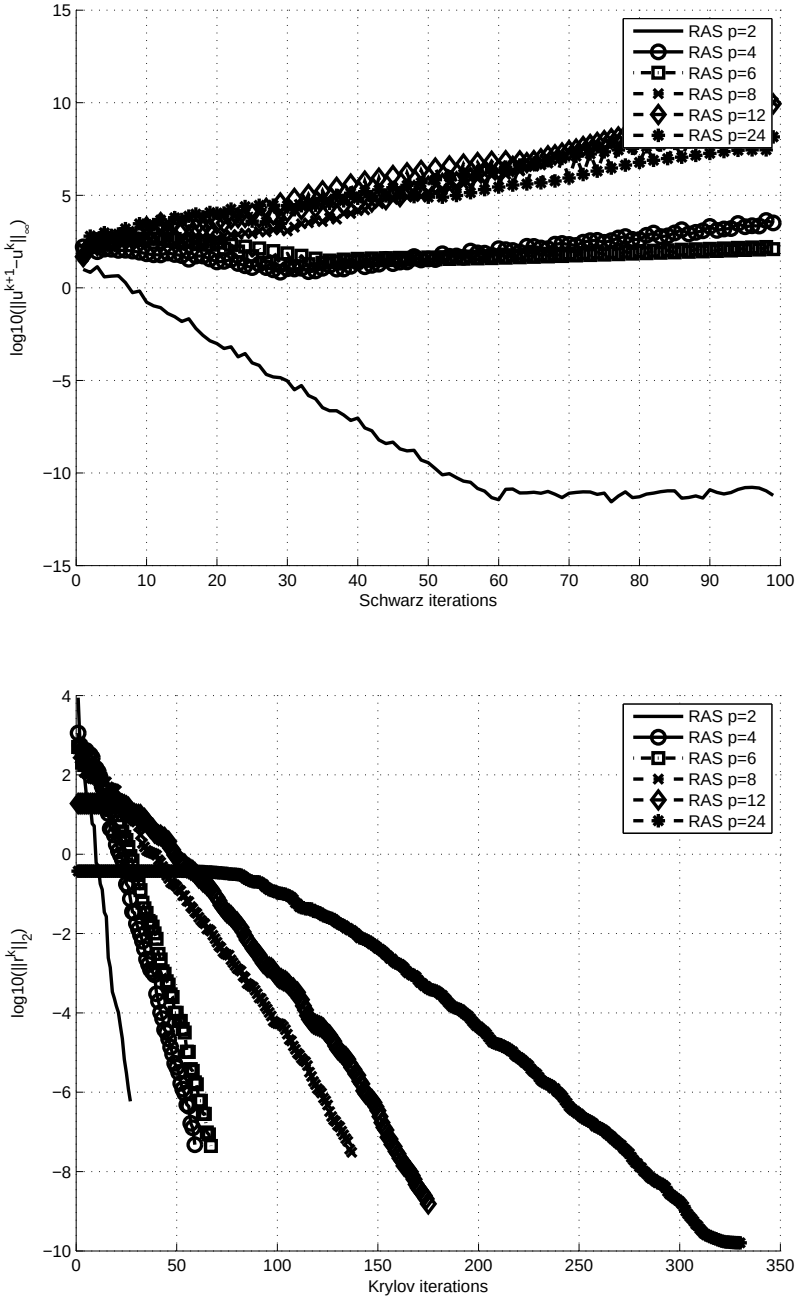

Figure 5.12: Solving the CASE_006 PR02 with RAS on p partitions (top) used as a solver (bottom) or a preconditioner for GMRES with overlap 2.

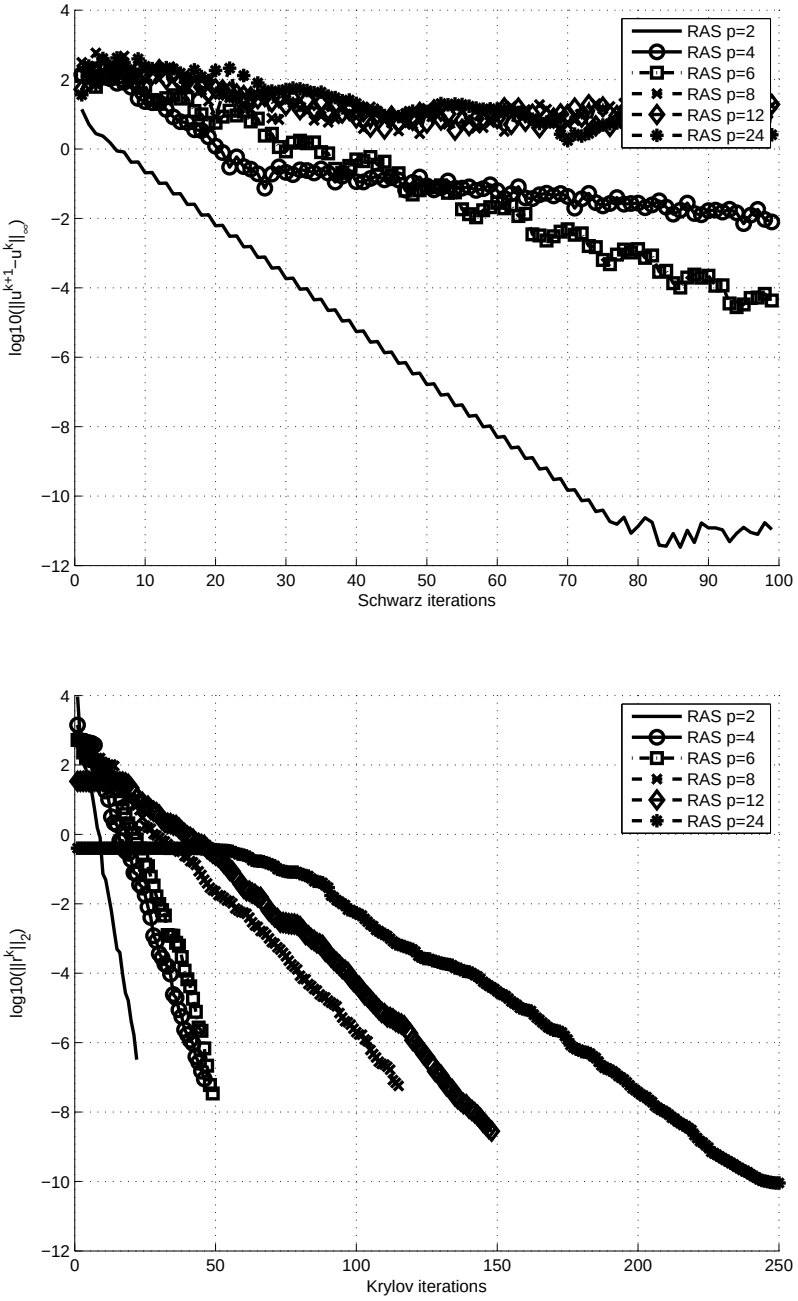

Figure 5.13: Solving the CASE_006 PR02 with RAS on p partitions (top) used as a solver (bottom) or a preconditioner for GMRES with 4.

5.1.2.2 ARAS for a convergent case on CASE_006 PR02

We consider a set of options for which the RAS iterative process converges. Let us take $p = 2$ and an overlap equal to two. The iterative process needs 52 iterations to converge with a tolerance $\varepsilon = 10^{-10}$, which is 12 iterations more than for the case without turbulence, whereas the GMRES(RAS) needs 25 iterations to converge, which is the same number of iterations as for the case without turbulence.
We apply Algorithm 9 to build both the base $\mathbb{U}_q$ and the error transfer operator on the coarse interface $P_{\mathbb{U}_q}$ with three different values of $q \in \{10,\ 20,\ 40\}$.

Figure 5.4 shows the q largest singular values computed with SLEPSc from the q interface solutions of the RAS iterative process. From one computation to another, nearly the same first values are computed. As for the CASE_005 FR02, there is no bend. All the singular values can be considered as significant.

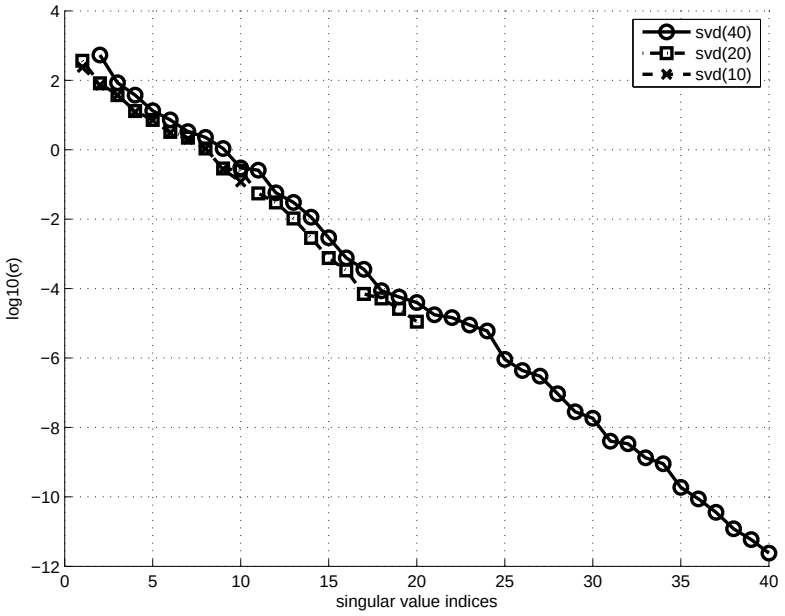

Figure 5.14: Singular values of the set of q interface's solutions of a RAS iterative process, for $p = 2$ and overlap 2.

Figure 5.15 shows the convergence history of both the Richardson process and the GMRES method with the RAS and ARAS(q) preconditioner. The resulting ARAS(q) iterative processes have an initial error which is less than the one of RAS. It characterizes the acceleration but contrary to the CASE_005 FR02, no linear convergence is retrieved after the acceleration. Each process has a different characteristic. For $q = 10$, the number of singular values computed seems to be insufficient since the first error is only one order less and the process diverges. For

$q = 20$, the first error has significantly decreased to around 10^{-5} but after diverges linearly repeating the same pattern. The divergence is slow. For $q = 40$, the first acceleration leads to an error around 10^{-10} and then stagnates near the same error for which the RAS stagnates. Thus, we can consider that this acceleration is the best acceleration we can have.

The convergence of the GMRES preconditioned by ARAS(q) for $q \in \{20,\ 40\}$ presents the same characteristics as for the RAS iterative process. The principal effect of the acceleration occurs at the beginning. For $q = 10$ the process starts with nearly the same residuals and after three applications proceeds to an acceleration. The divergence of the process seems to create a delay of the acceleration, after which the convergence behaviour of the Krylov iterative method is retrieved.

Figure 5.16 shows the convergence curves of both the Richardson process and the GMRES method with the RAS2 and ARAS2(q) preconditioners (multiplicative form of RAS and ARAS(q)). The RAS2 iterative process does not converge two times faster than the RAS.
For the GMRES, the number of iterations to reach the convergence is also reduced but not divided by two. The acceleration occurs at the first iteration. We also remark that, after the acceleration, the convergence behaviour of the classical method is retrieved as for the regular form.

Figure 5.17 shows the eigenvalues of the three approximations of the error transfer operator we get. Regarding the previous results, we assume that the operator obtained for $q = 40$ is a good and reliable approximation of the exact error transfer operator. Then we compare its eigenvalues to those of the operators computed with $q = 10$ and $q = 20$. For this case all the eigenvalues have a module less than one, characterizing the convergence of the RAS iterative process on the interface. We observe that the extreme values of the good approximation of P do not match those for $q = 10$. For $q = 10$, extreme values on the left and right side of the spectrum are far away from those for $q = 40$. This difference can be an explanation of the strong divergence of ARAS(10) since the operator with $q = 10$ does not represents the good extreme values of P and values which are greater than the one expected. For $q = 20$, the values are closer to the values expected but still don't match. Then the reduction of the spectrum by the Aitken acceleration can be efficient but with a disturbance of the iterative process. This is why the convergence is not retrieved.

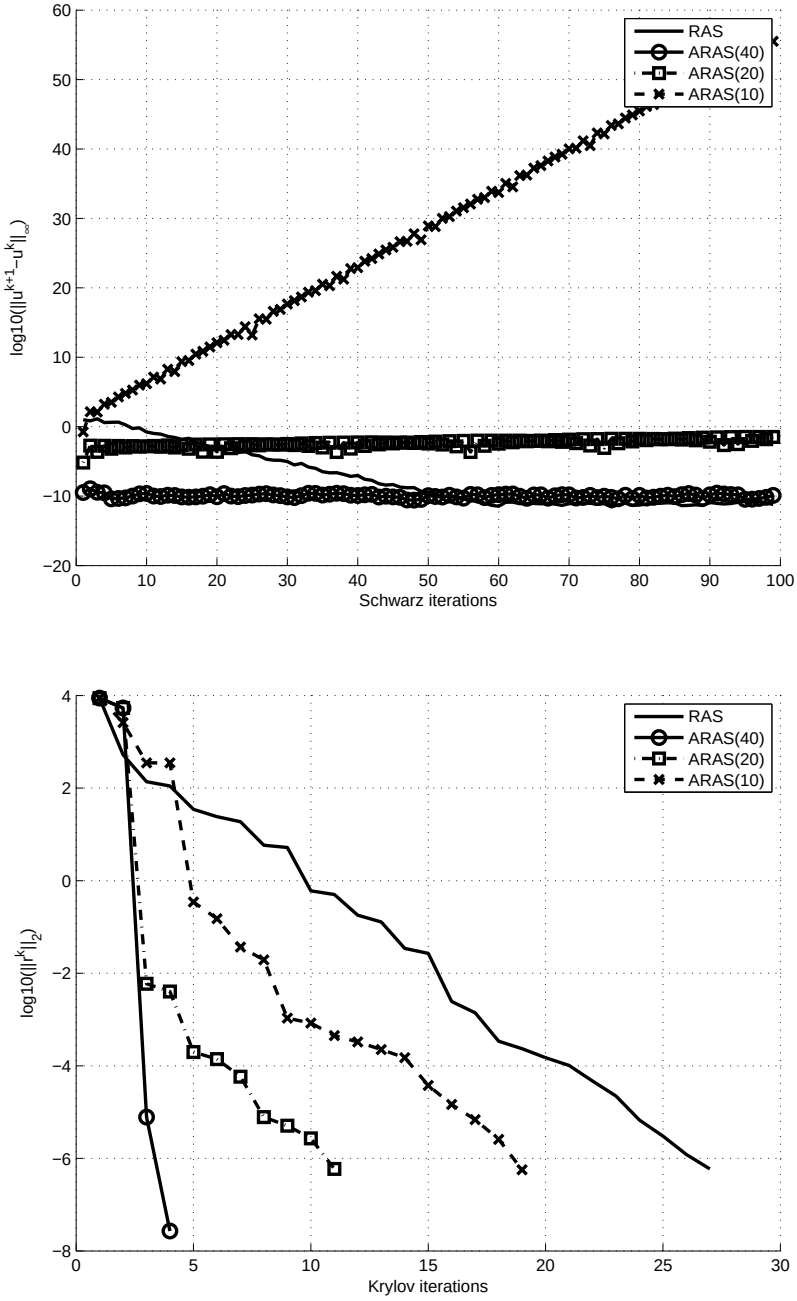

Figure 5.15: Solving the CASE_006 PR02 with RAS enhanced by Aitken on 2 partitions with overlap 2 in its regular form, (top) used as a solver (bottom) or a preconditioner for GMRES.

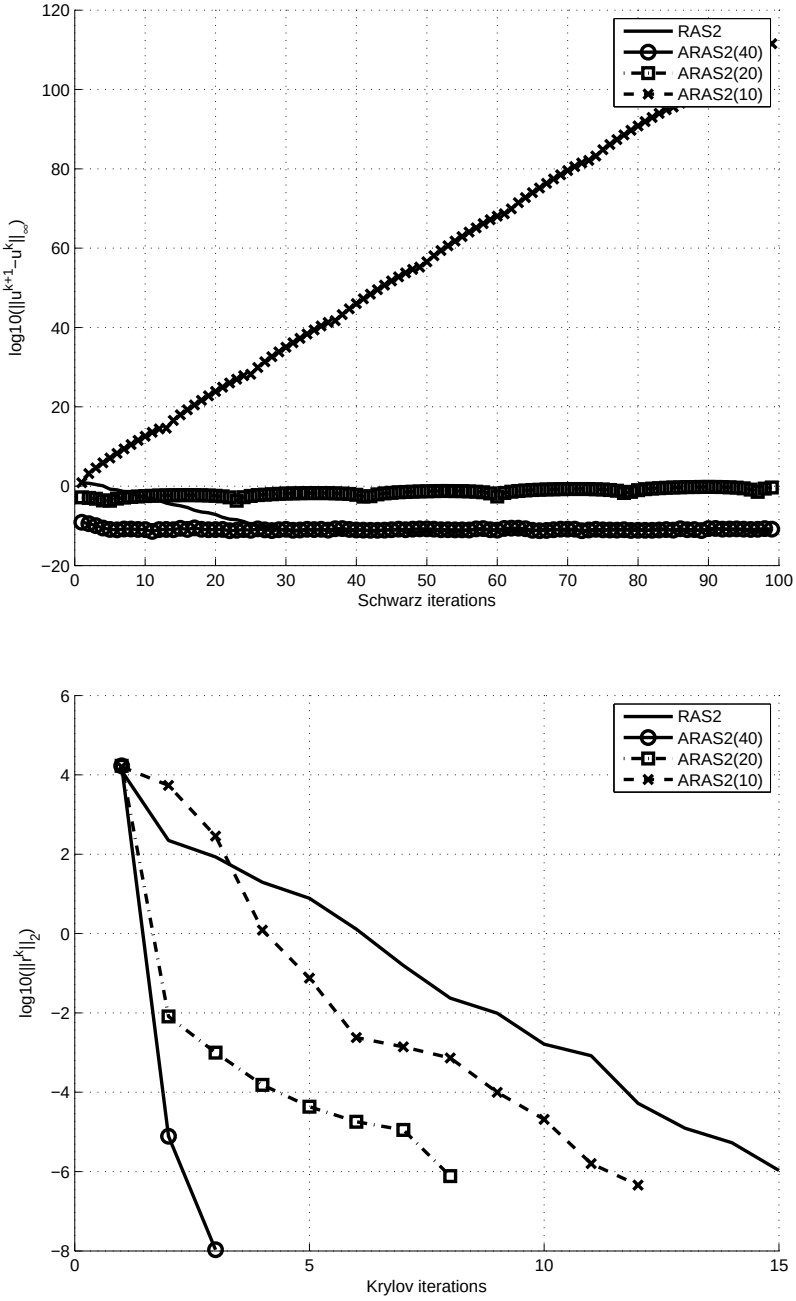

Figure 5.16: Solving the CASE_006 PR02 with RAS enhanced by Aitken on 2 partitions with overlap 2 in its multiplicative form, (top) used as a solver (bottom) or a preconditioner for GMRES.

