N° d'ordre : 4464 ANNÉE 2011

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Désiré NUENTSA WAKAM

préparée à l'unité de recherche INRIA
Institut National de Recherche en Informatique et Automatique
Composante universitaire ISTIC

Parallélisme et Robustesse dans les solveurs hybrides pour grands systèmes linéaires : Application à l'optimisation en dynamique des fluides

Thèse soutenue à Rennes
le 07 decembre 2011

devant le jury composé de :

Luc GIRAUD
Directeur de recherche, INRIA, Bordeaux / rapporteur

François-Xavier ROUX
Professeur associé, UPMC, Paris / rapporteur

Stéphane AUBERT
PDG, FLUOREM S.A, Lyon / examinateur

Laura GRIGORI
Chargée de recherche, INRIA, Saclay / examinateur

Bernard PHILIPPE
Directeur de recherche émérite, INRIA / examinateur

Jocelyne ERHEL
Directrice de recherche, INRIA / directrice de thèse

Édouard CANOT
Chargé de recherche, CNRS-IRISA / co-directeur de thèse

"À mon arrière grand-mère, ma grand-mère et ma maman, ces trois magnifiques femmes qui m'ont appris qu'en toutes choses, il faut des principes, de l'effort et de la persévérance"
Mères, puissiez-vous retrouver entre les lignes de ce rapport les prémices de votre éducation.

Remerciements

Et maintenant que le temps des récoltes est arrivé dans le bassin rennais, il me vient un grand plaisir de remercier vivement et sincèrement ceux qui ont contribué à ce que la graine puisse porter du fruit:

Ma profonde gratitude va en premier à Mme Jocelyne Erhel mon encadreur. Merci Jocelyne d'avoir guidé mes pas au cours de ces travaux de thèse, de m'avoir appris à travailler de façon autonome sans pour autant m'éparpiller. Merci pour toutes les corrections que tu as proposées aux premieres versions de ce manuscrit. Merci surtout pour ta disponibilité, ton encouragement et pour tous tes conseils par rapport à la poursuite de ma carrière professionnelle.

Je remercie M. Édouard Canot qui m'a encadré au debut de ma thèse et qui a été ensuite très disponible à apporter des éclaircissements chaque fois que j'ai eu recours à lui. Au delà de la recherche, j'aimerais te remercier, Édouard, pour ta constante sollicitude durant ces trois années, pour les repas en famille et pour les sorties à Lyon, Vannes et ailleurs.

Merci à M. Luc Giraud et M. François-Xavier Roux d'avoir accepté la charge d'évaluer cette thèse. J'ai toujours été très heureux d'échanger avec Luc à chaque fois lors de nos rencontres pendant les conférences et workshops et je lui en suis très reconnaissant. Merci à François-Xavier pour toutes les suggestions pour la suite de ce travail.

Merci à Mme Laura Grigori et M. Stéphane Aubert d'avoir accepté d'examiner cette thèse. Laura m'a beaucoup encouragé durant ma thèse à nouer des collaborations avec les autres chercheurs et je la remercie de tout coeur ici.

Je suis très reconnaissant envers M. Bernard Philippe avec qui j'ai eu beaucoup d'échanges durant ma thèse. Merci pour toutes les remarques qui m'ont aidé à mieux cerner certains aspects de mon travail. Au-delà de ses grandes qualités pédagogiques et de recherche, Bernard est une personne très simple et très aimable qui n'a pas hésité à venir vers moi chaque fois qu'il en avait l'occasion.

I would like to thank M. Bill Gropp for his welcome during my visit at NCSA and for suggesting many improvements in this work.

Un merci tout particulier à François Pacull de FLUOREM avec qui j'ai eu une collaboration intensive à la fin de cette thèse: merci François d'avoir pris tout le temps necessaire pour m'expliquer les problèmes liés à vos matrices, merci pour les nombreuses suggestions à l'amélioration de mon schéma hybride et merci pour tout le temps que tu as pris pour rédiger et améliorer une partie de notre article.

Merci à M. Emmanuel Kamgnia qui m'a donné l'occasion de faire de la recherche dans

ce domaine, merci d'avoir eu confiance en moi. J'espère avoir été à la hauteur de tes attentes.

Je tiens à remercier toute l'équipe SAGE au sein de laquelle j'ai fait cette thèse. Ceux que j'ai connu en premier: Fabienne et Cécile, Caroline, Noha, Amine, Guy et Mohamad, et tous les autres : Géraldine, Nadir, Baptiste, Denis, Sinda, Souhila, Julia et Mestapha. Je remercie tout particulierement Nadir pour toute son aide technique et pour sa bonne humeur légendaire qui a meublé mon passage dans l'équipe, Baptiste pour les nombreuses discussions qu'on a eu pendant notre fin de thèse et bien sûr sa serie quotidienne d'histoires drôles, Julia ma collègue de bureau pour l'ambiance et les fous rires qu'on a partagé durant ces trois années, et bien sûr Mestapha dont les précieux conseils de marathonien m'ont permis d'améliorer mes piètres performances.

J'aimerais exprimer toute ma gratitude envers ma famille qui m'a toujours soutenu sur tous les plans durant tout mon parcours académique. D'abord je rends un vibrant hommage à ces trois femmes de ma vie, ces étoiles du matin : mon arrière grand-maman, ma grand-mère et ma maman qui m'ont montré le prix de l'effort et qui ont dû faire des choix difficiles pour assurer notre éducation. Je pense ensuite à Basile Kuate mon oncle et tuteur, Luc Kamgue mon oncle et conseiller, Pascal Wafo mon oncle et ami : retrouvez ici toute ma reconnaissance pour tous vos sacrifices, merci.

Merci à toi Carole pour ton soutien au cours de ces dernières années et surtout pendant ma traversée du désert et merci pour ton oreille attentive à mes 'élucubrations philosophiques'.

J'adresse aussi mes remerciements à la communauté camerounaise pour son accueil à Rennes: merci à Mme Françoise Yamachui notre mère, Roméo Tatsambon et Guy Atenekeng pour leurs nombreux conseils au début de ma thèse et René PEMHA pour les conseils de vie qu'il n'a cessé de m'apporter. Les camerounais de Rennes étant nombreux, je ne pourrais pas tous les citer ici et je remercie sincèrement tous ceux que j'ai connus à l'ACR et qui ont su mettre la bonne ambiance lors de nos réunions.

Un merci spécial à tous ceux qui m'ont permis de m'évader de la recherche et plus particulierement à tous ceux qui n'ont jamais cessé de multiplier les coups 'tordus' pour m'éloigner de mon PC le week-end. Je pense spécialement à mon triplet 'favori' : Olivier, Jean Michel et Stéphane et mes deux jeunes 'filles': Nadège et Houdat. J'espère que les distances géographiques qui se creusent ne changeront rien à notre amitié. Je pense bien sûr à tous ceux avec qui on a organisé les differentes sorties et soirées 'jeux de société': Laure, Alain, Liliane, Achy, Belinda, Mauclère, Christophe, Francis et Cyrille. Un merci tout particulier à Romuald et Merlin : il n'y a pas si longtemps qu'on se connait mais c'est comme si on a toujours été des frères; continuez de partager votre bonne humeur, votre sincérité et votre joie de vivre.

Je remercie ici tous mes amis du club de Ngoa : Christiane, Agnès, Merline, Simplice, Rodrigue, Innocent, Dieudonné, Bernard. Bien que géographiquement séparés, vous avez pu rendre nos liens encore plus forts à travers les visites (super speed), toutes nos conferences téléphoniques, et bien sûr toutes les séries de blagues et d'histoires rocambolesques.

Le club des six : Marcel et Emma, Fadi, Oumoul et Mariette. Merci pour toutes les soirées studieuses et bien sûr les fous rires, gaffes et autres coups tordus qui ont meublé notre parcours académique aussi bien à Ngaoundéré qu'à Yaoundé.

Si éloignés et pourtant si présents : merci à David, Eugène, Danielle, Arlette, Désiré, Mariette et Ulrich de m'avoir montré que la distance ne compte pas dans une vraie amitié.

The last but not the least : my friend April Warren, thank you for your warm welcome during my stay in Urbana, it has been a pleasure getting to know you. You are a great person.

CONTENTS

Abstract

This thesis presents a set of numerical schemes that aim at solving large linear systems on parallel computers. The proposed approaches are part of a hybrid scheme where the direct and iterative methods are combined through domain decomposition techniques. The initial problem is first divided into subproblems by partitioning the coefficient graph of the system. The Schwarz-based methods are then used as preconditioners for GMRES-based Krylov methods.

We first consider a hybrid scheme using an explicit formulation of the multiplicative Schwarz preconditioner. We define two levels of data parallelism : the first level has been used to parallelize the GMRES method at the global level; we introduce the second level to solve the subsystems induced by the Schwarz preconditioner through a parallel direct method. We show that this splitting guarantee a certain robustness in the global hybrid approach by reducing the total number of partitions. Moreover, this approach enables a better usage of CPU resources allocated inside a compute node.

Then we study the convergence and the parallelism in the GMRES method which is used as the global accelerator in the hybrid method. The general observation is that the number of iterations in that method increases very fast with the number of partitions in the hybrid solver, and so the total CPU time. To limit this effect, we propose several implementations of the GMRES method with the deflation methods. These approaches formulate a deflation process either as an adaptive preconditioner or an augmented subspace basis technique. We show the usefulness of these approaches in their ability to limit the influence of the right choice of the Krylov basis size, and thus to avoid the stagnation of the global hybrid solver. Moreover, these approaches are very efficient to reduce the memory usage as well as the global CPU time and also the exchanged MPI messages between the working processors.

The benefits are given throughout this thesis on moderate to large linear systems arising from several applications fields, and mainly from design optimisation in computational fluid dynamics.

Mots-clés *: algebraic domain decomposition, hybrid direct/iterative methods, parallel multiplicative Schwarz, GMRES, adaptive deflation, fluid dynamics systems.*

Résumé

Cette thèse présente un ensemble de routines pour la résolution des grands systèmes linéaires creuses sur des architectures parallèles. Les approches proposées s'inscrivent dans un schéma hybride combinant les méthodes directes et itératives à travers l'utilisation des techniques de décomposition de domaine. Dans un tel schéma, le problème initial est divisé en sous-problèmes en effectuant un partitionnement du graphe de la matrice coefficient du système. Les méthodes de Schwarz sont ensuite utilisées comme outils de préconditionnements des méthodes de Krylov basées sur GMRES.

Nous nous intéressons tout d'abord au schéma utilisant un préconditionneur de Schwarz multiplicatif. Nous définissons deux niveaux de parallélisme: le premier est associé à GMRES préconditionné sur le système global et le second est utilisé pour résoudre les sous-systèmes à l'aide d'une méthode directe parallèle. Nous montrons que ce découpage permet de garantir une certaine robustesse à la méthode en limitant le nombre total de sous-domaines. De plus, cette approche permet d'utiliser plus efficacement tous les processeurs alloués sur un noeud de calcul.

Nous nous intéressons ensuite à la convergence et au parallélisme de GMRES qui est utilisée comme accélerateur global dans l'approche hybride. L'observation générale est que le nombre global d'itérations, et donc le temps de calcul global, augmente avec le nombre de partitions. Pour réduire cet effet, nous proposons plusieurs versions de GMRES basés sur la déflation. Les techniques de déflation proposées utilisent soit un préconditionnement adaptatif soit une base augmentée. Nous montrons l'utilité de ces approches dans leur capacité à limiter l'influence du choix d'une taille de base de Krylov adaptée, et donc à éviter une stagnation de la méthode hybride globale. De plus, elles permettent de réduire considérablement le coût mémoire, le temps de calcul ainsi que le nombre de messages échangés par les différents processeurs.

Les performances de ces méthodes sont démontrées numériquement sur des systèmes linéaires de grande taille provenant de plusieurs champs d'application, et principalement de l'optimisation de certains paramètres de conception en dynamique des fluides.

Mots-clés *: Décomposition de domaine algébrique, méthodes hybrides directes/iteratives, Schwarz multiplicatif parallèle, GMRES, Déflation adaptative, Calcul en dynamique des fluides.*

CHAPTER 1

Introduction

Le point de départ de ce travail se situe dans le cadre du projet LIBRAERO, financé par l'Agence Nationale de la Recherche et regroupant des partenaires industriels et académiques. L'un des objectifs de ce projet est l'amélioration d'un outil de paramétrisation dans la simulation numérique des écoulements (ou CFD pour Computational Fluid Dynamics) par la société FLUOREM, S.A. Cette amélioration passe par la résolution des systèmes linéaires issus des équations de Navier-Stokes moyennées (ou RANS pour Reynolds-Averaged Navier-Stokes). Les matrices-coefficients sont non symétriques, creuses et de grande taille (typiquement plus d'un million d'inconnues). Le but de cette thèse est donc d'étudier et d'améliorer des techniques existantes et aussi de développer de nouvelles approches pour résoudre efficacement ces systèmes linéaires sur des architectures parallèles. Dans cette thèse, plusieurs méthodes ont été proposées allant dans ce sens. De façon générale, ces méthodes s'appliquent à tout système linéaire creux. Une partie de cette thèse a donc été consacrée à fournir une implémentation standard des différentes approches dans des bibliothèques de type 'open-source' et à effectuer de nombreux essais numériques sur la grille de calcul Grid5K* et sur les supercalculateurs de l'IDRIS †.

Cette introduction générale donne tout d'abord un aperçu des différents critères de performance lors de la résolution sur des architectures parallèles de ces grands systèmes linéaires. Dans la littérature, il existe plusieurs méthodes qui remplissent certains de ces critères; nous en donnons quelques références. Pour d'autres critères, nous proposons ici plusieurs solutions pour les satisfaire. Cette thèse contient donc plusieurs parties qui sont plus ou moins connectés au travers de ces critères. Nous avons en effet choisi de construire chaque partie de façon indépendante. Ceci induit par conséquent que nous n'ayons pas pu éviter quelques répétitions dans l'assemblage du présent manuscrit. À la fin de ce chapitre, nous donnons un aperçu des differentes contributions relatifs à chaque partie de cette thèse.

1.1 Description générale du problème

Le problème à résoudre est formulé comme un système d'équations algébriques

$$Ax = b \tag{1.1}$$

*http://www.grid5000.fr
†http://www.idris.fr

La matrice A du système (1.1), fournie par FLUOREM correspond à la matrice jacobienne résultant des dérivées partielles de premier ordre de l'équation des fluides newtoniens pour des écoulements compressibles. Les dérivées sont calculées par rapport aux variables conservatives $(\rho, \rho u, \rho v, \rho w, \rho E, \rho k, \rho \omega)$ où ρ est la densité, (u, v, w) les composantes en 3D du champ de vitesse, E l'énergie totale, k et ω les facteurs de turbulence. Une présentation complète du problème physique ainsi que du schéma de discrétisation est présentée par Aubert *et al* [15]. Au cours de ce travail, le maillage physique n'étant pas fourni, nous n'avons tenu compte que des informations algébriques extraites de la matrice A. Entre autres caractéristiques, il est important de noter que A est non symétrique, sa partie symétrique $(A + A^T)/2$ n'est pas définie positive, ses éléments sont très hétérogènes à cause des différents niveaux de grandeur entre les variables du flux. Plusieurs autres caractéristiques relatives aux matrices utilisées dans ce document sont données en Annexe A. Celles-ci incluent notamment le conditionnement, le taux de remplissage et le degré de symétrie. Notons enfin que bien que la matrice A soit non symétrique (en valeurs), sa structure est symétrique en considérant le graphe-quotient de la matrice décrite comme suit

$$A' = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,N} \\ A_{2,1} & A_{2,2} & & A_{2,N} \\ \dots & & \ddots & \dots \\ A_{N,1} & A_{N,2} & \dots & A_{N,N} \end{bmatrix} \tag{1.2}$$

où $A_{I,J} \in \mathbb{R}^{b \times b}$, pour $1 \leq I, J, \leq N$, et $N = n/b$. Selon que l'on est en 3D ou en 2D, la taille des blocs vaut respectivement 7 ou 5.

Bien que nous nous soyons concentrés davantage sur ces systèmes qui viennent d'être décrits, il est important de souligner que les méthodes proposées s'appliquent à tout système linéaire à coefficients réels. Dans la suite du document, nous considérons donc de façon générale, et sauf indication contraire, un système tel qu'énoncé dans l'équation (1.1).

1.2 Résolution de systèmes linéaires et facteurs de performance

Au cours du temps, plusieurs techniques ont été proposées pour résoudre de plus en plus rapidement des systèmes linéaires de plus en plus grands. Elles peuvent être classées en deux catégories distinctes et extrêmes: les méthodes à usage général de type 'boîte noire' qui ne requièrent pas ou alors peu de connaissances sur le problème physique. Dans cette catégorie se trouvent les méthodes directes [45] et les méthodes itératives généralement basées sur les sous-espaces de Krylov [113, 131]. Dans l'autre catégorie, on retrouve les méthodes spécialisées pour certains classes de problèmes ayant des 'bonnes propriétés' physiques (Transformés de Fourier rapides[36], méthodes multipôles rapides [95], méthodes multigrilles [25],...). Nous nous concentrons ici sur les méthodes générales de la première catégorie.

De façon assez simple, ces méthodes sont performantes si elles permettent d'obtenir la solution du système avec un bon rapport 'qualité/prix'; la qualité étant prise ici comme l'erreur dans la solution x; le coût quant à lui représentant la quantité de ressources de calcul utilisées (temps de calcul, nombre de processeurs, charge d'utilisation des processeurs, mémoire requise, ...). De façon concrète, nous mesurons d'abord la qualité de la méthode par rapport à sa stabilité numérique; elle définit la capacité de la méthode à fournir une solution dont l'erreur est proportionnelle aux erreurs dans les données en entrée (*backward stability*). L'erreur relative dans la solution calculée $\hat{x}$ est bornée par $(\kappa(A) \cdot \|b - A\hat{x}\|/\|b\|)$

où $\kappa(A)$ est le conditionnement de A. Puisque $\kappa(A)$ est inhérent au système, on utilise généralement le résidu $\|b - A\hat{x}\|$ pour déterminer l'erreur dans la solution. Les méthodes directes sont généralement utilisées du fait de leur stabilité numérique. Cependant, les coûts mémoire requis par ces méthodes feront que l'on préfère les méthodes itératives pour les systèmes de très grandes tailles. Ces dernières en effet, requièrent moins de ressources mémoire et permettent d'obtenir, à partir d'itérations successives construites à faible coût, une solution approchée du système (1.1). Cependant, il est difficile de prédire à l'avance le nombre d'étapes nécessaires. Nous mesurons donc la qualité d'une méthode itérative d'abord par rapport à sa capacité à converger vers la solution (c'est-à-dire à fournir un petit résidu) en un nombre fini d'opérations : c'est ce que nous appelons dans la suite *la robustesse*. Il est aussi requis que la vitesse de convergence soit bonne. On cherche généralement à satisfaire à ces deux critères de qualité en préconditionnant la matrice A. Plusieurs approches hybrides ont donc été développées en prenant des variantes des méthodes directes comme préconditionneurs pour les approches itératives. Les approches plus récentes utilisent des techniques de décomposition de domaines pour combiner le direct et l'itératif et mieux exploiter l'architecture multi-niveaux des ordinateurs parallèles de plus en plus performants.

Dans cette section, nous rappelons, de façon succincte, quelques développements récents dans les méthodes directes et itératives, avec un accent sur leur implémentation parallèle et une restriction pour le cas des matrices non symétriques. À partir de ces deux approches, nous introduisons l'approche hybride qui est utilisée tout au long de cette thèse.

1.2.1 Méthodes directes parallèles

En 1801, Carl Friedrich Gauss se servit de la méthode d'élimination qui porte son nom pour déterminer l'orbite d'une comète [67]. Cette méthode est restée un standard dans les méthodes directes pour systèmes à coefficients non symétriques. L'idée principale est d'écrire la matrice A comme un produit $A = LU$ où L est triangulaire inférieure et U triangulaire supérieure. L'élimination de Gauss peut s'écrire récursivement comme

$$A = \begin{bmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \bar{A} \end{bmatrix} \begin{bmatrix} \alpha_{11} & u_{12}^T \\ 0 & I_{n-1} \end{bmatrix} \tag{1.3}$$

où $l_{21} = \dfrac{a_{21}}{\alpha_{11}}$, si $\alpha_{11} \neq 0$, $u_{12}^T = a_{12}^T$ et $\bar{A} = A_{22} - l_{21}u_{12}^T$. La matrice $\bar{A}$ est le *complément de Schur* de α_{11} par rapport à A. La factorisation LU de A est obtenue en répétant le processus sur le bloc $\bar{A}$. Lorsque la matrice A est dense, la factorisation se fait sur une matrice PAQ où P et Q sont choisis de façon à garantir la stabilité numérique, en évitant que des valeurs nulles ou trop petites se retrouvent sur la diagonale. Lorsque A est creuse, outre la stabilité, P et Q permettent de conserver au mieux la structure creuse de la matrice. En effet, lors de l'élimination de Gauss, des entrées qui étaient nulles peuvent devenir non nulles. Par exemple, dans l'opération $\bar{A} = A_{22} - l_{21}u_{12}^T$, si on suppose que $(A_{22})_{ij} = 0$, alors $\bar{A}_{ij} \neq 0$ si $(l_{21})_i \neq 0$ et $(u_{12})_j \neq 0$. C'est le phénomène de *remplissage*, qui dépend de la structure creuse de la matrice. Un des enjeux des méthodes directes est de concevoir des algorithmes de renumérotation pour limiter le remplissage dans les facteurs L et U tout en garantissant une bonne stabilité numérique. C'est un problème NP-complet [135]. Les approches existantes sont donc basées sur des heuristiques locales [94, 3], globales [63, 85] ou hybrides [10, 75]. Lorsque la matrice A est non-symétrique, la plupart de ces méthodes utilisent plutôt la structure creuse de la matrice $A + A^T$ ou A^TA pour obtenir une renumérotation symétrique. Les approches récentes proposent des

méthodes de renumérotation non symétrique en n'utilisant que la structure creuse de la matrice A [8, 38, 69].

De façon classique, la résolution du système (1.1) par une méthode de factorisation LU contient les principales étapes suivantes :

1. **Phase d'analyse** ou phase de renumérotation et factorisation symbolique : la matrice est renumérotée pour réduire le remplissage dans les facteurs L et U. Dans cette étape aussi sont déterminés les emplacements des nouveaux éléments qui seront créés lors de la factorisation. Les structures de données sont donc préparées à l'avance pour recevoir ces valeurs. Une tâche importante à cette étape est de calculer un graphe de dépendances dans le processus d'élimination. Les nœuds du graphe représentent l'élimination d'une ou de plusieurs colonnes. Les arcs du graphe représentent les dépendances entre les tâches d'élimination.

2. **Factorisation numérique** : Calcul des facteurs L et U. Cette étape n'est pas vraiment décorrélée de l'étape précedente. En effet, outre la permutation (dépendante de la structure) pour réduire le remplissage, il est necessaire ici de faire une autre permutation, qui dépend des valeurs au cours de la factorisation, pour assurer la stabilité numérique de la méthode.

3. **Phase de résolution** : à cette étape, la solution du système est calculée grâce à une substitution avant avec la matrice L et une substitution arrière avec la matrice U.

Derrière cette présentation basique se cachent en réalité une expertise et une littérature abondante relatives à chacune de ces différentes étapes. De façon globale, l'ensemble des techniques disponibles cherchent d'une part, (à l'exemple des méthodes pour matrices denses) à tirer profit de l'architecture des machines en permettant un accès régulier des données en mémoire, et d'autre part à accroître le parallélisme grâce à une répartition équilibrée de charges et une réduction des communications entre différents processeurs. Ces objectifs sont généralement difficiles à atteindre en même temps, en y associant aussi le souci d'obtenir une factorisation stable. Nous présentons ici de façon succincte deux solveurs proposant deux approches différentes et qui représentent, à notre avis, une bonne partie des progrès récents dans les méthodes directes parallèles, voir [5, 72] pour de plus amples détails :

SuperLU_DIST et la factorisation sur des supernœuds [91]. Cette approche est basée sur une factorisation de type *right-looking*[‡] et est particulièrement appropriée pour des matrices ayant une forte structure non symétrique. Dans la phase de factorisation symbolique, la matrice est partitionnée en utilisant la notion de *supernœuds non symétriques*[43] et distribuée aux processeurs organisés de facon bloc-cyclique sur une grille 2D. Le parallélisme à cette étape repose sur cette distribution bloc-cyclique et sur le partitionnement parallèle du graphe de la matrice $|A| + |A|^T$ [70]. À la sortie de cette phase est produit *l'arbre des séparateurs* qui donne la dépendance dans l'élimination des colonnes et qui permet aussi d'identifier les colonnes ayant la même structure creuse et pouvant être regroupées ensemble (supernœuds). Dans la phase de factorisation numérique, les opérations d'élimination et de mise à jour sont faites sur les supernœuds. Ceci permet de réduire les adressages indirects en mémoire et de mieux exploiter la hiérarchie des caches-mémoires

[‡]Aussitôt qu'une colonne est éliminée, les colonnes suivantes de la matrice sont mises à jour. Dans la version *left-looking*, la mise à jour d'une colonne est retardée jusqu'au moment où elle doit être éliminée.

(grâce à des noyaux d'opérations de type BLAS-2 et BLAS-3). Le parallélisme dans cette phase repose sur la structure de l'arbre des séparateurs. En effet, à chaque étape de la factorisation, les colonnes qui ne sont pas dépendantes peuvent être éliminées en même temps. L'une des principales fonctionnalités dans cette méthode est l'amélioration du parallélisme à travers l'utilisation du *pivot statique*. En effet, lorsque le pivot partiel est effectué pendant la factorisation numérique, les processeurs ont besoin de communiquer pour rechercher le pivot adéquat. Dans l'approche par pivot statique, il n'y a pas de permutation pendant la factorisation et la séquence d'élimination est connue à l'avance. Cela permet d'attribuer à l'avance et de façon statique les tâches aux differents processeurs. Le pivot statique requiert les principales étapes suivantes :

1. Lors de la phase d'analyse, les lignes et les colonnes de la matrice sont d'abord équilibrées et la matrice résultante est permutée pour avoir des éléments diagonaux assez grands par rapport aux éléments non diagonaux. Cette phase équivaut à trouver une matrice de permutation P_r, des matrices diagonales D_r et D_c et de faire la transformation $P_r \cdot D_r A D_c$

2. Lors de la factorisation numérique, les pivots inférieurs à un certain seuil sont remplacés. Plus précisement, étant donné un seuil ϵ, si un pivot $\alpha_{ii} < \sqrt{\epsilon} \cdot \|A\|_1$ alors α_{ii} est remplacée par $\sqrt{\epsilon} \cdot \|A\|_1$.

3. Dans la phase de résolution, un raffinement itératif est effectué si la solution n'est pas assez précise. Typiquement, il s'agit à partir de l'approximation x de calculer

$$\begin{array}{rl} r & = b - Ax \\ A \cdot dx & = r \text{ if } berr > \epsilon \\ x & = x + dx \end{array} \tag{1.4}$$

   où ϵ est la précision voulue et $berr = \max_i \dfrac{|r|_i}{(|A| \cdot |x| + |b|)_i}$.

Dans le chapitre 2, nous donnons quelques résultats numériques avec cette approche.

MUMPS et l'approche multifrontale [46, 4, 7]. Cette factorisation LU, basée sur une approche multifrontale, est particulièrement adaptée aux matrices à structure presque symétrique [5]. La phase d'analyse produit une séquence d'élimination sous forme de graphe de dépendances appelée *arbre d'assemblage*. Chaque nœud de l'arbre représente une factorisation d'une sous-matrice dense appelée *matrice frontale*. Par exemple, la matrice A ayant la structure suivante où X désigne des éléments non nuls

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} X & 0 & 0 & X & X \\ 0 & X & X & 0 & 0 \\ 0 & X & X & X & 0 \\ X & 0 & X & X & 0 \\ X & 0 & 0 & 0 & X \end{bmatrix} \end{array} \tag{1.5}$$

produit une factorisation ayant la structure

$$L+U = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} X & 0 & 0 & X & X \\ 0 & X & X & 0 & 0 \\ 0 & X & X & X & 0 \\ X & 0 & X & X & \bar{\mathbf{X}} \\ X & 0 & 0 & \bar{\mathbf{X}} & X \end{bmatrix} \end{array} \tag{1.6}$$

où $\bar{\mathbf{X}}$ représente le remplissage. Les matrices frontales sont définies comme suit

$$F_1 = \begin{matrix} 1 \\ 4 \\ 5 \end{matrix} \begin{bmatrix} X & X & X \\ X & U & U \\ X & U & U \end{bmatrix} \quad F_2 = \begin{matrix} 2 \\ 3 \end{matrix} \begin{bmatrix} X & X \\ X & U \end{bmatrix} \tag{1.7}$$

$$F_3 = \begin{matrix} 3 \\ 4 \end{matrix} \begin{bmatrix} X & X \\ X & U \end{bmatrix} \quad F_4 = \begin{matrix} 4 \\ 5 \end{matrix} \begin{bmatrix} X & X \\ X & U \end{bmatrix} \quad F_5 = [X] \tag{1.8}$$

où U designe la mise à jour nécessaire pour les variables dépendantes. La figure 1.2.1 donne le graphe de dépendance de ces matrices frontales.