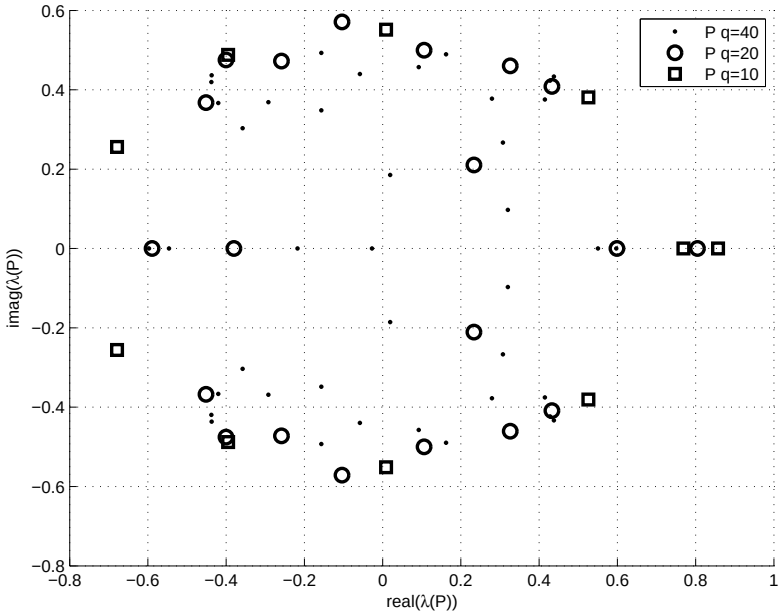

Figure 5.17: Eigenvalues of P in the $\mathbb{U}_q$ base computed from q iterations of RAS with $p = 2$ and overlap 2.

5.1.2.3 ARAS for a divergent case on CASE_006 PR02

We consider a set of options for which the RAS iterative process diverges. Let us take $p = 8$ and an overlap equal to two. The iterative process first converges slowly, and after twenty iterations starts to diverge. It goes from 10^2 to 10^{10} in nearly eighty iterations. We remark that those values are the same as for the non-turbulent case, shifted by six orders. When used as a preconditioner, the RAS method makes the GMRES converge in 138 iterations which is close to the number of iterations for the non-turbulent case.
We apply Algorithm 9 to build both the base $\mathbb{U}_q$ and the error transfer operator on the coarse interface $P_{\mathbb{U}_q}$ with four different values of $q \in \{30,\ 60,\ 90,\ 120\}$.

Figure 5.18 shows the q largest singular values computed with SLEPSc from the q interface solutions of the RAS iterative process. The more singular values we compute, the more we exhibit a bend. The spectrum is the same as for the non-turbulent case with eight partitions, but shifted by two orders. As for the non-turbulent case, we do not observe the interlacing of the singular values. The choice of the number of vectors to keep to build the error transfer operator, based on the value of the singular values, should not be sufficient as explained for the divergent case on CASE_005 FR02.

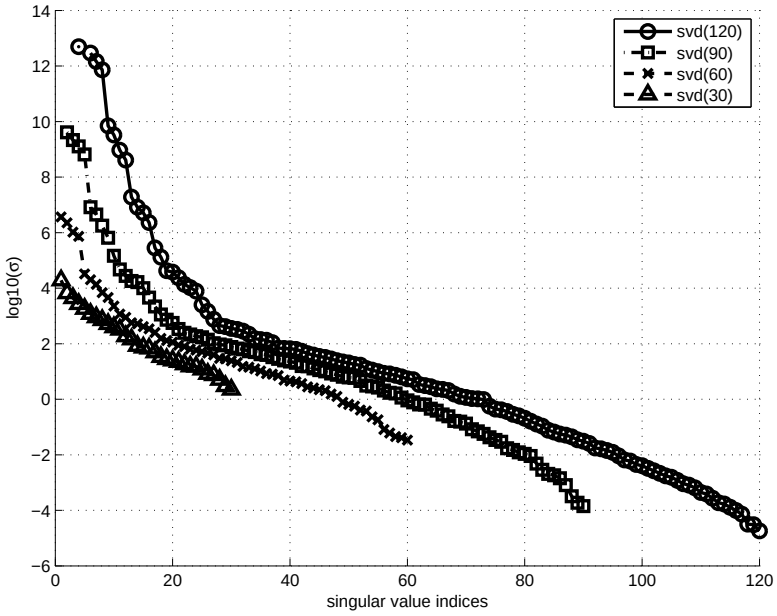

Figure 5.18: Singular values of the set of q interface's solutions of a RAS iterative process, for $p = 8$ and overlap 2.

The top of Figure 5.20 shows the convergence history of the Richardson process with the RAS and ARAS(q) preconditioner. The ARAS(q) iterative processes diverge. The acceleration does not annihilate the divergence. Contrary to the non-turbulent case, the more we keep singular values, the more the ARAS(q) method diverges. The stronger divergence may cause too much noise resulting from the SVD. As previously, the way we choose which vectors to keep is not reliable when the RAS process diverges.
The bottom of Figure 5.20 shows the convergence history of the GMRES preconditioned by the RAS and ARAS(q) preconditioner. Effects of the acceleration are clearly delayed by the divergence of the iterative process. A look at the curve of ARAS(30) confirms that for the 50 first iterations of GMRES, the residual is smaller than for ARAS(60), ARAS(90) and ARAS(120) but at the end converges in 117 iterations. For ARAS(90) and ARAS(120) the acceleration occurs after 60 iterations leading to a convergence in 113 and 100 iterations. Every ARAS(q) preconditioner is better than the RAS preconditioner even for a small number of singular values computed.

The top of Figure 5.21 shows the convergence history of the multiplicative form of RAS and ARAS(q) denoted by RAS2 and ARAS2(q) used as solver. Each multiplicative form diverges more strongly than the regular form. This behaviour was expected due to the fact than one iteration of RAS2 is equivalent to two

iterations of RAS.

The bottom of Figure 5.21 shows the convergence history of the GMRES preconditioned by RAS2 and ARAS2(q). The number of iterations to reach the convergence is worse than,or close to, the convergence of RAS2. Then considering that each iteration costs two iterations of RAS and that the cost of building ARAS(q) is larger than building RAS, it is not a good strategy to use. The fact that the acceleration is not efficient in the multiplicative form, shows that the error transfer operator is approximated with too much noise due to the too strong divergence of the iterative process.

Figure 5.19 shows the spectrum of each error transfer operator we computed. We see that the more we keep singular values, the more extreme a zone we describe. The real part is less than one, for $q \in \{60,\ 90,\ 120\}$ but the imaginary part can be greater than one. For $q = 30$ we observe an extreme value greater than one in module on the left part of the spectrum. As for the convergent case with $q = 10$, there are not enough singular values kept for $q = 30$. This result was expected since the RAS iterative process diverges.

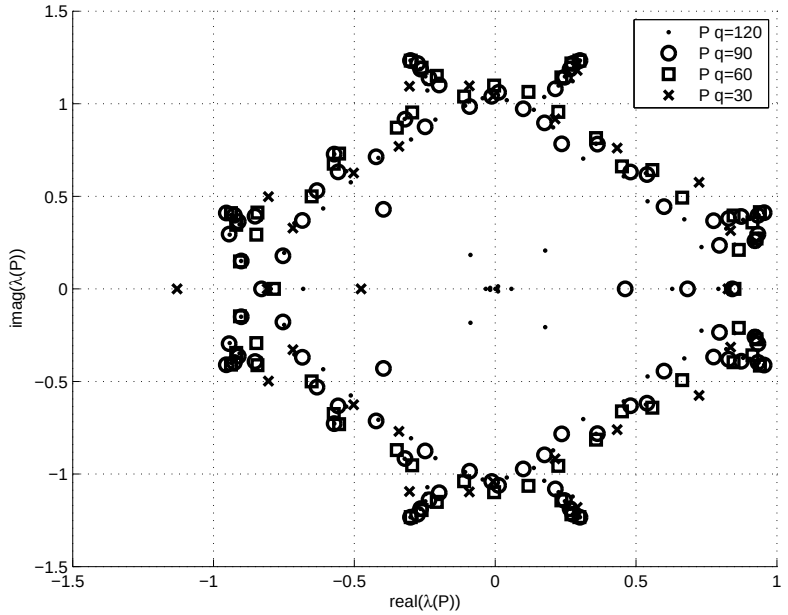

Figure 5.19: Eigenvalues of P in the $\mathbb{U}_q$ base computed from q iterations of RAS with $p = 8$ and overlap 2.

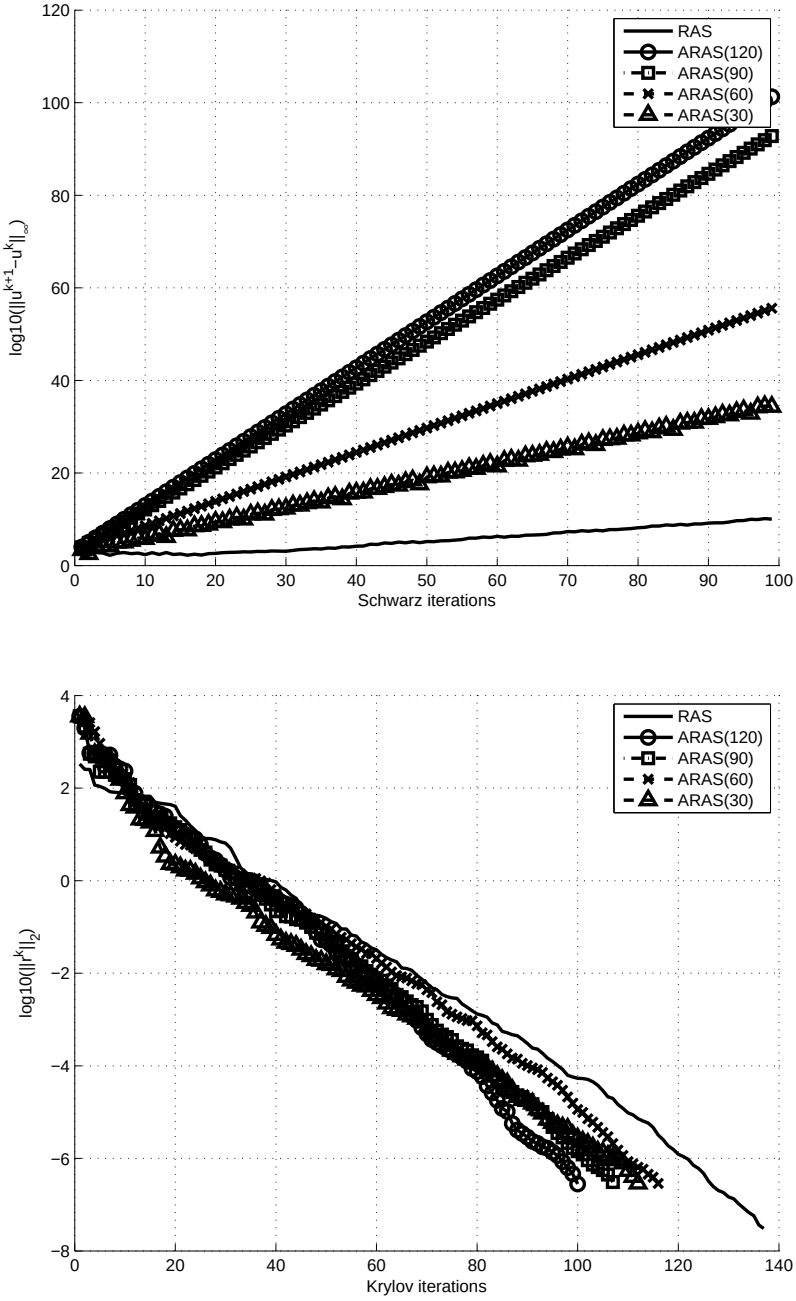

Figure 5.20: Solving the CASE_006 PR02 with RAS enhanced by Aitken on 8 partitions with overlap 2 in its regular form, (top) used as a solver (bottom) or a preconditioner for GMRES with (top).

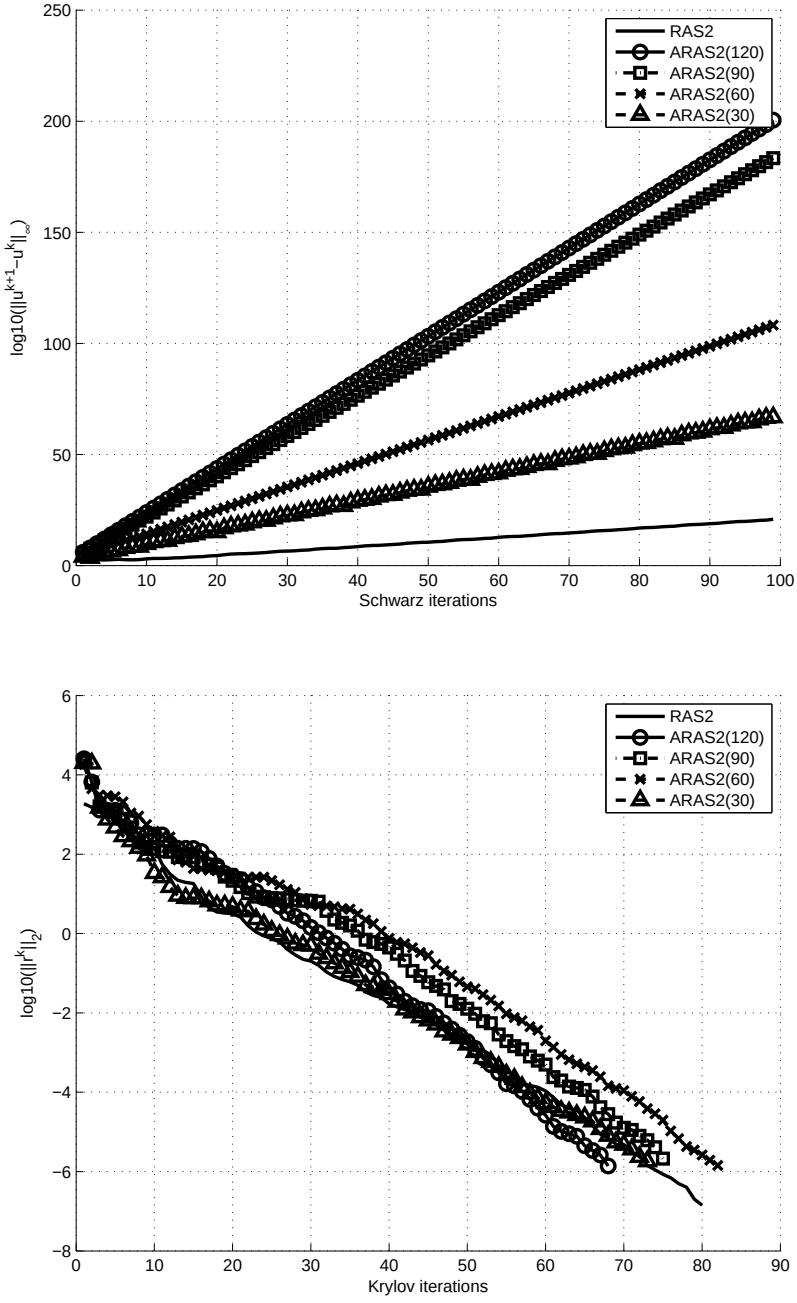

Figure 5.21: Solving the CASE_006 PR02 with RAS enhanced by Aitken on 8 partitions with overlap 2 in its multiplicative form, (top) used as a solver (bottom) or a preconditioner for GMRES.

5.1.3 Computing performances on $2D$ industrial cases

In this section we provide measures in terms of memory and time consumption on one of the $2D$ cases, the CASE_005 FR02. According to the numerical results, the convergent cases are similar in terms of iterations. For the divergent case, we saw that in the turbulent case it is not interesting to use the multiplicative form. A study on the non-turbulent case and the regular form of the preconditioners should be sufficient to discuss the efficiency of the implementation in this case.
Here, we chose to solve the local domain with a direct method. The method used is the LU factorization implemented in PETSc without option. A RCM ordering is used on local matrices to reduce the bandwidth.

Figure 5.22 presents the measure of the maximum local memory consumed during the run. The memory is practically all consumed by the local factorization. The global memory needed for the factorization stays constant as the number of processors increases. The maximum local memory decreases linearly as the number of processors increases. Due to the fact that the memory is allocated for the local factorization, the memory cost is nearly equal for ARAS(q) since $q \ll n$. The memory consumption can be mastered by a parallel distribution of the processor. Here we focus on the one-level MPI implementation but the concluding will be the same. The memory consumption is proportional to the number of processors involved in the factorization. Then, if a local factorization is too expensive, the number of processors can be increased.

Now, we observe the time spent in each part of the solution process of the code. We first measure the time spent in the local factorization which is the part of the time to build the RAS preconditioner. This time will be called "Factorization Time". We also measure the time spent in the building of the error transfer operator: "Build P Time". The sum of both times gives the time to build the ARAS(q) preconditioner. Finally we measure the time spent in the solution: "Solution Time".
Figure 5.23 shows that, as for the memory, the factorization time is reduced proportionally to the number of processors. This factorization time stays constant for each method. When q increases, the time to build P increases proportionally. This is a consequence of the arithmetic complexity calculated in section 3.6. Conversely, the solution time decreases while q increases. This reduction comes from the numerical behaviour and depends on the convergence of the preconditioned GMRES.
Figure 5.24 shows the time spent to build P and shows the solution time obtained for GMRES(RAS). A third measure consists of adding those times and comparing the sum to the solution time obtained for GMRES(RAS). If the ratio is less than one, then it is interesting to use ARAS(q) compared to RAS.
In all the runs done, none presents better time than the solution time with GMRES(RAS). For each q and for all the partitioning, the ratio between the

solution time and the reference time decreases while the ratio of the time to build P over the reference time increases.

Since the solution time with ARAS(q) is significantly smaller than RAS, it should be interesting to reuse the preconditioner if it is possible. The other point is that we must reduce the cost of building P. This should be done using Algorithm 8. Doing that, we can divide almost by two the time to build P, but without a guarantee of robustness.

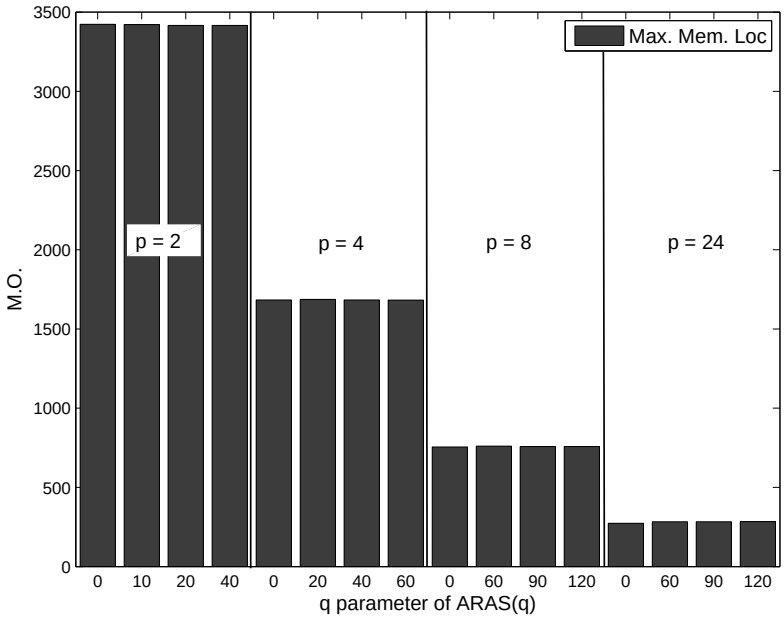

Figure 5.22: Maximum memory over processors while solving CASE_005 FR02 with GMRES preconditioned by ARAS(q), when $q = 0$ RAS is used.

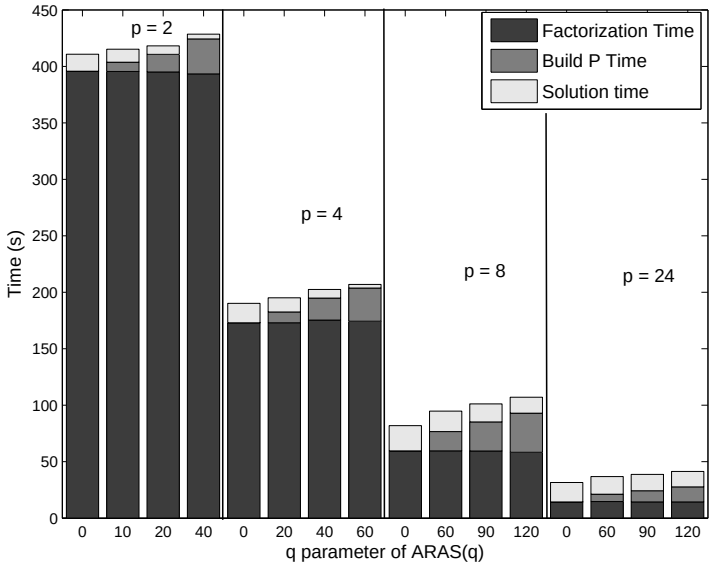

Figure 5.23: Time consumption while solving CASE_005 FR02 with GMRES preconditioned by ARAS(q), when $q = 0$ RAS is used.

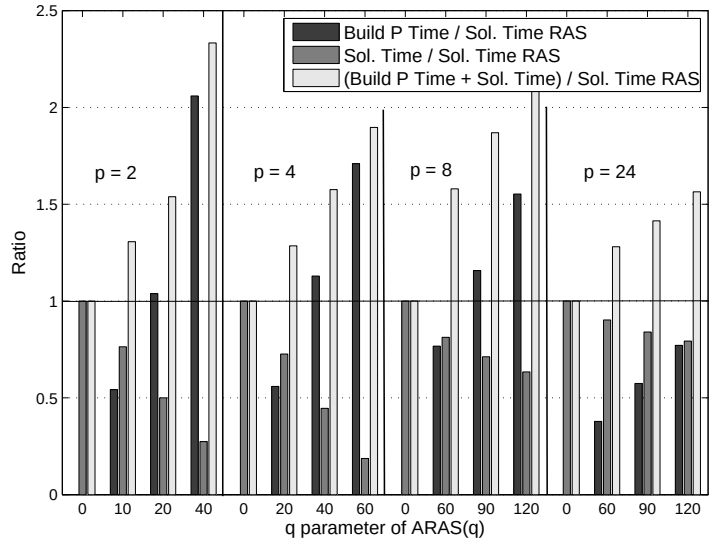

Figure 5.24: Ratio of time consumption for the building of P phase and the solution phase over the solution time spent for RAS while solving CASE_005 FR02 with GMRES preconditioned by ARAS(q), when $q = 0$ RAS is used.

5.2 $3D$ Darcy test cases

In section 4.3 we presented a code specially designed to solve large sparse linear system coming from the discretization of the $3D$ Darcy equation with strong variability of the permeability field. In this section we give some results obtained with this code, and we point out the different key points of the strategy chosen. Then we use the strategy we proposed to solve industrial problems. The goal is to show the numerical property of the preconditioner on this kind of problem and test the scalability of the code involved by the two-level MPI parallization coded in the PETSc framework.

5.2.1 Aitken Schwarz used as solver

5.2.1.1 Scalability

To be efficient we should keep the total number of macro-domains as small as possible. That is why we are not interested with strong scaling here. Performing strong scaling tests would consist in increasing the number of processors per macro-domains, which is equivalent to compute the strong scaling of the local solver.

To perform weak scaling tests without taking into account the weak scaling of the local solver, we fix the size of sub-domains and the number of sub-domains is increased. Results are provided in Table 5.2 where:

- Schwarz iteration: it corresponds to the computational time of local solution, and exchange of boundary conditions. The given time corresponds to the average time of the first five iterations.
- SVD: it is the time to compute the SVD of the first five snapshots. The SVD is computed on a single processor, so measurement includes communications.
- Aitken acceleration: includes time to compute the LU factorization of P and to compute the Aitken formula. This step is performed on a single processor. The measurement also includes communications.
- Number of iterations is the total number of Schwarz iterations performed to reach to the convergence.
- The memory requirement is the maximal allocated space for U during the computation.
- For the classic SVD, we know what the memory requirement would have been (i.e. the size of the matrix U is known) without running the test.
- The total time is the time needed to reach to the convergence.
- The ratio is given by the total time divided by the number of iterations performed.

Problem specifications	Subdomains	2	4	8	16	32
	Processors	128	256	512	1024	2048
	Unknowns (10^6)	28.8	55.6	109	216	425
Time of each step (s)	Schwarz iteration	5.3	5.1	5.3	5.2	5.7
	SVD	0.10	0.13	0.19	0.31	0.41
	Aitken acceleration	7.1	6.4	7.8	8.2	8.8
Memory requirement (Mo)	Classical SVD	8.9	31.5	73.4	157.2	422.6
Scalability	Number of iterations	19	27	48	184	566
	Total time (min)	4.6	5.3	9.6	36	102
	Ratio	0.242	0.196	0.200	0.195	0.180

Table 5.2: Computational times for different problems sizes with $(\lambda, \sigma) = (10, 1)$

The computational time for the first Schwarz iterations is stable when the number of domains grows.The small differences between these computational times come from the variation of the physical problem which implies a variation of the number of Krylov iterations. The local solution after each Schwarz step is used as a new initial guess for the next local solution. As a consequence, the first local solutions are more expensive for the first Schwarz iterations. For example, the very last iterations are computed in $1.2s$. We observe that computational times of the SVD and the Aitken formula grow with respect to the size of the problem, but they are still relatively small. These two steps are performed on a single processor but the size U involved in the SVD grows faster than the size of P involved in the Aitken formula. Furthermore this matrix is allocated by a single processor. Performing the SVD itself in parallel could be needed for very large problems.

The number of iterations gives a *numerical weak scaling*. We can see that the complexity is less than quadratic, which is acceptable for this kind of problem. Finally the ratio is nearly constant, so the weak scaling of our implementation corresponds well to the *numerical weak scaling*.

5.2.1.2 Domain decomposition strategy

Now, we propose to solve the same problem with different configurations concerning the domain decomposition. We set $\lambda = 10$ and $\sigma = 2$ and solve the equations on a global Cartesian grid of size $804 \times 336 \times 336$. The global grid is sliced in the Z direction. Each local operator A_i is symmetrized. We run the code with different numbers of partitions resulting in different numbers of macro-domains M_z in the Z direction. The different topologies chosen are presented in Table 5.3.

As for the $2D$ industrial case presented before, we first choose Algorithm 9 to perform the Aitken acceleration. Table 5.4 shows the time spent in each phase of the solution process. It appears that the time spent in the computation of the base $\mathbb{U}_q$ is about the same order as the time spent in the Schwarz solver. In the previous

Topology	M_z	P_z	P_y	P_x	ov.	n_z	n_y	n_x
1	4	6	4	4	11	34	83	83
2	4	6	4	4	19	35	83	83
3	6	4	4	4	11	35	83	83
4	8	3	4	4	11	36	83	83

Table 5.3: Choice of domain decomposition strategies for solving $3D$ Darcy equation on a cartesian grid of size $804 \times 336 \times 336$ with the Aitken-Schwarz solver.

section, we proposed to reduce the time of the computation of P induced by the application of the Schwarz solver on the vector of the base by using Algorithm 8. We run the code with this other algorithm with the same parameters. Table 5.5 shows that the time saved for each configuration is about 30% compared to the time spent using Algorithm 9.

In Figure 5.25, we observe three points:

- If the overlap increases, the Aitken-Schwarz method has a better rate of convergence because the acceleration is more efficient. The local solution uses, as an initial guess, the solution given by the Schwarz iteration. Hence the time spent in the Schwarz solver is all the shorter when the first acceleration is more efficient.

- When the number of domains increases, the convergence rate of the Aitken-Schwarz solver decreases. In fact, we observe that all the accelerations are less efficient while the number of macro-domains increases, causing the total time to increase.

- The Aitken acceleration obtained by using Algorithm 8 is less efficient than the one obtained with Algorithm 9. Nevertheless, the additional number of iterations is less than the number of Schwarz application on the base arising from the SVD required to compute the error transfer operator with Algorithm 9. We note that the first acceleration stays the more efficient. Hence, this strategy should be the best one for building a cheap preconditioner for this kind of linear system.

Topology	Schwarz Time (s)	SVD Time (s)	Apply on U Time (s)	Aitken Time (s)	Total Time (s)
1	724.257	3.57513	540.793	54.1986	1497.6
2	548.037	1.78533	398.856	31.1552	1125.9
3	649.654	5.73014	564.42	57.636	1478
4	652.238	10.1233	694.916	78.5006	1643.5

Table 5.4: Time spent in the different phases of the solution of the $3D$ Darcy equation on a cartesian grid of size $804 \times 336 \times 336$ with the Aitken-Schwarz solver using Algorithm 9. Topology denotes the topology chosen in Table 5.3. Schwarz Time is the total time spent in the Schwarz Solver. SVD Time is the total time spent in the SVD. Apply on U Time is the time spent in the application of one iteration of Schwarz on each base's vector. Aitken Time is the time spent applying the Aitken Formula.

Topology	Schwarz Time (s)	SVD Time (s)	Aitken Time (s)	Total Time (s)	Time saved
1	792.846	4.56104	19.8698	1008.9	32.633 %
2	618.285	2.42081	18.4242	806.94	28.329 %
3	755.304	8.51744	28.4039	1020.7	30.943 %
4	795.04	12.8469	37.8922	1101.4	32.986 %

Table 5.5: Time spent in the different phases of the solution of the $3D$ Darcy equation on a cartesian grid of size $804 \times 336 \times 336$ with the Aitken-Schwarz solver using Algorithm 8. Topology denotes the topology chosen in Table 5.3. Schwarz Time is the total time spent in the Schwarz Solver. SVD Time is the total time spent in the SVD. Aitken Time is the time spent applying the Aitken Formula. Time Saved is the percentage of Time saved compared to the solution with Algorithm 9.

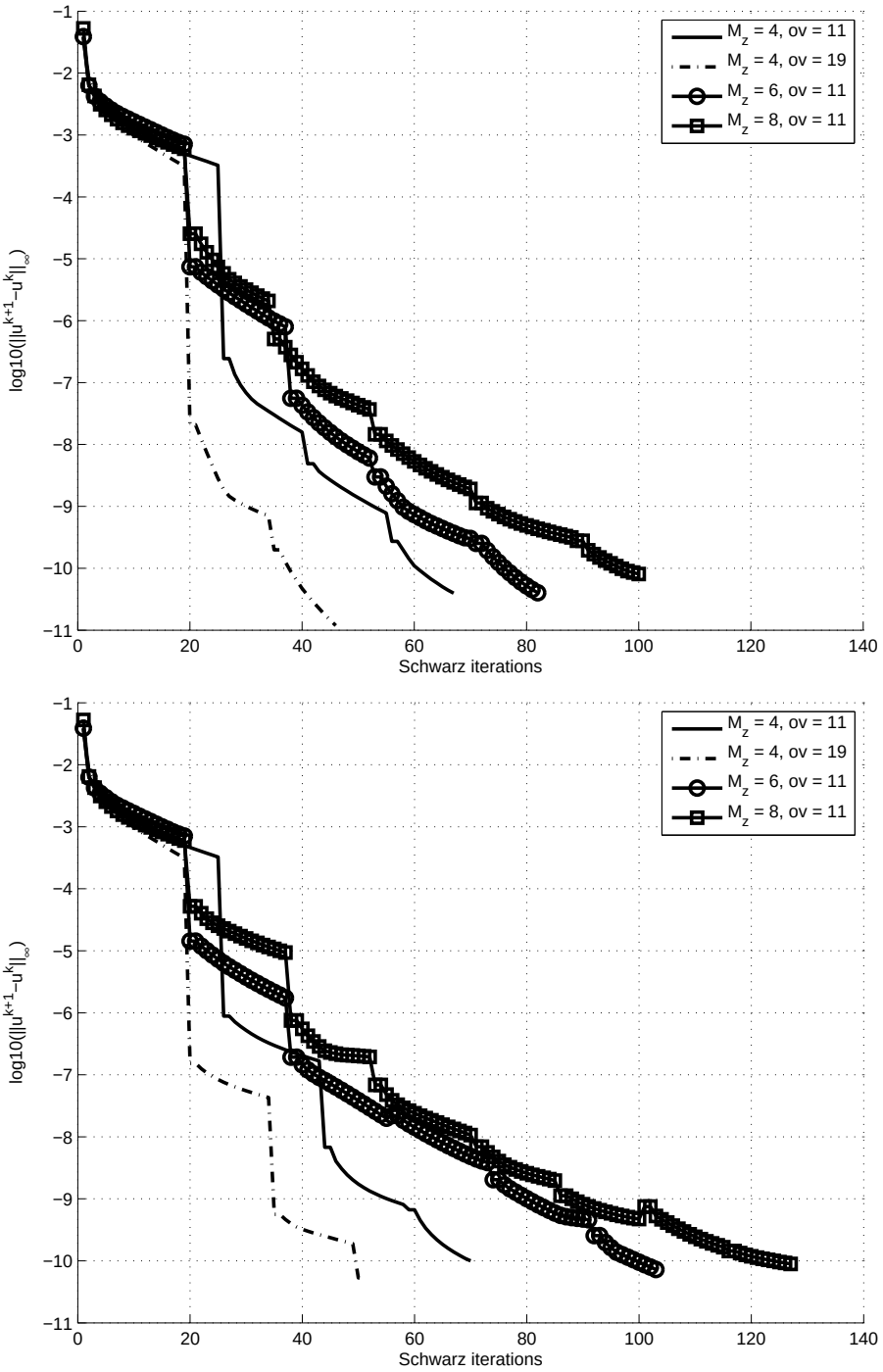

Figure 5.25: Solving the $3D$ Darcy equation on a cartesian grid of size $804 \times 336 \times 336$ with the parameters in Table 5.3 and (top) with Algorithm 9 without inversion, (bottom) with Algorithm 8 with inversion

The good scalability on $3D$ problems of the algorithms we have implemented and the way we implemented them will still be true for Aitken preconditioning techniques. The strategy of using a two-level MPI implementation is justified by the observation that the convergence rate and time consumption are deteriorated while increasing the number of partitions for the same number of processors. The best acceleration is performed at the first application of the Aitken formula. Then the fact that the ARAS preconditioning technique performs only the first acceleration guarantees a good enhancement.

5.2.2 Aitken Schwarz used as preconditioner

We consider here a cartesian grid of size $512 \times 128 \times 128$ with $\lambda = 10$ and $\sigma = 1$. The problem is partitioned with PARMETIS. The overlap is set to three. We want to evaluate different strategies of parallelization. We arbitrarily choose to solve the linear system over six partitions. We use the one-level code, given the usual strategy of one domain per process like most of the scientific library. For such a problem and configuration, the memory capability of a single processor is not sufficient to use a LU factorization. Then we must use an iterative method such as a GMRES left preconditioned sequentially by an ILU(1) factorization. This choice is not the best choice, due to the known properties of the operator we should choose the same local solver as in the Aitken-Schwarz solver dedicated to $3D$ Darcy equations: an algebraic multigrid solver. The choice of the GMRES is done because it will be used to solve local systems in the $3D$ industrial cases. Concerning the ARAS technique, we save computing time by applying a RAS preconditioner with a high tolerance (here it is set to $1e-3$) only for the building phase of the Aitken level. All the solution are computed with a tolerance set to 10^{-10}.