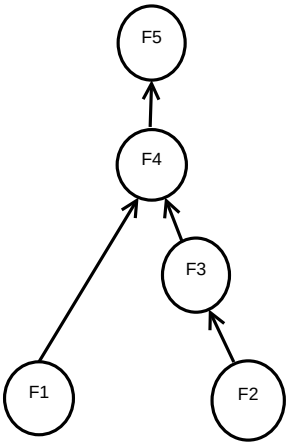


Figure 1.2.1: Graphe de dépendance dans une approche multifrontale

En pratique, lorsqu'une rangée de colonnes dans la matrice factorisée contient une même structure non nulle, elles peuvent être regroupées pour former des supernœuds. Dans ce cas, la matrice frontale ne contient plus une seule variable comme dans l'exemple de la figure 1.2.1 ,mais un ensemble de variables à éliminer.

Dans la phase de factorisation numérique, le parallélisme est exploité à deux niveaux: le premier se situe dans la structure du graphe d'élimination, deux branches de l'arbre peuvent être éliminées en même temps. Dans ce cas, les processeurs sont associés aux nœuds de l'arbre de façon statique après la phase d'analyse. Le deuxième se situe au niveau des opérations sur les matrices frontales denses associées à chaque nœud de l'arbre. En fonction de la taille de ces matrices, plusieurs processeurs sont affectés à son calcul. Vu que la taille de ces matrices frontales n'est pas toujours disponible à l'avance, le nombre de processeurs est défini de façon dynamique à l'exécution. Cela produit au final un ordonnancement hybride des processeurs : statique (au moins un processeur alloué par nœud après la phase d'analyse) et dynamique (à l'intérieur de chaque nœud).

Notons qu'à la difference de SuperLu_DIST, MUMPS utilise une stratégie de pivotage dynamique avec seuil à l'intérieur des matrices frontales. Lorsqu'un pivot acceptable (supérieur au seuil) ne peut pas être trouvée, l'élimination de la variable est affectée au nœud parent. L'avantage de cette approche est de produire une factorisation stable. L'inconvénient est l'espace mémoire requis pour stocker, non seulement la matrice frontale contenant les variables à éliminer, mais aussi les matrices de mise à jour transmises par les nœuds inférieurs de l'arbre. De plus, en présence du pivot dynamique, il est difficile d'avoir une bonne estimation de la mémoire à utiliser dans la phase de factorisation. Les développements récents ont introduit une fonctionnalité qui permet de stocker une partie des facteurs sur le disque lors de la factorisation [2]: c'est la factorisation *out-of-core*.

Cette approche procure l'avantage de travailler sur de très grandes matrices mais le débit de lecture/écriture du processeur vers le disque augmente considérablement le temps de factorisation.

La complexité des méthodes directes et leur demande croissante de mémoire ont conduit naturellement au développement des méthodes itératives.

1.2.2 Méthodes itératives parallèles

L'idée basique dans les méthodes itératives est de partir d'une estimation initiale x_0 de la solution au système (1.1) et de construire une suite d'itérés successifs $x_k, k = 0, 1, 2, \ldots$ qui converge vers la solution exacte x^*. L'équation (1.4) présente un exemple de méthode itérative pour améliorer la solution initiale fournie par la méthode directe. Les méthodes itératives présentent plusieurs avantages par rapport aux méthodes directes : elles sont en général plus faciles à implémenter car le noyau de calcul principal est basé sur le produit matrice-vecteur creux $y \leftarrow A{\cdot}x$. Puisque la matrice A n'est pas transformée, elles requièrent par conséquent moins de mémoire.

Cependant, le principal inconvénient dans les méthodes itératives est que leur convergence n'est pas garantie. Ce critère dépend du rayon spectral de la matrice utilisée pour formuler les itérées. Par exemple, les méthodes itératives 'basiques' de type Jacobi, Gauss-Seidel, SOR s'écrivent comme un schéma de point fixe

$$Mx_{k+1} = Nx_k + c \tag{1.9}$$

où M est une matrice inversible et $A = M - N$. Pour tout x_0, La suite d'itérées $x_k, k = 0, 1, 2, \ldots$ converge vers la solution du système (1.1) si le rayon spectral ρ de la matrice $M^{-1}N$ est inférieur à 1 [113]. Quand bien même la méthode itérative converge, il est important de savoir quelle est sa vitesse de convergence et surtout le nombre maximum d'étapes requises.

Les développements récents utilisent des approximations polynomiales de la forme

$$x_k = x_0 + p_{k-1}(A)r_0 \tag{1.10}$$

où x_0 est l'approximation initiale, $r_0 = b - Ax_0$ le résidu initial et $p_{k-1}(A)$ est un polynôme de degré au plus $k - 1$. La motivation pour ces méthodes vient du résultat suivant :

Proposition 1.2.1. *Il existe un polynôme q de degré au plus $n - 1$ tel que $A^{-1} = q(A)$.*

Voir [52, 113] pour la preuve. La difficulté réside alors dans la recherche du polynôme $q(A)$. La plupart des méthodes polynomiales se basent sur des projections dans le sous-espaces de Krylov $\mathcal{K}_k(A, r_0)$ pour approcher le polynôme $q(A)$, où $\mathcal{K}_k(A, r_0)$ est défini comme

$$\mathcal{K}_k(A, r_0) = span\{r_0, Ar_0, \ldots, A^{k-1}r_0\} \tag{1.11}$$

Etant donnée V_k, une base de $\mathcal{K}_k$, chaque itéré peut donc être exprimé sous la forme

$$x_k = x_0 + V_k y_k, \quad y_k \in \mathbb{R}^k \tag{1.12}$$

Les méthodes de Krylov diffèrent dans la façon de determiner le vecteur y_k.

La méthode GMRES : Dans le cas des matrices non-symétriques, en utilisant la relation (1.10) et en imposant la condition de Petrov-Galerkin $b - Ax_k \perp A\mathcal{K}_k$, on obtient la méthode de GMRES [114] qui minimise à chaque itération la norme du résidu r_k, i.e

$$\|r_k\|_2 = \min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|b - Ax\|_2. \tag{1.13}$$

La plus grande partie de nos contributions dans cette thèse est liée à cette méthode. Pour une description des autres méthodes de Krylov, voir par exemple [24, 52, 96, 113, 121]. Une particularité de GMRES est sa convergence monotone : la suite des normes de résidus est décroissante. De façon générale, l'avantage des méthodes de Krylov, et donc de GMRES, repose sur le fait que leur convergence est garantie, en arithmétique exacte, après n itérations. Il y a cependant intérêt, et cela pour plusieurs raisons, à ce que la méthode converge bien avant d'atteindre n itérations. En effet, le noyau de la méthode réside à chaque itération k, dans la détermination d'une base V_k. Génerer la base $V_k = [v_0, \ldots, v_k]$ requiert de construire une suite de puissances itérées à partir de la matrice A et du vecteur résidu initial r_0. Une base naturelle est donnée par la suite $v_0 = r_0, v_1 = Av_0, \ldots, v_k = Av_{k-1}$. Cependant, elle n'est pas stable numériquement. Pour pallier ce problème, on utilise généralement un procédé d'orthonormalisation pour produire la base V_k. De ce fait, en terme de stockage et de temps CPU, le coût de la méthode croît avec le nombre de vecteurs dans la base. Le coût de l'orthogonalisation augmente à chaque itération, ainsi que le nombre de vecteurs à stocker. De plus, dans une implémentation parallèle, il y aussi un intérêt à limiter le nombre de produits scalaires découlants du procédé d'orthonormalisation.

Plusieurs améliorations ont été proposées pour cette méthode, voir par exemple [121] pour un aperçu assez détaillé :

- Le redémarrage permet de limiter les coûts sus-cités. Il s'agit, après un certain nombre d'itérations, de se débarrasser de la base courante, et de prendre l'approximation courante comme point de départ pour un nouveau cycle. Bien que limitant les coûts, la conséquence immédiate est que la condition optimale de convergence (après n itérations) n'est plus garantie, car on ne minimise la norme du résidu que dans une petite partie du sous-espace de Krylov. Cette approche empêche aussi la convergence superlinéaire généralement observée [132]. Les méthodes de déflation [16, 28, 53, 87, 98] gardent des informations utiles pendant le rédemarrage pour améliorer la robustesse de cette approche. Au chapitre 4 et 5, nous montrons l'efficacité de ces approches sur notre problème.

- Dans l'implémentation du procédé d'Arnoldi généralement utilisé pour produire la base orthonormale V_k, les opérations essentielles sont les produits matrice-vecteur, les produits scalaires et les mises à jour des vecteurs par les coefficients d'orthogonalisation. Sur des architectures à mémoire distribuée, les produits scalaires induisent des communications globales qui peuvent avoir un effet important sur la scalabilité de la méthode. De plus, les opérations vecteur-vecteur dans ce procédé ne permettent pas d'exploiter efficacement la hiérarchie mémoire et la localité des données. Des implémentations alternatives ont été proposées [17, 41, 50, 51, 79, 83, 118, 107]. Elles diminuent la quantité de messages et offrent un meilleur noyau de calcul en dissociant la génération des vecteurs de la base de leur orthogonalisation. Cependant, leur stabilité numérique est un autre point de préoccupation. Au chapitre 6, nous utilisons une de ces approches où la base est construite avec des polynômes de Newton et nous utilisons les méthodes de déflation pour améliorer leur robustesse.

Préconditionnement: De façon pratique, on a recours aux méthodes de préconditionnement pour accélérer les méthodes de Krylov; c'est-à-dire que le système (1.1) est transformé pour obtenir

$$M_L A M_R (M_R^{-1} x) = M_L b, \tag{1.14}$$

où M_L et M_R sont des matrices inversibles et $x = M_R y$. M_L est appelé préconditionneur gauche et M_R est le préconditionneur droit. Le choix de M_L et M_R doit réaliser un compromis entre le fait que $M_L A M_R \approx I$ et que M_R et M_L ne soient pas trop coûteuses à appliquer. Le développement des préconditionneurs est en soi un grand axe de recherche dans la résolution des systèmes linéaires, voir par exemple [21] pour un aperçu assez large des approches récentes. Une grande partie des approches récentes utilisent les techniques empruntées aux méthodes directes pour construire des préconditionneurs adaptés aux méthodes itératives, produisant ainsi des *méthodes hybrides*. La première approche a été de produire une factorisation incomplète de la matrice $A = \tilde{L}\tilde{U} + R$ en limitant le remplissage dans les facteurs $\tilde{L}$ et $\tilde{U}$. Ces facteurs sont ensuite utilisés pour formuler le préconditionneur. La robustesse d'un tel préconditionneur dépend du critère pour inclure les nouveaux éléments créés lors de la factorisation. Les deux critères utilisés se basent sur le niveau de remplissage (ILU(k)) et/ou la valeur numérique des éléments créés (ILUT). Voir par exemple [113] pour une présentation détaillée de ces méthodes. De plus, il est difficile d'obtenir des méthodes de factorisation incomplète performantes au même titre que les factorisations 'complètes', et particulièrement sur des architectures parallèles. Les méthodes de décomposition de domaine ont permis naturellement de définir une nouvelle classe de méthodes hybrides exploitant au mieux ces architectures.

1.2.3 Approches hybrides basées sur une décomposition de domaine algébrique

Le terme *hybride* renvoie à la combinaison de plusieurs techniques, principalement les approches directes et itératives. Ce terme n'est pas nouveau. Par exemple dans [61], Gallivan, Sameh et Zlatev proposent en 1990 une méthode hybride adaptative qui utilise une factorisation incomplète comme préconditionneur à une méthode de Krylov et, si cette méthode hybride ne converge pas assez vite, ils passent à une factorisation complète.

Les méthodes hybrides actuelles cherchent à exploiter au maximum les avantages des méthodes directes et des méthodes itératives tout en exploitant au mieux les architectures parallèles. Elles sont basées sur des techniques de décomposition de domaine. En l'absence du graphe du maillage physique, la décomposition est faite de façon algébrique, c'est-à-dire que le graphe de la matrice est décomposée en sous-graphes et chaque sous-graphe correspond à un sous-domaine. Il existe un grand nombre de techniques permettant de produire une telle décomposition. Les logiciels comme METIS [84] et SCOTCH [35] sont des partitionneurs de graphe. L'objectif général est d'obtenir à peu près le même nombre de nœuds dans chaque sous-graphe. De plus, le nombre d'arcs qui relient les sous-graphes différents doit être minimal. A partir de la décomposition de la matrice en sous-domaines, on va ensuite distinguer deux grands types de méthodes de décomposition de domaine, selon que l'on autorise les sous-domaines à se recouvrir ou non :

Décomposition de domaine sans recouvrement: Les méthodes de cette classe sont dites d'interface. À partir du découpage en p sous-graphes, le système (1.14) est réordonné

de façon à avoir un système bloc de la forme suivante :

$$\begin{bmatrix} C_1 & & & & E_1 \\ & C_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & C_p & E_p \\ F_1 & F_2 & \dots & F_p & B \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ x_E \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \\ b_E \end{bmatrix} \tag{1.15}$$

Les blocs C_i correspondent aux variables intérieures à chaque sous-domaine, B est le bloc des séparateurs, les blocs E_i et F_i sont les variables d'interface entre C_i et B. Une élimination des blocs F_i produit

$$\begin{bmatrix} C_1 & & & & E_1 \\ & C_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & C_p & E_p \\ & & & & S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ y \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \\ \hat{b}_E \end{bmatrix} \tag{1.16}$$

où $S = B - \sum_{i=1}^{p} F_i C_i^{-1} E_i$ et $\hat{b}_E = b_E - \sum_{i=1}^{p} F_i C_i^{-1} b_i$. Le bloc S est le *complément de Schur*. Le système (1.15) peut donc être résolu en suivant les principales étapes :

1. Résoudre les systèmes $C_i z_i = b_i$ pour z_i.
2. Résoudre le système $Sy = b_E - \sum_i^p F_i z_i$ pour y.
3. Résoudre $C_i x_i = b_i - E_i y$ pour x_i.

Ces trois étapes offrent plusieurs possibilités de résolution donnant lieu à plusieurs implémentations [60, 73, 93]. De façon globale, les systèmes à l'étape 1 sont indépendants et peuvent donc être résolus en parallèle. La principale difficulté réside dans la résolution du système avec le complément de Schur à l'étape 2. Ceci est généralement fait avec une méthode itérative, puisqu'elle ne nécessite pas d'assembler la matrice S. Cependant, ce système requiert en général un préconditionneur pour que la méthode globale soit robuste. Et dans ce cas, il peut être utile d'avoir explicitement S ou du moins une approximation. Parmi les differentes approches, on distingue plusieurs cas: (1) On calcule une factorisation incomplète de S et on l'utilise comme préconditionneur; (2) On calcule une approximation de S et ensuite on effectue soit une factorisation LU, soit une factorisation ILU de l'approximation; Les facteurs produits sont donc utilisés comme préconditionneur pour le système à l'étape 2; (3) On construit des préconditionneurs de type Neumann-Neumann ou de type Schwarz à partir des compléments de Schur locaux.

En terme de parallélisme, ces approches sont efficaces si la taille des matrices C_i, et donc des variables indépendantes, est assez grande. Il peut arriver en effet, qu'à cause de plusieurs contraintes liées au partitionnement, qu'il existe une très grande interface entre les sous-domaines. De ce fait, la taille de S sera très grande et le parallélisme induit sera réduit. Les approches proposées dans [93, 71] définissent des versions récursives du complément de Schur.

Décomposition de domaine avec recouvrement: La deuxième catégorie des méthodes hybrides est basée sur une décomposition avec recouvrement et sont dites de Schwarz [113, 123, 127]. A l'origine, la motivation de ces approches était de résoudre en domaine continu des équations aux dérivées partielles (EDP) sur les domaines irréguliers [117].

Tout comme dans le cas des méthodes de Schur, le principe est de diviser le problème en sous-problèmes, ensuite de résoudre chacun des sous-problèmes et utiliser les solutions pour mettre à jour les interfaces entre ces sous-problèmes. Cette technique a été ensuite généralisée aux EDPs discretisés, où la matrice A (issue d'un schéma quelconque de discrétisation) est découpée en plusieurs sous-matrices, chaque sous-matrice correspondant directement à une partie du problème original.

L'approche utilisée ici et dans beaucoup d'approches récentes est algébrique. Au lieu de partir d'un domaine discret et de générer les sous-domaines, nous partons du graphe de la matrice et nous effectuons une décomposition en sous-graphes sur celle-ci. Ceci signifie que la décomposition peut ne pas correspondre à quelque chose de connu en terme de sous-problème physique. De façon expérimentale, ces approches sont viables et ont permis de développer des méthodes de résolution à usage général. Vu que la convergence de ces méthodes de Schwarz n'est prouvée que pour une classe de problèmes, l'approche habituelle est de les utiliser comme des préconditionneurs pour les méthodes de Krylov. On distingue la méthode de *Schwarz additive* et la méthode de *Schwarz multiplicative*. Elles correspondent respectivement à une formulation par blocs des méthodes de relaxation de type Jacobi et Gauss-Seidel. De ce fait, Schwarz multiplicatif offre un préconditionneur plus robuste que Schwarz additif, ceci pour le même schéma de découpage en sous-domaines. En environnement parallèle cependant, le préconditionneur de Schwarz additif est bien plus efficace que Schwarz multiplicatif. Nous l'utilisons donc comme méthode de préconditionnement dans les chapitres 4, 5 et 6.

Les travaux récents [13, 12] ont cependant exhibé une formulation de Schwarz multiplicatif qui offre un bon parallélisme lorsqu'elle est utilisée comme préconditionneur pour une version de GMRES. Une partie de ce travail a donc été consacrée à améliorer le parallélisme et la robustesse dans cette approche. Nous en discutons dans les chapitres 3 et 7.

Il est aussi important de noter qu'il existe une classe de méthodes de Schwarz dites optimisées qui permettent d'accroître la robustesse des méthodes de Schwarz [62, 128]. Dans le cadre du projet LIBRAERO, les possibilités de ces classes de méthodes sont explorées dans la thèse de Thomas Dufaud, voir par exemple [44]. Nous nous concentrons donc ici sur la version restrictive de Schwarz additif et la forme explicite de Schwarz multiplicatif.

Dans la section suivante, nous présentons de façon plus détaillée les méthodes qui ont été développées en se basant sur ces deux préconditionneurs.

1.3 Positionnement de la thèse et contributions

Les méthodes hybrides directes/iteratives basées sur une approche de type Schwarz ou Schur permettent de trouver un compromis entre les critères de performance recherchés : la robustesse, le parallélisme et l'utilisation mémoire, entre autres. Elles permettent dans les deux variantes d'utiliser une méthode itérative de type Krylov sur un système de même taille que le système global et d'utiliser les méthodes directes dans les sous-systèmes issus de la décomposition en sous-domaines. Dans cette thèse, notre approche est basée sur une décomposition avec recouvrement. Les méthodes développées s'appliquent dans certains cas sur des approches de type Schur.

Notre contribution dans cette thèse est d'accroître la robustesse tout en utilisant au maximum les opérations parallèles dans le schéma hybride; un accent est mis aussi sur la consommation mémoire. De façon succinte :

- Nous avons amélioré le parallélisme dans le préconditionneur de Schwarz en définissant deux niveaux d'opérations parallèles. Le premier niveau de parallélisme est

défini à travers tous les sous-domaines et est utilisé pour les operations à travers tous les sous-domaines. Le deuxième niveau est défini à l'intérieur de chaque sous-domaine et permet d'utiliser les méthodes directes parallèles pour les systèmes associées aux sous-domaines. Nous avons montré l'efficacité de cette approche avec des essais numériques utilisant Schwarz multiplicatif.

- Nous avons accru la robustesse et le parallélisme dans la méthode GMRES utilisée comme accélérateur global pour le schéma hybride. Dans cette partie, nous avons défini des versions adaptatives de la méthode de déflation pour accélérer la convergence de la méthode itérative tout en limitant la consommation mémoire et les points de synchronisations entre les processeurs. Un accent particulier a été mis sur l'analyse de plusieurs expériences numériques avec différents types de matrices.

- Nous avons pris soin de fournir des implémentations de ces méthodes qui permettent leur réutilisation en tant que modules pour les autres schémas hybrides. Ainsi, bien que nous nous soyons limités dans la présentation aux méthodes basées sur Schwarz, les versions robustes et parallèles de GMRES proposées ici peuvent être utilisées comme accélérateurs pour les méthodes de Schur. Outre ces modules de GMRES, nous avons amelioré l'interopérabilité du solveur GPREMS implémentant l'approche hybride basée sur la forme explicite de Schwarz multiplicatif.

Nous présentons maintenant le contenu des chapitres qui composent ce manuscrit:

1.3.1 Etude comparative de solveurs pour les systèmes issus de la dynamique des fluides

Dans le chapitre 2, nous effectuons une étude comparative de plusieurs solveurs linéaires dans un environnement de calcul distribué. L'objectif est de voir l'efficacité de certaines techniques existantes sur les systèmes issus du problème décrit en section 1.1. Plusieurs méthodes ont été testées notamment les méthodes directes tel que implementées par SuperLU_DIST et MUMPS (précédemment décrits). Nous avons aussi montré les limites, en terme de robustesse, des approches hybrides utilisant un préconditionnement de type ILU. L'instabilité de cette factorisation incomplète sur les systèmes soumis à l'étude a été explorée plus tard par Pacull *et al.* [106]. Plusieurs résultats n'ont pas été mentionnés dans cette partie, principalement ceux concernant l'approche multiniveaux basés sur une décomposition multigrille. La difficulté de ces approches réside dans la définition des différents niveaux de nœuds grossiers à partir du graphe de la matrice, et bien sûr les opérations d'interpolation et de restriction entre les différents niveaux de grille. Nous avons testé les différentes opérations proposées dans l'implémentation BoomerAMG de la suite HYPRE [54] et les résultats n'ont pas été concluants.

Ce chapitre a fait l'objet de l'article suivant : D. Nuentsa Wakam, J. Erhel, É. Canot, G.-A. Atenekeng-Kahou, *A comparative study of some distributed linear solvers on systems arising from fluid-dynamics simulations*, B. Chapman; F. Desprez; G. Joubert; A. Lichnewsky; F. Peters & T. Priol, (ed.) Parallel Computing: from Multicores and GPU's to Petascale (Proceedings of PARCO'09) IOS Press, pp. 51-58, 2010.

1.3.2 GMRES parallèle avec un préconditionneur Schwarz multiplicatif

L'objectif premier du chapitre 3 est de présenter les opérations parallèles présentes dans un schéma hybride formulé avec Schwarz multiplicatif. L'autre objectif est aussi d'accroître l'efficacité de la forme explicite du préconditionneur de Schwarz multiplicatif. En effet,

tout part de l'observation, assez classique d'ailleurs dans les méthodes de décomposition de domaine, qu'en augmentant le nombre de sous-domaines, la méthode hybride devient de moins en moins robuste. Puisque l'approche habituelle est d'associer un ou plusieurs sous-domaines à un processeur, le parallélisme est alors limité. Nous définissons donc dans ce travail un second niveau de découpage dans les sous-matrices induits pour permettre d'utiliser plus d'opérations parallèles. Cette approche a l'avantage d'associer les solveurs directs parallèles dans chaque sous-domaine avec une méthode itérative parallèle au niveau global. De plus, en limitant le nombre de sous-domaines, la convergence de la méthode globale est en général rapide. Cette approche s'applique aussi à Schwarz additif.

Ce chapitre a fait l'objet de l'article suivant : D. Nuentsa Wakam, G.-A. Atenekeng-Kahou, *Parallel GMRES with a multiplicative Schwarz preconditioner*, Revue Africaine de Recherche en Informatique et Mathématiques Appliquées., 2011, (à paraître).

Une version courte de cet article est apparu dans : D. Nuentsa Wakam, J. Erhel, É. Canot, *Parallélisme à deux niveaux dans GMRES avec un préconditionneur Schwarz multiplicatif*, E. Badouel; A. Sbihi & I. Lokpo,(ed.), Actes du CARI, pp. 189-196. 2010.

1.3.3 Préconditionnement de GMRES par déflation et Schwarz additif

Dans le chapitre 4, nous améliorons la robustesse de l'algorithme GMRES utilisé comme accélérateur dans le schéma hybride avec un préconditionneur de type Schwarz. La contribution première est basée sur une déflation adaptative et permet de réduire les effets négatifs du redémarrage dans GMRES. Cette approche a l'avantage de s'adapter de façon automatique lorsque la vitesse de convergence décroît. Elle permet donc, pour une taille de base fixée, de pourvoir augmenter le nombre de sous-domaines (et donc de processeurs) sans perte de robustesse dans la méthode hybride globale. Des essais numériques ont été effectués avec Schwarz additif. Ce chapitre a fait l'objet de l'article suivant :

D. Nuentsa Wakam, J. Erhel, W. D. Gropp, *Parallel adaptive deflated GMRES*, Lecture Notes in Computational Science and Engineering, Springer-Verlag, 2011 (in revision for the proceedings of the 20th international conference on Domain Decomposition).

1.3.4 Reduction de la mémoire dans les solveurs hybrides pour les systèmes issus de CFD

L'objectif du chapitre 5 est de valider la méthode précédente sur de grands cas tests. Pour cela, nous avons travaillé en étroite collaboration avec le partenaire industriel. La déflation permet de façon globale d'avoir un compromis entre GMRES avec ou sans redémarrage. Nous avons surtout montré l'avantage de cette approche en terme de réduction de coûts mémoire. Une particularité de cette méthode est qu'elle permet également d'accélérer la résolution de systèmes ayant plusieurs second membres. Ce chapitre est issu de l'article suivant :

D. Nuentsa Wakam, F. Pacull, *Memory Efficient Hybrid Algebraic Solvers for Large CFD Linear Systems*, Computer & Fluids, submitted, 2011 (special issue of the 23rd internation conference on Parallel Computational Fluid dynamics).

1.3.5 Parallélisme et robustesse dans GMRES avec une base de Newton augmentée

Dans le chapitre 6, nous proposons une implémentation de GMRES qui accroît à la fois le parallélisme et la robustesse en utilisant une base augmentée. La base de Krylov est construite avec des polynômes de Newton, ce qui permet de réduire les synchronisations

et les communications entre les processeurs. La déflation accroît la robustesse en limitant les effets d'une base trop petite et en permettant d'utiliser un grand nombre de sous-domaines. Une version adaptative de la déflation est également proposée ici dans le but principal d'éviter une stagnation de la méthode itérative. Les résultats sont obtenus, non seulement avec des systèmes issus du problème à l'étude, mais également des équations de convection-diffusion. Ce chapitre est issu de l'article suivant :

D. Nuentsa Wakam, J. Erhel, *Parallelism and robustness in GMRES with the Newton basis and the deflation of eigenvalues*, Electronic Transactions on Numerical Analysis, submitted, 2011.

1.3.6 Analyse globale du parallélisme et de la robustesse dans les schémas hybrides

L'objectif du chapitre 7 est de donner une vue d'ensemble du schéma hybride avec ses composantes algorithmiques. Nous donnons quelques critères pour le partitionnement de graphe, la formulation des préconditionneurs de Schwarz associés et leur implémentation parallèle et enfin les différentes versions de l'accélérateur basé sur GMRES. L'objectif est aussi de montrer, avec des cas tests pratiques, les améliorations qui ont été proposées à chaque étape.

CHAPTER 2

A comparative study of some distributed linear solvers on systems arising from fluid dynamics simulations

Joint work with
Jocelyne ERHEL, Edouard CANOT, Guy-Antoine ATENEKENG-KAHOU

Abstract: This paper presents a comparative study of some distributed solvers on a set of linear systems arising from Navier-Stokes equations and provided by an industrial software. Solvers under consideration implement direct, iterative or domain decomposition methods and most of them are freely available packages. Numerical tests with various parameters are made easier by developing a unified toolbox that links with interface functions provided by these libraries. The intensive numerical tests performed on various sets of processors reveal the good performance results achieved by the recently proposed parallel preconditioner for Krylov methods based on an explicit formulation of multiplicative Schwarz [14].

Keywords : Fluid dynamics simulation, large linear systems, distributed solvers, parallel preconditioning, Multiplicative Schwarz, Additive Schwarz.