Figure 5.26 shows the time consumption for each key phase of the solution for six domains in the configurations we chose. The figure denotes too large a time consumption for the solution phase for the usual one-level MPI strategy. The total time is reduced using the ARAS(24) preconditioner. For the two-level MPI strategy we try two different choices of local solver: one using a local iterative solver and another using a parallel direct solver because the distribution of local systems over twelve processors allows the code to run a LU factorization locally with less than two G.B. per process, which is good enough on machine B.
When the local solver is a GMRES preconditioned by a RAS with local solution approximated by ILU(1), the solution time decreases significantly by a factor of six using 12 processors per domain. The fact we use a high tolerance in the buiding of the approximation of the error transfer operator allows us to save computing time. The time to build the preconditioner is less than the solution time. Moreover, the good property of the ARAS preconditioner makes the solution time decreases. Then the total solution time is smaller for ARAS than for RAS.
When the local solver is a parallel direct solver (here it is SUPERLU_DIST), the code runs faster than with the local iterative solver chosen before. However, the same conclusion as for the $2D$ industrial case can be made: effort must be made to reduce the time to build the error transfer operator. This should be done choosing Algorithm 8 as we illustrated in the previous sub-section.

The strategy of using a local direct solver is still highly memory demanding. For example, here, the two G.B per process is the critical limit. It is recommended to use the parallel strategy with an iterative local solver due to memory issues. If this is the case, then the ARAS(q) technique becomes an interesting choice. We note that the solution time could be reduced by re-using the Krylov bases at each

application of ARAS, instead of computing all the direction descent vectors every time.

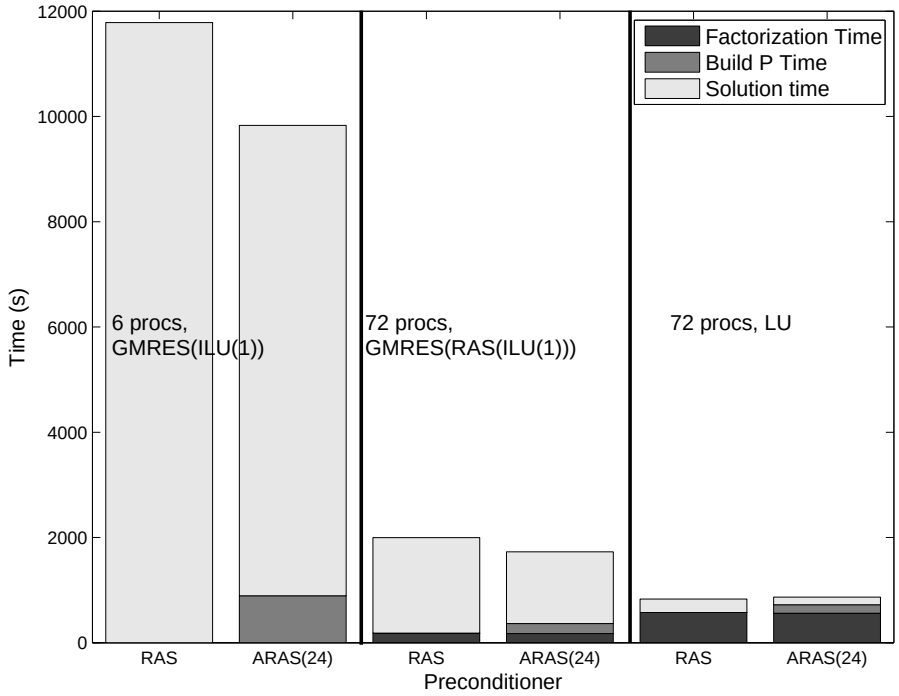

Figure 5.26: Time consumption while solving the $3D$ Darcy equation on a cartesian grid of size $512 \times 128 \times 128$ with Algorithm 9 without inversion, for 6 partitions with three different parallelisms.

In order to point out the scalability of the ARAS preconditioning technique on $3D$ problems, we change the configuration and set up 72 domains, one domain per processor. We choose the strategy with local iterative solvers. The size of the interface for six partitions is equal to 194735. For 72 partitions it is 811491. We set the size of the base to compute the acceleration to 24 for six partitions and 72 for 72 partitions.
Figure 5.27 shows that the preconditioning is more efficient for a small number of partitions, but we observe a very good numerical scalability of the ARAS preconditioning technique.
Table 5.6 shows the corresponding time and memory spent while running the code in both configurations. Those results suggests to choose the one-level strategy.

The number of partitions can be very important compared to the number of domain we need to choose for efficient computing for the Aitken-Schwarz solver. The preconditioning strategy scales very well on this $3D$ problem.

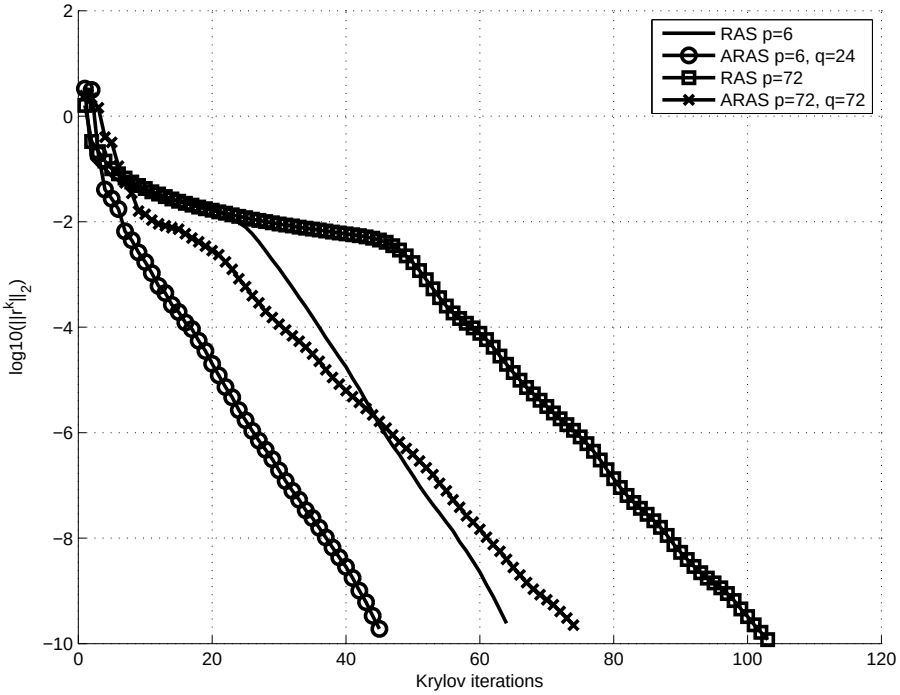

Figure 5.27: Solving the $3D$ Darcy equation on a cartesian grid of size $512 \times 128 \times 128$ with Algorithm 9 without inversion, for six or 72 partitions. Iterative solvers are chosen for the local solution.

Procs.	Parts.	Prec. Time (s)	Total Loc. (M.O.)	Max. Mem.
72	6	RAS	1997.524	815
72	6	ARAS(24)	1729.120	831
72	72	RAS	588.881	466
72	72	ARAS(72)	588.00	535

Table 5.6: Time and memory consumption while running the code with six or 72 partitions among 72 processors.

For this kind of $3D$ problem the ARAS preconditioning is very efficient also. It applies a second level on a coarse interface represented by a small number of vectors compared to the size it represents. Algorithm 9 must be chosen if a competitive solver is required. There is no need for using a second level of MPI for this grid size. We need to run $3D$ industrial test cases to see if their characteristics (large number of data dependencies compared to the global size) do not disturb the scalability. If it is the case, the two-level MPI implementation should avoid this problem, as for the Aitken-Schwarz solver.

5.3 $3D$ CFD test cases

In this section we focus on a $3D$ industrial case we presented in chapter 1. We pointed out in Table 1.2 that the number of data dependencies between domains can be greater than the global size of the operator with a small number of partitions. Here, we want to observe the effect of such a growth of the data dependencies while solving the linear system with a GMRES preconditioned by RAS and ARAS(q). We choose three domain decomposition topologies, $p =\in \{12,\ 24,\ 72\}$ with an overlap equal to one. We choose a unique number $q = 12$ to define the coarse interface.
The ARAS(q) left preconditioning is computed with the one-level MPI implementation and with local iterative solvers (here it is a GMRES left preconditioned by ILU). The tolerance for the local solution for the building of the acceleration is set to $1e{-}3$. The tolerance for the local solution during the solving phase is set to $1e{-}10$.

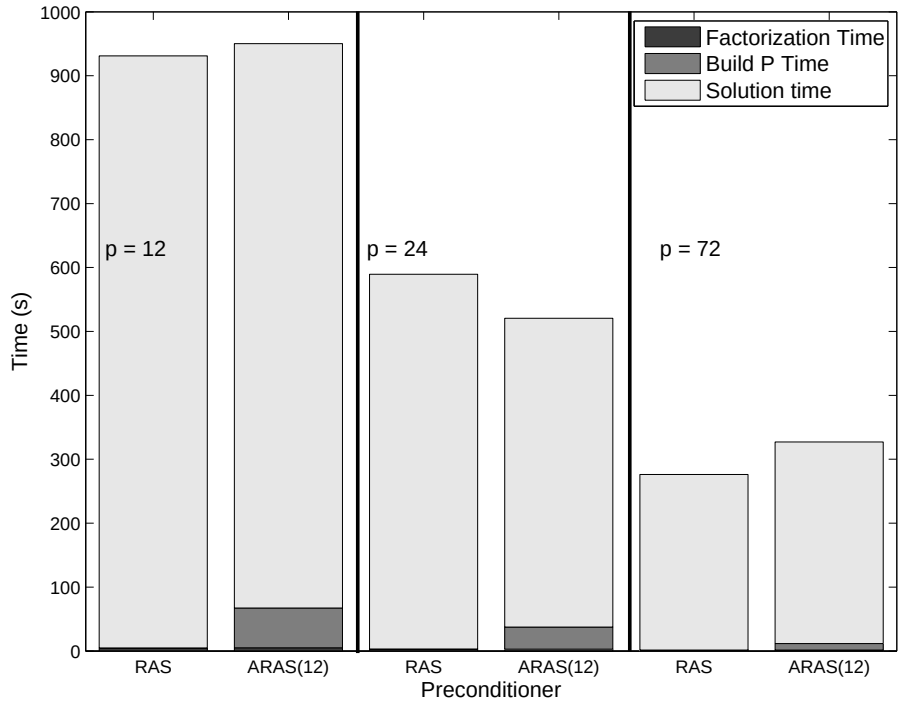

Figure 5.28: Time consumption while solving the CASE_017.

Figure 5.28 shows the time consumption of the one-level MPI PETSc implementation of ARAS, on the CASE_017 RM07, for a number of processors $p \in \{12,\ 24,\ 72\}$ with overlap equal to one. The factorization time (ILU(1)) can be neglected compared to the solution time. The solution time is reduced while the number of processors increases. The time spent in the building of the error transfer operator is also reduced.

For $p \in \{12,\ 72\}$, ARAS(12) does not reduce the solution time. For $p = 24$, it reduces the solution time.

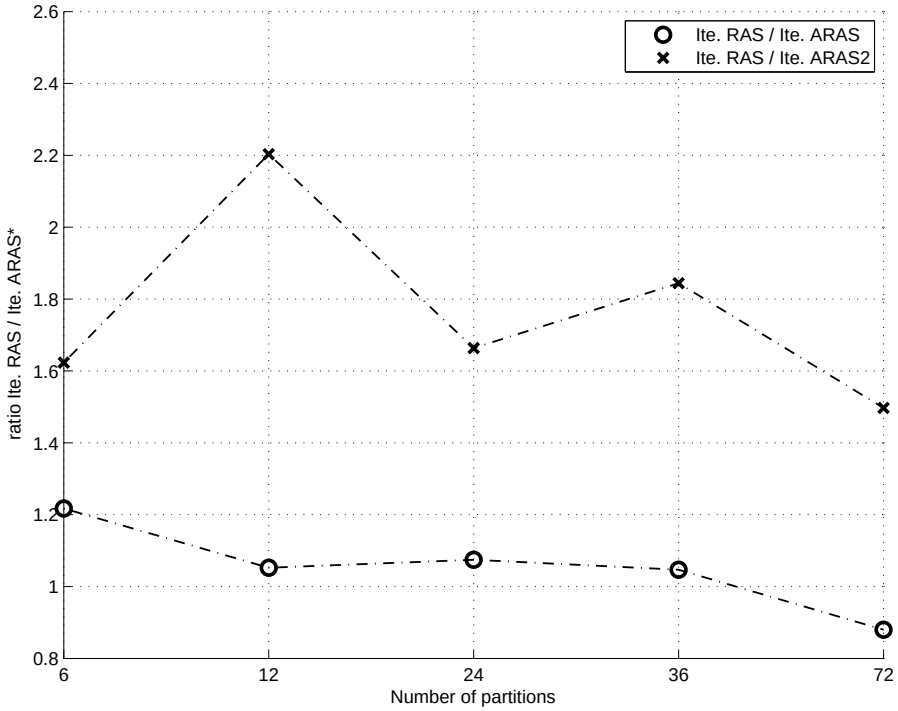

Figure 5.29: Numerical efficiency of ARAS and ARAS2 compared to RAS for a small number of vector bases (36).

Figure 5.29 presents the numerical efficiency of ARAS(q) and ARAS2(q) compared to RAS.
The curve with circles shows that the best efficiency of ARAS appears for the smallest number of partitions chosen. For $p \in \{12,\ 24,\ 36\}$ the efficiency is nearly the same, with a little bit more efficiency for $p = 24$. For $p = 72$ the Aitken technique deteriorate the convergence rate of RAS.
The curve with crosses shows that the best efficiency of ARAS2 appears for $p = 12$ and is greater than 2.2. For $p \in \{6,\ 24,\ 36\}$ the efficiency is nearly the same, with a little bit more efficiency for $p = 36$.
The ARAS2(12) technique is then a good choice to reduce the solution time for $p = 12$, and the ARAS(12) technique is a good choice to reduce the solution time for $p = 6$.

Table 5.7 presents the convergence for each configuration tested in Figure 5.28. The configuration with $p = 72$ leads to a loss of convergence of about four orders of magnitude. This should be the consequence of a bad partitioning and too many

p	q	Ite.	Rel. Residual (GMRES)	$\|\|Ax-b\|\|_2/\|\|b\|\|_2$
12	0	141	5.4201e-14	5.2641e-09
12	12	134	8.6801e-14	3.2475e-08
24	0	173	5.4239e-14	9.6885e-10
24	12	165	1.2449e-13	5.6873e-09
72	0	256	4.0725e-14	**1.8957e-05**
72	12	287	1.4511e-13	**1.5880e-05**

Table 5.7: Solution information on CASE_017 RM07 for $p \in \{12,\ 24,\ 72\}$ and the overlap set to one.

data dependencies between domains.

For too a large number of partitions with too large a number of data dependencies compared to the size of the domain, the RAS preconditioner is not as efficient as wished. We consider that it is too hazardous to solve this kind of industrial problem with a large number of partitions in order to save time and memory computing. The two-level MPI implementation should be a good choice for solving those problems on a large number of processors and save computing. Nevertheless, the local solutions need more special care than for the $3D$ Darcy problems. Developments are in progress.

Chapter 6

Conclusions and outlook

6.1 French version - Version française

Une nouvelle technique de préconditionnement multi-niveaux a été proposée. Les difficultés numériques rencontrées lors de la résolution de systèmes linéaires issus de la différentiation automatique des solutions de Navier-Stokes compressible suivant les paramètres de simulation ont motivés la conception de cette technique.

Dans le chapitre 2 nous avons présenté la convergence purement linéaire de la méthode de décomposition de domaine de Schwarz. Certaines techniques multi-niveaux, considérées soit comme méthodes de résolution, soit comme méthodes de préconditionnement, reposent sur la formulation d'un problème sur l'interface. Considérant la convergence, ou divergence, purement linéaire de la méthode de Schwarz sur l'interface, nous nous intéressons à l'accélération de la convergence par la technique d'accélération d'Aitken vectorielle. Cette technique est intimement liée à la méthode du complément de Schur.

Dans le chapitre 3 nous avons proposé différentes techniques pour créer un espace d'approximation dans lequel effectuer l'accélération d'Aitken. Deux approches différentes ont été mises en évidence :

(a) Troncature de l'opérateur dans une base complète construite analytiquement.

(b) Approximation de l'opérateur dans une base construite explicitement.

Tandis que la méthode (a) requiert une discrétisation régulière ou de fortes hypothèses sur l'opérateur discret A, la méthode (b) est algébrique. De plus, la méthode (a) se montre plus sensible aux perturbations que la méthode (b). Dans la catégorie des méthodes de type (b), nous nous intéressons particulièrement à l'approximation dans une base provenant de la Décomposition en Valeurs Singulières (SVD) des solutions à l'interface, d'une méthode itérative ayant une convergence purement linéaire. Sur des considérations d'analyse numérique, la décroissance des valeurs singulières donne un bon critère de sélection des vecteurs de base qui permettent de représenter la solution sur l'interface. La méthode d'Aitken-Schwarz devient alors une méthode avec un critère algébrique pour effectuer l'accélération de la convergence. Elle peut donc être appliquée sans considérer la discrétisation des équations.
L'approximation dans un tel espace peut conduire à une perte de précision lors de

l'accélération. Afin de compenser cette perte, nous avons étudié l'intégration de l'accélération d'Aitken dans une forme de Richardson d'un processus de Schwarz Additif Restreint. L'idée sous-jacente est que le préconditionneur RAS génère une suite de solutions sur l'interface Γ *qui converge linéairement. On construit un préconditionneur qui intègre l'accélération d'Aitken directement dans sa formulation pour donner un nouveau processus itératif.*
Ceci mène à la formulation du préconditionneur Aitken Schwarz Additif Restreint (ARAS) originel. L'approximation de l'accélération a été ajoutée et conduit au processus ARAS(q), où q est un paramètre définissant l'espace grossier. Quand la base est construite avec la SVD des solutions sur l'interface de RAS, q représente le nombre de valeurs singulières retenu dans le processus de sélection.
Dans le but de comprendre les caractéristiques de convergence de la technique de préconditionnement ARAS(q), nous avons proposé une étude de convergence et plus précisément, une analyse théorique quand l'approximation est effectuée par troncature de l'opérateur dans une base complète construite analytiquement.
Placé dans le contexte dans lequel nous souhaitons utiliser cette technique, une attention particulière est portée sur l'analyse du coût du préconditionneur. Nous avons exprimé que l'application de ARAS(q) était, en terme de nombre d'opérations, du même orde que le coût de RAS lorsque q est suffisamment petit comparé au nombre de dépendances de données entre les sous-domaines. Le coût de construction du préconditionneur dépend de q et dépend de l'algorithme qui est choisi pour approximer l'opérateur de transfert d'erreur. Nous avons présenté le coût de la construction la plus robuste du préconditionneur.