2.1 Problem Definition

In this paper, we are interested in finding a good solver for a class of large linear systems

$$Ax = b \tag{2.1}$$

where $A \in \mathbb{R}^{n \times n}$ is a real and unsymmetric sparse matrix, x, $b \in \mathbb{R}^n$ are respectively solution and right-hand side vectors. The matrix A corresponds to the global Jacobian matrix resulting from the partial first-order derivatives of the Reynolds-averaged Navier-Stokes equations. The derivatives are done with respect to the conservative fluid variables. There are various linear solvers libraries freely available and a task of finding a good one

among them (for our set of linear systems) is not easy by itself. Although a theoretical analysis of the problem can suggest a class of solver, it is necessary to consider numerical comparisons on the problem being solved. These comparisons include, but are not limited to, memory usage, reliability, parallel efficiency, CPU time and accuracy in the final solution. So in this work, we present a comparative study of some distributed linear solvers on the above-mentioned set of linear systems. We do not have the pretension to consider all existing distributed solvers in this short study neither all aspects in the solvers as in [5, 72]. At least, we expect this numerical study to suggest which method is appropriate for this problem. This study is also motivated by the performance achieved on these systems using the parallel preconditioned GMRES with the explicit formulation of multiplicative Schwarz [11]. As this kind of study needs many tests with various parameters, we have found useful to design a unified interface that helps us to link uniformly to the interfaces provided by the solvers. However, we should stress on the fact that our main goal in this work is not to offer a generic framework such as Trilinos [77] or Numerical Platon [126] but to test and compare each method suitable for our set of linear problems; so the toolbox is designed primarily to switch between all the solvers under study in some easy and uniform way. The paper is organized as follows. In the next section, the distributed solvers we used in this study are listed. The third part gives an overview of the toolbox. The section 2.4 is the major part of this work: it is devoted to the experimental comparisons. Concluding remarks are given at the end.

2.2 Distributed Linear Solvers

Traditionally speaking, the solvers suitable for the system (2.1) are based either on sparse direct or iterative methods. But with the actual state-of-art, the separation between these two classes is tight. Presently, techniques from the first class are used as preconditioners into the second class. Even in the second class, there are a variety of techniques based on Krylov subspace methods or multilevel methods (Multigrid, Domain decomposition). We first consider the solution with two distributed direct solvers, namely SuperLU_DIST [91] and MUMPS [4]. They are representative of two widely-used techniques in this class. Almost all aspects in both packages have been thoroughly compared [5] using a collection of matrices of reasonable size. Our guess is that the need of memory will become a bottleneck with our present collection of matrices. In fact, this memory usage can be reduced significantly when direct methods are used in incomplete form as preconditioner for iterative methods. So, in this work, we consider EUCLID [80], the recommended ILU preconditioner in HYPRE [54] library. Secondly, we focus on domain decomposition methods. Acting as preconditioners for the Krylov subspace methods (essentially GMRES method), they make use of previous methods to solve (more or less) the local problems induced by the decomposition.

When using Domain Decomposition methods to solve PDE equations, a classical scheme is to consider the splitting from the computational domain. Here, we consider rather a load balancing partitioning based on the adjacency graph of the matrix. Schur complement approaches use a partitioning without overlap while Schwarz methods are applied to partitions that are allowed to overlap. First, we consider the pARMS [93] package based on the first group. In the second group, we use the additive Schwarz preconditioner in the PETSc package [18].

The convergence of the Schwarz methods is better with a successive correction of the residual vector over the subdomains. This is the case in the Multiplicative Schwarz. However, it leads to an inefficient preconditioner in parallel environment due to the high de-

pendencies of data between the subdomains. In a recent work [14], the authors proposed an explicit formulation of this preconditioner in order to dissociate the computation of the residual vector from the preconditioner application. This explicit form is used in conjunction with the parallel version of GMRES proposed in [50]. Hence, the preconditioned Newton-basis is first constructed in a pipeline over all processors [11]; then, a parallel version of QR factorization [118] is called to get an orthogonal basis. In this study, we use the result of that work which is expressed in the PETSc format and available in a library named as GPREMS (Gmres PREconditioned by Multiplicative Schwarz)*.

2.3 Environment of Tests

Our main goal here is to build a ready-to-use interface toolbox such that we can uniformly test any method presented above. The PETSc installer tool is used to build compatible libraries of some of the solvers under study. Figure 2.3.1 gives a simplified overview of this architecture. The *routines for sparse matrix format* are provided to read data of systems from files (matrices and right-hand sides). These data can be in compressed Harwell Boeing format, in Coordinate Matrix Market format, or in compressed block sparse row. The *parameter routines* define *classes* that are used to select options for solvers as well as other parameters either from XML files or PETSc-style database options. At the top level, the test routines define interface functions to all solvers under consideration. So, we need only to choose a solver, edit or generate parameter file and give it to the test routine along with matrix and right-hand side file. At the end of execution, the main statistics are returned in html (XML) or text file via the *statistics routines*. As our toolkit has a capability to switch between solvers transparently, it can be used to select automatically a particular solver given some properties of the linear system being solved such as the size of the matrix or its structural symmetry. However, as we shall see shortly with the results, this decision making is not easy.

2.4 Experimental Comparisons

Tests are carried out using the Grid'5000 experimental testbed, on *paradent* cluster in the Rennes site. Each compute node is a dual-cpu and each cpu is a quadricore Carri System CS-5393B (Intel Xeon L5420 at 2.5GHz) with a 32 GB shared memory. In the following, only one cpu is working in each node as no shared-memory programming paradigm was used. All nodes are connected through a Gigabyte Ethernet switch.

2.4.1 Test Matrices

All the matrices presented here are freely available upon request at [55]. In table 2.1, we list the characteristics of some of them. Integers n and nnz are respectively the size and

Table 2.1: Matrices of test

Idx	Matrix	n	nnz	origin
1	CASE_05	161,070	5,066,996	2D linear cascade turbine
2	CASE_07	233,786	11,762,405	2D linear cascade compressor
3	CASE_10	261,465	26,872,530	3D hydraulic gate case
4	CASE_17	381,689	37,464,962	3D jet engine compressor

*This library will be soon available for public use

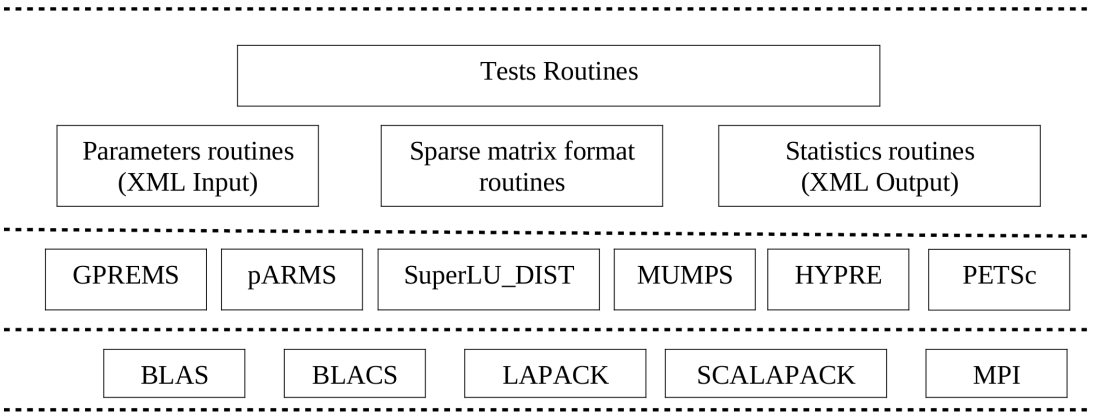


Figure 2.3.1: Architecture of our toolbox

the number of the nonzeros of the matrix.

2.4.2 Numerical Behavior, Parallel Efficiency and Fill-in with Direct Solvers

We consider the minimum degree (MD) and the nested dissection (ND) ordering. As the two direct packages (MUMPS and SuperLU_DIST) accept any pivotal sequence, any ordering method can be used. So we have used METIS as nested dissection ordering in both solvers. With approximate minimum degree (AMD) in SuperLU_DIST, we have observed that the fill-in produced was less than that in multiple minimum degree (MMD); however the factorization time is larger. So we have preferred to use the default ordering provided, *i.e.* MMD in SuperLU_DIST and AMD in MUMPS.

First, the accuracy in the computed solution is considered. In table 2.2, we give the relative residual norm in the solution, *i.e.* $||b - Ax||/||b||$. Tests are done on 4 processors but the results are roughly the same on 8 or 16 processors. We have observed that in some cases, depending on the use of HSL-MC64 routine to permute large elements on diagonal, the computed solution could be wrong. With CASE_05 for instance, when MC64 is used after a nested dissection ordering, both methods do not achieve a good accuracy. With CASE_07, the situation is more complicated. Either with minimum degree or nested dissection ordering, the solution produced with SuperLU_DIST is not accurate. On the other side, without MC64 permutation, all systems are solved somehow accurately with both methods.

After the accuracy, we look at the increasing of memory needed during the factorization. So in table 2.3, the ratio of fill-in in factored matrices is given with respect to the nonzeros in the initial matrix i.e $fill = nnz(L + U - I)/nnz(A)$. The fill-in is larger when the permutation is performed to obtain large diagonal elements, particularly with the nested dissection ordering. As a result, it takes much more time to factorize the matrix, particularly for the largest case. In table 2.4, this preordering effect is shown for

Table 2.2: Numerical behavior : Relative residual norm (4 processors)

Matrix	Ordering	SuperLU_DIST		MUMPS	
		No MC64	MC64	No MC64	MC64
CASE_05	MD	3.6e-14	3.5e-14	1e-13	9.4e-14
	ND	3.6e-14	1.3e-02	8e-14	29.9e-01
CASE_07	MD	1.8e-16	9.9e-01	4.7e-16	1.2e-13
	ND	3.6e-14	7.05e-01	3.6e-16	5.1e-10
CASE_10	MD	8.1e-13	8.8e-13	1.2e-12	1.5e-12
	ND	6.3e-16	8.3e-13	1.2e-12	1.4e-12
CASE_17	MD	7.3e-14	9e-12	6.3e-13	1.3e-10
	ND	7.5e-14	4.4e-13	8.2e-13	2.3e-10

the overall CPU time on 16 processors. Observe that it takes twice CPU time with MC64 preordering in both methods. Surprisingly with METIS, the time in SuperLU_DIST is ten times larger when this preordering step is performed despite the fact that the fill-in is not so large as shown in table 2.3. For the MUMPS solver, these results confirm the advices that the maximum transversal should not be applied on matrices with nearly symmetry structure[5]

Table 2.3: Ratio of fill-in ($fill = nnz(L+U-I)/nnz(A)$) : 4 processors

Matrix	Ordering	SuperLU_DIST		MUMPS	
		No MC64	MC64	No MC64	MC64
CASE_05	MD	13	17	12	20
	ND	10	64	10	15
CASE_07	MD	30	32	30	34
	ND	22	126	21	27
CASE_10	MD	21.8	23.8	20.9	25.1
	ND	17.6	95.4	17.4	21.8
CASE_17	MD	115	138	100	119
	ND	61	74	58	77

Table 2.4: CASE_17: Effect of preprocessing on the CPU time (16 processors)

	SuperLU_DIST		MUMPS	
Ordering	No MC64	MC64	No MC64	MC64
MMD/AMD	3050	4045	4228	8229
METIS	1098	17342	1960	3605

The last aspects of interest are the overall time and the parallel efficiency. In table 2.5, we consider these aspects on the matrix CASE_17. T is the time in seconds while *Acc* and *Eff* are respectively the acceleration and the efficiency with respect to the time on 4 nodes. In this part, all tests are done without the permutation by MC64 as it leads in some cases to huge fill-in and consequently, large CPU factorization time. Note that MUMPS is slightly better than SuperLU_DIST on 4 processors. However, SuperLU_DIST performs better when we increase the number of processors. Moreover, it scales better than MUMPS. This result may come from the relatively slow network interconnecting the nodes [31]. Also in SuperLU_DIST, the amount of communication is reduced during the numerical

factorization by using the static pivoting.

Table 2.5: Parallel efficiency with CASE_17

Ordering	Solver	P=4	P=8			P=16		
		T	T	Acc	Eff	T	Acc	Eff
METIS	SuperLU_DIST	3923	2073	1.89	0.94	1098	3.57	0.89
	MUMPS	3598	2969	1.21	0.6	1960	1.83	0.45

2.4.3 Parallel Behavior of Preconditioners

In the following, we strike to see the convergence of GMRES with the preconditioners mentioned in section 2.2; namely the parallel ILU preconditioner (EUCLID) in HYPRE, the restricted additive Schwarz(ASM) available in PETSc, the Explicit Form of Multiplicative Schwarz(EFMS) used in GPREMS and the left Schur complement (SC) associated with the flexible GMRES in pARMS. To solve the local systems, we have used MUMPS in the case of ASM and EFMS while ILUK is used as approximate solver in the case of SC. The maximum number of iterations allowed is 4000 and the relative tolerance for the convergence is 10^{-9}. The size of the Krylov basis is 64 for the CASE_05 and CASE_07 cases and 128 for the largest ones.

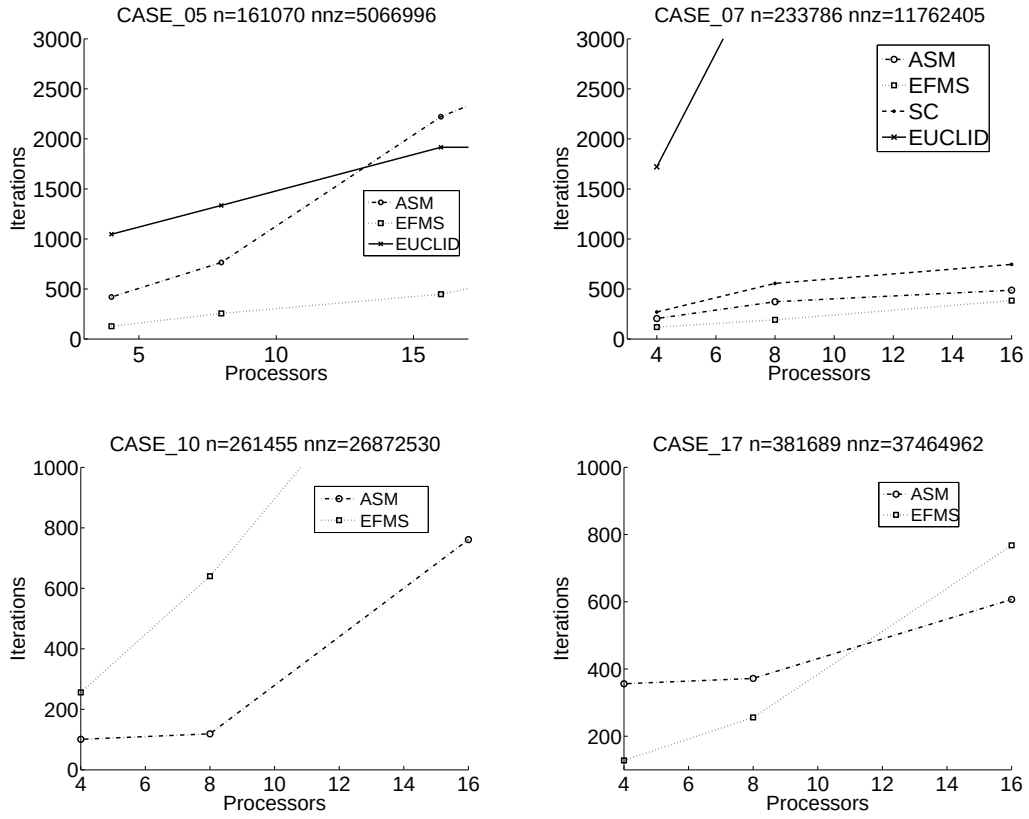


Figure 2.4.1: Number of iterations of GMRES

In figure 2.4.1 the number of iterations is given as a function of the computing nodes. On small systems (CASE_05, CASE_07) with all preconditioners, this number of iterations grows very fast with the number of processors. In many cases, the maximum number of

iterations is reached before convergence. See for instance the CASE_07 with EUCLID on 8 nodes or more. For the largest cases, CASE_10 and CASE_17, GMRES with SC or EUCLID does not converge, whatever the number of nodes used. Thus, only ASM and EFMS are taken into account. On a small number of processors, with all cases but the CASE_10, EFMS gives less number of iterations than ASM. With the CASE_10, ASM performs better than EFMS. However, for more than 8 processors, the number of iterations increases very fast.

Figure 2.4.2 gives the time needed to converge to the right solution with respect to the number of processors. For the smallest case and the largest case, we compare direct solvers to preconditioned GMRES. METIS ordering is used in the two direct solvers without MC64 ordering. For the CASE_05, SuperLU_DIST and MUMPS are clearly faster than preconditioned GMRES. Also, the CPU time with ASM and EFMS tends to increase in CASE_05 and CASE_10. For the largest case, GMRES with ASM or EFMS performs better than direct solvers. However, ASM is better than EFMS with more than 4 processors. The main reason is that in EFMS, the residual vector is corrected in a pipeline through the subdomains whereas this correction is done almost simultaneously in ASM. On the other side, GPREMS do perform well regarding the number of iterations as shown in fig. 2.4.1.

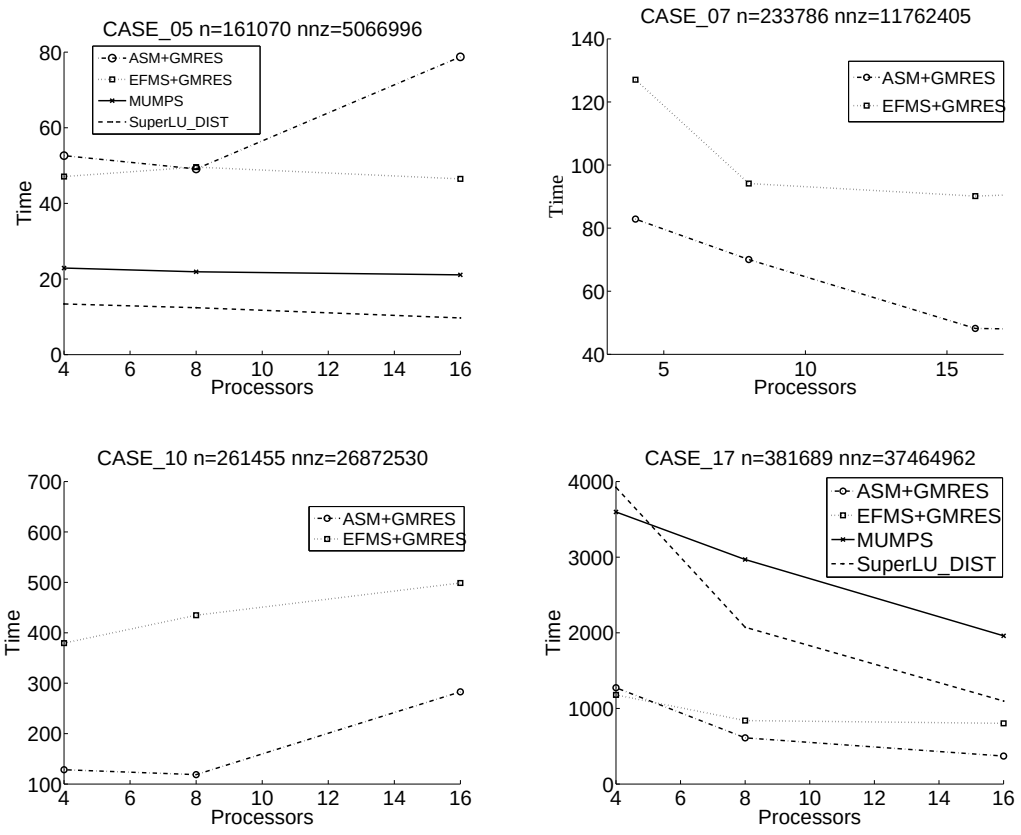


Figure 2.4.2: CPU time

2.5 Concluding Remarks

In this paper, we are interested in the numerical solution of some sparse linear systems issued from actual industrial CFD cases. The distributed solvers we have used are based ei-

ther on direct, iterative or hybrid techniques. Usually direct solvers are robust, but we have observed here that they could fail to solve some of these systems with some non-obvious parameters. However, on small cases, they are markedly more efficient than other methods used in this study. Also, we have tested the ILU-EUCLID and the left Schur Complement preconditioner in pARMS library but on our set of linear systems, Schwarz-based preconditioners should be preferred. So, in this last category, the restricted additive Schwarz preconditioner performs well when it is associated with a direct solver on subdomains. However, we still need to take a very large Krylov basis which could be a bottleneck in the case of larger systems. Finally, one motivation in this work was to show the significant performance achieved by the parallel GMRES when it is preconditioned by one iteration of the multiplicative Schwarz method. The results prove that this preconditioner is competitive among other domain decomposition methods. However, it still suffers from poor scalability. So we are investigating ways to improve this aspect by using some multilevel techniques.

Acknowledgment This work was supported by ANR-RNTL under the LIBRAERO contract. Experiments were carried out using the Grid'5000 experimental testbed[†].

[†]https://www.grid5000.fr

CHAPTER 3

Parallel GMRES with a multiplicative Schwarz preconditioner

Joint work with
Guy-Antoine ATENEKENG-KAHOU

Abstract: This paper presents a robust hybrid solver for linear systems that combines a Krylov subspace method as accelerator with a Schwarz-based preconditioner. This preconditioner uses an explicit formulation associated with one iteration of the multiplicative Schwarz method. The Newton-basis GMRES, which aim at expressing a good data parallelism between subdomains is used as accelerator. In the first part of this paper, we present the pipeline parallelism that is obtained when the multiplicative Schwarz preconditioner is used to build the Krylov basis for the GMRES method. This is referred as the *first level of parallelism*. In the second part, we introduce a *second level of parallelism* inside the subdomains. For Schwarz-based preconditioners, the number of subdommains are kept small to provide a robust solver. Therefore, the linear systems associated with subdomains are solved efficiently with this approach. Numerical experiments are performed on several problems to demonstrate the benefits of using these two levels of parallelism in the solver, mainly in terms of numerical robustness and global efficiency.

Keywords: domain decomposition, preconditioning, multiplicative Schwarz, Parallel GMRES, Newton basis, multilevel parallelism

3.1 Introduction

In this paper, we are interested in the parallel computation of the solution of the linear system (3.1)

$$Ax = b \tag{3.1}$$

with $A \in \mathbb{R}^{n\times n}$, $x, b \in \mathbb{R}^n$. Over the two past decades, the GMRES iterative method proposed by Saad and Schultz [114] has been proved very successful for this type of systems, particularly when A is a large sparse nonsymmetric matrix. Usually, to be robust, the

method solves a preconditioned system (3.2)

$$M^{-1}Ax = M^{-1}b \quad \text{or} \quad AM^{-1}(Mx) = b \tag{3.2}$$

where M^{-1} is a preconditioner operator that accelerates the convergence of the iterative method.

On computing environments with a distributed architecture, preconditioners based on domain decomposition are of natural use. Their formulation reduces the global problem to several subproblems, where each subproblem is associated with a subdomain; therefore, one or more subdomains are associated with a node of the parallel computer and the global system is solved by exchanging informations between neighboring subdomains. Generally, in domain decomposition methods, there are two ways of deriving the subdomains : (i) from the underlying physical domain and (ii) from the adjacency graph of the coefficient matrix A. In any of these partitioning techniques, subdomains may overlap. Overlapping domain decomposition approaches are known as Schwarz methods while non-overlapping approaches refer to Schur complement techniques. Here, we are interested in preconditioners based on the first class. Depending on how the global solution is obtained, the Schwarz method is *additive* or *multiplicative* [123, Ch. 1]. The former approach computes the solution of subproblems simultaneously in all subdomains. It is akin to the block Jacobi method; therefore, the additive Schwarz method has a straightforward implementation in a parallel environment [29]. Furthermore, it is often used in conjunction with Schur complement techniques to produce hybrid preconditioners [32, 66, 115].

The multiplicative Schwarz method builds a solution of the global system by alternating successively through all the subdomains; it is therefore similar to the block Gauss-Seidel method on an extended system; Thus, compared to the additive approach, it will theoretically require fewer iterations to converge. However, good efficiency is difficult to obtain in a parallel environment due to the high data dependencies between the subdomains. The traditional approach to overcome this is through graph coloring by associating different colors to neighboring subdomains. Hence, the solution in subdomains of the same color could be updated in parallel [123, Ch. 1]. Recently, a different approach has been proposed [11, 14]. In that work, the authors proposed an explicit formulation associated with one iteration of the multiplicative Schwarz method. This formulation requires that the matrix is partitioned in block diagonal form [12] such that each block has a maximum of two neighbors; from this explicit formula, the residual vector is determined out of the computation of the new approximate global solution, and therefore could be parallelized through sequences of matrix-vector products and local solutions in subdomains.

The first purpose of this paper is to present the parallelism that is obtained while using this explicit formulation to build the preconditioned Krylov basis for the GMRES method. This is achieved through the Newton basis implementation proposed in [17] and applied in [50, 118]. The usual inner products and global synchronizations are avoided across all the subdomains and the resulted algorithm leads to a pipeline parallelism. We will refer to this as a *first-level of parallelism* in GMRES. The second and main purpose of our work here is to further use parallel operations when dealing with subdomains. Generally, for Schwarz-based preconditioners, the number of subdomains are kept small to guarantee the convergence and consequently, the linear systems associated with subdomains can be very large. It is therefore natural to introduce a *second-level of parallelism* when solving those subsystems. This approach is further motivated by the architecture of the current parallel computers made from several interconnected nodes and multi-core processors inside each node. Indeed, these two levels of parallelism use efficiently the compute resources by dividing tasks across and inside all the allocated nodes of the parallel computer. A similar

approach has been recently used to enhance scalability of hybrid preconditioners based on the additive Schwarz preconditioner for the Schur complement techniques [65].

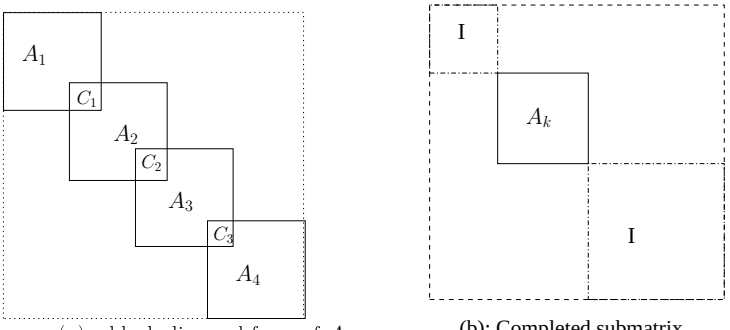
The remaining part of this paper is organized as follows. Section 3.2 recalls the explicit formulation of the Multiplicative Schwarz preconditioner. After that, a parallel implementation of the preconditioned Newton-basis GMRES is given. Section 3.3 provides the second level of parallelism introduced to solve linear systems in subdomains. As those systems should be solved several times with different right-hand sides, the natural way is to use a parallel third-party solver based on LU factorization. In section 3.4, we provide intensive numerical results that reveal good performance of this parallel hybrid solver. The matrices of tests are taken either from public academic repositories or from industrial test cases. Concluding remarks and future directions of this work are given at the end of the paper.

3.2 A parallel version of GMRES preconditioned by multiplicative Schwarz

In this section, the explicit formulation of the multiplicative Schwarz method is introduced. Then we show how to use it efficiently as a parallel preconditioner for the GMRES method. A good parallelism is obtained due to the use of the Newton-Krylov basis.

3.2.1 Explicit formulation of the multiplicative Schwarz preconditioner

From the equation (3.1), we consider a permutation of the matrix A into p overlapping partitions A_i. We denote by C_i the overlapping matrix between A_i and A_{i+1} $(i = 1, \ldots, p-1)$. Here, each diagonal block has a maximum of two neighbors (see Figure 3.2.1.a). Assuming that there is no full line (or column) in the sparse matrix A, such partitioning can be obtained from the adjacency graph of A by means of profile reduction and level sets [12]. We define $\bar{A}_i$ (resp. $\bar{C}_i$) the matrix A_i (resp. C_i) completed by identity to the size of A (see Figure 3.2.1.b).



(a) : block-diagonal form of A (b): Completed submatrix

Figure 3.2.1: Partitioning of A into four subdomains

If $\bar{A}_i$ and $\bar{C}_i$ are nonsingular matrices, then the matrix associated with one iteration of the classical multiplicative Schwarz method is defined [14] as :

$$M^{-1} = \bar{A}_p^{-1}\bar{C}_{p-1}\bar{A}_{p-1}^{-1}\bar{C}_{p-2}\ldots\bar{A}_2^{-1}\bar{C}_1\bar{A}_1^{-1}. \tag{3.3}$$

This explicit formulation is very useful to provide stand-alone algorithm for the essential operation $y \leftarrow M^{-1}x$ used in iterative methods. However, since dependencies between

subdomains are still present in this expression, no efficient parallel algorithm could be obtained for this single operation. Now, if a sequence of vectors v_i should be generated such that $v_i \leftarrow M^{-1}Av_{i-1}$, then a pipeline computation can be setup between all the v_i's and between all the subdomains for each single vector. This is described in Section 3.2.2.1. For the preconditioned Krylov method, the v_i's are simply the vectors basis of the Krylov subspace. If the classical Arnoldi process is used to generate those basis vectors, then the presence of global communication would destroy the pipelined computation. We therefore rely on the Newton basis implementation described in the next section.