Dans le chapitre 4, on suppose que les méthodes de décomposition de domaine présentent en général de bonnes propriétés pour être utilisées sur des machines parallèles compte tenu de l'idée naturelle de décomposition en plusieurs sous-problèmes qui peuvent être résolus indépendamment les uns des autres. En considérant cette propriété, nous avons présenté les principaux points de la méthode qui peuvent être parallélisés. Une première étude a été effectuée sur la méthode de Schwarz.
Deux implémentations ont été présentées. L'une, est dédiée à un type particulier de problème : la résolution d'un système linéaire issu de la discrétisation en Volumes Finis des équations de Darcy $3D$ *avec une forte variabilité de la perméabilité sur une grille cartésienne* $3D$*. La décomposition de domaine est basée sur un partitionnement physique du domaine selon la dimension principale. Chaque sous-domaine est lui-même partitionné afin d'accroître le parallèlisme du code. Les résolutions locales sont effectuées par une méthode multi-grille algébrique. Différents algorithmes d'accélération adaptative sont implémentés et permettent au programme de réduire la consommation de mémoire et d'obtenir de bonnes performances au fil de son exécution. Le code permet de tester la méthode sur de très grands problèmes et d'effectuer des tests d'extensibilité sur des machines massivement parallèles.*
La seconde implémentation est l'implémentation totalement algébrique du préconditionneur ARAS(q) dans l'environnement de la bibliothèque de calcul PETSc. Le partitionnement est effectué avec un partitionneur tel que PARMETIS et ne repose

plus sur le maillage sous-jacent. Afin d'apporter l'amélioration du préconditionneur Schwarz, nous avons dû ré-écrire le préconditionneur RAS. Notre implémentation révèle les mêmes performances que celles de l'implémentation initiale mais nous permet d'accéder à l'ensemble des dépendances de données ainsi qu'au recouvrement qui sont nécessaires à l'écriture de l'accélération d'Aitken. Une implémentation de ARAS et ARAS2 a été proposée, s'appuyant sur l'interface PC_SHELL de PETSc. Comme l'implémentation est dédiée à la résolution de tous systèmes linéaires sans considération des équations sous-jacentes et du maillage, l'accélération d'Aitken est effectuée dans une base issue de la SVD de q solutions d'un processus itératif RAS sur l'interface. Une contribution originale de ce travail de développement au sein de la bibliothèque PETSc est la conception d'un second niveau de parallélisation sous le paradigme MPI. La distribution des données et les communications ont été pensées pour pouvoir utiliser les fonctions existantes de PETSc. Cette nouvelle implémentation ne nécessite pas la ré-écriture de l'accélération d'Aitken. L'implémentation à deux niveaux MPI permet à l'utilisateur de RAS ou ARAS(q) d'exploiter différentes stratégies de parallélisation quand ceci est nécessaire. L'extensibilité de cette implémentation est la même que celle dédiée aux équations de Darcy $3D$.

Des résultats de performances, à la fois numérique et informatique, ont été présentés sur trois types de systèmes linéaires dans le chapitre 5. Le code PETSc a d'abord été éprouvé sur les cas de CFD $2D$. Deux problèmes, l'un sans turbulence, l'autre avec turbulence, ont été résolus sous différentes configurations avec l'implémentation à un niveau MPI de RAS et ARAS(q). Les bonnes propriétés numériques de ARAS(q) sont observées pour des configurations où le processus itératif RAS converge ou diverge. Cependant, ceci a montré que le temps passé dans la construction de l'opérateur de transfert d'erreur dans une base d'approximation peut s'avérer trop chèr comparé au temps de résolution.
Comme deuxième classe de tests, nous avons présenté des résultats de la méthode d'Aitken-Schwarz avec SVD comme méthode de résolution ou de préconditionnement sur les problèmes de Darcy $3D$. Des tests d'extensibilité ont été accomplis sur la méthode de résolution d'Aitken-Schwarz et ont montré une très bonne extensibilité en terme de parallélisation et de calcul informatique. Mais quand le nombre de domaines croît, l'efficacité numérique de l'accélération d'Aitken diminue si le nombre de vecteurs pour décrire l'interface est trop petit. Les deux algorithmes de construction de la base impliquant la SVD ont été testés. L'algorithme sans inversion, mentionné comme étant le plus robuste, conduit à de meilleures propriétés numériques. Bien que l'algorithme avec inversion, mentionné comme étant plus sensible, montre une perte de précision dans l'accélération, son calcul est moins coûteux et permet un gain de temps avoisinant les 30%.
Quand elle est utilisée comme une méthode de préconditionnement sur les problèmes $3D$, la technique ARAS(q) apparaît comme étant moins sensible à l'augmentation du nombre de sous-domaines. Elle montre de très bonnes propriétés pour un préconditionneur deux niveaux sur les problèmes $3D$. Nous avons proposé deux manières différentes d'économiser les opérations informatiques dans la phase de construction

du préconditionneur :

(a) Si les résolutions locales sont effectuées par une méthode de résolution itérative, la tolérance de la méthode locale peut être élevée (ici, nous avons utilisé une tolérance pour le résidu relatif du GMRES de $1e-3$*) lors de la construction de la base SVD et de l'opérateur de transfert d'erreur.*

(b) Choisir l'algorithme SVD avec inversion et éviter une deuxième application du préconditionneur RAS sur la base complète de la SVD pour construire l'opérateur de transfert d'erreur.

Enfin, nous avons testé l'implémentation de ARAS(q) sur un problème de CFD $3D$*. Nous avons montré les difficultés de ce test, et notamment le fait que le nombre de dépendances de données peut être plus grand que la taille du problème global. Le principal problème est que ce phénomène est couplé à une forte divergence du processus itératif RAS. Dès lors, seulement un petit nombre de vecteurs de base peut être calculé et l'accélération peut détériorer la convergence en ajoutant beaucoup trop de bruit. Un autre problème concerne le préconditionneur RAS : quand le nombre de partitions est trop grand sur ce problème, la convergence à la solution est détériorée également. Ainsi, la proposition d'utiliser l'implémentation à deux niveaux MPI devient nécessaire afin de conserver un nombre raisonnable de dépendances de données et une bonne convergence à la solution.*
À ce stade des travaux, la résolution des systèmes locaux demande une attention particulière lorsqu'il s'agit de les résoudre en parallèle. En effet, les opérateurs locaux devraient être partitionnés autrement qu'en blocs de lignes et demandent une étude de leurs propriétés. Ceci n'était pas un inconvénient pour le cas Darcy $3D$ *compte tenu du fait que les sous-domaines sont discrétisés de la même manière que le problème global.*

Nous avons mis en lumière les gains numériques apportés par cette méthode et sa prallélisation efficace. Certains points concernant les difficultés rencontrées avec les matrices provenant de la différentiation automatique en CFD ont été mis en évidence.
Dans la plupart des configurations, le processus itératif RAS diverge. Alors, seulement peu de vecteurs peuvent être gardés pour éviter l'introduction de bruit. Deux idées devraient être suivies pour améliorer la qualité de la SVD quand le processus RAS diverge :

(a) Une relaxation peut être ajoutée dans le processus itératif pour atténuer la divergence.

(b) Les solutions à l'interface pourraient être normalisées avant de réaliser la SVD. En opérant ainsi, on espère que les valeurs les plus fortes deviennent du même ordre et permettent d'obtenir une SVD plus riche.

Du point de vue de l'implémentation, l'application de RAS sur la base complète est calculée itérativement sur une suite de vecteurs. Des produits Matrice-Matrice

pourraient remplacer la boucle sur les opérations Matrice-Vecteur.
Pour la stratégie mettant en oeuvre les méthodes de résolution itératives locales, il est possible de réduire le temps des résolutions locales qui sont répétées à chaque pas en gardant les vecteurs de Krylov. Autrement l'utilisation d'un Flexible-GMRES devrait être testée.
Enfin, la capacité d'un partitionneur à produire une bonne décomposition de domaine pour une méthode de Schwarz devrait être étudiée. L'idée est de produire un système local avec de bonnes propriétés pour la factorisation LU complète ou incomplète. Des méthodes existent pour tester si la factorisation incomplète est de bonne qualité mais il n'y a pas de technique pour produire automatiquement un sous-opérateur qui soit un bon candidat pour une factorisation LU.

6.2 English version - Version anglaise

A new multilevel preconditioning technique has been proposed. The design of this technique was motivated by the numerical difficulties encountered when solving matrices coming from automatic differentiation of compressible Navier-Stokes solutions with respect to the simulation parameters.

In Chapter 2, we presented the purely linear convergence of the Schwarz domain decomposition method. Some multilevel techniques, viewed as solvers or preconditioners, are based on a formulation of a problem on the interface. Considering the purely linear convergence or divergence of the Schwarz method on the interface, we focus on the acceleration of the convergence by the vectorial Aitken acceleration technique. This technique is intimately related to the Schur complement method.

In Chapter 3, we proposed different techniques to create an approximation space to perform an Aitken acceleration. Two different approaches were highlighted:

(a) Truncation of the operator in a complete base built analytically.

(b) Approximation of the operator in a base built explicitly.

While method (a) requires a regular discretization or strong assumption on the discretized operator A, method (b) is algebraic. Moreover, method (a) seems to be more sensitive to perturbations than method (b). In the class of method (b) we devoted a particular interest to the approximation in a base arising from the Singular Value Decomposition of the interface solutions of an iterative method with a pure linear convergence. According to numerical analysis, the descent of the singular values gives a good criterion to select the basis vectors which allows us to represent the solution on the interface. Then, the Aitken-Schwarz method becomes a method with an algebraic criterion to accelerate the convergence. It can be applied without consideration of any discretization.
The approximation in such a space can lead to a lack of accuracy in the acceleration.

In order to compensate for this problem, we study the integration of the Aitken acceleration into the Richardson form of a Restricted Additive Schwarz process. We follow the idea that such a preconditioner must generate a sequence of solutions on the interface Γ with a Restricted Additive Schwarz iterative process, and accelerate the convergence of the Schwarz process from this original sequence applying the Aitken formula. Then, the accelerated solution on the interface replaces the last solution.
This proposition leads to the formulation of the original Aitken Restricted Additive Schwarz preconditioner (ARAS). The approximation of the acceleration is added and leads to the ARAS(q) process, where q is a parameter to define the coarse space. When the base is built from the Singular Value Decomposition of the interface's solutions of RAS, q represents the number of singular values kept in the selection process.
In order to understand the convergence behaviour of the ARAS(q) preconditioning technique, we proposed a convergence study and more precisely, a theoretical analysis when the approximation is done by truncation of the operator in a complete base built analytically.
In the context in which we want to use this technique, special care is given to the analysis of the cost of this preconditioner. We showed that the application of ARAS(q) has approximately the same cost as applying RAS when q is sufficiently small compared to the number of data dependencies between sub-domains. The cost of the computation of the preconditioner depends on q and depends on the algorithm which is chosen to approximate the error transfer operator. We presented the cost of building the preconditioner the robust way.

In Chapter 4, we assumed that domain decomposition methods, in general, present good properties to be used on parallel machines due to their natural method of splitting a problem into several sub-problems which can be solved independently. Considering this characteristic, we presented the main points of the method which can be parallelised. A first study is done on the Schwarz method.
Two implementations were presented. One implementation is dedicated to a special kind of problem: the solution of linear systems coming from the finite volume discretization of the $3D$ Darcy equations with strong variability of the permeability on large $3D$ Cartesian grids. The domain decomposition is based on a physical partitioning into the leading dimension. Each sub-domain is also partitioned in order to increase the parallelism of the code. The local solutions are performed by an algebraic multigrid method. Different algorithms of adaptive accelerations are implemented and enable the program to adaptively save computation and memory achieving good performances. The code enables us to test the method on very large problems and evaluate its scalability on massively parallel machines.
The second implementation is the fully algebraic implementation of the ARAS(q) preconditioner in the PETSc framework. The partitioning is done with a partitioner such as PARMETIS and is no longer based on the mesh. In order to provide the enhancement of the Schwarz preconditioner we had to re-write the RAS

preconditioner. Our implementation has the same performance as the initial implementation, but allows us to access all the data dependencies and overlap necessary to write the Aitken acceleration. An implementation of ARAS and ARAS2 is proposed through the PC_SHELL interface. As the implementation is designed for solving any linear systems without consideration of the underlying equations or the mesh, the Aitken acceleration is performed in a base arising from the SVD of q interface solutions of the RAS iterative process. An original contribution of this work concerning the development into the PETSc framework is the design of a second level of MPI parallelism. The data distribution and the communications have been designed to use the existing routines of PETSc. This new implementation does not need a re-writing of the Aitken acceleration. The two-level MPI implementation enables the user of a RAS or ARAS(q) preconditioner to exploit different strategies of parallelism when necessary. The scalability of the strategy is the same as for the code dedicated to the $3D$ Darcy equations.

Results to understand the performances, both numerical and computational, were presented on three types of linear systems in Chapter 5. The PETSc code ran, at first, on $2D$ CFD test cases. Two problems, one without turbulence and one with turbulence, were solved with different configurations with the one-level MPI version of RAS and ARAS(q). Good numerical behaviours of ARAS(q) were observed for configurations where the RAS iterative process converges or diverges. Nevertheless, it was shown that the time spent in building the error transfer operator in a base of approximation can be too expensive compared to the solution time.
As a second class of tests, we presented results of the Aitken-Schwarz method with SVD as the solver and preconditioner on the $3D$ Darcy problem. Tests of scalability were performed on the solver and showed good computational scalability. When the number of domains increases, however, the efficiency of the Aitken acceleration decreases if the number of vectors to describe the interface is too small. Two algorithms for building the acceleration in the SVD space were tested. The algorithm without inversion, called more robust, leads to better numerical properties. Although the algorithm with inversion, called more sensitive, shows a lack of accuracy in the acceleration, its computation is cheap and enables us to save around 30% of the computational time.
When used as preconditioner on those $3D$ problems, the ARAS(q) preconditioning technique appears to be less sensitive to increasing the number of sub-domains. It shows very good properties for a two-level preconditioner for $3D$ problems. We proposed two different ways to save computation in the building phase of the preconditioner:

(a) if local solutions are performed by an iterative solver, the tolerance of the local solver can be high (here we used $1e-3$ as the tolerance for the relative residual of the GMRES) when building the SVD base and transfer operator.

(b) we choose the algorithm with SVD and inversion to avoid a second application of the preconditioner on the entire SVD base to compute the error transfer

operator.

Finally, we tested the ARAS implementation on a $3D$ industrial problem. We emphasized some difficulties of this test, and notably the fact that the number of data dependencies can be larger than the global size of the problem. The principal issue is that this phenomenon is coupled to a strong divergence of the RAS iterative process. Therefore only a very few number of basis vectors can be computed and the acceleration can deteriorate the convergence, adding too much noise. Another problem concerns the RAS preconditioner: when the number of partitions is too large on this problem, the convergence to the solution is also deteriorated. Hence, the proposition of using the two-level MPI implementation should be necessary in order to keep a reasonable number of data dependencies and good convergence to the solution.
At this point, the local system requires special care to be solved on several processors while using an iterative method. This is because we need to partition the local sub-systems with a different partitioning than a block partitioning and study the properties of the problems. This was not a problem for the $3D$ Darcy problem due to the fact that the sub-domains were discretised the same way as the global problem.

We highlighted numerical benefits of the method and efficient parallelism. Some points have been revealed in this study regarding to the difficulties provided by the matrices arising from the automatic differentiation in CFD with respect to the parameters.
In most of the configurations, the RAS iterative process diverges. Then, only a few vectors can be kept to avoid introducing noise. Two ideas should be followed to enhance the quality of the SVD when the RAS process diverges:

(a) A relaxation can be added in the iterative process to dampen the divergence.

(b) The interface solution could be normalized before performing the SVD. Doing this, we hope that the strongest values become of the same order and enable a richest SVD.

Concerning the implementation, the application of RAS on the entire base is computed iteratively on a sequence of vectors. Matrix-matrix operations could replace the loop over matrix-vector operations.
For the strategy with local iterative solvers, it is possible to reduce the time of the local solutions which are repeated at each iteration by keeping the Krylov vectors. Otherwise, the use of a flexible GMRES should be tested.
The ability for a partitioner to provide a good domain decomposition for the Schwarz method should be studied. The idea is to produce local systems with good properties for complete and incomplete LU factorisation. Methods exist to test if an incomplete factorisation is good, but there is no technique to automatically produce a sub-operator which is a good guess for ILU factorisation.

Appendix A

Tables of running information of the ARAS implementation

This appendix is related to chapter 5. It shows the performances of the PETSc code presented in chapter 4 on $2D$ industrial cases, with different sets of parameters. Table A.1 concerns the results obtained on CASE_005 FR02 and Table A.2 concerns the results obtained on the CASE_006 PR02. The study presented in chapter 5 is based on those results.
The general configuration is a one-level MPI implementation of a GMRES preconditioned by a RAS or ARAS(q) technique over p processes.
In each table,

- p denotes the number of partitions and here, the number of processors.
- *Prec.* denotes the kind of preconditioner chosen for solving with a full GMRES. It could be one of RAS, RAS2, ARAS or ARAS2.
- *Ov.* denotes the overlap chosen for the domain decomposition.
- n denotes the size of the global artificial interface Γ.
- q denotes the size of the approximation space chosen to perform the Aitken acceleration. It is zero if *Prec.* is RAS or RAS2.
- *Fact. Time* is the computational time spent in the local LU factorization.
- *Build P Time* denotes the time spent to compute the approximation base $\mathbb{U}_q$ and the operator $P_{\mathbb{U}_q}$.
- *It.* denotes the number of iterations of Full GMRES.
- *Rel. Res.* denotes the relative residual of the GMRES at the end of the execution.
- *Sol. Time* denotes the computational time spent in the solution time.
- *Tot. Time* denotes the total computational time.
- *Max. Mem. Loc.* denotes the maximum amount of memory locally allocated.
- *||b-Ax||/||b||* denotes the relative error to the solution.