3.2.2 Background on GMRES with the Newton basis

A left preconditioned restarted GMRES(m) method minimizes the residual vector $r_m = M^{-1}(b - Ax_m)$ in the Krylov subspace $x_0 + \mathcal{K}_m$ where x_0 is the initial approximation and x_m the current iterate. If r_0 is the initial residual vector, then $\mathcal{K}_m$ is defined as

$$span\{r_0, M^{-1}Ar_0, \ldots, (M^{-1}A)^{m-1}r_0\}. \tag{3.4}$$

The new approximation is of the form $x_m = x_0 + V_m y_m$ where y_m minimizes the euclidian norm $||r_m||_2$. The most time consuming part in this method is the construction of the orthonormal basis V_m of $\mathcal{K}_m$; the Arnoldi process is generally used for this purpose [113]. It constructs the basis and orthogonalizes it in the same time. The effect of this is the presence of global communication between all the processes. Hence, no parallelism could be obtained across the subdomains with this approach. However, synchronisation points can be avoided by decoupling the construction of V_m into two independent phases: first the basis is generated *a priori* then it is orthogonalized.

Many authors proposed different ways to generate this *a priori* basis [17, 133, 40]. With shifts λ_j and scaling factors μ_j $(j = 1, \ldots, n)$, Bai and Reichel [17] define the Newton basis as :

$$\widehat{V}_{m+1} = [\mu_0 r_0, \mu_1(M^{-1}A - \lambda_1 I)r_0, \ldots, \mu_m \prod_{j=1}^{m}(M^{-1}A - \lambda_j I)r_0]. \tag{3.5}$$

The values λ_j are chosen as approximate eigenvalues of $M^{-1}A$ and ordered with the modified Leja ordering [17] to get a well-conditioned basis. From a chosen initial vector $v_0 = r_0/||r_0||$, a sequence of vectors $v_1, v_2, \ldots, v_m$ is generated as follows: If $\lambda_j \in \mathbb{R}$:

$$v_j = \sigma_j(M^{-1}A - \lambda_j I)v_{j-1} \tag{3.6}$$

If $\lambda_j \in \mathbb{C}$:

$$v_j = \sigma_j(M^{-1}A - Re(\lambda_j)I)v_{j-1} \tag{3.7}$$

$$v_{j+1} = \sigma_{j+1}(M^{-1}A - \lambda_j I)(M^{-1}A - \bar{\lambda}_j I)v_{j-1} \tag{3.8}$$

where

$$\sigma_j = 1/||(M^{-1}A - \lambda_j I)v_{j-1}||. \tag{3.9}$$

To avoid global communication, the vectors v_j are normalized at the end of the process. Hence, the scalars μ_j from the equation (3.5) are easily computed as the product of scalars σ_j. These steps are explained in detail in sections 3.2.2.1 and 3.2.2.2. At this point, we get a normalized basis V_m such that

$$M^{-1}AV_m = V_{m+1}T_m \tag{3.10}$$

where T_m is a rectangular matrix formed with the scalars $1/\sigma_j$ and λ_j. V_{m+1} is be orthogonalized using a QR factorization :

$$V_{m+1} = Q_{m+1}R_{m+1}. \tag{3.11}$$

As we show in section 3.2.2.1 and following the distribution of vectors in Figure 3.2.2.(b), the v_is are distributed in blocks of consecutive rows between all the processors; However, their overlapped regions are not duplicated between neighboring processors. Thus, to perform the QR factorization, we use an algorithm introduced by Sameh [116] with a parallel implementation provided by Sidje [118]. Recently, a new approach called TSQR has been proposed by Demmel et al. [42] which aims to minimize the communications and better use the BLAS kernel operations. Their current algorithm used in [97] is implemented with POSIX threads and is used when a whole matrix V_m is available in the memory of one SMP node.

So far, at the end of the factorization, we get an orthogonal basis Q_{m+1} implicitly represented as a set of orthogonal reflectors. The matrix R_{m+1} is available in the memory of the last processor. To perform the minimization step in GMRES, we derive an Arnoldi-like relation [113, Section 6.3] using equations (3.10) and (3.11)

$$M^{-1}AV_m = Q_{m+1}R_{m+1}T_m = Q_{m+1}\bar{G}_m. \tag{3.12}$$

Hence the matrix $\bar{G}_m$ is in Hessenberg form and the new approximate solution is given by $x_m = x_0 + V_m y_m$ where the vector y minimizes the function J defined by

$$J(y) = ||\beta e_1 - \bar{G}_m y||, \quad \beta = ||r_0|| \quad e_1 = [1, 0, \ldots, 0]^T \in \mathbb{R}^m. \tag{3.13}$$

The matrix $\bar{G}_m$ is in the memory of the last processor. Since $m << n$, this least-square problem is sequentially and easily solved using a QR factorization of $\bar{G}_m$. More details on the form of T_m and the algorithm to compute $\bar{G}_m$ can be found in [17, 118]. The outline of the GMRES algorithm with the Newton basis is in [50].

3.2.2.1 Parallel processing of the preconditioned Newton basis

In this section, we generate the Krylov vectors $v_j, (j = 0, \ldots, m)$ of V_{m+1} from the equation (3.6). Consider the partitioning of the Figure 3.2.1 with a total of p subdomains. At this point, we assume that the number of processes is equal to the number of subdomains. Thus, each process computes the sequence of vectors $v_j^{(k)} = (M^{-1}A - \lambda_j I)v_{j-1}^{(k)}$ where $v_j^{(k)}$ is the set of block rows from the vector v_j owned by process P_k. The kernel computation reduces to some extent to two major operations $z = Ax$ and $y = M^{-1}z$. For these operations, we consider the matrices and vector distribution on Figure 3.2.2. On each process P_k, the overlapping submatrix with the process P_{k+1} is zeroed to yield a matrix B_k for the matrix-vector multiplication. In Figure 3.2.2.(b), the distribution for the vector x is plotted. The overlapping parts are repeated on all processes. Hence, for each subvector $x^{(k)}$ on process P_k, $x^{(k)^u}$ and $x^{(k)^d}$ denote respectively the overlapping parts with $x^{(k-1)}$ and $x^{(k+1)}$. The pseudocode for the matrix-vector product $z = Ax$ follows then in Algorithm 1.

Now we consider the matrix distribution in Figure 3.2.2.(a) for the second operation $y = M^{-1}z$. According to relation (3.3), each process k solves locally the linear system $A_k t^{(k)} = z^{(k)}$ for $t^{(k)}$ followed by a product $y^{(k)^d} = C_k t^{(k)^d}$ with the overlapped matrix C_k. However, the process P_k should receive first the overlapping part of $y^{(k-1)^d}$ from the process P_{k-1}. Algorithm 2 describes the application of the preconditioner M^{-1} to a vector z. The form of the M^{-1} operator produces a data dependency between neighboring

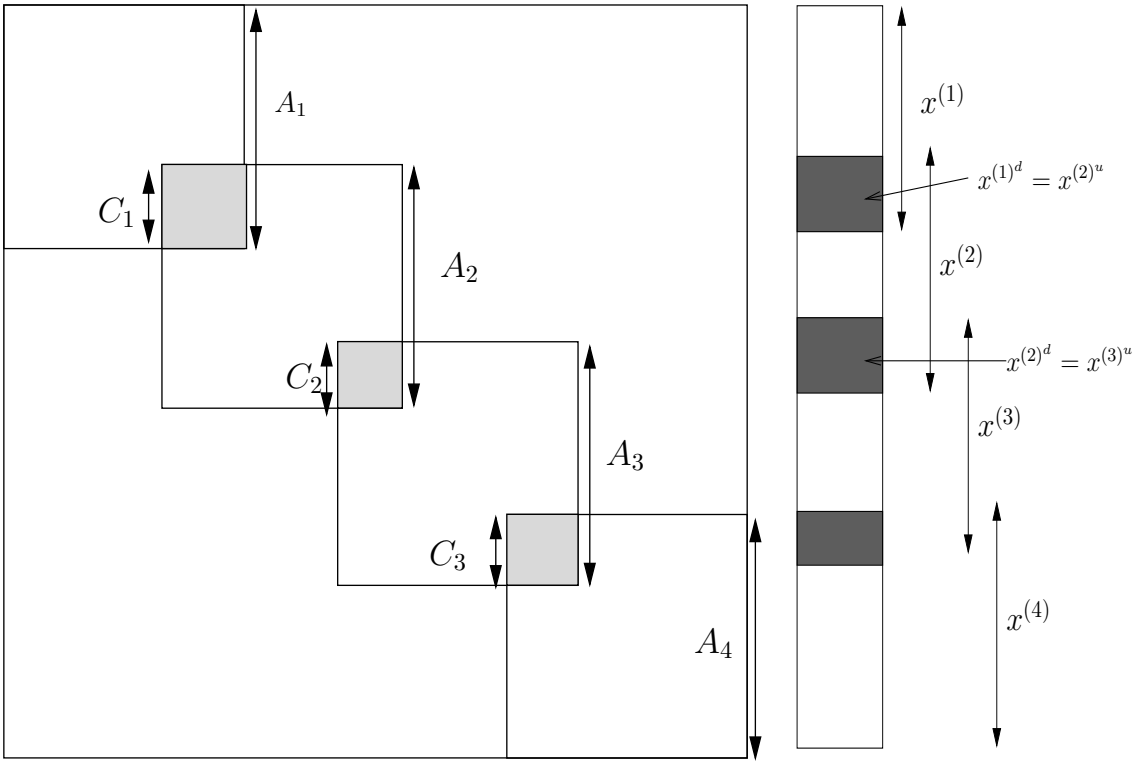


Figure 3.2.2: Distribution of the matrices and vectors for the operation $y \leftarrow M^{-1}x$

Algorithm 1 $z = Ax$

1: /* *Process* P_k *holds* B_k, $x^{(k)}$ */
2: $k \leftarrow myrank()$
3: $z^{(k)} \leftarrow B_k x^{(k)}$; /* *local matrix-vector product* */
4: **if** $k < p$ **then**
5: Send $z^{(k)^d}$ to process P_{k+1}
6: **end if**
7: **if** $k > 1$ **then**
8: Receive $z^{(k-1)^d}$ from process P_{k-1}
9: $z^{(k)^u} \leftarrow z^{(k)^u} + z^{(k-1)^d}$
10: **end if**
11: /* *Communication for the consistency of overlapped regions* */
12: **if** $k > 1$ **then**
13: Send $z^{(k)^u}$ to process P_{k-1}
14: Receive $z^{(k+1)^u}$ from process P_{k+1}
15: $z^{(k)^d} \leftarrow z^{(k+1)^u}$
16: **end if**
17: **return** $z^{(k)}$

processes. Hence the computation of a single vector $v_j = (M^{-1}A - \lambda_j I)v_{j-1}$ is sequential overall the processses. However since the v_j are computed one after another, a process can start to compute its own part of the vector v_j even if the whole previous vector v_{j-1} is not available. We refer to this as a *pipeline parallelism* since the vectors v_j are computed across all the processes as in a pipeline. This would not be possible with the Arnoldi process as all

the global communications introduce synchronizations points between the processes and prevent the use of the pipeline; see for instance the data dependencies implied by this process in Erhel [50].

Algorithm 2 $y = M^{-1}z$

1: /* Process P_k holds A_k, $z^{(k)}$ */
2: $k \leftarrow myrank()$
3: **if** $k > 1$ **then**
4: Receive $y^{(k-1)^d}$ from process P_{k-1}
5: $z^{(k)^u} \leftarrow y^{(k-1)^d}$
6: **end if**
7: Solve local system $A_k y^{(k)} = z^{(k)}$ for $y^{(k)}$
8: **if** $k < p$ **then**
9: $y^{(k)^d} = C_k y^{(k)^d}$
10: Send $y^{(k)^d}$ to process P_{k+1}
11: **end if**
12: /* Communication for the consistency of overlapped regions */
13: **if** $k > 1$ **then**
14: Send $y^{(k)^u}$ to process P_{k-1}
15: **end if**
16: **if** $k < p$ **then**
17: Receive $y^{(k+1)^u}$ from process P_{k+1}
18: $y^{(k)^d} = y^{(k+1)^u}$
19: **end if**
20: **return** $y^{(k)}$

To better understand the actual *pipeline parallelism*, we recall here all the dependencies presented in [125]. For the parallel matrix-vector product in Algorithm 1, if we set $z = h(x) = Ax$, then $z^{(k)} = h_k(x^{(k-1)}, x^{(k)}, x^{(k+1)})$ and the Figure 3.2.3.(a) illustrates these dependencies. For the preconditioner application $y \leftarrow M^{-1}z$ as written in Algorithm 2, we set $y = g(z) = M^{-1}z$, then $y^{(k)} = g_k(y^{(k-1)}, z^{(k)}, z^{(k+1)})$ and dependencies are depicted on Figure 3.2.3.(b). Finally, for the operation $y \leftarrow M^{-1}Ax$, if $y = f(x) = AM^{-1}x = h \circ g(x)$, then $y^{(k)} = f_k(y^{(k-1)}, x^{(k)}, x^{(k+1)}, x^{(k+2)})$ and we combine the two graphs to have the dependencies on Figure 3.2.4.a. Hence the dependency $x^{(k-1)}$ is combined with that of $y^{(k-1)}$. In the pipeline flow computation $v_j = f(v_{j-1})$, the dependency $x^{(k+2)}$ in Figure 3.2.4.b delays the computation of the $v_j^{(k)}$ until the subvector $v_{j-1}^{(k+2)}$ is available. If there is a good load balancing between all the subdomains and if τ denotes a time to compute a subvector $x^{(k)}$ including the time of all the required MPI communications, then the time to compute one vector is

$$t(1) = p\tau \tag{3.14}$$

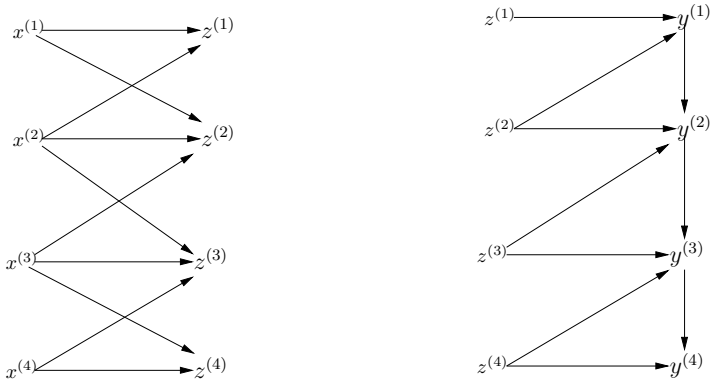
and the time to compute m vectors of the basis follows

$$t(m) = p\tau + 3(m-1)\tau. \tag{3.15}$$

Finally, the first vector is available after $p\tau$ and then, a new vector is produced every 3τ (see Figure 3.2.5). The efficiency e_p of the overall algorithm is therefore computed as :

$$e_p = \frac{mt(1)}{pt(m)} = \frac{p\tau m}{p(p\tau + 3(m-1)\tau)} = \frac{m}{p + 3(m-1)}. \tag{3.16}$$

Hence, the efficiency grows with the value of m but is limited by the Amdhal law to 1/3 when m tends to the infinity.



(a): Graph for the matrix-vector (b): Graph for applying M^{-1}

Figure 3.2.3: Dependency graphs

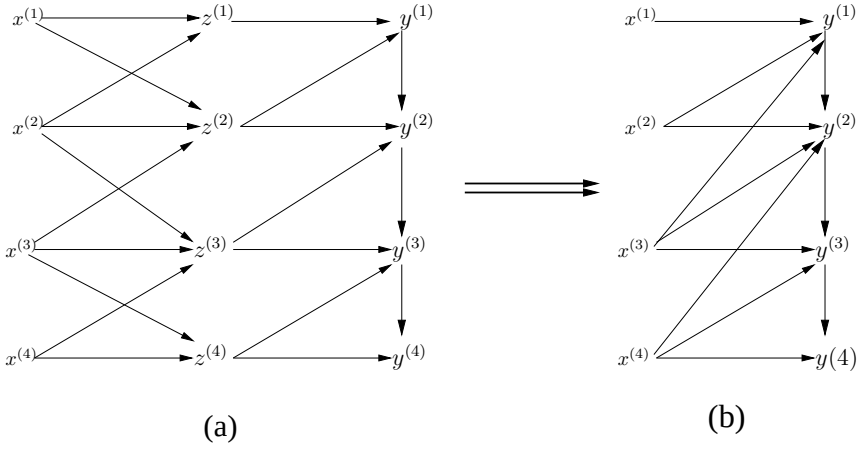


Figure 3.2.4: Dependency graph for $y = M^{-1}Ax$

3.2.2.2 Computation of shifts and scaling factors

So far, we have not yet explained how to compute the shifts λ_i and the scaling factors σ_i of the equation (3.6) and (3.9).

In order to get a well-conditioned Krylov basis, Reichel [109] and Bai and Reichel [17] suggest the use of approximate eigenvalues of $M^{-1}A$ as the shifts λ_j in the Newton basis polynomials. After one cycle of the classical GMRES(m) with the Arnoldi process, these shifts are obtained cheaply by computing the m eigenvalues of the leading $m \times m$ principal submatrix of the output Hessenberg matrix. These values, known as the Ritz values of $M^{-1}A$, are sorted using the modified Leja ordering [109] grouping together the complex conjugate pairs. Recently, Philippe and Reichel [107] proved that, in some cases, roots of the Chebychev polynomials can also be used efficiently as shifts for this Newton basis.

The scalars σ_i used to normalize the vectors of $\widehat{V}_{m+1}$ are determined without using global reduction operations. Indeed, such operations introduce synchronisation points between all the processes and consequently destroy the pipeline parallelism. The Algorithms

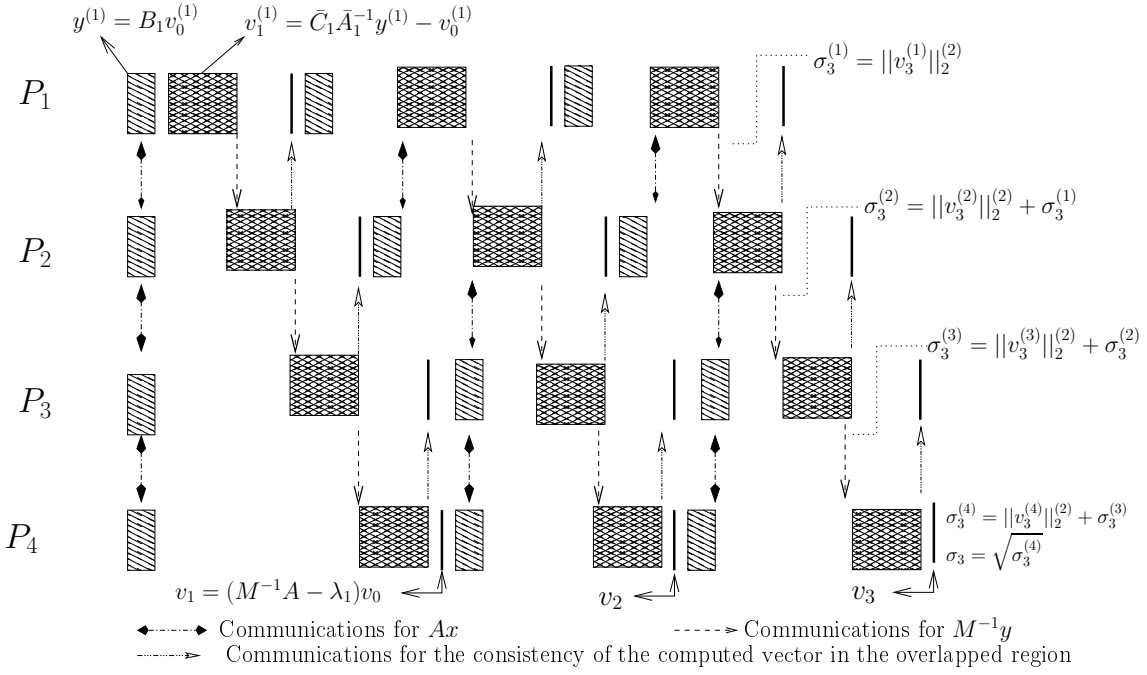


Figure 3.2.5: Double recursion during the computation of the Krylov basis

1 and 2 compute in some extent the sequence $v_j = (M^{-1}A - \lambda_i I)v_{j-1}$. We are interested in the scalars $\sigma_j = ||v_j||_2$. For this purpose, on process P_k, we define by $\hat{v}_j^{(k)}$ the subvector $v_j^{(k)}$ without the overlapping part. In the Algorithm 2, after the line 8, an instruction is therefore added to compute the local sum $\sigma_j^{(k)} = ||\hat{v}_j^{(k)}||_2^2$. The result is sent to the process P_{k+1} at the same time as the overlapping subvector at line 10. The next process P_{k+1} receives the result and adds it to its own contribution. This is repeated until the last process P_p. At the end, the scalar $\sqrt{\sigma_j^{(p)}}$ gives the 2-norm of v_j. An illustration is given in the Figure 3.2.5 during the computation of the vector v_3.

3.3 Enhancing the parallelism in subdomains.

In this section, we propose two levels of parallelism to enhance the robustness and the efficiency of the method.

3.3.1 Motivations for two levels of parallelism

Preconditioners based on domain decomposition do no scale very well, particularly when the coefficient matrix of the linear system is nonsymmetric or symmetric indefinite or when the underlying PDE is far from elliptic. In those cases generally, the number of iterations of the preconditioned GMRES increases very fast with the number of subdomains and consequently the total time to converge is also increased. It becomes essential to keep constant and small the global number of subdomains in order to provide a robust iterative method.

In the particular case of the multiplicative Schwarz preconditioner, the startup time $p\tau$ of the pipeline parallelism introduced in section 3.2.2.1 and the identity 3.16 shows that

the efficiency is limited by p, the number of subdomains. Thus the method should be more efficient if the number of vectors m to compute is large enough to annihilate this startup time. In other words, p should not be very large compared to m. Figure 3.3.1 gives a theoretical efficiency of the method with different values of p and m. It can be seen that the more the subdomains are, the more m should be large in order to get a significant efficiency; usually, no assumption should be made on the value of m as it depends on the problem difficulty. It is therefore essential to act more on p in order to enhance the parallel efficiency of the method.

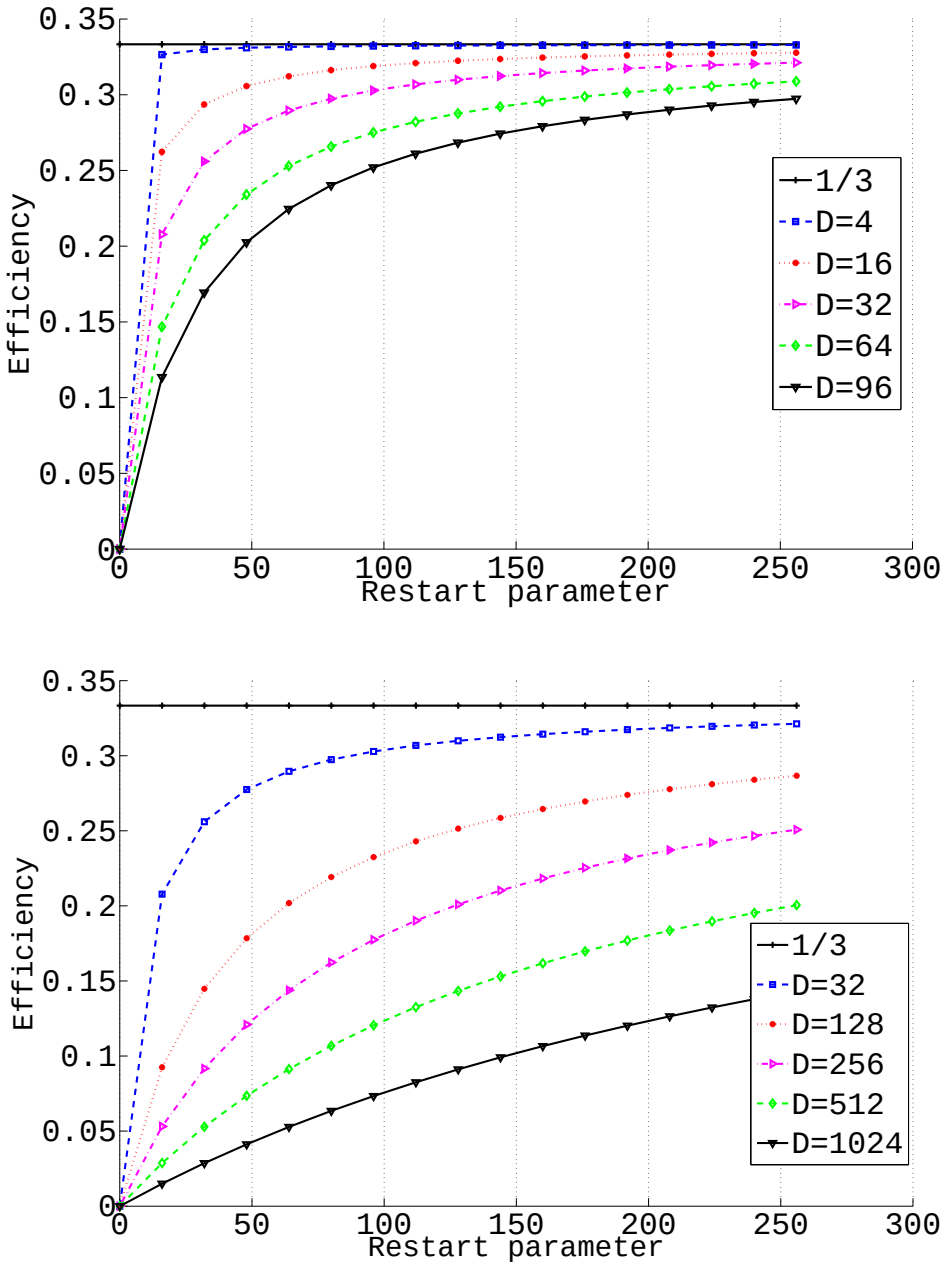


Figure 3.3.1: Theoretical efficiency of GPREMS 1

If one subdomain is assigned to only one processor on modern supercomputers, then

using a small number of subdomains as suggested by the above discussion could be a limitation of the method. If there are more processors than subdomains then a logical approach is to assign one subdomain to several processors and then to define several groups of processors, one for each subdomain. This distribution of data follow naturally the architecture of present parallel computers made with several interconnected nodes and with several processors inside one node. Hence all processors in one node deal with data inside the memory of this node. The next section shows how this second level of parallelism is used within the subdomains.

3.3.2 Practical implementation

Tasks to parallelize inside the subdomain are mainly the solution of linear systems induced by the application of the preconditioner M^{-1}. As those systems should be solved several times with the same coefficient matrix, it is natural to use a direct method. This approach is usually known as hybrid direct/iterative technique. The parallelism inside the subdomains relies on the message passing paradigm instead of a shared-memory model. The motivation for this choice is that many high-performant and public domain solvers are based on MPI [4, 76, 91]. Note that with the message passing model, a subdomain can be distributed on more than one node if the physical memory is small on that node.

So far with the two levels of data distribution on compute units, the solver goes through all the following steps

1. ***Initialization*** *:* During the first step, the global matrix A is permuted in block diagonal form by the host process (see Figure 3.2.1). After that, the local matrices A_k and C_k should be distributed to other processes. If a distributed solver is used in subdomains, then the processors are dispatched in multiple communicators. Figure 3.3.2 shows a distribution of four nodes with four processes each around two levels of MPI communicators. The first level is intended for the communication across the subdomains. Hence in this communicator, the submatrices are distributed to the *level* 0 processes (i.e $P_{k,0}$ where $k = 0 \dots p-1$ and p the number of subdomains). For each subdomain k, a second communicator is created between the *level* 1 processes to manage communications inside the subdomains (i.e $P_{k,j}$ where $j = 0 \dots p_k - 1$ and p_k the number of processes in the subdomain k).

2. ***Setup*** *:* In this phase, the symbolic and numerical factorization are performed on submatrices A_k by the underlying local solver. This step is purely parallel across all the subdomains. At the end of this phase, the factors L_k and U_k reside in the processors responsible for the subdomain k. This is totally managed by the local solver. Prior to this phase, a preprocessing step can be performed to scale the elements of the matrix.

3. ***Solve*** *:* This is the iterative phase of the solver. With an initial guess x_0, the solver computes all the equations (3.6-3.13) as outlined here:

   (a) Perform one cycle of GMRES with the Arnoldi process to compute the shifts λ_j.

   (b) Pipeline computation of V_{m+1}

   (c) Parallel QR factorization $V_{m+1} = Q_{m+1}R_{m+1}$

   (d) Sequential computation of $\bar{G}_m$ such that $M^{-1}AV_m = Q_{m+1}\bar{G}_m$ on the process $P_{p-1,0}$.

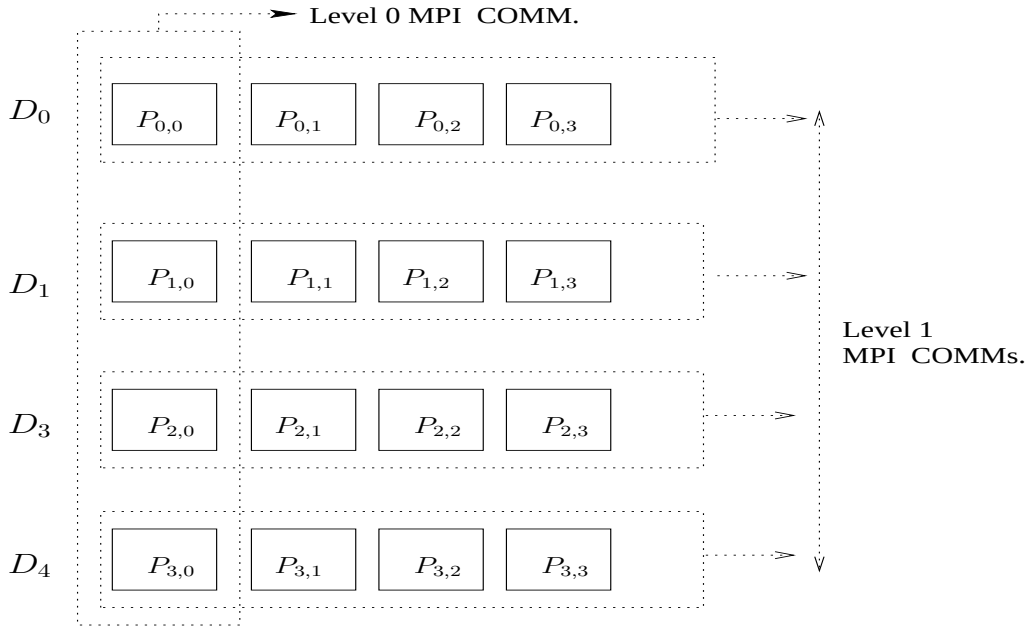


Figure 3.3.2: MPI communicators for the two levels of parallelism

(e) Sequential solution of least-square problem (3.13) for y_m on the process $P_{p-1,0}$.

(f) Broadcast the vector y_m to processes $P_{k,0}$

(g) Paralle computation of $x_m = x_0 + V_m y_m$

The convergence is reached when $||b-Ax|| < \epsilon||b||$ otherwise the process restarts with $x_0 = x_m$. When two levels of parallelism are used during the computation of V_m, *level* 1 processors perform multiple backward and forward sweeps in parallel to solve local systems. After the parallel QR factorization in step 3c, the explicit Q factor is never formed explicitly. Indeed, only the unfactored basis V is used to apply the new correction as shown in step 3g. Nevertheless, a routine to form this factor is provided in the solver with the courtesy of Sidje [118].

3.4 Numerical experiments

In this section, we perform several experiments to give the numerical robustness of GPREMS and the benefits of using two levels of data distribution. We start by giving the software architecture of the solver in subsection 3.4.1 and the test cases in subsection 3.4.2.

3.4.1 Software and hardware framework

The solver is named GPREMS* (GMRES PRECONDITIONED BY MULTIPICATIVE SCHWARZ). It is intended to be deployed on distributed memory computers that communicate through message passing (MPI). The parallelism in subdomains is based either on message passing or threads model depending on the underlying solver. The whole library is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation) [18, 19]. The motivation of this choice is the uniform access to a wide range of external packages for the linear systems in subdomain. Moreover, the PETSc package provides several optimized functions

*A public license will be released soon

and data structures to manipulate the distributed matrices and vectors. Figure 3.4.1 gives

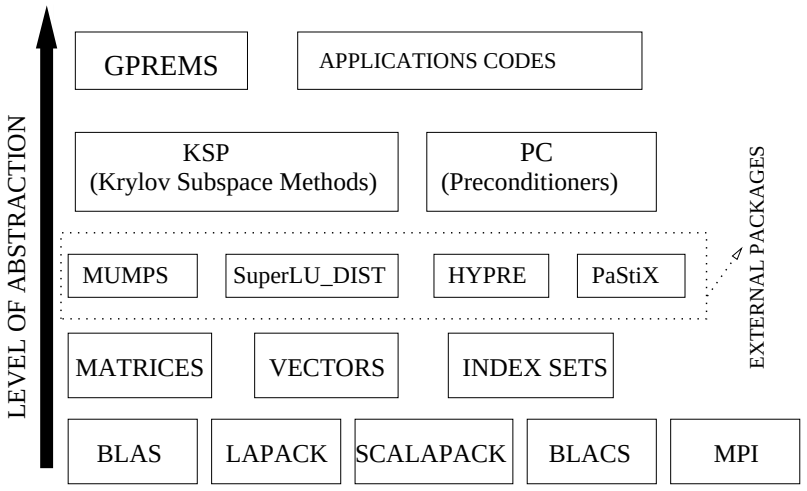


Figure 3.4.1: GPREMS library in PETSc

the position of GPREMS in the abstraction layer of PETSc. We give only the components that are used by GPREMS. For a complete reference on PETSc environment, please refer to [18]. Although GPREMS uses the PETSc environment, it is not distributed as a part of PETSc libraries. Nevertheless, the library is configured easily once a usable version of PETSc is available on the targeted architecture.

All the tests in this paper are performed on the IBM p575 SMP nodes connected through the Infiniband DDR network. Each node is composed of 32 Power6 processors sharing the same global memory. A Power6 processor is a dual-core 2-way SMT with a peak frequency at 4.7 GHz. A total of 128 nodes is available in this supercomputer named *Vargas*[†] which is part of the French CNRS-IDRIS supercomputing facility.

3.4.2 Test matrices

Table 3.1: General properties of the four test matrices

Matrix	Size	Entries	Source
PARA_04	153,226	2,930,882	UFL
CASE_04	7,980	2,930,882	FLUOREM
CASE_07	233,786	11,762,405	FLUOREM
CASE_09	277,095	30,000,952	FLUOREM
CASE_17	381,689	37,464,962	FLUOREM

In Table 3.1, the main characteristics of the test cases are listed. The first matrix PARA-4 arises from 2D semiconductor device simulations and is taken from the University of Florida (UFL) Sparse Matrix Collection [39]. The remaining test cases are provided by FLUOREM [55], a software editor in fluid dynamics simulations. They correspond to linearized Navier-Stokes equations.

[†]http://www.idris.fr/su/Scalaire/vargas/hw-vargas.html

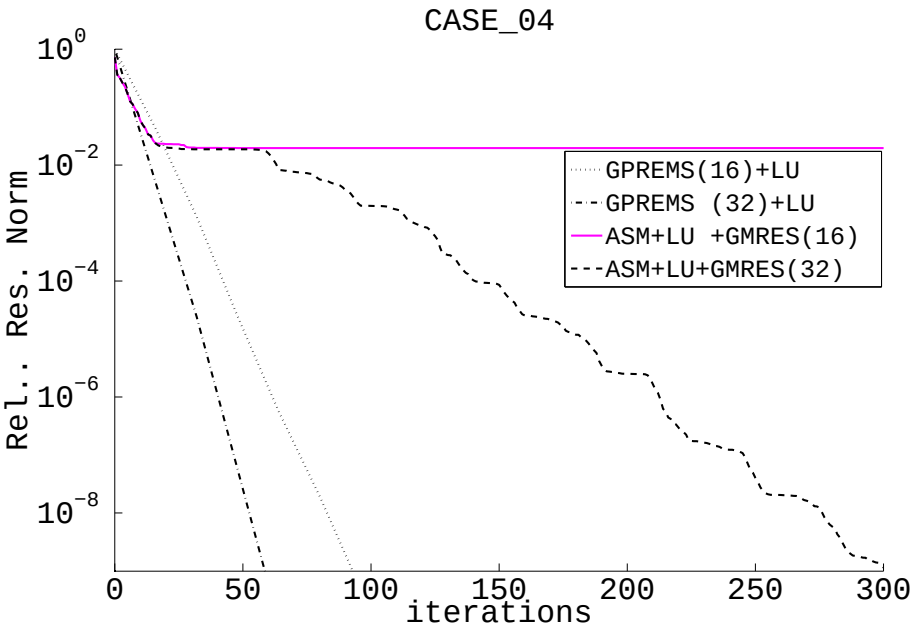


Figure 3.4.2: The multiplicative Schwarz approach (GPREMS) compared to the restricted additive Schwarz (ASM) on CASE_004; 16 and 32 vectors in the Krylov subspace at each restart; 4 subdomains (block diagonal partitioning in GPREMS and Parmetis for ASM); LU factorization on local matrices with MUMPS package

3.4.3 Numerical robustness of GPREMS

The main motivation of using the multiplicative Schwarz method is its robustness compared to the additive Schwarz implementation. There are some cases where the latter fails. In this case, the former can be a good alternative. We illustrate such situation on FLUOREM problem CASE_04. It is worth to note that previous work has been done to test similar problems on other hybrid solvers based on Schur complement techniques. Here we compare the multiplicative Schwarz in GPREMS with the restricted additive Schwarz (ASM) method used as a preconditioner for GMRES. The implementation provided in PETSc release 3.0.0-p2. In the latter, the Krylov basis is built with the modified Gram-Schmidt (MGS) method and the input matrix is partitioned in 4 subdomains using ParMETIS. With ASM, we note in Figure 3.4.2 that GMRES(16) stagnates from the first restart (the plain line). It is necessary to form 32 Krylov vectors at each restart in order to achieve a fair accuracy (dash curve). For GMRES(m), the size m of the Krylov subspace is critical. Generally, it is a trial and error process to get a good value of m. When GPREMS is used, a stagnation is less likely to occur. In this test case, a good convergence is obtained in a few number of iterations with the two values of m. For instance, the relative residual norm drops to 10^{-8} in less than 100 iterations (dot and dash-dot lines).

3.4.4 Benefits of two levels of parallelism

In this part, we give the gain of using two levels of data distribution. We first consider the problem CASE_017 listed in Table 3.1. The geometry of this 3D case is a jet engine compressor. This test case is difficult to solve as shown in a short comparative study involving some distributed linear solvers [103]. From this study, solvers based on overlapping Schwarz decomposition provide an efficient way to deal with such problems. However, the

number of subdomains should be kept small to provide a good convergence. In Table 3.2, we keep 4 and 8 subdomains and we increase the total number of processors. As a result, almost all the steps in GPREMS get a noticeable speedup. Typically, with 4 subdomains, when only one processor is active, the time to setup the block matrices is almost 203 s. and the time spent in the iterative loop (under Time/Iter) is 622 s. Moving to 8 active processors decreases these times to 59 s. and 181 s. respectively. The same observation can be done when using 8 subdomains, in which case the overall time drops from 540 s. to 238 s. Note that the overall time includes the first sequential step that permutes the matrix in block-diagonal form and distributes the block matrices to all active processors. *Intranode speedup and efficiency* are reported in the last two columns of Table 3.2 . This is different from the ones computed with only one level of parallelism. The main objective here is to evaluate if it is worthy to add more processors in a subdomain. If d is the number of subdomains, $s_p = T_d/T_p$ and $e_p = s_p/p$ where T_p is the CPU time on p processors. For 4 subdomains, using 8 processors in each subdomain gives a speedup of 2.88. This speedup is 2.28 when 8 subdomains and 64 processors are used. The major speedup is observed for the setup phase thanks to the direct solver MUMPS [4]. The solve phase shows a noticeable speedup as well. The overall efficiency is decreasing very fast due to the fact that all the processors in one subdomain are scheduled in one SMP node. During the computations, all the processors share the same memory bandwith and access irregularly data in the global memory. With the low granularity and the irregular access of data in sparse matrix computations, the efficiency is determined mostly by the size of the memory bandwith rather than the clock rate of processors. Nevertheless, the two levels of parallelism help to keep busy the processors in the SMP node as they would be idle otherwise.

Table 3.2: Benefits of the two-levels of parallelism for various phases of GPREMS on CASE_17 with a restart of 64 and MUMPS direct solver in subdomains

#D	#Proc.	Iter.	CPU Time (s.)						
			Init	Setup	Solve	Time/Iter	Total	s_p	e_p
4	4	128	70.78	203.67	622.94	4.87	897.39	-	-
	8	128	70.77	125.77	411.42	3.21	607.96	1.48	0.74
	16	128	69.79	73.15	280.41	2.19	423.35	2.12	0.53
	32	128	70.77	59.44	181.45	1.42	311.66	2.88	0.36
8	8	256	69.91	71.73	399.34	1.56	540.98	-	-
	16	256	70.36	42.99	343.25	1.34	456.59	1.18	0.59
	32	256	71.61	28.72	206.7	0.81	307.02	1.76	0.44
	64	256	71.85	20.85	144.79	0.57	237.49	2.28	0.28

We further point out the benefits of adding several processors in each subdomain for the problems PARA_4, CASE_07 and CASE_09. The CPU times for the main phases in the solver are reported in Table 3.3 and 3.4: the setup time, the Iterative loop time (under Solve), the mean time spent in each iteration (under Time/Iter). The total time includes the time for the preprocessing step which is not reported. The size of the Krylov basis is 32 for PARA_4, and 40 for CASE_07 and CASE_09. MUMPS[4] is used as direct solver in each subdomain. The iterative process stops when the relative residual norm $||b - Ax||/||b||$ is less than 10^{-8}. The statistics show a good improvement with two levels of parallelism. As noted before, the major improvement is in the setup phase but the iterative phase benefits from this approach as well. It is worth to note the time spent for each single iteration. Indeed, this time decreases, although not fast, as more processors are added in subdomains.

Table 3.3: CPU Time of GPREMS on PARA_4

#Proc.	#D	PARA_4			
		Setup	Solve	Time/Iter	Total
4	4	11.75	58.31	0.40	72.84
8	4	8.28	58.68	0.41	69.74
	8	3.79	46.40	0.24	52.97
16	4	5.79	40.55	0.28	49.11
	8	2.86	31.64	0.16	37.26
	16	1.12	41.85	0.17	46.09
32	4	4.57	33.80	0.23	41.15
	8	1.78	20.94	0.11	25.47
	16	0.86	31.10	0.13	35.09
64	8	1.51	17.97	0.09	22.25

Table 3.4: Setup and solve phase of GPREMS on CASE_07 and CASE_09

#Proc.	#D	CASE_07			CASE_09		
		Setup	Solve	Time/Iter	Setup	Solve	Time/Iter
4	4	19.05	77.38	0.64	57.00	121.85	1.52
16	4	10.27	47.86	0.40	24.94	60.18	0.75
48	12	1.95	28.49	0.09	4.72	51.99	0.22
96	12	1.67	21.75	0.07	3.45	38.08	0.16

3.5 Concluding remarks

In this paper, we give an implementation of a parallel solver for the solution of linear systems on distributed-memory computers. This implementation is based on an hybrid technique that combines a multiplicative Schwarz preconditioner with a GMRES accelerator. It is known that the multiplicative formulation of the Schwarz method does not have a natural parallelism. Thanks to the Newton basis GMRES implementation, a good pipeline parallelism can be obtained through the subdomains. It is also admitted that Schwarz-based preconditioners do not scale very well with the number of subdomains. In this work, we implement two levels of data distribution to limit the number of subdomains. To this end, we define two levels of parallelism during the computation of the orthonormal basis needed by GMRES : The first level is expressed through pipeline operations across all the subdomains. The second level uses a parallelism inside third-party solvers to build the solution of subsystems induced by the domain decomposition. It is obvious that even with these two levels of parallelism, the proposed approach cannot compete with multilevel methods based on Schur complement techniques. Nevertheless, there are some problems where Schwarz preconditioners offer more robustness.

The experimental tests have pointed out this robustness on tests cases arising from linearized Navier Stokes equations. This robustness is enhanced with a direct solver for linear systems in subdomains. We have shown that the gain of using a parallel solver inside the subdomains is two-fold: the convergence is guaranteed when the number of processors grows as the number of subdomains remains the same. The global efficiency increases as we add more processors in subdomains. For large block matrices, the use of a direct solver in subdomains implies to access the whole memory of a SMP node. Therefore, this approach keeps busy all the processors of the SMP node and enables a good usage of allocated computing resources.

However, more work needs to be done to achieve very good scalability on massively parallel computers. Presently, an attempt to increase the number of subdomains up to 32 increases the number of iterations as well. So we are investigating ways to maintain the latter as small as possible. A first attempt is to use more than two levels of splitting but the efficiency of such approach is not always guaranteed, specifically if the linear system arises from non-elliptic partial differential equations. An ongoing work is to keep the two-levels of splitting for the preconditioner operator and then to further accelerate the GMRES method with spectral informations gathered during the iterative process.

Acknowledgments

The authors wish to thank Jocelyne ERHEL and Bernard PHILIPPE from INRIA-Rennes for helpful discussions on this work. This work was performed using HPC resources from GENCI-IDRIS (Grand Equipement National de Calcul Intensif - Institut du Développement et des Ressources en Informatique Scientifique). Early experiments were carried out using the Grid'5000 experimental testbed (https://www.grid5000.fr).

A short version of this paper appeared in the proceedings of the CARI conference, see [102].

CHAPTER 4

Parallel Adaptive Deflated GMRES

Joint work with
Jocelyne ERHEL and William D. GROPP

Abstract: Many scientific libraries are currently based on the GMRES method as a Krylov subspace iterative method for solving large linear systems. The restarted formulation known as GMRES(m) has been extensively studied and several approaches have been proposed to reduce the negative effects due to the restarting procedure. A common effect in GMRES(m) is a slow convergence rate or a stagnation in the iterative process. In this situation, it is less attractive as a general solver in industrial applications. In this work, we propose an adaptive deflation strategy which retains useful information at at time of restart to avoid stagnation in GMRES(m) and improve its convergence rate. We give a parallel implementation in the PETSc package. The provided numerical results show that this approach can be effectively used in the hybrid direct/iterative methods to solve large-scale systems.

Keywords: *Hybrid Solver, Adaptive GMRES, Deflated Restarting, Parallel Preconditioning*

4.1 Introduction

The GMRES method due to [114] is widely used, thanks to its monotonic convergence properties, as a Krylov subspace method for solving large and sparse linear systems. Due to memory and computational requirements, the restarted GMRES (noted as GMRES(m)) is generally used. At the time of restart, information from the previous Krylov subspace is discarded and the orthogonality between successive Krylov subspaces is not preserved. The worst case is when the successive generated Krylov subspaces are very close. As a result, there is no significant reduction in the residual norm and the iterative process may stagnate. Deflation techniques are a class of acceleration strategies that collects useful information at the time of restart mainly to avoid this stagnation and improve the convergence rate. The main idea behind these methods is to remove the smallest eigencomponents from the residual vector as they are known to slow down the convergence of GMRES.

In a practical use of a deflation strategy, it is necessary to define the number of eigenvalues to deflate. As the deflation process induces additional operations to GMRES(m), it is interesting as well to know *a priori* if the deflation will be beneficial. In this work, we propose an adaptive deflated GMRES(m) which aims at enhancing the convergence of GMRES(m) by adaptively extracting the spectral information needed to speedup the convergence. The adaptive strategy is based on a (near) stagnation test which defines if the deflation process is needed or not and if more accurate spectral information are required. Although we use a stagnation test similar to that in [124], our approach is different since we assume that the restart length m is fixed. This work is motivated by the convergence behavior of GMRES when it is used with a Schwarz preconditioner. As the number of subdomains increases, the eigenvalues are less and less clustered. The restarting may have the disadvantage to discard the smallest eigenvalues before their convergence. The proposed adaptive strategy will thus keep these spectral values in the Krylov subspace until their convergence.

The remaining part of this report is organized as follows: in Section 4.2, we first recall the basis of the deflation technique applied as a preconditioner and we derive the adaptive strategy. In Section 4.3, we discuss on the parallel implementation. Section 4.4 is focused on numerical experiments to show the benefits of this scheme on a real industrial CFD test case.

4.2 Adaptive preconditioner for the deflated GMRES(m)

We are interested in the solution of the linear system

$$Ax = b \tag{4.1}$$

The GMRES method is among the best methods to solve this system when the coefficient matrix A is nonsingular and nonsymmetric. For large linear systems, the restarted version should always be used to reduce the memory and computational requirements. The deflated GMRES has been proposed to reduce the negative effects of the restarting procedure. The general idea behind these methods is to add to the Krylov subspace an approximation of the invariant subspace associated to the smallest eigenvalues. In [53], this is carried out by defining a preconditioner that is equal to the projected matrix onto the approximated invariant subspace and is taken as the identity on the orthogonal subspace. Hence, given $U = [u_1, \ldots, u_r] \in \mathbb{R}^{n \times r}$ the r-dimensional basis of the invariant subspace associated to the eigenvalues to deflate, the preconditioner is defined as

$$M_D^{-1} \equiv I_n + U(|\lambda_n| T_r^{-1} - I_r)U^T, \quad T = U^T B U, \tag{4.2}$$

where λ_n is the largest eigenvalue in magnitude, I_n and I_r are the identity matrices and B the initial preconditioned matrix. Since M_D^{-1} is nonsingular, the eigenvalues of the resulted matrix $M_D^{-1}B$ or BM_D^{-1} are $\lambda_{r+1}, \ldots, \lambda_n, |\lambda_n|$ with a multiplicity at least r. It is therefore expected to get a faster convergence rate with this preconditioner since the r smallest eigencomponents that slow down the convergence are deflated. This assumes that U is a good approximation of the basis of the selected invariant subspace. For large matrices however, the cost of accurately computing U (as suggested in [53] and later in [28]) may induce a significant overhead. This process should be carried out only when it is necessary, for instance to avoid stagnation.

We thus propose here an adaptive strategy that detects a near-stagnation in the iterative process or a slow reduction in the residual norm. This approach is based upon the work by

Algorithm 3 DGMRES(m, k, r): Restarted GMRES with adaptive deflation

input (m, $itmax$, ϵ, k, smv, bgv, $rmax$);
Set $B \equiv AM^{-1}$, M^{-1} is any external preconditioner
$r_0 = b - Ax_0$; $\quad U = [\,]$; $\quad M_D = I$; $\quad it = 0$; $\quad r = 0$;
while ($\|r_0\| > \epsilon$)
  Arnoldi process on B to get $BM_D^{-1}V_m = V_{m+1}\bar{H}_m$. See [114]
  $x_m = x_0 + M_D^{-1}M^{-1}V_m y_m$, y_m solution of $min\|\beta e_1 - \bar{H}_m y_m\|_2$;
  $r_m = b - Ax_m$, $it \leftarrow it + m$;
  If ($\|r_m\| > \epsilon$ and $it < itmax$) **then**
    $Iter = m * log(\frac{\epsilon}{\|r_m\|})/log(\frac{\|r_m\|}{\|r_0\|})$;
    If($(Iter > smv * (itmax - it)$ **and** $r < rmax)$ **then**
      Compute k Schur vectors of B noted X. See [53]
      Orthogonalize X against U
      Compute $T = \begin{bmatrix} U & X \end{bmatrix}^T B \begin{bmatrix} U & X \end{bmatrix} \equiv \begin{pmatrix} U^TBU & U^TBX \\ X^TBU & X^TBX \end{pmatrix}$
      Increase U by X; $r \leftarrow r + k$;
      If$(Iter > bgv * (itmax - it)$) **then**
        Improve U as indicated in [28, section 3]
      EndIf
      Factorize T $\quad$ Set $M_D^{-1} \equiv I_n + U(|\lambda_n|T^{-1} - I_r)U^T$
    End If
  End If
  $x_0 = x_m$, $\quad r_0 = r_m$
end while

[124] in which the Krylov subspace is adaptively increased along the cycles of GMRES(m); Here, we find it natural to enrich the subspace with the eigencomponents that slow down the convergence. The main steps are given in Algorithm 3. First, m steps of the Arnoldi process are performed to compute the orthonormal basis V_m. It also creates an upper Hessenberg matrix $H_m = V_m^T B V_m$ which is the restriction of B onto the $m-$dimensional Krylov subspace. Then, a least-squares problem is solved to minimize the residual norm in the Krylov subspace. At the time of restart, if the desired residual norm is not achieved, a stagnation test is computed to determine if a deflation process could be beneficial to accelerate the convergence. This test considers the convergence rate over the previous restart cycles and evaluates the number of iterations ($Iter$) needed to achieve the desired accuracy. If $Iter$ is greater than the remaining number of steps (bounded by a small multiple smv of the number of iterations allowed), then data are computed to update the preconditioner associated to the deflation process. This test is therefore used to reduce the iteration counts in GMRES(m). To detect a near-stagnation, we use another test which considers a large multiple bgv of the remaining number of steps. In this case, a harmonic projection is carried out to accurately compute the eigenvalues and continuously update the previous estimation of U.

4.3 Implementation notes

We now give some details about the implementation of Algorithm 3 on distributed-memory computers. The programming model is SPMD (Single Program Multiple Data) and communications are done using the message-passing interface (MPI). The adjacency graph of the input sparse matrix is first built. PARMETIS is then used to partition the vertices of the graph into D disjoint vertices. From this partitioning, the matrix is distributed such

that each processor holds a contiguous chunk of rows corresponding to the vertices it owns. The right hand side and all other vectors (Krylov basis, invariant basis) are distributed accordingly. Note that the goal of this data distribution is to get a good load balance and to minimize communication during matrix-vector multiply and preconditioning steps. When the additive Schwarz preconditioner is used, an overlapping partitioning can be defined by taking recursively adjacent vertices from the initial disjoint partitions.

The main parallel operations in Algorithm 3 so far are the matrix-vector multiply, scalar products, and the application of M^{-1} and M_D^{-1}. M^{-1} can be any parallel preconditioner as long as it implements the basic operation $v_j \leftarrow M^{-1}v_i$. In our tests, the restricted additive Schwarz has been used as defined in [30]. It is then necessary in the setup phase to factorize in each process the block matrices A_p corresponding to the restriction of A onto the defined subdomains. M_D^{-1} is applied to a distributed vector v_j in a straightforward manner given the data distribution described above. This implies r all-to-all communications to compute the projection onto the invariant subspace. There is no additional communication for the other terms since the $r \times r$ dense matrix T is owned by each process.

We provide an implementation of this method using the PETSc package (see [18]). The original implementation of the built-in *KSP GMRES* has been modified to provide the data needed for the deflation and to apply the resulting preconditioner to generate the Krylov basis. Although the current presentation does not discuss the choice of side of preconditioning, the implementation does define left and right preconditioning. Note that the current adaptive preconditioning can be associated with any other preconditioner available in the package or defined by the end user since we provide generic interface similar to the other Krylov subspace methods in the package. The resulted KSP module (named as DGMRES) is available in PETSc release 3.2.

4.4 Numerical experiments

This section presents some numerical results to prove the efficiency of the proposed approaches. The test problem arises from design optimization in computational fluid dynamics. The physical model is a 3D flow simulation in a jet engine compressor rotor. The physical equations are the Reynolds-Averaged Navier-Stokes for compressible flows, discretized using the finite volume method as presented by [15]. The matrices have been extracted from the software Turb'Opty^{TM} designed by the FLUOREM company. They are also available in the University of Florida sparse matrix collection (see [39]) under the name $RM07R$ in the FLUOREM group. The matrix is nonsymmetric and indefinite with a size 272,635 and 37,355,908 nonzero entries. Other test cases can be found in [105].

With this test case so far, previous studies have shown the limits of some existing solvers in terms of memory usage and numerical accuracy (see [103]). [106] have proved as well the instability of the ILU factorization to approximate the solution of linear subsystems. In our hybrid approach, we therefore rely on a direct solver within each subdomain, such as MUMPS [4].

4.4.1 Benefits of the deflated restarting

We now give the main benefits of using the deflated GMRES with the additive Schwarz method (ASM). It is known that one level ASM is a weak preconditioner when the number of subdomains D gets large. The size of the Krylov subspace m could then be increased to enhance the robustness of the global method. However, choosing a good size m of the Krylov subspace is a trial-and-error process. With the adaptive deflation, we show

experimentally that the method is robust for various values of m and D. Moreover, using a large number of subdomains reduces the memory required to handle the submatrices by the direct solver. Hence it is expected that the time to factorize these matrices and the memory required will get smaller as D increases. This is reported in the last column of Table 4.1. We also report the number of matrix-vector multiplies and the global CPU time with respect to the number of subdomains D. We then compare the restarted version (GMRES(m)) with the deflated version (DGMRES(m, k)), where $m = 48$ and 64. A dash in a field means that the relative residual norm of 10^{-8} is not reached after 2500 iterations. It can be observed that DGMRES provides reliable and faster convergence than the classical restarted GMRES. It also gives a faster method since significantly fewer iterations are needed. Furthermore, the method reveals a substantial acceleration as the number of processors increases. Note that without the deflation, this acceleration will not be obtained since the number of matrix-vector multiplies increases hugely with the subdomains. For instance, this behavior can be seen with GMRES(64) when using $D = 16$ and $D = 32$.