A.1 Running information of CASE_005 PR02

Table A.1: Runs information on CASE_005 FR02 using the one level MPI version of ARAS in PETSc

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	$\|\|b - Ax\|\|/\|\|b\|\|$
2	RAS	2	3696	0	395.691s	0	31	2.4881e-13	15.061s	410.752s	3423m	1.2231e-11
2	RAS2	2	3696	0	393.438s	0	16	2.9137e-13	16.02s	409.458s	3418m	2.1449e-11
2	ARAS	2	3696	10	395.522s	8.168s	22	2.9207e-13	11.507s	415.197s	3422m	1.5368e-11
2	ARAS2	2	3696	10	394.072s	7.899s	11	5.2618e-13	12.124s	414.095s	3416m	3.8145e-11
2	ARAS	2	3696	20	395.036s	15.633s	12	3.4591e-13	7.537s	418.206s	3416m	2.2829e-11
2	ARAS2	2	3696	20	399.046s	16.085s	6	8.1128e-13	8.347s	423.478s	3417m	5.4725e-11
2	ARAS	2	3696	40	393.353s	31.015s	3	6.1195e-16	4.124s	428.492s	3416m	6.3559e-13
2	ARAS2	2	3696	40	398.602s	32.173s	2	3.2692e-16	5.124s	435.899s	3411m	1.2338e-11
2	RAS	4	3640	0	401.135s	0	25	1.4518e-13	13.143s	414.278s	3487m	6.7415e-12
2	RAS2	4	3640	0	406.228s	0	13	1.4113e-13	14.394s	420.622s	3482m	7.8782e-12
2	ARAS	4	3640	10	399.769s	8.013s	16	1.6464e-13	9.565s	417.347s	3486m	4.7378e-12
2	ARAS2	4	3640	10	400.231s	7.917s	9	2.3063e-13	10.367s	418.515s	3481m	1.0085e-11
2	ARAS	4	3640	20	400.687s	16.444s	6	2.8408e-13	5.49s	422.621s	3480m	2.0356e-11
2	ARAS2	4	3640	20	401.361s	15.846s	3	3.3242e-13	5.885s	423.092s	3481m	3.0501e-11

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
2	ARAS	4	3640	40	402.443s	32.287s	3	3.2916e-16	4.274s	439.004s	3480m	2.0094e-12
2	ARAS2	4	3640	40	400.388s	31.266s	2	1.9602e-16	4.972s	436.626s	3475m	5.3157e-12
4	RAS	2	7455	0	173.054s	0	61	5.1284e-13	17.2s	190.254s	1683m	1.4707e-10
4	RAS2	2	7455	0	179.043s	0	32	1.7982e-13	17.936s	196.979s	1675m	5.8899e-11
4	ARAS	2	7455	20	173.008s	9.611s	43	4.8978e-13	12.494s	195.113s	1687m	2.7652e-10
4	ARAS2	2	7455	20	173.126s	9.798s	26	5.7193e-13	15.006s	197.930s	1682m	5.5102e-10
4	ARAS	2	7455	32	172.983s	15.231s	31	5.3792e-13	9.436s	197.650s	1682m	2.3339e-10
4	ARAS2	2	7455	32	173.953s	15.652s	18	4.7534e-13	11.065s	200.670s	1681m	1.2200e-09
4	ARAS	2	7455	40	175.402s	19.427s	23	5.8928e-13	7.669s	202.498s	1683m	2.1331e-10
4	ARAS2	2	7455	40	173.970s	19.43s	13	4.3731e-13	8.607s	202.007s	1682m	2.7038e-10
4	ARAS	2	7455	60	174.390s	29.399s	5	3.6135e-13	3.22s	207.009s	1682m	1.2897e-10
4	ARAS2	2	7455	60	172.861s	29.426s	3	4.7403e-13	3.762s	206.049s	1682m	1.0760e-10
4	ARAS	2	7455	80	172.983s	38.78s	4	6.0416e-14	2.977s	214.740s	1684m	7.2115e-11
4	ARAS2	2	7455	80	173.167s	39.177s	2	4.2218e-13	3.258s	215.602s	1684m	1.9558e-10
4	RAS	4	7511	0	178.886s	0	47	5.5883e-13	14.05s	192.936s	1742m	1.1822e-10
4	RAS2	4	7511	0	177.692s	0	25	1.5249e-13	14.703s	192.395s	1738m	2.4578e-11
4	ARAS	4	7511	20	181.315s	10.152s	29	3.4730e-13	9.45s	200.917s	1745m	5.6532e-10
4	ARAS2	4	7511	20	177.751s	10.145s	16	8.4285e-13	10.394s	198.290s	1742m	1.1833e-09

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	$\|\|b - Ax\|\|/\|\|b\|\|$
4	ARAS	4	7511	40	177.918s	20.304s	9	4.3854e-13	3.976s	202.198s	1741m	1.0530e-09
4	ARAS2	4	7511	40	177.672s	20.208s	5	4.7968e-13	4.793s	202.673s	1741m	9.7661e-10
4	ARAS	4	7511	80	178.006s	40.658s	3	5.2598e-14	2.786s	221.450s	1742m	2.1377e-10
4	ARAS2	4	7511	80	178.631s	40.003s	2	2.7007e-14	3.267s	221.901s	1741m	3.2422e-11
6	RAS	2	11571	0	84.902s	0	66	2.6802e-13	11.996s	96.898s	996m	2.4005e-10
6	RAS2	2	11571	0	84.721s	0	34	4.1067e-13	12.367s	97.088s	995m	1.7350e-10
6	ARAS	2	11571	30	85.185s	9.317s	42	4.4160e-13	8.274s	102.776s	995m	1.1161e-09
6	ARAS2	2	11571	30	84.613s	9.386s	25	7.1792e-13	9.627s	103.626s	995m	2.0457e-09
6	ARAS	2	11571	48	85.158s	15.001s	35	5.0449e-13	7.07s	107.229s	995m	7.7505e-10
6	ARAS2	2	11571	48	84.949s	14.793s	19	6.8769e-13	7.686s	107.428s	996m	8.8092e-10
6	ARAS	2	11571	60	84.639s	18.486s	32	3.6906e-13	6.692s	109.817s	996m	1.9139e-10
6	ARAS2	2	11571	60	84.605s	18.508s	17	4.9819e-13	7.067s	110.180s	996m	5.4985e-10
6	ARAS	2	11571	90	84.607s	27.814s	27	4.7670e-13	5.848s	118.269s	999m	2.6552e-10
6	ARAS2	2	11571	90	84.622s	27.724s	14	4.3724e-13	6.281s	118.627s	999m	7.1045e-10
6	ARAS	2	11571	120	84.786s	37.234s	30	3.7282e-13	6.367s	128.387s	1002m	2.8255e-10
6	ARAS2	2	11571	120	84.627s	37.165s	15	6.9431e-13	6.532s	128.324s	1000m	1.2447e-09
6	RAS	4	11683	0	92.284s	0	47	5.1802e-13	9.638s	101.922s	1085m	2.4840e-10
6	RAS2	4	11683	0	92.282s	0	29	3.7752e-13	11.724s	104.006s	1084m	3.2601e-10

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
6	ARAS	4	11683	30	92.319s	10.106s	30	4.5610e-13	6.702s	109.127s	1086m	1.7703e-09
6	ARAS2	4	11683	30	92.176s	10.131s	18	2.9512e-13	7.943s	110.250s	1084m	6.7018e-10
6	ARAS	4	11683	60	92.234s	20.183s	24	2.6951e-13	5.732s	118.149s	1088m	4.3986e-10
6	ARAS2	4	11683	60	92.175s	20.155s	13	2.1841e-13	6.228s	118.558s	1086m	5.4751e-10
6	ARAS	4	11683	120	92.121s	40.544s	23	2.5809e-13	5.585s	138.250s	1092m	1.1770e-09
6	ARAS2	4	11683	120	92.144s	40.468s	12	5.5436e-13	5.9s	138.512s	1089m	1.0766e-09
8	RAS	2	15505	0	59.636s	0	142	4.7186e-13	22.269s	81.905s	755m	4.9138e-10
8	RAS2	2	15505	0	60.097s	0	86	4.1495e-13	26.376s	86.473s	750m	4.4517e-10
8	ARAS	2	15505	30	58.658s	8.645s	122	5.8769e-13	19.326s	86.629s	759m	6.6061e-10
8	ARAS2	2	15505	30	59.712s	8.674s	80	7.8554e-13	24.655s	93.041s	752m	9.6768e-09
8	ARAS	2	15505	48	64.200s	13.764s	117	5.2332e-13	18.677s	96.641s	760m	7.4670e-10
8	ARAS2	2	15505	48	58.348s	13.725s	70	9.2887e-13	21.714s	93.787s	751m	5.8469e-09
8	ARAS	2	15505	60	59.561s	17.088s	113	5.6965e-13	18.094s	94.743s	760m	5.8549e-10
8	ARAS2	2	15505	60	58.411s	17.143s	71	8.4629e-13	22.18s	97.734s	751m	1.8348e-08
8	ARAS	2	15505	90	59.489s	25.774s	97	5.9890e-13	15.86s	101.123s	758m	6.7733e-10
8	ARAS2	2	15505	90	59.883s	25.801s	62	9.1923e-13	19.561s	105.245s	751m	2.9232e-08
8	ARAS	2	15505	120	58.376s	34.58s	87	5.7342e-13	14.117s	107.073s	758m	1.2887e-09
8	ARAS2	2	15505	120	59.689s	34.371s	55	7.0028e-13	17.445s	111.505s	752m	4.7041e-09

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
8	RAS	4	16023	0	70.476s	0	129	4.4935e-13	23.084s	93.560s	861m	3.1063e-10
8	RAS2	4	16023	0	69.433s	0	75	3.7936e-13	26.2s	95.633s	859m	3.9473e-10
8	ARAS	4	16023	30	69.757s	9.781s	107	4.5190e-13	19.456s	98.994s	865m	6.4476e-10
8	ARAS2	4	16023	30	69.038s	9.77s	65	9.2031e-13	22.987s	101.795s	858m	1.1914e-08
8	ARAS	4	16023	60	69.646s	19.488s	86	4.2094e-13	15.985s	105.119s	863m	5.9271e-10
8	ARAS2	4	16023	60	68.855s	19.524s	57	7.0805e-13	20.584s	108.963s	858m	4.0596e-09
8	ARAS	4	16023	120	68.765s	39.406s	56	5.4123e-13	10.898s	119.069s	860m	9.6520e-10
8	ARAS2	4	16023	120	68.700s	39.174s	34	8.4272e-13	12.771s	120.645s	857m	1.5320e-09
12	RAS	2	20979	0	25.695s	0	163	7.2396e-13	19.961s	45.656s	421m	6.6093e-10
12	RAS2	2	20979	0	25.747s	0	96	8.5933e-13	22.825s	48.572s	419m	5.8311e-10
12	ARAS	2	20979	12	26.081s	2.641s	156	6.1933e-13	19.264s	47.986s	426m	7.5086e-10
12	ARAS2	2	20979	12	25.748s	2.636s	92	6.3385e-13	21.992s	50.376s	418m	1.1862e-09
12	ARAS	2	20979	30	25.727s	6.671s	143	7.8111e-13	17.731s	50.129s	427m	8.6042e-10
12	ARAS2	2	20979	30	25.692s	6.658s	90	1.0353e-12	21.577s	53.927s	420m	6.4078e-09
12	ARAS	2	20979	60	25.890s	13.318s	134	6.7994e-13	16.596s	55.804s	427m	8.3967e-10
12	ARAS2	2	20979	60	25.710s	13.297s	92	8.2325e-13	21.878s	60.885s	421m	1.2343e-08
12	ARAS	2	20979	90	25.693s	20.005s	120	6.0517e-13	14.976s	60.674s	426m	8.5236e-10
12	ARAS2	2	20979	90	25.696s	19.986s	80	8.4678e-13	19.377s	65.059s	425m	9.2331e-09

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	$\|\|b - Ax\|\|/\|\|b\|\|$
12	ARAS	2	20979	120	25.702s	26.753s	104	6.7291e-13	13.326s	65.781s	432m	7.9725e-10
12	ARAS2	2	20979	120	26.225s	26.77s	74	9.6476e-13	18.26s	71.255s	432m	1.2508e-08
12	RAS	4	22428	0	30.601s	0	143	7.9955e-13	20.631s	51.232s	487m	7.1211e-10
12	RAS2	4	22428	0	30.739s	0	82	9.8313e-13	22.77s	53.509s	487m	1.2924e-09
12	ARAS	4	22428	30	31.207s	7.896s	121	7.6511e-13	17.49s	56.593s	492m	1.2604e-09
12	ARAS2	4	22428	30	30.579s	7.906s	90	7.6823e-13	25.412s	63.897s	489m	6.5557e-08
12	ARAS	4	22428	60	47.896s	21.549s	110	7.5160e-13	23.49s	92.935s	493m	1.9212e-09
12	ARAS2	4	22428	60	30.767s	15.721s	78	8.7974e-13	22.032s	68.520s	488m	1.2157e-07
12	ARAS	4	22428	120	43.178s	31.654s	78	6.2883e-13	12.597s	87.429s	501m	1.8933e-09
12	ARAS2	4	22428	120	30.651s	31.797s	53	8.5977e-13	15.426s	77.874s	500m	1.7617e-08
24	RAS	2	36239	0	14.376s	0	278	7.0041e-13	17.264s	31.640s	273m	7.3355e-09
24	RAS2	2	36239	0	14.386s	0	167	9.4329e-13	19.615s	34.001s	269m	4.7122e-08
24	ARAS	2	36239	30	14.380s	3.241s	264	7.3193e-13	16.842s	34.463s	282m	5.2667e-09
24	ARAS2	2	36239	30	14.397s	3.235s	163	1.1255e-12	19.809s	37.441s	277m	6.5445e-08
24	ARAS	2	36239	60	14.690s	6.536s	244	8.0859e-13	15.575s	36.801s	283m	5.4372e-09
24	ARAS2	2	36239	60	15.607s	6.531s	161	1.1359e-12	19.411s	41.549s	281m	1.6170e-07
24	ARAS	2	36239	90	14.385s	9.921s	225	8.0280e-13	14.5s	38.806s	283m	6.2505e-09
24	ARAS2	2	36239	90	14.380s	9.886s	156	1.0071e-12	18.867s	43.133s	287m	4.2054e-08

Continued on next page

Table A.1 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	$\|\|b - Ax\|\|/\|\|b\|\|$
24	ARAS	2	36239	120	14.428s	13.31s	211	7.7474e-13	13.695s	41.433s	284m	6.0730e-09
24	ARAS2	2	36239	120	14.391s	13.328s	155	1.0329e-12	18.961s	46.680s	281m	1.7198e-07
24	RAS	4	39109	0	18.477s	0	215	7.3545e-13	16.867s	35.344s	332m	4.2917e-09
24	RAS2	4	39109	0	18.457s	0	120	8.5152e-13	18.172s	36.629s	330m	1.2059e-08
24	ARAS	4	39109	30	18.511s	4.167s	196	9.0238e-13	15.693s	38.371s	339m	4.5571e-09
24	ARAS2	4	39109	30	18.473s	4.168s	131	9.6646e-13	19.994s	42.635s	336m	2.4994e-08
24	ARAS	4	39109	60	18.445s	8.396s	184	7.5547e-13	14.848s	41.689s	341m	4.2154e-09
24	ARAS2	4	39109	60	18.484s	8.394s	146	9.9981e-13	22.189s	49.067s	343m	8.0372e-07
24	ARAS	4	39109	120	18.943s	17.258s	161	7.2343e-13	12.972s	49.173s	342m	6.1055e-09
24	ARAS2	4	39109	120	18.449s	17.323s	134	9.2029e-13	20.367s	56.139s	343m	1.7210e-07

A.2 Running information of CASE_006 FR02

Table A.2: Runs information on CASE_006 PR02 using the one level MPI version of ARAS in PETSc

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
2	RAS	2	3696	0	396.432s	0	26	5.8952e-07	13.183s	409.615s	3423m	7.6325e-05
2	RAS2	2	3696	0	398.562s	0	14	1.0691e-06	14.508s	413.070s	3418m	5.9353e-05
2	ARAS	2	3696	10	396.180s	7.878s	18	5.6615e-07	9.979s	414.037s	3422m	7.4107e-05
2	ARAS2	2	3696	10	393.220s	7.734s	11	4.5566e-07	11.827s	412.781s	3417m	4.6030e-05
2	ARAS	2	3696	20	393.343s	15.609s	10	5.8997e-07	6.843s	415.795s	3416m	1.6436e-04
2	ARAS2	2	3696	20	393.419s	15.837s	7	7.6876e-07	9.273s	418.529s	3417m	1.1732e-04
2	ARAS	2	3696	40	396.411s	30.075s	3	2.7017e-08	4.144s	430.630s	3416m	3.5441e-06
2	ARAS2	2	3696	40	393.856s	31.018s	2	1.0733e-08	4.926s	429.800s	3411m	9.9744e-06
2	ARAS	2	3696	60	398.981s	46.523s	3	7.5328e-09	4.04s	449.544s	3416m	3.1572e-06
2	ARAS2	2	3696	60	393.093s	50.331s	2	6.6678e-09	5.031s	448.455s	3412m	1.0396e-05
2	RAS	4	3640	0	400.883s	0	21	3.2299e-07	11.548s	412.431s	3481m	2.4343e-05
2	RAS2	4	3640	0	400.802s	0	11	1.0686e-06	11.742s	412.544s	3476m	6.1363e-05
2	ARAS	4	3640	10	402.886s	7.938s	13	6.4153e-07	8.21s	419.034s	3486m	8.2195e-05
2	ARAS2	4	3640	10	403.488s	8.135s	7	1.0106e-06	9.211s	420.834s	3481m	5.8247e-05