Table 4.1: RM07R : Benefits of using DGMRES with an additive Schwarz preconditioner and an overlap of 1. The deflation process reduces the total number of iterations and helps to use a large number of subdomains and thus a large number of processors. Here, the number of processors is indeed equal to the number of subdomains.

D	GMRES(48)		DGMRES(47,1)			GMRES(64)		DGMRES(63,1)		
	Matvecs	Time	Matvecs	Time	r	Matvecs	Time	Matvecs	Time	r
16	551	230	212	173.4	3	355	193.8	208	168.9	2
32	-	-	533	109.2	4	2217	244.6	455	94.6	7
64	-	-	410	56.8	4	-	-	453	50.8	7
128	-	-	791	51.5	15	-	-	638	44.3	8

4.4.2 Adaptive DGMRES and Full GMRES

From the robustness standpoint, the full GMRES approach is more reliable than the restarted version even with the deflation process. However as the size of the basis grows, it should be more sensitive to round-off errors. To illustrate this behavior, we consider two formulations of the Arnoldi process, namely the classical Gram-Schmidt (CGS) and the modified Gram-Schmidt (MGS) algorithms. The former is sometimes preferred since it provides good kernel operations in parallel environments. In the PETSc package, for instance, it is used by default in the GMRES implementation as the orthogonalization method with a possible iterative refinement strategy. In Figure 4.4.1, the residual history is displayed with respect to the number of matrix-vector products. The method stops when the relative residual norm is 10^{-10}. It can then be noticed that with CGS, stagnation occurs in the full GMRES (in solid line) due to severe cancellation in the algorithm and consequently a loss of orthogonality. This does not happen when the basis is small since the round-off errors are not propagated very far and DGMRES (dash-dotted line) converges at the desired accuracy even with CGS. Note that although good accuracy is finally achieved in full GMRES with MGS (dashed line), it will require much more memory to store all the vectors of the growing Krylov basis (265 vectors in this case). In DGMRES, the Krylov basis is stored just for one cycle. Only the invariant basis U is stored over the restart cycles together with vectors $M^{-1}AU$ to reduce the matrix-vector counts. Thus in this example, only $63 + 7 \times 2 = 77$ vectors are stored. Note also that this number can be further reduced by using a smaller Krylov basis since convergence is still good, as shown

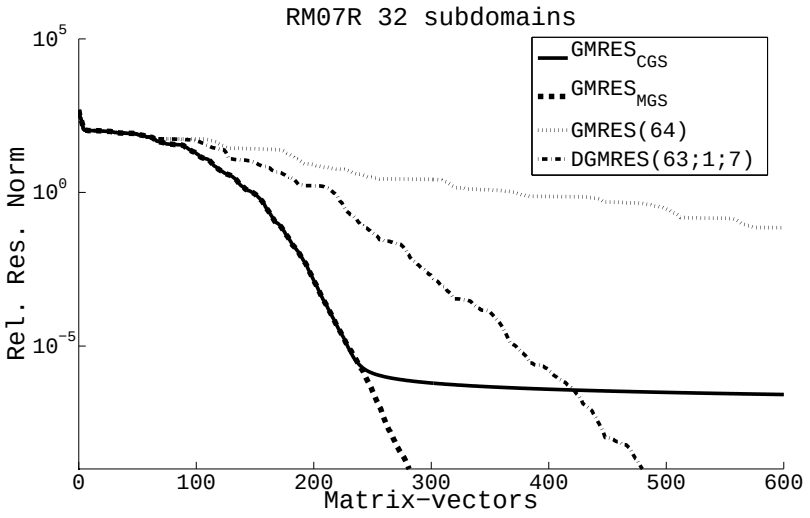


Figure 4.4.1: Convergence of full GMRES, GMRES(m) and DGMRES(m, k, r) with classical Gram-Schmidt(CGS) and modified Gram-Schmidt (MGS) orthogonalization scheme. k is the number of eigenvalues to extract at each detected stagnation and r is the total number of eigenvalues extracted at the convergence. 32 subdomains are used in the additive Schwarz method with a 1-overlap.

in Table 4.1.

4.5 conclusion

We have designed an adaptive deflation strategy that can be used for preconditioned GMRES. We show in this paper that the proposed algorithm can be used to improve the robustness and reduce both CPU time and memory required by hybrid solvers based on a one level additive Schwarz method. We have implemented this method in the new module DGMRES of the PETSc library.

Acknowledgments : This work is funded by the French National Agency of Research under the contract ANR-TLOG07-011-03 LIBRAERO. The work of the first author was done while visiting the NCSA at Urbana-Champaign in the context of the Joint laboratory INRIA-University of Illinois. Experiments in this paper have been carried out using the *parapide* cluster in the GRID'5000 experimental testbed (see `https://www.grid5000.fr`). We thank the referees for providing many instructive comments.

CHAPTER 5

Memory Efficient Hybrid Algebraic Solvers for Large CFD Linear Systems

Joint work with
François PACULL,

Abstract: This paper deals with the solution of large and sparse linear systems arising from design optimization in Computational Fluid Dynamics. From the algebraic decomposition of the input matrix, a hybrid robust direct/iterative solver is often defined with a Krylov subspace method as accelerator, a domain decomposition method as preconditioner and a direct method as subdomain solver. The goal of this paper is to reduce the memory requirements and indirectly the computational cost at different steps of this scheme. To this end, we use a grid-point induced block approach for the data storage and the partitioning part, a Krylov subspace method based on the restarted GMRES accelerated by deflation, a preconditioner formulated with the restricted additive Schwarz method and an the splitting of aerodynamic/turbulent fields at the subdomain level. Numerical results are presented with industrial test cases to show the benefits of these choices.

***Keywords**:Hybrid linear solvers ; Deflated GMRES ; Restricted additive Schwarz preconditioner ; FieldSplit preconditioner ; Weighted graph partitioning ;PETSc ; CFD optimization ;Jacobian matrix*

5.1 Introduction

This paper deals with sparse linear systems in CFD involving large Jacobian matrices. Solving industrial problems of this type requires the implementation of efficient parallel algorithms. The hybrid algebraic solvers consist in a global iterative solver on one side, a local direct solver on the other, connected together through a preconditioner induced by a domain decomposition method. Both types of solvers, i.e. iterative and direct, are highly memory consuming when applied to these CFD matrices. The motivation of the present work is to lower this memory usage, so as to increase the size of the systems solved and meet the ever-growing demand for large meshes. The actual hybrid solver described

in this paper is based on the GMRES algorithm [114], the Restricted Additive Schwarz (RAS) method [30] as well as LU decomposition. We show that a significant reduction of the demanded memory can be achieved by using some various techniques at every level of the hybrid solver: the GMRES global level, the RAS domain decomposition one, and the LU local one. Implementations are extensively based on the Portable Extensible Toolkit for Scientific computations (PETSc) library [19, 18, 20].

The remainder of this paper is organized as follows. Section 5.2 is a general introduction covering the linear systems, the linear solver as well as the memory aspect. Section 5.3 details the different approaches implemented in the present study. Section 5.4 presents an application of the points developed in section 5.3 on a selection of test matrices. Finally, we summarize some important results in section 5.5.

5.2 Context

Before considering some numerical aspects related to the hybrid solver, it is important to describe the category of CFD matrices addressed in this paper.

5.2.1 The Family of Linear Systems

Although CFD and parametrization are not the topics of this paper, we give in this subsection some information related to these fields that are relevant to the linear systems presented in Section 5.4. We are dealing with the system:

$$Ax = b \tag{5.1}$$

where A is an $n \times n$ sparse matrix with real entries, representing the Jacobian of the discretized flow equations' residual with respect to the fluid unknowns. In the current paper, the linearized PDEs are the Reynolds-Averaged Navier-Stokes equations for compressible flows of Newtonian fluids, discretized using the finite-volume method. The cases presented in the following are all based on multi-block structured meshes, consisting of piecewise structured arrays of hexahedra. These linear systems result from the parameterization software Turb'Opty^{TM} relying on automatic differentiation, as first presented by S. Aubert *et al* in [15]. These Jacobian matrices are evaluated near enough from a stationary point of the flow dynamical system, so as to have all of their eigenvalues featuring a strictly positive real part. Since this equilibrium might be very close to unstable, some eigenvalues may lay arbitrarily close to the imaginary axis, while the maximum imaginary-part magnitude is rather linked to the Reynolds number of the flow.

The computational stencil being very small compared to the global number of grid points, the matrix A presents a high degree of sparsity even if the reduced bandwidth may still be relatively large in 3D, where the regular stencil uses 25 mesh nodes. Because variables within a grid point are coupled by the equations in a tight and invariable manner, the initial ordering of A is based on the grid-point approach: the values of the n_u unknowns at each grid point are stuck together as a n_u-tuple, leading to a Jacobian matrix made of tiny dense $n_u \times n_u$ blocks. For example, in a 3D case with the $k-\omega$ turbulence model [134], we have the following 7-tuple of conservative variables: $(\rho, \rho u, \rho v, \rho w, \rho E, \rho k, \rho\omega)$, where variable ρ denotes the density, (u, v, w) the velocity, E the total internal energy, k the turbulent kinetic energy and ω the specific turbulent dissipation rate. So the matrix

A is structured in the following way:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,N} \\ A_{2,1} & A_{2,2} & & A_{2,N} \\ \vdots & & \ddots & \vdots \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{bmatrix} \tag{5.2}$$

where $A_{I,J} \in \mathbb{R}^{n_u \times n_u}$ for $1 \leqslant I, J \leqslant N$ and $N = n/n_u$ is the number of mesh nodes. Note that in this paper, we use the capital I, J letters for referring to the indices of these sub-matrices of order n_u. Another noteworthy feature of A is the high degree of heterogeneity among the entries of the matrix even within these small sub-blocks, related to the different levels of magnitude of the flow variables.

An important point about the discretization is that the second-order Jameson-Schmidt-Turkel centered scheme [82] is used for the convective part, which induces the diagonal blocks to be weak in magnitude compared to the extra-diagonal blocks, but generally relatively well conditioned thanks to artificial dissipation terms. These extra-diagonal block entries can have very large negative or positive values. Also, the symmetric part $(A + A^T)/2$ of the Jacobian is not positive-definite.

Finally, matrix A is characterized by its anisotropy, which is twofold: the first kind is due to the highly directionally-dependent convective and acoustic waves inherent to compressible flows, and the second one to the large cell ratio in the boundary layers and far field.

We now give a short overview of the solver implemented with the PETSc library.

5.2.2 The Hybrid Algebraic Solver

The system (5.1) could be solved using a direct solver, but as the size of the matrix increases, the number of fill-in elements in the L and U factors grows drastically, which prohibits the use of these solvers on industrial problems that can easily make use of millions of mesh nodes. Also, basic iterative methods are found to be ineffective for the targeted linear systems and multigrid methods, outrageously difficult to adapt. The group of Krylov subspace methods is the only industrial option left. Matrix A being nonsymmetric, the GMRES method [114] is appealing with its monotonic convergence behavior resulting from an optimality condition indirectly imposed at each iteration. However, this recurring minimization process does have arithmetic and memory costs, both increasing with respect to the iteration steps, and mainly originating from the orthogonalization in the Arnoldi process and the Krylov basis storage (see [113] for more details). Restarting the process after a given number of iterations is the way of limiting these costs compared to the aforementioned full GMRES approach: the monotonicity property is kept but not the global optimality condition, reduced to the local space built in-between the successive restarts.

In order to aim a good convergence speed of the iterative solver, preconditioning is always applied to A. The preconditioning of massive nonsymmetric matrices with large off-diagonal elements still remains an active field of research and an *a priori* optimal strategy is usually hard to guess. If we note M the preconditioner matrix and consider right-preconditioning, the aim is to find a cheap approximate of A such that AM^{-1} is not too ill-conditioned, has its eigenvalues as much clustered as possible and is not at a too large distance from normality. This kind of approximate is usually chosen within the family of domain decomposition methods, in which the PDEs locality is fully exploited in order to get independent problems on the subdomains and thus an induced preconditioner that is

appropriate for parallel computing. The preconditioner used in the following is based on the RAS method [30], which relies on overlapping subdomains.

We now consider the quotient graph of A, which is the block-wise adjacency graph related to the physical mesh, and call W the set of vertices $\{1, 2, ..., N\}$. The graph partitioning is performed using ParMETIS [86], splitting the domain into p non-overlapping subdomains, thus leading to p disjoint sub-sets $\{W_i\}_{1 \leqslant i \leqslant p}$, whose union is equal to W. Given that $W_i^0 = W_i$ for $1 \leqslant i \leqslant p$, we can recursively define the δ-overlap partition of W:

$$W_i^{\delta+1} = W_i^\delta \cup \{J \in W \ \ s.t. \ \ (I, J) \text{ is an edge of the quotient graph and } I \in W_i^\delta\}. \quad (5.3)$$

Now we note n_i^δ the size of the vector sub-space spanned by the blocks with indices in W_i^δ, for $1 \leq i \leq p$ and $\delta = 0, 1, 2, ...$ and so on. We call $R_i^\delta \in \mathbb{R}^{n_i^\delta \times n}$ the restriction operator from $\mathbb{R}^n$ to the i-th vector sub-space associated to W_i^δ. Therefore, the local operators are:

$$A_i^\delta = R_i^\delta A (R_i^\delta)^T, \ \text{ for } \ 1 \leq i \leq p. \quad (5.4)$$

Then the Restricted Additive Schwarz (RAS) preconditioner is the following one:

$$M^{-1} = \sum_{i=1}^{p} (R_i^0)^T (A_i^\delta)^{-1} R_i^\delta. \quad (5.5)$$

where $R_i^0 \in \mathbb{R}^{n_i^\delta \times n}$ is the restriction operator from $\mathbb{R}^n$ to the i-th vector sub-space associated to W_i^0, that is, without overlap. An important point is that M^{-1} is never explicitly constructed: given a vector $v \in \mathbb{R}^n$, the process with rank i only deals with the local part of the $M^{-1}v$ matrix-vector product, which is $(A_i^\delta)^{-1} R_i^\delta \ v$.

Regarding the factorization of the A_i^δ's, it is found that incomplete factorization suffers from instability when applied to the matrices presented in the sub-section 5.4.1 [106]. Since this wide subject is beyond the scope of this paper and still under investigation, complete LU factorization is always used in the following, which allows the qualitative evaluation of the different methods without depending on the worth of a given incomplete factorization: what works well with a LU factorization as a subdomain solver could also be useful with an incomplete LU decomposition of good quality, associated or not to a local iterative solver if ever an exact subdomain solve is required. The multifrontal sparse direct solver MUMPS [4, 7] is used here to compute a LU decomposition of $A_i^\delta = L_i^\delta U_i^\delta$ and apply the forward and backward substitutions to a local vector.

The memory cost of a LU decomposition being very large, especially when the envelope of A_i^δ cannot be substantially reduced as for the 3D cases, we limit the overlap to $\delta = 1$, yielding the following preconditioner:

$$M^{-1} = \sum_{i=1}^{p} (R_i^0)^T (U_i^1)^{-1} (L_i^1)^{-1} R_i^1. \quad (5.6)$$

Lastly, we mention a few details about the solver: the modified Gram-Schmidt algorithm is used to orthogonalize the Arnoldi vectors, the preconditioner is positioned on the right side of the operator, the initial solution vector is null, and finally, the system is initially scaled with left and right diagonal matrices, such that the row and column norms of A are roughly close to one. This last point appears to be beneficial to the GMRES convergence behavior, as it condenses the distribution of the eigenvalues of A.

Having considered the main solver features, it is now possible to look in short into its memory consumption.

5.2.3 The Memory Issue

Whether the preconditioner is induced by a Schwarz or a Schur domain decomposition method, the memory used by the hybrid solver is mainly composed of the L_i and U_i subdomain factors on one side, and the Arnoldi vectors on the other.

As the GMRES iterations proceed, new basis vectors are computed and orthogonalized against the previous vectors in the basis. When the method restarts after a fixed number of iterations, say m, the number of vectors generated in one pass is bounded by m and the memory complexity is $\mathcal{O}(n \times m)$. In the full GMRES, this number of vectors is as high as n since the method converges theoretically after n iterations in exact arithmetics or more specifically after d iterations where d is the degree of the minimal polynomial of AM^{-1} [114]. Hence the worst memory cost is $\mathcal{O}(n \times d)$. Although the preconditioner M^{-1} is used to relax this bound, it is difficult to predict the number of steps needed to converge as it depends both on the problem and preconditioner. Hence no assumption could be made *a priori* for the memory required to store the Krylov basis. For large systems, a very rough empirical rule when incomplete LU is used within the subdomains and when the factorization is good enough to lead to convergence, is that about half of the memory is dedicated to the factors' storage while the other half is taken progressively by the Arnoldi vectors of the full GMRES. We propose in the sub-section 5.3.4 an implementation of a restarted GMRES based on deflation [53] and also an adaptive strategy to control the memory usage. Compared to the full GMRES, the proposed strategy requires less memory on the supplied test cases. This deflated GMRES is also likely to be more robust than the restarted GMRES for the same basis length. In addition, edge weights are used in the domain partitioning process in order to improve the convergence rate of the GMRES process, and so, to reduce both Krylov basis and deflation subspace sizes. This will be described in sub-section 5.3.2.

Finally, recall that the discretization 3D stencil involves 25 mesh nodes, which implies that the average number of scalar entries per row is about 175 in the 7 unknowns per node case. As a consequence, the memory required in 3D cases for storing the complete factors for each subdomain solve is exorbitant. As proposed by the PETSc library, we can reduce this memory burden by splitting each subdomain operator into two-by-two block matrices and only factorize the diagonal blocks while using a multiplicative block strategy to approach the subdomain inverse, as presented in the sub-section 5.3.3. This technique belongs to the family of physics-based preconditioners.

We now present the proposed techniques aiming at the memory load alleviation.

5.3 Some Key Elements in Memory Usage

The first basic way to improve the solver is to use the specific storage data structures offered by PETSc, in order to take advantage of the matrix block constitution as shown is (5.2), and possibly conjointly use point-block algorithms.

5.3.1 Scalar vs Block Data Format

If we note $Bnnz_A$ the number of non-zero blocks in A, the block-CSR format corresponds to three arrays: a $\{val(K),\ 1 \leqslant K \leqslant Bnnz_A\}$ array pointing toward the stored dense blocks, a $\{col(K),\ 1 \leqslant K \leqslant Bnnz_A\}$ array containing the block column indices of the blocks stored in val and a $\{row(J),\ 1 \leqslant J \leqslant N+1\}$ array of pointers toward the beginning of each stored block row in val and col. If we assume that the val vector is made of 8 bytes

scalars, using double precision floating point, while the two other vectors are made of 4 bytes integers, the required memory for the storage of A is:

$$Bnnz_A \times n_u^2 \times 8 + (Bnnz_A + N + 1) \times 4 \text{ bytes.} \tag{5.7}$$

The same matrix stored in a non-block scalar CSR format needs the following storage memory:

$$Bnnz_A \times n_u^2 \times 8 + (Bnnz_A \times n_u^2 + n + 1) \times 4 \text{ bytes.} \tag{5.8}$$

The size of the *col* array is thus multiplied by a factor n_u^2 and the *row* one by a factor close to n_u. It is indeed advantageous to store matrix A as a block matrix as apposed to a scalar matrix. Also, the data storage format of A may differ from the one used for the preconditioner M^{-1}, which may be imposed by the software library used for the factorization part, as it is the case with MUMPS, requiring a scalar storage. Such a storage of A is also found to be more effective regarding the computational time: when using blocks, the sparse matrix consisting in a collection of small dense matrices of constant size, operations on an upper level than blocks are treated as scalar operations while lower operations involving block entries may be optimized.

Given these advantages of using a block data format, it is also efficient to base the partitioning on the quotient graph, which is the reduced graph adjacent to A based on the blocks instead of the scalar entries position. This allow to save some memory and computational time during the partitioning step. In addition, the quotient graph being related to the physical mesh, partitions based on it are probably more natural since they do not split the strongly coupled unknowns located at a same mesh node among different processes.

We now look at the combinatorial problem of partitioning the quotient adjacent graph of A.

5.3.2 The Partitioning

In the process of partitioning, ParMETIS divides the set W into p sub-parts $\{W_i^0\}_{1 \leqslant i \leqslant p}$. At first, the partitioner must be given weights assigned to each vertex in W and to each edge (I, J) of the quotient graph. The default choice is to assign a unit weight to each vertex and to each edge. The edge cut is the set of all edges that have one vertex in one sub-part and the other vertex in another one, while the edge cut weight is the sum of all these edge weights. While dividing the domain, the partitioner aims at minimizing the edge cut weight, which purpose is the reduction of the total amount of communication in the solver. The constraint of this minimization process is to build sub-parts with nearly equal total vertex weight, for an homogeneous load balancing.

The idea implemented here is to take into account the coupling between the mesh nodes, due to the physics or the mesh cell aspect. The graph given to the partitioner being undirected, we use a symmetric definition for the weight $W_{I,J}$ of unordered (I, J) vertex pairs:

$$W_{I,J} = \frac{1}{2} \left(\|A_{I,J}\|_F + \|A_{J,I}\|_F \right), \tag{5.9}$$

where $\|.\|_F$ denotes the Frobenius norm. This matrix norm is chosen for its computational simplicity and its evenness with respect to all the block entries. Given these edge weights, ParMETIS is trying to avoid cutting the domain through strongly coupled edges. This results in reducing the coupling between the subdomains while increasing the inner subdomain coupling. The vertex weights are kept at one.

In Figure 5.3.1 and Figure 5.3.2, a physical domain is partitioned by ParMETIS into 8 subdomains with uniform weights or with weights based on the Frobenius norm. The geometry is a RAE airfoil profile with a C-shaped mesh. The flow simulation is done with a far-field Mach number of 0.3. The lines of strongest coupling follow the streamlines except for two different area: in front of the leading edge and along the suction and pressure sides, where the lines are perpendicular to the wing wall boundary, and in the wake of the wing, where the lines are vertical. The partitioning on Figure 5.3.1 exhibits numerous cuts across the strongly coupled lines, for example in the sub-part located entirely in the suction side boundary layer, or in the sub-part stretched all along the wing's wake. The partitioning on Figure 5.3.2 does avoid these heavy weighted edge cuts. We remark on this last figure that some subdomain may consist in non-connected region when using the Frobenius norm edge weights.

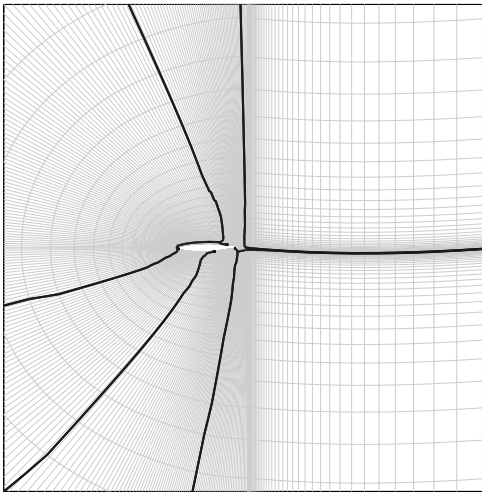


Figure 5.3.1: Partitioning into 8 subdomains using uniform edge weights

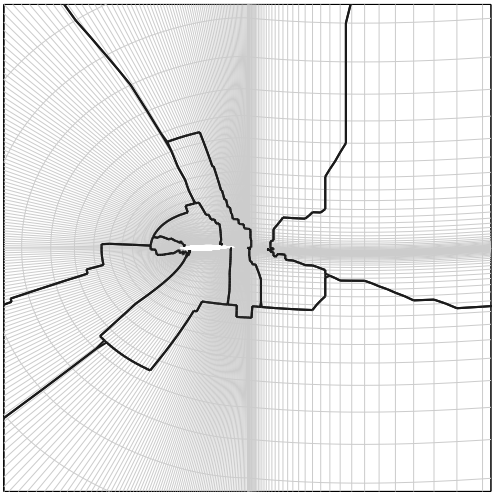


Figure 5.3.2: Partitioning into 8 subdomains using Frobenius norm edge weights

Alternatively, we present in the next sub-section a significant way of reducing the memory occupied by the LU factors, while having a relatively small impact on the effect of the domain-decomposition induced preconditioner.

5.3.3 Splitting the Fields

The local operator for each subdomain of the preconditioner is A_i^δ, for $1 \leq i \leq p$. For the purpose of reducing the number of decomposition factors, these local matrices can easily be splitted by PETSc into a two-by-two block matrix form:

$$A_i^\delta \equiv \begin{bmatrix} A_{1\,1} & A_{1\,2} \\ A_{2\,1} & A_{2\,2} \end{bmatrix}. \tag{5.10}$$

Since the five aerodynamic variables (ρ, ρu, ρv, ρw, ρE) are mutually strongly coupled and so are the two turbulent ones (ρk, $\rho \omega$), we propose an aerodynamic/turbulent variable

splitting, where $A_{1\,1}$ corresponds to the turbulent variables (block size equals 2), $A_{2\,2}$ corresponds to the aerodynamic variables (block size equals 5), while $A_{1\,2}$ and $A_{2\,1}$ carry one-way information about the coupling between the aerodynamic and the turbulent variables.

We used in this paper the multiplicative approach, which is the following:

$$(A_i^\delta)^{-1} \approx \begin{bmatrix} A_{1\,1} & 0 \\ A_{2\,1} & A_{2\,2} \end{bmatrix}^{-1} = \begin{bmatrix} A_{1\,1}^{-1} & 0 \\ -A_{2\,2}^{-1} A_{2\,1} A_{1\,1}^{-1} & A_{2\,2}^{-1} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & A_{2\,2}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{2\,1} & I \end{bmatrix} \begin{bmatrix} A_{1\,1}^{-1} & 0 \\ 0 & I \end{bmatrix}. \tag{5.11}$$

From (5.11), we see that applying this operator is simple and only requires factorizing the diagonal blocks:

$$(A_i^\delta)^{-1} \approx \begin{bmatrix} I & 0 \\ 0 & U_{2\,2}^{-1} L_{2\,2}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{2\,1} & I \end{bmatrix} \begin{bmatrix} U_{1\,1}^{-1} L_{1\,1}^{-1} & 0 \\ 0 & I \end{bmatrix}. \tag{5.12}$$

For all the cases tested so far, there was only a very little difference regarding the GMRES convergence behavior between assigning the turbulent variables to $A_{1\,1}$ and the aerodynamic ones to $A_{2\,2}$ or the converse situation. Also, in the majority of the cases, the convergence penalty as compared to the full factorization of the subdomain block is reasonable as we will see in the next section. However, the benefit of such a technique is multiple. First of all, this represents roughly 40% less entries in the factorization matrices, since a matrix based on 7×7 blocks is substituted by two matrices with the same structural pattern and based respectively on 5×5 and 2×2 blocks. This allow the saving of computational time as well, each time the preconditioner is applied, and improves the homogeneity of the entries within the blocks that are factorized.

However, we believe that in some rare cases, this fieldsplit technique can deteriorate the preconditioner. Inverting the two-by-two block matrix from (5.11) requires that $A_{1\,1}$ and $A_{2\,2}$ are both nonsingular, but one of these two blocks might be close to singular, while the global matrix (5.10) is far from singular. From our experience, the turbulent variable block is the one that might be, in some particular subdomains, at a close distance to singularity.

In the next sub-section, we describe a deflation technique, consisting in removing from AM^{-1} an invariant subspace that handicaps the restarted GMRES behavior.

5.3.4 Deflation

The Krylov subspace accelerator is based on the GMRES method proposed by Saad and Schultz [114]. Its superlinear convergence has been related to the convergence of Ritz values [132] which occurs automatically when the Krylov subspace is large enough. In a practical implementation, the GMRES method restarts the iterative process after a given number of iterations, say m and takes the last iterate of the previous cycle as the initial guess for the current cycle. The main motivation of the restarting procedure is to reduce the cost of storing the Krylov basis and to maintain its orthogonality. However, the accumulated spectral information gathered from the Arnoldi iteration are lost at the time of restart. As a result, the method could experience a slow convergence or even a stagnation if the restart occurs too early.

The eigenvalue deflation has been proposed to enhance the convergence rate of the restarted GMRES by keeping the Ritz values at the time of restart for the subsequent restart cycles [16, 28, 53, 87, 98, 99]. This work is based on the approaches (DEFLATED-GMRES(m, r) and DEFLATION(m)) developed in [53, 28] which build a preconditioner from the basis of an invariant subspace associated with the eigenvalues to deflate. We briefly present the key steps of this approach and we use the adaptive strategy proposed

by Sosonkina *et al* [124] to bound the size of the invariant subspace associated to the deflation.

Let P be an invariant subspace of dimension r corresponding to the r smallest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_r$ of AM^{-1}. Let the columns of U form an orthonormal basis for P. The deflated GMRES proposed in [53] is based on the following theorem:

Theorem 5.3.1. *If $T = U^T AM^{-1}U$ and $\tilde{M} = I_n + U(|\lambda_n|T - I_r)U^T$, then M is nonsingular and*

$$\widetilde{M}^{-1} = I_n + U(|\lambda_n|^{-1} - I_r)U^T, \tag{5.13}$$

and the eigenvalues of $AM^{-1}\widetilde{M}$ are $\lambda_{r+1}, \lambda_{r+2}, \ldots, \lambda_n, |\lambda_n|$, the last term has a multiplicity at least r.

This theorem is formulated for a preconditioned matrix AM^{-1}. The proof in [53] is still valid in this case because we still use the basis of an invariant subspace associated to the smallest eigenvalues of the preconditioned matrix. Likewise, the result is valid for a left preconditioned matrix $M^{-1}A$. Now, if $\widetilde{M}^{-1}$ is used as preconditioner in the restarted GMRES(m), we would likely have a better convergence than in the restarted version since the r smallest eigencomponents that slow down the convergence are removed from the system [53].

To build the preconditioner $\widetilde{M}^{-1}$, new eigencomponents should be computed at each restart to form the basis U. In the absence of the exact smallest eigenvalues of the preconditioned matrix, the Ritz values from the Arnoldi iteration can be used as approximate values. At step m, the Arnoldi process produces the Hessenberg matrix H_m and the basis vectors $V_{m+1} = [v_1, v_2, \ldots, v_m, v_{m+1}]$ such that $AM^{-1}V_m = V_{m+1}\bar{H}_m = V_m H_m + h_{\{m+1,m\}} v_{m+1} e_m^T$. Let $(\lambda, V_m y)$ be an approximate eigenpair extracted from the subspace $span\{V_m\}$ where y is an m-dimensional vector. Using a standard orthogonal projection technique, we get:

$$\begin{aligned} V_m^T(AM^{-1} - \lambda I)V_m y = 0 \\ \Rightarrow H_m y = \lambda y. \end{aligned} \tag{5.14}$$

The eigenvalues of H_m are known as the Ritz values of AM^{-1}. After each restart, a Schur decomposition of H_m is computed and its eigenvalues are ordered in increasing order. We extract k Schur vectors S corresponding to the smallest eigenvalues of H_m. Therefore, $U = V_m S$ approximate the Schur vectors of AM^{-1} corresponding to its smallest eigenvalues. Instead of using the Ritz values,it appears from previous work [28, 34, 99] that using the harmonic Ritz values yield better approximation of the smallest eigenvalues of AM^{-1}. Now, using the subspace $span\{AM^{-1}V_m\}$ for the orthogonality condition, the Galerkin condition writes:

$$(AM^{-1}V_m)^T(AM^{-1} - \lambda I)V_m y = 0 \tag{5.15}$$

$$\Rightarrow (H_m + h_{\{m+1,m\}} H_m^{-T} e_m e_m^T)y = \lambda y. \tag{5.16}$$

As before, the Schur vectors S corresponding to the smallest eigenvalues are extracted from the harmonic-Ritz eigenvalue problem in Equation (5.16). At each restart of GMRES, the DEFLATED-GMRES(m, k) will thus compute k Schur vectors noted X from the eigenvalue problem in Equation (5.16). After that, the new vectors are orthogonalized against the other vectors in U. At the beginning of the iterative process in GMRES, U does not contain any vector. For multiple right-hand side systems with the same matrix, the basis U may be kept at the convergence of the first system for subsequent solves. So far with the new set of Schur vectors $[U, X]$, the matrix T is updated using a block decomposition

and a LU factorization. The new basis U is increased by X for the next restart cycles and the subsequent applications of the preconditioner in Equation (5.13). We now analyze the memory consumption of DEFLATED-GMRES(m,k) and we compare it to that of GMRES(m) and full GMRES.

As in GMRES(m), the number of basis vectors is bounded in DEFLATED-GMRES(m,k) by m. At each restart, k new Schur vectors are added in the basis U. The matrix $AM^{-1}U$ is kept as well to save operations during the computation of T. The memory cost is thus $2nqk$ for q restarts. As advised in [53], this cost can be bounded by stopping the deflation after a fixed number of restarts. This is equivalent to bound the size of the basis U, say r. Now the DEFLATED-GMRES(m,k) will require less memory than the full GMRES if $rk < d$ where d is the number of iterations in the full GMRES. Note that the computational cost of DEFLATED-GMRES(m,k) and the memory needed by U increase with r. To limit these costs, we introduce an adaptive strategy which determines the convergence rate before adding new Schur vectors in U. This strategy is based upon the work by Sosonkina *et al* [124] in which the Krylov basis length is adaptively increased if the desired residual norm cannot be met in the remaining number of steps allowed. Here, we rather increase the basis U with more approximations of the smallest eigencomponents that slow down the convergence. The main steps are given in Algorithm 4. As stated above, the method starts with an empty basis U. At each restart, the convergence rate is computed and the estimated number of remaining steps is derived ($Iter$). If the desired accuracy ϵ cannot be met during these steps, new Schur vectors are computed to update the basis U. Note that $\widetilde{M}^{-1}$ is never formed explicitly. Its expression is rather used when necessary.

Algorithm 4 DGMRES$(m,k,rmax)$ - Restarted GMRES with adaptive deflation

Require: m, $itmax$, ϵ, k, r,smv,$rmax$;
1: Set $B \equiv AM^{-1}$, M^{-1} is the RAS preconditioner
2: $r_0 = b - Bx_0$; $\quad U = [\,]$; $\quad \widetilde{M} = I$; $\quad it = 0$; $r = 0$;
3: **while** $(||r_0|| > \epsilon)$ **do**
4: *Run a GMRES(m) cycle (m iterations) on* $B\widetilde{M}^{-1}$ *to get* x_m *and* r_m *(See [114])*
5: $it \leftarrow it + m$;
6: **if** $(||r_m|| > \epsilon$ and $it < itmax)$ **then**
7: $Iter = m * log(\frac{\epsilon}{||r_m||})/log(\frac{||r_m||}{||r_0||})$;
8:
9: **if** $(Iter > smv * (itmax - it)$ **and** $r < rmax\ )$ **then**
10: *Estimate k smallest eigenvalues of* BM^{-1} *(See Equation 5.14 or 5.16)*
11: *Compute data for* $\widetilde{M}^{-1}$ *(Theorem 5.3.1);* $\quad r = r + k$;
12: **end if**
13: **end if**
14: $x_0 = x_m$, $\quad r_0 = r_m$
15: **end while**

In the next section, we present results of numerical experiments, illustrating on three test cases the behavior that can be anticipated from the previous sub-sections.

5.4 Results

We start by briefly describing the test cases and the computational platform. Then the results are organized into three parts: the edge weights, FieldSplit and deflation impact studies.

5.4.1 The Test Cases

Table 5.1 introduces the three test cases discussed in the present paper, two of which (RM07R and HV15R) can be found in the University of Florida sparse matrix collection [39]. Figures 5.4.1, 5.4.2 and 5.4.3 display their respective structural pattern, which are nearly symmetric.

Case	n	N	n_u	nnz_A
VV11R	277095	39585	7	30000952
RM07R	381689	54527	7	37464962
HV15R	2017169	288167	7	283073458

Table 5.1: matrix order (n), number of mesh nodes (N), block size (n_u) and number of non-zero entries (nnz_A) of the test cases

The RM07R matrix is a 3D case of a jet engine compressor with frozen turbulence. Hence the rows and columns associated to turbulence could be removed (block size would be 5) from the original matrix. The VV11R matrix is a low Mach number case of an axisymmetric vane: only a thin section of the vane is simulated, using some periodic boundary conditions. The HV15R matrix is a 3D car engine fan case, the flow exhibits a low Mach number. As for the RM07R case, only one inter-blade channel is simulated in association to some periodic boundary conditions. The periodicity as well as the structured mesh sub-blocks coupling can be observed in the natural matrix structural pattern for the three cases.

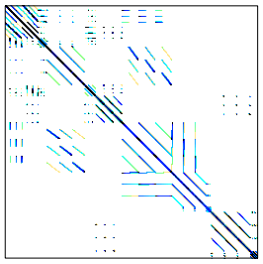


Figure 5.4.1: Structural pattern of VV11R

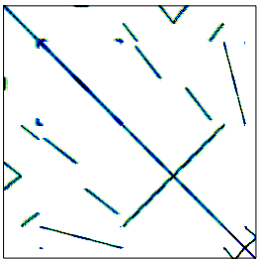


Figure 5.4.2: Structural pattern of RM07R

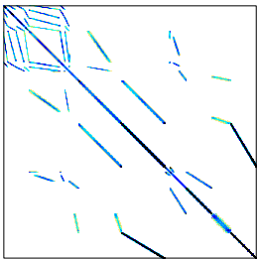


Figure 5.4.3: Structural pattern of HV15R

5.4.2 The platform of tests

Experiments are done on a distributed memory supercomputer *Vargas** which has 3,584 Power6 CPUs. Each Power6 CPU is a dual-core 2-way SMT with a peak frequency at 4.7 GHz. The computer is made of 112 nodes connected through an Infiniband network. Each node has 32 Power6 CPUs that access 128GB of local memory in a non-uniform way (hardware NUMA nodes). The programming model is SPMD-like with a message passing paradigm between the processes.

*http://www.idris.fr/su/Scalaire/vargas/hw-vargas.html

5.4.3 ParMETIS Edge Weights

The results given in this section evaluate the benefits of considering non-uniform edge weights during the matrix graph partitioning, as described in sub-section 5.3.2. In order to only assess one technique at a time, the full GMRES without FieldSplit is always used in this sub-section.

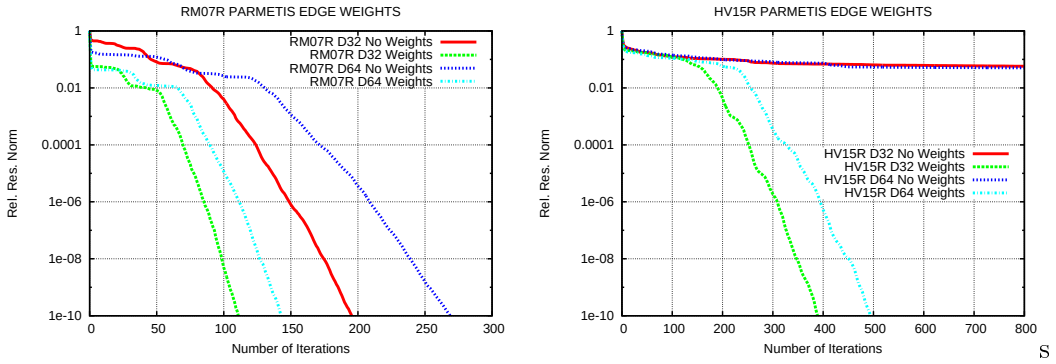


Figure 5.4.4: RM07R & HV15R: Influence of the edge weights in the convergence of the GMRES method, D32 indicates that 32 subdomains have been obtained from the ParMETIS partitioning

The results shown in Figure 5.4.4 confirm that this approach produces a good partitioning for the RAS preconditioner. We consider indeed 32 and 64 partitions on the RM07R and HV15R test cases and compare the GMRES convergence curves obtained with or without edge weights. It can be seen indeed that the fastest convergences in the hybrid solver are obtained when the edge weights are used during the partitioning. On RM07R for instance, the GMRES requires respectively 117 and 149 iterations for 32 and 64 subdomains. Without edge weights, it requires almost 200 iterations for 32 subdomains and more than 250 iterations for 64 subdomains. The effect of this approach is even stronger for HV15R. With uniform edge weights, the produced partitions do not contain enough relevant information inside each subdomain. As a result, the preconditioned GMRES stagnates after a few steps. Since no restart is done here in the GMRES, all the basis vectors are stored until convergence. Hence, using the edge weights will not only reduce the number of iterations but also the memory required to store those vectors. Sub-section 5.4.5 show how this memory can be further reduced.

5.4.4 The Aerodynamic/Turbulent FieldSplit

In this sub-section, we study the benefits of splitting the aerodynamic and turbulent fields during the factorization process as described in sub-section 5.3.3. We consider the two test cases VV11R and HV15R that contain turbulent variables. The full GMRES is always used in this sub-section, along with the Frobenius norm based edge weights in the partitioning process.

Table 5.4.4 reports the CPU time, the number of GMRES iterations as well as the memory required to store the L_i and U_i matrices from the factorization of the subdomain matrices. This memory size does not take into account any other part of the hybrid solver.

It can be seen, as expected, that the memory usage is saved about 40% by using the FieldSplit in the RAS preconditioner, if only the factorization size is considered. On the

D	With FieldSplit					Without FieldSplit				
	Setup	**Time**	**TI**	**ITS**	**MEM**	**Setup**	**Time**	**TI**	**ITS**	**MEM**
VV11R										
8	4.40	46.25	0.45	94	2.58	8.92	49.93	0.44	94	4.27
16	2.17	26.27	0.19	130	2.5	4.11	34.33	0.24	127	4.18
32	1.09	19.34	0.11	171	2.47	1.99	23.73	0.13	168	4.08
HV15R										
32	159.28	1464.0	2.13	612	78.58	401.41	1536.0	2.92	389	131.67
64	49.189	901.5	1.15	739	67.69	141.42	1024.0	1.79	494	115.21

Table 5.2: Benefits of the Fieldsplit for VV11R & HV15R; **D**: Number of subdomains and number of MPI processes, **Setup (s)** : CPU time for the factorization phase, **Time (s)**: Maximum CPU time overall processes, **TI (s)** : Average time per iteration, **ITS**: Number of iterations in GMRES (Matrix-vector product and preconditioning), **MEM (GB)**: Total memory overall processes used to store the preconditioner in GigaBytes

other side, for the HV15R case, the GMRES requires significantly more iterations when the FieldSplit is used, e.g. 612 against 389 in the 32 subdomains case. Over a set of five different matrices tested with the Fieldsplit, the HV15R case is the only case for which this behavior is observed. The VV11R case is representative of the usual FieldSplit results, which are that the convergence curves are almost the same with or without FieldSplit, e.g. 171 against 168 iterations in the 32 subdomains case. The HV15R behavior can be explained by the fact that without FieldSplit, the inverse of the subdomain matrices provides a more accurate expression of the RAS preconditioner. We suppose that either the aerodynamic or the turbulent block in one of the subdomains, at least, might be ill-conditioned, or that the extra-diagonal blocks in the FieldSplit two-by-two block decomposition are of extreme importance. Nevertheless, the fieldsplit approach produces the best CPU time at the different parts of the hybrid approach. The shorter setup time is explained by the amount of fill-in in the factors L_i and U_i. The average time per GMRES iteration is mainly depending on the time required to apply the preconditioner, which is also function of the size of the factored submatrices. This explains why the average time per iteration is lower when the FieldSplit is used.

5.4.5 DGMRES

The results presented in the previous sections use the non restarted version of GMRES. Hence, no assumption could be made on the memory required during the iterative phase to store the basis vectors. In this section, we present the benefits of using the adaptive deflation to reduce this memory usage as presented in section 5.3.4. Table 5.4.5 compare these strategies together with the classical restarted GMRES.

The results presented in the previous sub-sections use the non-restarted version of GMRES, without any consideration of the memory required to store the basis vectors during the iterative phase. In this section, we present the benefits of using the adaptive deflation to reduce this memory usage as presented in sub-section 5.3.4. Table 5.4.5 compare these strategies together with the classical restarted GMRES. The non-uniform edge weights are always used in this sub-section, while the FieldSplit is discarded.

When determining the memory usage in this iterative phase, we only take into account the vectors that have a size of n, the initial problem size. Let $nvec$ be this number of vectors. In the full GMRES (denoted as FULL-GMRES in the following), $nvec$ is equivalent to the number of iterations. In GMRES(m) it is equal to the basis length m. In DGMRES($m, r, rmax$), $nvec = m + 2r$ where r, bounded by $rmax$, is the number of

VV11R										
D	FULL-GMRES			GMRES(48)			DGMRES(32,2,15)			
	ITS	**Time**	**MEM**	**ITS**	**Time**	**MEM**	**ITS**	**Time**	r	**MEM**
16	127	34	33.5	288	66	12.6	194	45	2	9.5
32	168	24	44.4	670	72	12.6	318	37	2	9.5

RM07R										
D	FULL-GMRES			GMRES(48)			DGMRES(32,2,15)			
	ITS	**Time**	**MEM**	**ITS**	**Time**	**MEM**	**ITS**	**Time**	r	**MEM**
16	92	214	23.9	169	297	12.4	115	250	9	13.0
32	117	103	30.4	355	260	12.4	160	139	11	14.0
64	149	66	38.7	860	166	12.4	206	53	12	14.6

HV15R										
D	FULL-GMRES			GMRES(150)			DGMRES(96,10,70)			
	ITS	**Time**	**MEM**	**ITS**	**Time**	**MEM**	**ITS**	**Time**	r	**MEM**
32	389	1536	5980.1	(8.4E-08)	3,316	2304.0	510	2002	50	3010.5
64	494	1024	7598.08	(6.8E-01)	-	-	635	1500	61	3624.9

Table 5.3: Benefits of Deflation for VV11R, RM07R & HV15R; **ITS**: Number of GMRES iterations (or reduction in residual norm when the maximum number of iterations is reached; **Time (s)** : Maximum CPU time overall processes to compute the solution. **MEM (MB)**: Total memory overlall processes to store the basis vectors and data needed for the deflation; r : Size of the invariant basis U at the convergence.

extracted Harmonic Ritz vectors. In double precision, a scalar uses 8 bytes of memory. Hence, the total memory required in this phase of the hybrid solver is equal to $8 \times n \times nvec$ bytes. To show the benefits of using DGMRES, the restart length is choosen smaller than that in GMRES(m), the goal being to have about the same memory bound for these two restarted methods. Table 5.4.5 reports the memory usage, as just described, with respect to the three test cases and various number of subdomains. As expected, it can be noticed that for all configurations, DGMRES($m, r, rmax$) and GMRES(m) require less memory than FULL-GMRES. This is more noticeable with the largest test cases HV15R. Indeed, the ratio of the memory required in FULL-GMRES as compared to GMRES and DGMRES is close to 2. A second observation is that the FULL-GMRES memory usage increases with the number of subdomains, which is induced by the increased number of iterations. In DGMRES, only the size of the basis U increases.

The main empirical observation here is that DGMRES provide a trade-off between FULL-GMRES and the restarted GMRES. Compared to FULL-GMRES, the main benefit of DGMRES is that it bounds the memory required by the basis vectors. In FULL-GMRES, no assumption could be made *a priori* for the memory required to save the basis vectors. The immediate consequence is that the memory is allocated on the fly. In DGMRES, it is always possible to estimate the memory that will be used and thus to allocate this memory beforehand. Compared to the restarted GMRES, the provided numerical results as well as numerous other experiments show that DGMRES is more robust and exhibits a better convergence rate, which is illustrated in Figure 5.4.5: it can be seen that GMRES(m) tends to stagnate after each restart. By increasing the restart length m, the convergence rate is improved but the negative effects due to restarting remains. With DGMRES, only the size of the invariant basis is adaptively increased.

Besides the memory aspect, there is another benefit to DGMRES, in the particular case of a sequence of linear systems involving the same matrix and different Right-Hand Side (RHS) vectors. Then the basis U may be kept from one system to another. We mention

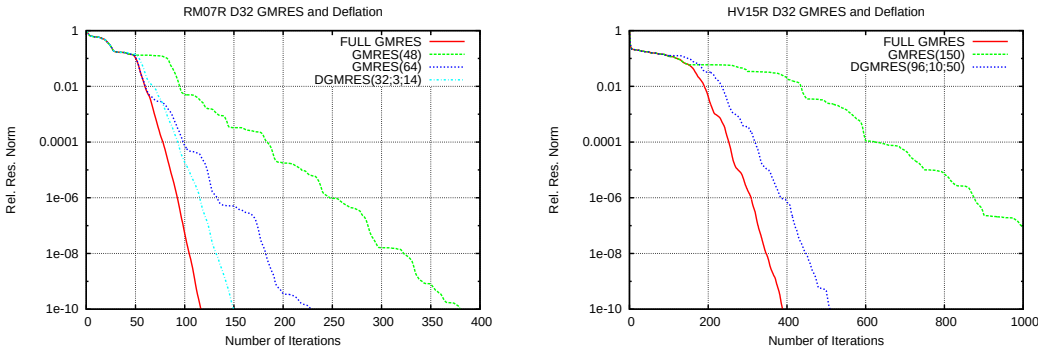


Figure 5.4.5: RM07R & HV15R : Benefits of the Deflation in GMRES(m). 32 subdomains are used.

here that multiple RHS cases are commun in the field of CFD optimization, each RHS accounting for a given geometric parameter for example [15]. We give in Table 5.4 the benefits of such approach on the test problems RM07R and HV15R, for which three and two distinct RHS vectors are used respectively. In the hybrid solver, the partitioning of the matrices and the factorization on the subdomain matrices are done only once. Thereafter, the iterative phase is performed successively for the distinct RHS vectors. For the first linear system, the initial U basis is empty. New vectors are then added adaptively within the maximum size allowed during the first solve. For the subsequent solves, the basis U is used at the first iteration to build the preconditioner associated to deflation, and is continuously and adaptively increased until the maximum size is reached.

RM07R									
	FULL-GMRES			DGMRES(32, 2, 15)			GMRES(48)		
RHS #	1	2	3	1	2	3	1	2	3
ITS	144	141	135	209	138	119	864	(3.4E-09)	368
HV15R									
	FULL-GMRES			DGMRES(96, 10, 70)			GMRES(150)		
RHS #	1	2		1	2		1	2	
ITS	494	529		664	445		(6.8E-01)	-	

Table 5.4: Number of iterations for multiple right-hand-side systems. In DGMRES, the basis U is kept from one subdomain to another. 64 subdomains are used in the additive Schwarz. Numbers in parentheses are the reduction in residual norm when the maximum number of steps is reached.

In Table 5.4, we give the number of iterations obtained for each linear system. It is clearly seen that DGMRES requires more iterations for the first system. However, for the second system, DGMRES performs better. It is worth to note that more sophisticated Krylov methods are available to solve sequences of linear systems by treating all the RHS at once [113, 100], however, the adaptive approach presented here does not require all the RHSs to be available at the same time. This could be used, for example, to design memory-efficient iterative solvers within the subdomains, associated to incomplete factorization. However, we leave it as future work as it requires more investigation.

5.5 Conclusion

It is found that within the framework of hybrid linear solvers, the association of the DGMRES algorithm, the FieldSplit technique and the weighted edge partitioning is efficient regarding memory requirements for the systems involving compressible flow Jacobian matrices. In this paper, the type of linear systems as well as the numerical methods and the associated results on different test cases have been presented. To summarize, the edge weights based on the block norm are found to produce a partitioning that greatly improves the RAS preconditioner. The drawback is an increase of the process communication due to a larger edge cut set. Then, the aerodynamic/turbulent FieldSplit allows a substantial saving of the factorization memory size, while also decreasing the CPU time per iteration. The penalty of this technique is a lower convergence rate in some ill-conditioned cases as compared to the non-split version. Finally, the DGMRES algorithm provides a good trade-off between the FULL-GMRES and the restarted GMRES, by bounding the memory required for the basis vectors and exhibiting a better convergence rate than the restarted GMRES. A future work direction involves using a robust incomplete instead of the complete LU factorization as a subdomain preconditioner, associated with the DGMRES algorithm with a deflation basis recycled over the global iterations, as a subdomain solver. The global method would be the FGMRES-DR [64] preconditioned by RAS.

Acknowledgement: This work was funded by the French National Agency of Research under the contract ANR-TLOG07-011-03 LIBRAERO. Numerical experiments have been done on the VARGAS supercomputer from GENCI-IDRIS (Grand Equipement National de Calcul Intensif - Institut du Développement et des Ressources en Informatique Scientifique). Preliminary tests have been carried out using the GRID'5000 experimental testbed (https://www.grid5000.fr).

CHAPTER 6

Parallelism and robustness in GMRES with the Newton basis and the deflation of eigenvalues

Joint work with
Jocelyne ERHEL,

Abstract: The GMRES iterative method is widely used as Krylov subspace accelerator for solving sparse linear systems when the coefficient matrix is nonsymmetric and indefinite. The Newton basis implementation has been proposed on distributed memory computers as an alternative to the classical approach with the Arnoldi process. The aim of our work here is to introduce a modification based on deflation techniques. This approach builds an augmented subspace in an adaptive way to accelerate the convergence of the restarted formulation. In our numerical experiments, we show the benefits of using this implementation with hybrid direct/iterative methods to solve large linear systems.

Keywords: *Augmented Krylov subspaces, Adaptive Deflated GMRES, Newton basis, Hybrid linear solvers*

6.1 Introduction

In this paper, we are interested in the solution of large systems of linear algebraic equations

$$Ax = b, \tag{6.1}$$

where A is a $n \times n$ real nonsingular matrix, b and x are n-dimensional real vectors. Practical algorithms transform the original problem (6.1) to the following

$$M_L^{-1} A M_R^{-1} \tilde{x} = M_L^{-1} b, \quad \tilde{x} = M_R x \tag{6.2}$$

where M_L^{-1} and M_R^{-1} are the action of preconditioning the system at left ($M_R = I$), at right ($M_L = I$) or both. On parallel computers, we assume that these preconditioners are

formulated from some algebraic decomposition of the input matrix. However, they can be any approximation of the inverse of the matrix A and we refer the reader to the survey on the preconditioning techniques [21]. These preconditioners are generally combined with Krylov subspace methods as accelerators. The GMRES method [114] is widely used in this context. From this method, many improvements have been proposed to enhance its robustness and parallel efficiency; see for instance [17, 28, 34, 50, 53, 51, 98, 118, 79, 107]. In this work, we propose a new formulation of the method which combines two main approaches, namely the Newton basis GMRES [17] and the augmented basis for the restarted GMRES [98]. Our approach benefits from the enhanced parallelism in the former and the robustness in the latter. For the sake of clarity, we give here the formulation of the GMRES algorithm as first proposed by Saad and Schultz [114].

We consider in this paper the right preconditioned matrix $B \equiv AM^{-1}$. The proposed algorithms can be derived with less effort for the left preconditioned matrix. Given an initial guess x_0, the GMRES method finds the j approximate solution x_j of the form

$$x_j \in x_0 + \mathcal{K}_j(B, r_0), \tag{6.3}$$

where $r_0 = b - Bx_0$ is the initial residual vector and $\mathcal{K}_j(B, r_0)$ is the k-th Krylov subspace defined as

$$K_j(B, r_0) = span\{r_0, Br_0, \ldots, B^{j-1}r_0\}. \tag{6.4}$$

The goal behind GMRES is to minimize at each step the Euclidian norm of the residual, i.e

$$\|b - Bx_j\| = \min_{u \in x_0 + \mathcal{K}_j(B, r_0)} \|b - Bu\|. \tag{6.5}$$

An orthonormal basis $V_{j+1} = [v_0, \ldots, v_j]^*$ of $\mathcal{K}_{j+1}(B, r_0)$ is generated such that

$$v_0 = r_0/\beta, \quad \beta = \|r_0\|, \quad BV_j = V_{j+1}H_{j+1,j} = V_jH_j + h_{\{j+1,j\}}v_je_j^T, \tag{6.6}$$

It is therefore proved [113] that (6.5) reduces to

$$\|\beta e_1 - H_{j+1,j}y_j\| = \min_{y \in \mathbb{R}^j} \|\beta e_1 - H_{j+1,j}y\| \tag{6.7}$$

and the approximate solution x_j can be written as

$$x_j = x_0 + M^{-1}V_jy_j. \tag{6.8}$$

Our work combines two improvements of this method. In GMRES(m), the method restarts at some step m to save the storage and the computational requirements as the iterations proceed. The deflated and augmented approaches [16, 28, 53, 87, 98] keep some useful information at the time of the restart to enhance robustness. We briefly review these methods in Section 6.2. The second improvement builds the orthonormal basis with a parallel algorithm and reduces the number of exchanged MPI messages on distributed-memory computers. In the original formulation indeed, the basis V is built and orthogonalized by the modified Gram-Schmidt implementation (MGS) of the Arnoldi process (V_j is referred to as *Arnoldi basis*). This process induces a high communication overhead due to the numerous inner products. For instance a GMRES cycle of m iterations requires approximately $\frac{1}{2}(m^2 + 3m)$ global communications for the inner products. On high latency networks, the start-up time due to these collective communication can easily dominate. Moreover, the kernel operations in MGS have a very low granularity which does not fully benefit from

*Throughout this paper, we use a zero-based numbering for all the vectors in the basis

the computer architecture. In the classical Gram-Schmidt implementation (CGS) of the Arnoldi process, the communication time can be reduced by accumulating and broadcasting multiple inner products together. However the low granularity of the kernel operations in the orthogonalization procedure remains because of the sequential form of the Arnoldi process. Moreover, CGS is more sensitive to rounding errors than MGS [81]. Alternative implementations [17, 41, 50, 51, 79, 83, 118, 107] have been proposed. They divide the process into two main phases : first, a nonorthogonal basis of the Krylov subspace is generated and orthogonalized as a group of vectors in the second phase. As first proposed by Bai, Hu and Reichel [17], the *a priori* basis is built with the aid of Newton polynomials. We will refer to this as the *Newton-basis GMRES*. Later on, the orthogonalization is done by replacing the vector-vector operations of the MGS method by the task of computing a QR factorization of a dense method. De Sturler [41] analyzes the parallel implementation of the second phase and suggests a distributed MGS factorization to overlap communication with computation. Sidje and Philippe [119], Erhel [50] and Sidje [118] use a different orthogonalization strategy called RODDEC which combines the Householder factorization with Givens rotations and requires only point-to-point communication. Demmel et al [42] propose a different QR factorization called *Tall Skinny* QR (TSQR) which reorganizes the computation to reduce the memory access and exploit the data locality.