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
2	ARAS	4	3640	20	401.608s	16.003s	5	5.6908e-07	5.085s	422.696s	3480m	3.5400e-05
2	ARAS2	4	3640	20	408.672s	15.302s	4	2.2918e-07	6.428s	430.402s	3481m	1.5037e-05
2	ARAS	4	3640	40	399.673s	31.631s	3	6.4906e-08	4.164s	435.468s	3480m	4.4234e-06
2	ARAS2	4	3640	40	402.945s	31.974s	2	3.8666e-08	5.147s	440.066s	3475m	1.1378e-05
4	RAS	2	7455	0	172.897s	0	58	4.7533e-08	16.268s	189.165s	1683m	1.1670e-04
4	RAS2	2	7455	0	172.942s	0	30	3.0938e-07	16.886s	189.828s	1675m	2.4465e-04
4	ARAS	2	7455	20	176.847s	9.846s	39	1.7936e-07	11.823s	198.516s	1684m	8.8462e-04
4	ARAS2	2	7455	20	175.472s	9.71s	22	1.4067e-06	13.049s	198.231s	1681m	1.2079e-02
4	ARAS	2	7455	40	178.830s	19.535s	21	1.4203e-07	7.228s	205.593s	1681m	5.9824e-04
4	ARAS2	2	7455	40	175.221s	19.532s	11	8.5783e-07	7.526s	202.279s	1682m	1.9388e-03
4	ARAS	2	7455	60	173.287s	29.447s	9	1.3551e-07	4.263s	206.997s	1682m	7.7906e-04
4	ARAS2	2	7455	60	172.634s	25.587s	4	1.3323e-06	3.956s	202.177s	1682m	3.0710e-03
4	ARAS	2	7455	80	173.098s	39.363s	7	2.1280e-07	3.737s	216.198s	1685m	7.8403e-04
4	ARAS2	2	7455	80	176.677s	38.985s	4	3.7224e-07	4.207s	219.869s	1684m	2.8428e-04
4	RAS	4	7511	0	184.052s	0	45	8.9861e-08	13.77s	197.822s	1742m	4.1706e-04
4	RAS2	4	7511	0	180.283s	0	23	3.0034e-07	13.925s	194.208s	1737m	1.0703e-03
4	ARAS	4	7511	20	180.912s	10.218s	27	1.9347e-07	8.989s	200.119s	1745m	5.6256e-04
4	ARAS2	4	7511	20	183.498s	10.229s	16	9.0938e-07	10.565s	204.292s	1742m	5.7459e-03

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
4	ARAS	4	7511	40	188.116s	20.403s	10	2.3604e-07	4.615s	213.134s	1741m	3.7485e-04
4	ARAS2	4	7511	40	178.666s	20.305s	5	1.5859e-06	4.866s	203.837s	1741m	2.9715e-03
4	ARAS	4	7511	80	181.830s	40.84s	9	8.4156e-08	4.356s	227.026s	1742m	1.3654e-04
4	ARAS2	4	7511	80	180.099s	40.996s	5	7.3858e-07	4.898s	225.993s	1742m	3.2755e-03
6	RAS	2	11571	0	84.805s	0	66	4.4300e-08	12.076s	96.881s	996m	7.5392e-05
6	RAS2	2	11571	0	85.586s	0	34	1.9227e-07	12.482s	98.068s	995m	3.6756e-04
6	ARAS	2	11571	12	84.696s	3.773s	56	9.6868e-08	10.436s	98.905s	998m	6.0978e-04
6	ARAS2	2	11571	12	84.711s	3.818s	33	9.0757e-07	12.129s	100.658s	995m	8.2999e-02
6	ARAS	2	11571	30	84.874s	9.349s	46	1.3592e-07	8.97s	103.193s	996m	8.8046e-04
6	ARAS2	2	11571	30	85.195s	9.479s	26	7.6382e-07	9.93s	104.604s	995m	2.2680e-02
6	ARAS	2	11571	60	84.890s	18.514s	40	9.3168e-08	7.989s	111.393s	996m	1.9610e-03
6	ARAS2	2	11571	60	84.705s	18.622s	24	4.5372e-07	9.283s	112.610s	996m	1.4444e-02
6	ARAS	2	11571	90	84.869s	27.812s	37	1.0204e-07	7.592s	120.273s	1000m	9.9166e-04
6	ARAS2	2	11571	90	84.666s	27.737s	20	7.8578e-07	8.263s	120.666s	999m	5.5156e-02
6	ARAS	2	11571	120	84.551s	37.073s	35	7.6575e-08	7.123s	128.747s	1004m	3.2391e-04
6	ARAS2	2	11571	120	84.649s	37.278s	20	1.3077e-06	8.011s	129.938s	1001m	7.6977e-02
6	RAS	4	11683	0	92.370s	0	48	3.4237e-08	9.908s	102.278s	1085m	1.2339e-04
6	RAS2	4	11683	0	92.236s	0	28	1.8320e-07	11.338s	103.574s	1084m	3.0851e-03

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
6	ARAS	4	11683	30	92.506s	10.155s	32	9.2803e-08	7.073s	109.734s	1086m	1.2126e-03
6	ARAS2	4	11683	30	93.037s	10.155s	19	5.8781e-07	8.29s	111.482s	1085m	1.9036e-02
6	ARAS	4	11683	60	92.211s	20.186s	27	1.7689e-07	6.249s	118.646s	1088m	1.1502e-02
6	ARAS2	4	11683	60	92.436s	20.114s	17	1.2152e-06	7.544s	120.094s	1086m	5.9229e-02
6	ARAS	4	11683	120	92.141s	40.515s	28	1.1124e-07	6.442s	139.098s	1093m	1.8618e-03
6	ARAS2	4	11683	120	92.531s	40.612s	19	1.2526e-06	8.306s	141.449s	1091m	2.3878e-01
8	RAS	2	15505	0	59.637s	0	136	3.0638e-08	21.446s	81.083s	755m	1.2017e-04
8	RAS2	2	15505	0	59.740s	0	79	1.4192e-07	24.325s	84.065s	749m	7.1981e-04
8	ARAS	2	15505	16	59.747s	4.693s	119	5.3936e-07	18.893s	83.333s	759m	3.2278e-03
8	ARAS2	2	15505	16	59.255s	4.634s	66	4.8591e-06	20.487s	84.376s	750m	4.9888e-01
8	ARAS	2	15505	30	59.523s	8.623s	111	2.8738e-07	17.908s	86.054s	757m	6.1101e-03
8	ARAS2	2	15505	30	60.003s	8.697s	72	1.8351e-06	22.336s	91.036s	750m	1.1901e-01
8	ARAS	2	15505	48	59.972s	13.807s	119	3.1134e-07	19.039s	92.818s	759m	5.5007e-03
8	ARAS2	2	15505	48	59.703s	13.778s	84	1.5178e-06	25.901s	99.382s	754m	3.4439e-01
8	ARAS	2	15505	60	59.737s	17.276s	115	2.8715e-07	18.505s	95.518s	760m	2.4579e-02
8	ARAS2	2	15505	60	61.507s	16.993s	81	1.4272e-06	24.908s	103.408s	753m	2.3509e-01
8	ARAS	2	15505	90	59.254s	25.639s	106	3.0838e-07	17.089s	101.982s	760m	1.6119e-02
8	ARAS2	2	15505	90	59.277s	25.607s	74	2.1246e-06	23.06s	107.944s	754m	2.6310e-01

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
8	ARAS	2	15505	120	59.589s	34.474s	99	2.7858e-07	15.954s	110.017s	759m	5.5780e-03
8	ARAS2	2	15505	120	61.468s	34.21s	67	1.3835e-06	20.854s	116.532s	754m	2.5133e-01
8	RAS	4	16023	0	69.603s	0	114	5.6714e-08	20.555s	90.158s	859m	5.6455e-04
8	RAS2	4	16023	0	68.880s	0	63	2.5968e-07	22.285s	91.165s	857m	2.7827e-03
8	ARAS	4	16023	30	69.128s	9.751s	98	6.4935e-07	17.792s	96.671s	863m	8.8693e-03
8	ARAS2	4	16023	30	69.125s	9.734s	64	1.8858e-06	22.665s	101.524s	858m	1.2429e+00
8	ARAS	4	16023	60	68.785s	19.534s	87	6.0482e-07	16.126s	104.445s	863m	5.9284e-03
8	ARAS2	4	16023	60	68.535s	19.539s	63	2.5640e-06	22.49s	110.564s	860m	1.2612e+00
8	ARAS	4	16023	120	69.238s	39.526s	67	6.1850e-07	12.767s	121.531s	862m	4.3584e-02
8	ARAS2	4	16023	120	68.816s	39.195s	46	1.8522e-06	16.677s	124.688s	859m	1.1633e+00
12	RAS	2	20979	0	25.709s	0	174	1.5415e-09	21.385s	47.094s	422m	5.0550e-06
12	RAS2	2	20979	0	25.725s	0	92	5.7606e-08	21.904s	47.629s	418m	2.0423e-04
12	ARAS	2	20979	12	25.714s	2.647s	155	1.0663e-07	21.803s	50.164s	425m	7.0272e-04
12	ARAS2	2	20979	12	25.747s	2.633s	96	1.3910e-06	22.942s	51.322s	419m	5.0750e-02
12	ARAS	2	20979	30	25.865s	6.579s	148	1.7260e-07	18.331s	50.775s	427m	1.3086e-03
12	ARAS2	2	20979	30	25.897s	6.688s	93	1.9389e-06	22.089s	54.674s	422m	3.6116e-02
12	ARAS	2	20979	60	25.728s	13.35s	143	1.4230e-07	17.615s	56.693s	428m	4.5309e-04
12	ARAS2	2	20979	60	26.003s	13.352s	122	1.8590e-06	28.912s	68.267s	425m	1.7962e+00

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	||b - Ax||/||b||
12	ARAS	2	20979	120	25.713s	26.898s	137	1.4126e-07	17.356s	69.967s	432m	9.4629e-04
12	ARAS2	2	20979	120	25.728s	26.747s	97	4.7017e-06	23.399s	75.874s	431m	5.4627e-02
12	RAS	4	22428	0	30.581s	0	147	2.8053e-09	21.059s	51.640s	487m	5.7828e-06
12	RAS2	4	22428	0	30.715s	0	77	7.7126e-08	21.589s	52.304s	486m	1.2583e-03
12	ARAS	4	22428	30	64.442s	13.546s	119	2.0454e-07	26.643s	104.631s	492m	1.2051e-03
12	ARAS2	4	22428	30	34.066s	7.782s	76	2.0312e-06	21.428s	63.276s	487m	2.9865e-02
12	ARAS	4	22428	60	30.914s	15.599s	116	2.1011e-07	16.998s	63.511s	493m	1.0320e-03
12	ARAS2	4	22428	60	31.499s	15.768s	92	1.4888e-06	25.98s	73.247s	489m	4.6685e-02
12	ARAS	4	22428	120	30.761s	31.675s	128	1.8417e-07	18.546s	80.982s	502m	4.6449e-01
12	ARAS2	4	22428	120	30.793s	31.896s	92	1.4336e-06	26.082s	88.771s	502m	1.0293e-01
24	RAS	2	36239	0	15.027s	0	500	1.4200e-10	31.921s	46.948s	284m	1.2014e-04
24	RAS2	2	36239	0	14.401s	0	162	2.8865e-08	18.963s	33.364s	269m	1.9385e-03
24	ARAS	2	36239	24	14.612s	2.581s	250	2.1637e-07	18.577s	35.770s	279m	8.3867e-03
24	ARAS2	2	36239	24	14.558s	2.592s	196	2.2347e-06	28.929s	46.079s	278m	1.3986e+02
24	ARAS	2	36239	30	14.418s	3.261s	245	4.3416e-07	15.679s	33.358s	281m	2.3818e-02
24	ARAS2	2	36239	30	14.385s	3.238s	141	7.6291e-06	17.08s	34.703s	275m	3.8821e-01
24	ARAS	2	36239	60	14.388s	6.548s	244	3.2889e-07	15.68s	36.616s	283m	2.7190e-03
24	ARAS2	2	36239	60	14.393s	6.538s	168	2.2425e-06	20.126s	41.057s	282m	6.1326e-01

Continued on next page

Table A.2 – continued from previous page

p	Prec.	Ov.	n	q	Fact. Time	Build P Time	It.	Rel. Res.	Sol. Time	Tot. Time	Max. Mem. Loc.	$\|\|b - Ax\|\|/\|\|b\|\|$
24	ARAS	2	36239	120	14.418s	13.322s	233	3.4344e-07	15.156s	42.896s	285m	6.0710e-03
24	ARAS2	2	36239	120	14.449s	13.315s	170	2.0333e-06	20.688s	48.452s	282m	1.0605e+00
24	RAS	4	39109	0	18.544s	0	500	7.9742e-11	39.656s	58.200s	346m	1.0730e-03
24	RAS2	4	39109	0	19.457s	0	114	2.9586e-08	17.318s	36.775s	330m	8.7528e-04
24	ARAS	4	39109	30	18.562s	4.17s	184	7.4420e-07	14.824s	37.556s	339m	2.2852e-02
24	ARAS2	4	39109	30	19.902s	4.176s	120	5.4976e-06	18.4s	42.478s	336m	8.5747e-01
24	ARAS	4	39109	60	18.474s	8.399s	217	3.8548e-07	17.377s	44.250s	343m	4.2938e-02
24	ARAS2	4	39109	60	19.426s	8.451s	156	8.7372e-05	23.745s	51.622s	343m	7.4277e+02
24	ARAS	4	39109	120	18.438s	17.326s	260	4.6484e-07	20.616s	56.380s	348m	1.1710e+01
24	ARAS2	4	39109	120	18.422s	17.304s	186	1.0314e-04	28.359s	64.085s	344m	9.9450e+02

Bibliography

[Amestoy *et al.* 2001] P. R. Amestoy, I. S. Duff, J. Koster and J.-Y. L'Excellent. *A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling.* SIAM Journal on Matrix Analysis and Applications, vol. 23, no. 1, pages 15–41, 2001. (Cited on page 101.)

[Amestoy *et al.* 2006] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet. *Hybrid scheduling for the parallel solution of linear systems.* Parallel Computing, vol. 32, no. 2, pages 136–156, 2006. (Cited on page 101.)

[Axelsson 1996] O. Axelsson. Iterative solution methods, chapitre 5. Cambridge University Press, 1996. (Cited on page 65.)

[Balay *et al.* 1997] S. Balay, W. D. Gropp, L. Curfman McInnes and B. F. Smith. *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries.* In E. Arge, A. M. Bruaset and H. P. Langtangen, Modern Software Tools in Scientific Computing, pages 163–202. Birkhäuser Press, 1997. (Cited on page 104.)

[Balay *et al.* 2008] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith and H. Zhang. *PETSc Users Manual.* Rapport technique ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008. (Cited on page 105.)

[Balay *et al.* 2009] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith and H. Zhang. *PETSc Web page*, 2009. http://www.mcs.anl.gov/petsc. (Cited on page 104.)

[Bar-On & Leoncini 2002] I. Bar-On and M. Leoncini. *Reliable parallel solution of bidiagonal systems.* Numer. Math., vol. 90, pages 415–440, 2002. (Cited on page 58.)

[Baranger *et al.* 2008] J. Baranger, M. Garbey and F. Oudin-Dardun. *The Aitken-like acceleration of the Schwarz method on nonuniform Cartesian grids.* SIAM J. Sci. Comput., vol. 30, no. 5, pages 2566–2586, 2008. (Cited on pages 59 and 71.)

[Barberou *et al.* 2003] N. Barberou, M. Garbey, M. Hess, M. Resh, T. Rossi, J. Toivanen and D. Tromeur-Dervout. *Efficient metacomputing of elliptic linear and non-linear problems.* Journal of Parallel and Distributed Computing, vol. 63, no. 5, pages 564–577, 2003. (Cited on page 59.)

[Beauwens 2004] R. Beauwens. *Iterative solution methods.* Appl. Numer. Math., vol. 51, no. 4, pages 437–450, 2004. (Cited on pages 25, 28 and 64.)

[Berenguer *et al.* 20YYa] L. Berenguer, T. Dufaud, T. Pham and D. Tromeur-Dervout. *On-the-fly Singular Value Decomposition for Aitken's Acceleration of the Schwarz Domain Decomposition Method.* In Proc. Int. Conf. Parallel Computing 2011, 20YY. Submitted. (Cited on pages 90 and 96.)

[Berenguer *et al.* 20YYb] L. Berenguer, T. Dufaud and D. Tromeur-Dervout. *Aitken's acceleration of the Schwarz process using singular value decomposition for heterogeneous 3D groundwater flow problems.* Computers and Fluids, 20YY. Submitted. (Cited on pages 90 and 96.)

[Boursier *et al.* 2005] I. Boursier, D. Tromeur-Dervout and Y. Vassilevsky. *Aitken-Schwarz methods with nonmatching finite elements and spectral elements grids for the parallel simulationof an underground waste disposal site modelized upscaling.* In G. Winter, A. Ecer, J.Periaux, N.Satofuka and P.Fox editors, Proc. Int. Conf. PARCFD04, 2005. (Cited on page 52.)

[Bramble *et al.* 1986] J. H. Bramble, J. E. Pasciak and A. H. Schatz. *The construction of preconditioners for elliptic problems by substructuring. I.* Math. Comp., vol. 47, no. 175, pages 103–134, 1986. (Cited on pages 28, 36, 40 and 41.)

[Brent & Luk 1985] R.P. Brent and F.T. Luk. *The solution of singular-value and symmetric eigenvalue problems on multiprocessors arrays.* SIAM J. ScI. Stat. Comput., vol. 6, no. 1, pages 69–84, 1985. (Cited on page 58.)

[Cai & Sarkis 1999] X.-C. Cai and M. Sarkis. *A restricted additive Schwarz preconditioner for general sparse linear systems.* SIAM J. Sci. Comput., vol. 21, no. 2, pages 792–797 (electronic), 1999. (Cited on pages 27 and 36.)

[Cai *et al.* 1994] X.-C. Cai, W. D. Gropp and D. E. Keyes. *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems.* Numer. Linear Algebra Appl., vol. 1, no. 5, pages 477–504, 1994. (Cited on page 36.)

[Cai *et al.* 2004] X.-C. Cai, M. Dryja and M. Sarkis. *A restricted additive Schwarz preconditioner with harmonic overlap for symmetric positive definite linear systems.* Cubo, vol. 6, no. 4, pages 73–95, 2004. (Cited on page 36.)

[Carrier *et al.* 1988] J. Carrier, L. Greengard and V. Rokhlin. *A fast adaptive multipole algorithm for particle simulations.* SIAM J. Sci. Statist. Comput., vol. 9, no. 4, pages 669–686, 1988. (Cited on page 58.)

[Carvalho *et al.* 2000] L. M. Carvalho, L. Giraud and P. Le Tallec. *Algebraic two-level preconditioners for the Schur complement method.* SIAM J. Sci. Comput., vol. 22, no. 6, pages 1987–2005, 2000. (Cited on pages 28 and 40.)

[Carvalho *et al.* 2001] L. M. Carvalho, L. Giraud and G. Meurant. *Local preconditioners for two-level non-overlapping domain decomposition methods.* Numer.

Linear Algebra Appl., vol. 8, no. 4, pages 207–227, 2001. (Cited on pages 28 and 40.)

[Chevalier & Nataf 1998] P. Chevalier and F. Nataf. *An optimized order 2 (OO2) method for the Helmholtz equation.* C. R. Acad. Sci. Paris Sér. I Math., vol. 326, no. 6, pages 769–774, 1998. (Cited on pages 27 and 34.)

[Ciarlet 1994] P. G. Ciarlet. Introduction à l'analyse numérique matricielle et à l'optimisation, chapitre 5. Masson, 1994. (Cited on page 65.)

[Davis & Hu 20YY] T. A. Davis and Y. Hu. *The University of Florida Sparse Matrix Collection.* ACM Transactions on Mathematical Software (to appear), 20YY. http://www.cise.ufl.edu/research/sparse/matrices. (Cited on pages 10 and 16.)

[de Sturler 1995] E. de Sturler. *Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer.* Appl. Numer. Math., vol. 19, no. 1-2, pages 129–146, 1995. Massively parallel computing and applications (Amsterdam, 1993–1994). (Cited on pages 25 and 28.)

[de Sturler 1996] E. de Sturler. *Nested Krylov methods based on GCR.* J. Comput. Appl. Math., vol. 67, no. 1, pages 15–41, 1996. (Cited on pages 26 and 30.)

[Demmel & Kahan 1990] J. Demmel and W. Kahan. *Accurate Singular values of bidiagonal matrices.* SIAM J. ScI. Stat. Comput., vol. 11, no. 2, pages 873–912, 1990. (Cited on page 58.)

[Dongara 1983] J.J. Dongara. *Improving the accuracy of computed singular values.* Siam J. Sci. Stat. Comput., vol. 4, no. 4, pages 712–719, 1983. (Cited on page 58.)

[Drmač & Veselić 2008a] Z. Drmač and K. Veselić. *New fast and accurate Jacobi SVD algorithm. I.* SIAM J. Matrix Anal. Appl., vol. 29, no. 4, pages 1322–1342, 2008. (Cited on page 58.)

[Drmač & Veselić 2008b] Z. Drmač and K. Veselić. *New fast and accurate Jacobi SVD algorithm. II.* SIAM J. Matrix Anal. Appl., vol. 29, no. 4, pages 1343–1362, 2008. (Cited on page 58.)

[Dryja & Widlund 1990] M. Dryja and O. B. Widlund. *Some domain decomposition algorithms for elliptic problems.* In Iterative methods for large linear systems (Austin, TX, 1988), pages 273–291. Academic Press, Boston, MA, 1990. (Cited on page 36.)

[Dufaud & Tromeur-Dervout 2010a] T. Dufaud and D. Tromeur-Dervout. *Adaptive Aitken-Schwarz Method for Non Separable Operator on Multiprocessor Systems.* In Rupak Biswas and NASA Advanced Supercomputing Division

NASA Ames Research Center, Parallel Computational Fluid Dynamics Recent Advances & Future Directions, pages 297–305. DEStech Publications, 2010. (Cited on pages 90, 96 and 101.)

[Dufaud & Tromeur-Dervout 2010b] T. Dufaud and D. Tromeur-Dervout. *Aitken's acceleration of the restricted additive Schwarz preconditioning using coarse approximations on the interface.* C. R. Math. Acad. Sci. Paris, vol. 348, no. 13-14, pages 821–824, 2010. (Cited on pages 46, 59 and 73.)

[Dufaud & Tromeur-Dervout 2011] T. Dufaud and D. Tromeur-Dervout. *Numerical Investigations and Parallel Implementation of the ARAS2 Preconditioning Technique.* In P. Ivanyi and B.H.V. Topping, 95 PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED, GRID AND CLOUD COMPUTING FOR ENGINEERING, 2011. (Cited on pages 73, 90 and 104.)

[Dufaud & Tromeur-Dervout 20YYa] T. Dufaud and D. Tromeur-Dervout. *ARAS2 preconditioning technique for CFD industrial cases.* In ddm.org, Domain Decomposition Method 2011, San Diego, CA, 20YY. Submitted. (Cited on pages 46 and 59.)

[Dufaud & Tromeur-Dervout 20YYb] T. Dufaud and D. Tromeur-Dervout. *Efficient parallel implementation of the fully algebraic preconditioning technique ARAS2.* Advances in Engineering Software, 20YY. Submitted. (Cited on pages 90 and 104.)

[Efstathiou & Gander 2003] E. Efstathiou and M. J. Gander. *Why restricted additive Schwarz converges faster than additive Schwarz.* BIT, vol. 43, no. suppl., pages 945–959, 2003. (Cited on pages 28 and 36.)

[Eisenstat & Ming 1995] S.C. Eisenstat and G. Ming. *A divide-and-conquer algorithm for the bidiagonal SVD.* SIAM J. Matrix Anal. Appl., vol. 16, no. 1, pages 79–92, 1995. (Cited on page 58.)

[Eisenstat *et al.* 1983] S. C. Eisenstat, H. C. Elman and M. H. Schultz. *Variational iterative methods for nonsymmetric systems of linear equations.* SIAM J. Numer. Anal., vol. 20, no. 2, pages 345–357, 1983. (Cited on pages 26, 27, 29, 30 and 35.)

[Engquist & Zhao 1998] B. Engquist and H.-K. Zhao. *Absorbing boundary conditions for domain decomposition.* Appl. Numer. Math., vol. 27, no. 4, pages 341–365, 1998. Absorbing boundary conditions. (Cited on pages 26, 27, 31, 33 and 34.)

[Flannery *et al.* 2007] B. Flannery, W. H. Press, S. Teukolsky and W. Vetterling. Numerical recipes: The art of scientific computing. Cambridge University Press, third édition, 2007. (Cited on page 57.)

[Frayssé *et al.* 2009] V. Frayssé, L. Giraud and S. Gratton. *Algorithm 881: a set of flexible GMRES routines for real and complex arithmetics on high-performance computers.* ACM Trans. Math. Software, vol. 35, no. 2, pages Art. 13, 12, 2009. (Cited on pages 26 and 30.)

[Frullone & Tromeur-Dervout 2006] A. Frullone and D. Tromeur-Dervout. *A new formulation of NUDFT applied to Aitken-Schwarz DDM on nonuniform meshes.* In Parallel Computational Fluid Dynamics 2005, pages 493–500, 2006. (Cited on pages 50, 52, 59 and 71.)

[Gander *et al.* 2002] M. J. Gander, F. Magoulès and F. Nataf. *Optimized Schwarz methods without overlap for the Helmholtz equation.* SIAM J. Sci. Comput., vol. 24, no. 1, pages 38–60 (electronic), 2002. (Cited on pages 27 and 34.)

[Gander *et al.* 2007] M. J. Gander, L. Halpern, F. Magoulès and F.-X. Roux. *Analysis of patch substructuring methods.* Int. J. Appl. Math. Comput. Sci., vol. 17, no. 3, pages 395–402, 2007. (Cited on pages 28 and 38.)

[Garbey & Tromeur-Dervout 2001] M. Garbey and D. Tromeur-Dervout. *Two level domain decomposition for Multiclusters.* In T. Chan & Al editors, 12th Int. Conf. on Domain Decomposition Methods DD12, pages 325–339. ddm.org, 2001. (Cited on pages 27 and 34.)

[Garbey & Tromeur-Dervout 2002] M. Garbey and D. Tromeur-Dervout. *On some Aitken like acceleration of the Schwarz method.* Internat. J. Numer. Methods Fluids, vol. 40, no. 12, pages 1493–1513, 2002. LMS Workshop on Domain Decomposition Methods in Fluid Mechanics (London, 2001). (Cited on pages 27, 34, 59 and 71.)

[Garbey 2005] M. Garbey. *Acceleration of the Schwarz Method for Elliptic Problems.* SIAM J. Sci. Comput., vol. 26, no. 6, pages 1871–1893, 2005. (Cited on pages 50, 51, 59, 67 and 71.)

[Gerardo-Giorda & Nataf 2005] L. Gerardo-Giorda and F. Nataf. *Optimized Schwarz methods for unsymmetric layered problems with strongly discontinuous and anisotropic coefficients.* J. Numer. Math., vol. 13, no. 4, pages 265–294, 2005. (Cited on pages 27 and 34.)

[Giraud *et al.* 2010] L. Giraud, S. Gratton, X. Pinel and X. Vasseur. *Flexible GMRES with deflated restarting.* SIAM J. Sci. Comput., vol. 32, no. 4, pages 1858–1878, 2010. (Cited on pages 26 and 30.)

[Golub & Kahan 1965] G. Golub and W. Kahan. *Calculating the singular values and pseudo-inverse of a matrix.* J. Siam Numer. Anal. Ser. B, vol. 2, no. 2, pages 205–224, 1965. (Cited on page 58.)

[Gropp & Keyes 1992] W. D. Gropp and D. E. Keyes. *Parallel performance of domain-decomposed preconditioned Krylov methods for PDEs with locally*

uniform refinement. SIAM J. Sci. Statist. Comput., vol. 13, no. 1, pages 128–145, 1992. (Cited on page 36.)

[Haidar 2008] Azzam Haidar. *On the parallel scalability of hybrid linear solvers for large 3D problems.* PhD thesis, INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE, 2008. (Cited on page 91.)

[Hernandez *et al.* 2005] V. Hernandez, J. E. Roman and V. Vidal. *SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems.* ACM Trans. Math. Software, vol. 31, no. 3, pages 351–362, 2005. (Cited on page 108.)

[Jessup & Sorensen 1994] E.R. Jessup and D.C. Sorensen. *A parallel Algorithm for computing the singular value decomposition of a matrix.* SIAM J. Matrix Anal. Appl., vol. 15, no. 2, pages 530–548, 1994. (Cited on page 58.)

[Karypis *et al.* 2003] G. Karypis, K. Schloegel and V. Kumar. *PARMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library.* Rapport technique Version 3.1, University of Minnesota, Department of Computer Science and Engineering, 2003. (Cited on pages 12 and 18.)

[Khoromskij & Wittum 2004] B. N. Khoromskij and G. Wittum. Numerical solution of elliptic differential equations by reduction to the interface, volume 36 of *Lecture Notes in Computational Science and Engineering.* Springer-Verlag, Berlin, 2004. (Cited on pages 28 and 40.)

[Li & Saad 2006] Z. Li and Y. Saad. *SchurRAS: a restricted version of the overlapping Schur complement preconditioner.* SIAM J. Sci. Comput., vol. 27, no. 5, pages 1787–1801 (electronic), 2006. (Cited on pages 28 and 37.)

[Lions 1988] P.-L. Lions. *On the Schwarz alternating method. I.* In First International Symposium on Domain Decomposition Methods for Partial Differential Equations (Paris, 1987), pages 1–42. SIAM, Philadelphia, PA, 1988. (Cited on pages 27 and 33.)

[Lions 1990] P.-L. Lions. *On the Schwarz alternating method. III. A variant for nonoverlapping subdomains.* In Third International Symposium on Domain Decomposition Methods for Partial Differential Equations (Houston, TX, 1989), pages 202–223. SIAM, Philadelphia, PA, 1990. (Cited on page 33.)

[Marini & Quarteroni 1989] L. D. Marini and A. Quarteroni. *A relaxation procedure for domain decomposition methods using finite elements.* Numer. Math., vol. 55, no. 5, pages 575–598, 1989. (Cited on pages 27 and 33.)

[Martikainen *et al.* 2002] J. Martikainen, R. A. E. Mäkinen, T. Rossi and J. Toivanen. *A preconditioner for linear elasticity problems.* In Domain decomposition methods in science and engineering (Lyon, 2000), Theory Eng. Appl. Comput. Methods, pages 429–436. Internat. Center Numer. Methods Eng. (CIMNE), Barcelona, 2002. (Cited on pages 25 and 29.)

[Mascagni & Srinivasan 2000] M Mascagni and A Srinivasan. *Algorithm 806. SPRNG: A scalable library for pseudorandom number generation (vol 26, pg 436, 2000).* ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, vol. 26, no. 4, pages 618–619, 2000. (Cited on page 108.)

[Meijerink & van der Vorst 1981] J. A. Meijerink and H. A. van der Vorst. *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems.* J. Comput. Phys., vol. 44, no. 1, pages 134–155, 1981. (Cited on pages 25 and 28.)

[MPI 2009] The MPI Forum, University of Tennessee, KnoxVille, Tennessee. *MPI: A Message-Passing Interface Standard, Version 2.2*, September 4 2009. http://www.mpi-forum.org/docs/mpi-2.2/. (Cited on page 97.)

[Notay 2010] Y. Notay. *An aggregation-based algebraic multigrid method.* Electronic Transactions on Numerical Analysis, vol. 37, 2010. (Cited on page 101.)

[Parks *et al.* 2006] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson and S. Maiti. *Recycling Krylov subspaces for sequences of linear systems.* SIAM J. Sci. Comput., vol. 28, no. 5, pages 1651–1674 (electronic), 2006. (Cited on pages 26 and 31.)

[Quarteroni & Valli 1999] A. Quarteroni and A. Valli. Domain decomposition methods for partial differential equations. Numerical Mathematics and Scientific Computation. The Clarendon Press Oxford University Press, New York, 1999. Oxford Science Publications. (Cited on pages 26 and 31.)

[Saad & Schultz 1986] Y. Saad and M. H. Schultz. *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Statist. Comput., vol. 7, no. 3, pages 856–869, 1986. (Cited on pages 26 and 29.)

[Saad 1993] Y. Saad. *A flexible inner-outer preconditioned GMRES algorithm.* SIAM J. Sci. Comput., vol. 14, no. 2, pages 461–469, 1993. (Cited on pages 26 and 30.)

[Saad 2003] Y. Saad. Iterative methods for sparse linear systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, second édition, 2003. (Cited on pages 26 and 30.)

[St-Cyr *et al.* 2007] A. St-Cyr, M. J. Gander and S. J. Thomas. *Optimized multiplicative, additive, and restricted additive Schwarz preconditioning.* SIAM J. Sci. Comput., vol. 29, no. 6, pages 2402–2425 (electronic), 2007. (Cited on pages 27, 28, 33 and 36.)

[Starke 1994] G. Starke. *Alternating direction preconditioning for nonsymmetric systems of linear equations.* SIAM J. Sci. Comput., vol. 15, no. 2, pages 369–384, 1994. Iterative methods in numerical linear algebra (Copper Mountain Resort, CO, 1992). (Cited on pages 25 and 29.)

[Stoer & Bulirsch 2002] J. Stoer and R. Bulirsch. Introduction to numerical analysis, volume 12 of *Texts in Applied Mathematics*. Springer-Verlag, New York, third édition, 2002. Translated from the German by R. Bartels, W. Gautschi and C. Witzgall. (Cited on pages 45 and 61.)

[Tromeur-Dervout & Vassilevski 2006] D. Tromeur-Dervout and Y. Vassilevski. *Choice of initial guess in iterative solution of series of systems arising in fluid flow simulations*. J. Comput. Phys., vol. 219, no. 1, pages 210–227, 2006. (Cited on pages 26 and 31.)

[Tromeur-Dervout 2009] D. Tromeur-Dervout. *Meshfree Adaptative Aitken-Schwarz Domain Decomposition with application to Darcy Flow*. In BHV Topping and P Ivanyi, PARALLEL, DISTRIBUTED AND GRID COMPUTING FOR ENGINEERING, volume 21 of *Computational Science Engineering and Technology Series*, pages 217–250. SAXE-COBURG PUBLICATIONS, 2009. (Cited on pages 55, 59 and 64.)

[van Gijzen 1995] M. B. van Gijzen. *A polynomial preconditioner for the GMRES algorithm*. J. Comput. Appl. Math., vol. 59, no. 1, pages 91–107, 1995. (Cited on pages 25 and 28.)

[Vuik 1995] C. Vuik. *New insights in GMRES-like methods with variable preconditioners*. J. Comput. Appl. Math., vol. 61, no. 2, pages 189–204, 1995. (Cited on pages 26 and 30.)

[Young & Jea 1980] D. M. Young and K. C. Jea. *Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods*. Linear Algebra Appl., vol. 34, pages 159–194, 1980. (Cited on pages 26 and 31.)

Contribution to the development of Aitken Restricted Additive Schwarz preconditioning and application to linear systems arising from automatic differentiation of compressible Navier-Stokes solutions with respect to the simulation's parameters.

Abstract: A two level preconditioner, based on the Aitken acceleration technique of a sequence of q interface's solution vectors of the Restricted Additive Schwarz iterative process, is designed. This new technique, called ARAS(q), uses a coarse approximation of the solution on the interface. Different methods are discussed, leading to the development of an approximation technique by Singular Value Decomposition of the sequence of vectors. Parallel implementations of Aitken-Schwarz methods are proposed, and the study leads to a fully algebraic one-level and two-level MPI implementation of ARAS(q) written into the PETSc library framework. This fully parallel and algebraic code gives an adaptive tool to solve linear systems such as those arising from automatic differentiation of compressible Navier-Stokes solution with respect to the simulation's parameters.

Keywords: Aitken acceleration, Algebraic technique, Coarse approximation space, Parallel computation, Resticted Additive Schwarz, Singular Value Decomposition, Two-level preconditioning, Two-level MPI implementation

---

Contribution au développement du préconditionnement Aitken Schwarz Additif Restreint et son application aux systèmes linéaires issus de la différentiation automatique des solutions de Navier-Stokes dépendant des paramètres de la simulation.

Résumé : Un préconditionneur à deux niveaux, reposant sur la technique d'accélération d'Aitken d'une suite de q vecteurs solutions de l'interface d'un processus itératif de Schwarz Additif Restreint, est conçu. Cette nouvelle technique, dénomée ARAS(q), utilise une approximation grossière de la solution sur l'interface. Différentes méthodes sont proposées, aboutissant au développement d'une technique d'approximation par Décomposition en Valeures Singulières de la suite de vecteurs. Des implémentations parallèles des méthodes d'Aitken-Schwarz sont proposées et l'étude conduit à l'implémentation d'un code totalement algébrique, sur un ou deux niveaux de parallélisation MPI, écrit dans l'environnement de la bibliothèque PETSc. Cette implémentation pleinement parallèle et algébrique procure un outil flexible pour la résolution de systèmes linéaires tels que ceux issus de la différentiation automatique des solutions de Navier-Stokes dépendant des paramètres de la simulation.

Mots clefs : Accélération d'Aitken, Calcul parallèle, Décomposition en Valeures Singulières, Espace d'approximation grossier, Implémentation MPI deux niveaux, Préconditionnement deux niveaux, Schwarz Additif Restreint, Technique algèbrique