Our proposal in this work is to combine the Newton-basis GMRES with the augmented and deflated GMRES. The new approach is simple and can be used together with any of the previous orthogonalization strategies once the augmented *a priori* basis is built. The motivation of our work is two-fold: previous studies [107] have shown that when the size of the Newton basis grows, the vectors become increasingly dependent. As a result, the method may experience a slow convergence rate. With the new approach, the basis is kept small and augmented with some useful approximate eigenvectors. The second motivation is related to GMRES(m) preconditioned by domain decomposition methods. Indeed, with Schwarz-based preconditioners, when the number of subdomains increases, the preconditioner becomes less and less robust and the method requires more iterations to converge. In this situation, the basis length is usually increased to prevent the stagnation. In the proposed approach, we show that by adding adaptively more approximate eigenvectors, the convergence rate is improved. Recently, Mohiyuddin et al [97] and Hoemmen [79] propose a new formulation in their *communication avoiding* GMRES which do not require the Krylov basis length to be equal to the number of vectors generated *a priori*. Their formulation builds the Krylov basis with several steps of the Arnoldi process where each step builds a set of vectors with the Newton polynomials. Our proposed approach can be used as well with their formulation for the problems that are very sensitive to the restarting procedure in GMRES. We show indeed in Section 6.3 how the augmented basis can be formulated in their proposed approach. The remaining part of this paper is organized as follows : in section 6.2, we review briefly how the deflation of eigenvalues is used in the restarted GMRES. In section 6.3, we derive the new approach combining deflation to the Newton basis GMRES and we discuss on the parallel implementation. Section 6.4 is focused on numerical experiments to show the benefits of the proposed approach.

6.2 Restarted GMRES accelerated by deflation

A practical implementation of GMRES is based on restarting a minimum residual iteration when the correction space reaches a given dimension m. At the time of restart, information from the previous Krylov subspace is discarded and the orthogonality between successive Krylov subspaces is not preserved. The worst case is when the successive generated Krylov

subspaces are very close. As a result, there is no significant reduction in the residual norm and the iterative process stagnates. Deflation techniques are a class of acceleration strategies that collects useful information at the time of restart mainly to avoid this stagnation and improve the convergence rate. The main idea behind these methods is to remove the smallest eigencomponents from the residual vector as they are known to slow down the convergence of GMRES [132]. For a general analysis of acceleration strategies in the minimal residual methods, we refer the reader to Eirman, Ernst, and Schneider [48]. For the general Krylov subspace methods, the recent reviews in [52, 121] are also of great interest.

In deflation techniques, the Krylov subspaces are enriched by some approximation of invariant subspaces associated to a selected group of eigenvalues (generally the smallest ones). Two strategies are often used, namely by preconditioning the linear system [28, 50, 87] or by augmenting the Krylov subspace [98, 99].

Consider U the basis of a nearly invariant subspace spanned by r (approximated) eigenvectors computed from the previous Krylov subspaces, the preconditioner used to deflate the corresponding eigenvalues is given by [53] :

$$\bar{M}^{-1} = I_n + U(|\lambda_n|T^{-1} - I_r)U^T \tag{6.9}$$

where $T = U^T BU$ and λ_n approximates the largest eigenvalue. When U is an exact B-invariant basis, the eigenvalues of the preconditioned matrix BM^{-1} are $\lambda_{r+1}, \ldots, \lambda_n, |\lambda_n|$ with a multiplicity at least r. It is therefore expected that GMRES(m) will converge faster since the smallest eigencomponents that slow down the convergence are deflated. The actual implementation in [53] and the improvements in [28] rely on the approximation of U which is updated at each restart by computing the Ritz values from the Arnoldi relation in Equation (6.6) to yield a more accurate basis. In [16], U is computed by the Implicit-Restarted Arnoldi (IRA) process and the result is used to form a left preconditioner. The adaptive preconditioner by Kharchenko and Yeremin [87] is built such that the Ritz values (which approximate the largest eigenvalues of B) are translated to a cluster around one. With a large number of processors, the communication overhead may dominate when applying the preconditioner in Equation (6.9) to form the basis vectors. As we express the goal to reduce the global communications, we rather rely on the augmented approaches [98, 99].

The augmented approaches, form the new approximation with a projection onto a subspace $\mathcal{C} = \mathcal{K}_m(B, r_0) + \mathcal{W}$, where $\mathcal{W} = span\{u_0, \ldots, u_{r-1}\}$. Minimal changes are required to the existing kernel operations as the vectors are directly added to the existing Krylov basis. Moreover, when the vectors $u_0, \ldots, u_{r-1}$ are the harmonic Ritz vectors, Morgan [99] shows that the new searching subspace $\mathcal{C}$ is itself a Krylov subspace and writes

$$\mathcal{C} = \mathcal{K}_m(B, r_0) + \mathcal{W} = \mathcal{K}_{m+r}(B, q_m(B)r_0) \tag{6.10}$$

where $q_m(B)$ is a polynomial of degree at most m.

6.3 Deflated GMRES in the Newton basis

In this section, We derive the new implementation of the GMRES algorithm where the Krylov subspace is spanned by the Newton polynomials and augmented with eigenvectors. Regarding the previous Newton basis implementations [17, 50, 118, 79], the new approach uses the deflation strategies to recover the information that are lost at the time of restart. Hence for the problems that are sensitive to the restarting procedure, our implementation should converge faster than the previous approaches for the same basis length. Regarding

the GMRES-E by Morgan [98], our approach communicates less and should produce better computational kernels. Compared to the GMRESDR of Morgan [99] however, we have not investigated whether the proposed augmented basis is itself a Krylov basis and we left it as future work.

6.3.1 Augmenting the Newton basis

We now derive the proposed approach. Our motivation is to get a Arnoldi-like relation for the augmented basis when the eigenvectors are added at the end of the Newton basis. Let B be the preconditioned matrix, x_0 an initial guess and r_0 the initial residual vector. A m-dimensional Krylov subspace is spanned by the Newton polynomials applied to r_0 of the form

$$\mathcal{P}_{j+1}(B) := \sigma_{j+1}(B - \lambda_{j+1}I)\mathcal{P}_j(B), \quad j = 0, 1, 2, \ldots, \tag{6.11}$$

where $\mathcal{P}_0(B) := r_0$, σ_j and $\lambda_j \in \mathbb{R}$ (See [17, 107]). We discuss later on the choice of these scalars. Let $\mathcal{W} = span[u_0, u_1, \ldots, u_{r-1}]$ be an approximate basis of an invariant subspace associated to r selected eigenvalues, let $s = m+r$ and $\mathcal{C}_{s+1}$ the augmented subspace defined by

$$\mathcal{C}_{s+1} = span\{r_0, (B - \lambda_1 I)r_0, \ldots, \prod_{j=1}^{m}(B - \lambda_j I)r_0\} + \mathcal{W}. \tag{6.12}$$

Proposition 6.3.1. *There exists a rectangular matrix $\mathtt{W}_s \in \mathbb{R}^{n\times s}$, basis of $\mathcal{C}_s$, a matrix $\mathtt{V}_{s+1} \in \mathbb{R}^{n\times(s+1)}$, whose columns form an orthogonal set of vectors, such that*

$$B\mathtt{W}_s = \mathtt{V}_{s+1}\bar{\mathtt{H}}_s = \mathtt{V}_s\mathtt{H}_s + \mathtt{h}_{s+1,s}v_{s+1}e_s^T, \tag{6.13}$$

where $\bar{\mathtt{H}}_s \in \mathbb{R}^{(s+1)\times s}$ is a upper Hessenberg matrix. Moreover, the vector $x_s \in \mathbb{R}^n$ given by

$$x_s = x_0 + M^{-1}\mathtt{W}_s y_s, \tag{6.14}$$

where $y_s \in \mathbb{R}^n$ solves the least-square problem $J_s(y)$ defined by

$$J_s(y) = \|\beta e_1 - \bar{\mathtt{H}}_s y\|_2, \quad \beta = \|r_0\|_2 \tag{6.15}$$

minimizes the residual norm $\|b - Ax_s\|$ over $x_0 + M^{-1}\mathcal{C}_s$.

Proof. From $k_0 = r_0/\|r_0\|_2$, a set of vectors k_j can be generated such that

$$\sigma_{j+1}k_{j+1} = \begin{cases} (B - \lambda_{j+1}I)k_j & \text{if} \quad 0 \le j \le m-1 \\ \\ Bu_{j-m} & \text{if} \quad m \le j \le s-1. \end{cases} \tag{6.16}$$

where λ_j and σ_j, $(j = 1, 2, \ldots)$ are user-specified scalars. We discuss in sections 6.3.2.1 and 6.3.2.2 on their optimal choice. In matrix form, the relation (6.16) writes

$$\begin{cases} BK_m = K_{m+1}\bar{T}_m \\ BU_r = K_r D_r \end{cases} \tag{6.17}$$

with $K_{m+1} = k_0, k_1, \ldots, k_m$, $K_r = k_{m+1}, \ldots, k_s$ and

$$\bar{T}_m = \begin{bmatrix} \lambda_1 & & & & & \\ \sigma_1 & \lambda_2 & & & & \\ & \sigma_2 & \lambda_3 & & & \\ & & \ddots & \ddots & & \\ & & & & \lambda_{m-1} & \\ & & & & \sigma_{m-1} & \lambda_m \\ & & & & & \sigma_m \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}, \tag{6.18}$$

and $D_r = diag\{\sigma_{m+1}, \ldots, \sigma_s\} \in \mathbb{R}^{r \times r}$.

A QR factorization on K_{s+1} yields

$$K_{s+1} = \mathtt{V}_{s+1}\mathtt{R}_{s+1}. \tag{6.19}$$

Defining $\mathtt{W}_s = \left[\begin{array}{cc} K_m & U_r \end{array} \right]$ and using equations (6.17) and (6.19), we get

$$B\mathtt{W}_s = \mathtt{V}_{s+1}\mathtt{R}_{s+1} \left[\begin{array}{cc} \bar{T}_m & 0 \\ 0 & D_r \end{array} \right]. \tag{6.20}$$

The first part is thus proved (Equation 6.13) by defining

$$\bar{\mathtt{H}}_s = \mathtt{R}_{s+1} \left[\begin{array}{cc} \bar{T}_m & 0 \\ 0 & D_r \end{array} \right] = \left[\begin{array}{c} H_s \\ h_{s+1,s} e_s^T \end{array} \right]. \tag{6.21}$$

The second part of the proposition is similar to the optimality property in the augmented GMRES [34, Algorithm 2.1]. Consider an arbitrary vector $x_s = x_0 + M^{-1}\mathtt{W}_s y$ in the affine space $x_0 + M^{-1}\mathcal{C}_s$, the corresponding residual vector can be expressed as:

$$\begin{aligned} b - Ax_s &= b - A(x_0 + M^{-1}\mathtt{W}_s y) && (6.22) \\ &= r_0 - \mathtt{V}_{s+1}\bar{\mathtt{H}}_s y \\ &= \beta k_0 - \mathtt{V}_{s+1}\bar{\mathtt{H}}_s y \\ &= \mathtt{V}_{s+1}(\beta e_1 - \bar{\mathtt{H}}_s y) && (6.23) \end{aligned}$$

where $\beta = \|r_0\|$ and $e_1 = [1, 0, \ldots, 0]^T$. If we denote by $J_s(y)$ the function

$$J_s(y) = \|b - A[x_0 + M^{-1}\mathtt{W}_s y]\|_2,$$

it comes from Equation 6.23 and the fact that $\mathtt{V}_{s+1}$ is orthogonal that

$$J_s(y) = \|\beta e_1 - \bar{\mathtt{H}}_s y\|_2. \tag{6.24}$$

Thus by taking the vector $y_s \in \mathbb{R}^m$ which minimizes the function $J_s(y)$, the approximate solution $x_s = x_0 + M^{-1}\mathtt{W}_s y_s$ will have the smallest residual in $x_0 + M^{-1}\mathcal{C}_s$. □

We refer to the matrix $\mathtt{W}_s$ as the *augmented Newton basis* of the subspace $\mathcal{C}_s$ and the induced GMRES is the *augmented Newton basis GMRES*.

This proof assumes that the basis vectors $k_j, j = 0, \ldots, s$ are generated through one pass in the kernel computation of Equation 6.16. There are some situations where m is too large to guarantee a good robustness (ill-conditioned basis) or good performance (best value for data locality in multicore nodes). In a recent work, Hoemmen [79] uses the μ-step Arnoldi of Kim and Chronopoulos [88][†] in his *Arnoldi(μ, t)* to build the basis vectors through multiple passes of the kernel computation in Equation 6.16. Hence the process of computing s basis vectors is divided into t steps where each step generates μ basis vectors with the Newton polynomials. The restart length is thus $s = \mu \cdot t$. We show in the following that the proposition 6.3.1 holds in the case of a μ-step basis. We explain the basis idea with $t = 2$.

Let k_0 be the starting vector and $s = \mu \cdot t + r = m + r$. As from the first part of Equation 6.16 we generate the sequence of vectors

$$\sigma_{j+1} k_{j+1} = (B - \lambda_{j+1} I) k_j, 0 \leq j \leq \mu - 1. \tag{6.25}$$

[†]The original method is referred to as s-step Arnoldi instead of μ-step Arnoldi but we choose μ here to differentiate with the size s of our augmented basis.

It comes that

$$BK_{\mu}^{(0)} = K_{\mu+1}^{(0)} \bar{T}_{\mu}^{(0)} \tag{6.26}$$

where $K_{\mu+1}^{(0)} = [k_0, k_1, \ldots, k_\mu] \in \mathbb{R}^{n \times \mu+1}$, $\bar{T}_{\mu}^{(0)} \in \mathbb{R}^{\mu+1 \times \mu}$ is a bidiagonal matrix. A QR factorization of $K_{\mu+1}^{(0)}$ gives

$$K_{\mu+1}^{(0)} = V_{\mu+1}^{(0)} R_{\mu+1}^{(0)} \tag{6.27}$$

and thus

$$BK_{\mu}^{(0)} = V_{\mu+1}^{(0)} \bar{\mathtt{H}}_{\mu}^{(0)} = V_{\mu}^{(0)} \mathtt{H}_{\mu}^{(0)} + h_{\mu+1,\mu} V_{\mu+1}^{(0)} e_\mu e_\mu^T, \tag{6.28}$$

where $\bar{\mathtt{H}}_{\mu}^{(0)} = R_{\mu+1}^{(0)} \bar{T}_{\mu}^{(0)}$and e_μ is the μ^{th} unit vector. This first step is just the derivation of the Arnoldi-like relation for the (non-augmented) Newton basis. Note that we don't have a mathematically equivalent Arnoldi relation as in Equation 6.6. $\bar{\mathtt{H}}_{\mu+1}^{(0)}$ is not equal in exact arithmetics to the Hessenberg matrix $\bar{H}$ of that equation as we avoid dealing with the term $(R_{\mu+1}^{(0)})^{-1}$. However, the columns of $V_{\mu+1}^{(0)}$ form an orthogonal basis of a μ-dimensional Krylov basis. From its last column as a starting vector, we can thus build the second μ-step basis. At this step, we add the eigenvectors in the subspace by augmenting the μ-step basis.

Let $k_\mu = V_{\mu+1}^{(0)} e_\mu$, a μ-step augmented basis is generated as follows:

$$\sigma_{j+1} k_{j+1} = \begin{cases} (B - \lambda_{\mu-j+1} I) k_j & \text{if} \quad \mu \leq j \leq m-1 \\ \\ B u_{j-m+1} & \text{if} \quad m \leq j \leq s-1. \end{cases} \tag{6.29}$$

In matrix form, we get

$$B \begin{bmatrix} K_{\mu}^{(1)} & U_r \end{bmatrix} = K_{\mu+r+1}^{(1)} \begin{bmatrix} \bar{T}_{\mu}^{(1)} & 0 \\ 0 & D_r \end{bmatrix} \tag{6.30}$$

The matrices $\bar{T}_{\mu}^{(1)}$ and D_r are similar to the matrices in Equation 6.13. At this point, to avoid loss of orthogonality, the vectors $K_{\mu+r+1}^{(1)} = [k_\mu, k_{\mu+1}, \ldots, k_{\mu+m+r}]$ should be orthogonalized against the previous vectors $V_{\mu}^{(0)}$. This can be done with a block Gram-Schmidt method which is equivalent to write[‡]

$$\hat{K}_{\mu+r+1}^{(1)} = \left(I - V_{\mu}^{(0)} (V_{\mu}^{(0)})^T \right) K_{\mu+r+1}^{(1)} \tag{6.31}$$

The vectors $\hat{K}_{\mu+r}^{(1)}$ are now orthogonal to the basis vectors but it remains to orthogononalize them between each other. This can be done with a dense QR factorization to produce

$$\hat{K}_{\mu+r+1}^{(1)} = V_{\mu+r+1}^{(1)} R_{\mu+r+1}^{(1)}. \tag{6.32}$$

From Equations 6.28 and 6.30, we get

$$B \begin{bmatrix} K_{\mu}^{(0)} & K_{\mu}^{(1)} & U_r \end{bmatrix} = \begin{bmatrix} V_{\mu}^{(0)} & K_{\mu+r+1}^{(1)} \end{bmatrix} \begin{bmatrix} \mathtt{H}_{\mu}^{(0)} & 0 \\ h_{\mu+1,\mu} e_1 e_\mu^T & \bar{C}_{\mu+r} \end{bmatrix} \tag{6.33}$$

where

$$\bar{C}_{\mu+r} = \begin{bmatrix} \bar{T}_{\mu}^{(1)} & 0 \\ 0 & D_r \end{bmatrix}$$

[‡]Note that the first vector k_μ is already orthogonal to $V_{\mu}^{(0)}$ but we choose to orthogonalize it again.

Knowing that a QR factorization update have been perfomed on $K^{(1)}_{\mu+r+1}$, we get from Equations 6.31 and 6.32 that,

$$\left[\begin{array}{cc} V^{(0)}_{\mu} & K^{(1)}_{\mu+r+1} \end{array}\right] = \left[\begin{array}{cc} V^{(0)}_{\mu} & V^{(1)}_{\mu+r+1} \end{array}\right] \left[\begin{array}{cc} I & (V^{(0)}_{\mu})^T K^{(1)}_{\mu+r+1} \\ 0 & R^{(1)}_{\mu+r+1} \end{array}\right]. \tag{6.34}$$

Substituting 6.34 in 6.33, we get

$$BW_s = V_{s+1}\bar{\mathtt{H}}_s \tag{6.35}$$

where $W_s = \left[\begin{array}{ccc} K^{(0)}_{\mu} & K^{(1)}_{\mu} & U_r \end{array}\right]$, $\mathtt{V}_{s+1} = \left[\begin{array}{cc} V^{(0)}_{\mu} & V^{(1)}_{\mu+r} \end{array}\right]$ and

$$\bar{\mathtt{H}}_s = \left[\begin{array}{cc} I_{\mu,\mu} & (V^{(0)}_{\mu})^T K^{(1)}_{\mu+r+1} \\ 0 & R^{(1)}_{\mu+r+1} \end{array}\right] \left[\begin{array}{cc} \mathtt{H}^{(0)}_{\mu} & 0 \\ h_{\mu+1,\mu} e_1 e_{\mu}^T & \bar{C}_{\mu+r} \end{array}\right].$$

From the fact that $K^{(1)}_{\mu+r+1}e_1$ is orthogonal to $V^{(0)}_{\mu}$ and that $R^{(1)}_{\mu+r+1}e_1 = e_1$, we get

$$\bar{\mathtt{H}}_s = \left[\begin{array}{cc} \mathtt{H}^{(0)}_{\mu} & (V^{(0)}_{\mu})^T K^{(1)}_{\mu+r+1}\bar{C}_{\mu+r} \\ h_{\mu+1,\mu} e_1 e_{\mu}^T & R^{(1)}_{\mu+r+1}\bar{C}_{\mu+r} \end{array}\right] \tag{6.36}$$

which is an Hessenberg matrix. The first part of the proposition 6.3.1 is thus proved and the second part is similar to the previous proof.

The GMRES with the μ-step Newton basis is useful to control the conditioning of the basis generated with the Newton polynomials by choosing a suitable value of μ. On multicore nodes, a well chosen value of μ will also improve the data locality during the computation of the kernel computations (Generation of the basis and orthogonalization) [42, 79]. The drawback with this formulation is that when the new set of basis vectors is orthogonalized against all the previous vectors already computed, it is important to perform a good QR factorization update. Sometimes when a block Gram-Schmidt process is used, a reorthogonalization strategy should be performed to avoid loss of orthogonality, see for instance [81]. This process induces more computational cost as the number t of steps increases. As for the scalar formulation, the augmented basis will thus help to reduce this cost by reducing the number of steps t. We do not further investigate in this direction and we focus in this paper to the basic implementation of a $(\mu+r)$-step augmented Newton basis.

6.3.2 AGMRES : Augmented Newton-basis GMRES

This section discuss the parallel implementation of the GMRES method where the Newton basis is augmented with a few selection of approximate eigenvectors. The main steps are outlined in Algorithm 6.3.2.

If we compare AGMRES with the previous related implementations of the GMRES method, we can make the following observations :

- Compared to the standard GMRES method, AGMRES produces efficient kernel computations during the generation of the orthogonal Krylov basis in steps 5 and 6. However, in addition to the basis W_s, it keeps the orthogonal system V_s for computing the eigenvectors.
- The GMRES-E of Morgan [98] keeps a second basis as well. However its implementation is based on the Arnoldi process. It will thus communicate more for the same convergence behaviour. Our implementation includes an adaptive strategy that will allow to increase the number of extracted eigvenvectors if necessary.

Algorithm 5 AGMRES(m, r, l) : Augmented Newton-basis GMRES

Require: $x_0, m, itmax, \epsilon, r, l, r_{max}, smv, bgv$;

1: **Perform** one cycle of GMRES(m) [114, Algorithm 4] to find a new approximation x_m, the residual r_m and the matrices H_m and V_m satisfying Equation 6.6. **if**$(\|r_m\| < \epsilon)$ **return**
2: Set $x_0 \leftarrow x_m$ and $r_0 \leftarrow r_m$; $\beta = \|r_0\|$;
3: **Compute** m Ritz values $\{\lambda_j\}_{j=1}^{j=m}$ of AM^{-1} from H_m and order them with the Leja ordering [17]; **If** $(r > 0)$ extract r Ritz vectors U for the augmented basis.
4: **while** $(\|r_0\| > \epsilon)$ **do**
5: **Compute** K_{s+1} from Equation 6.16 together with $\bar{T}_m$ and D_r and derive $\mathtt{W}_s$
6: **Compute** the QR factorization $K_{s+1} = \mathtt{V}_{s+1} R_{s+1}$
7: **Compute** the $(s+1) \times s$ Hessenberg matrix $\bar{\mathtt{H}}_s$ from Equation 6.21
8: **Solve** $y_s = min \|\beta e_1 - \bar{\mathtt{H}}_s y\|_2$
9: **Compute** $x_s = x_0 + M^{-1}\mathtt{W}_s y_s$ $r_s = b - Ax_s$, $it \leftarrow it + s$
10: **if**$(\|r_s\| < \epsilon$ **or** $it > itmax)$ **return**
11: Set $x_0 \leftarrow x_s$ and $r_0 \leftarrow r_s$;
12: **if** $r > 0$ **then**
13: $Iter = s * log\left(\frac{\epsilon}{\|r_s\|}\right) / log\left(\frac{\|r_s\|}{\|r_0\|}\right)$
14: **if** $(Iter > smv * (itmax - it))$ **then**
15: **if** $((Iter > bgv * (itmax - it))$ **and** $(r < r_{max})$ **and** $(l > 0))$ **then**
16: $r \leftarrow r + l$ /*Increase the number of eigenvalues to deflate*/
17: **end if**
18: **Update** the r eigenvectors $u_1, u_2, \ldots, u_r$ from the harmonic Ritz values of $B \equiv M^{-1}A$
19: **end if**
20: **end if**
21: **end while**

- Compared to CA-GMRES of Hoemmen [79], our implementation is limited to one μ-step Newton basis. However, we show in the previous section how an augmented basis can be defined for more than one μ-step basis. For the same restart length, CA-GMRES(μ, t) and GMRES$(\mu \cdot t)$ produce the same convergence behaviour. AGMRES$(\mu \cdot t, r)$ is more likely to produce a faster convergence than these two approaches when the convergence rate is affected by the restarting procedure.

So far, the algorithm starts with an initial approximation of the solution vector x_0 (in practice, we use a zero vector), the size m of the Krylov basis, the maximum number of iterations $itmax$ allowed and the desired accuracy ϵ. The remaining input values are used for the augmented basis: the number of eigenvectors r that are added at each step; the parameters l, r_{max}, smv and bgv for the adaptive strategy, see section 6.3.2.6. The main steps of the algorithm are the computation of the shifts (steps 1 and 3), the generation of the augmented Newton basis at step 5 and its orthogonolization in section 6. The approximated solution is updated at step 9. At step 18, we update the eigenvectors to be added in the Newton basis. The adaptive strategy is implemented in steps 12-20 All these steps are explained in the next sections.

6.3.2.1 Computation of the shifts

The generation of the Krylov subspace with the Newton polynomials uses the scalars λ_j, $j = 1, \ldots, m$ to produce a stable basis. Bai et al [17] show that an optimal choice would be to use the eigenvalues of B numbered according to the following modified *Leja order* (see [109]):

$$\begin{cases} |\lambda_1| = \max_{j=1,\ldots,m} |\lambda_j| \\ \prod_{k=1}^{j} |\lambda_{j+1} - \lambda_k| = \max_{l=1,\ldots,m} \prod_{k=1}^{j} |\lambda_l - \lambda_k|, \quad j = 1, \ldots, m-1 \end{cases} \tag{6.37}$$

In practice, the spectrum of B is not available and very expensive to compute. In this situation, the Ritz values of B which are the eigenvalues of the Hessenberg matrix H_m in Equation 6.6 are used. This implies that m steps of the Arnoldi process should be performed to get these values. At step 1, we perform one cycle of the Arnoldi-GMRES method. From this step, we get an approximation of the solution x_m and the associated residual r_m. This vector is used as the initial search direction for the Newton basis GMRES from step 4. At step 3, each process computes the eigenvalues of its own copy of the Hessenberg matrix H_m and order them with the Leja ordering. This step uses so far the parallelism inside the matrix-vector product and the preconditioning operation. But it requires global communication as pointed out in section 6.1. Note that when m gets large, it may be expensive to perform this step of the Arnoldi-GMRES method. The cost here is compared to one step of the Newton basis GMRES mainly in terms of granularity and MPI messages volume. In practice, we use a small value of m to show the benefits of augmenting the basis. Nevertheless, if a large basis should be used, a solution could be to use a μ-step basis as explained in the previous section. Another solution, as advised by Philippe and Reichel [107] is to perform a Arnoldi-GMRES with a smaller basis to get a subset of these values. From this subset, a convex hull is defined and continuously updated with new values collected during the Newton-basis GMRES iterations.

6.3.2.2 Computation of the Newton basis with scaling factors

The first $m+1$ vectors of K_{m+1} can be generated using Algorithm 1.1 in [118]; then it is easy to generate the last r vectors from U_r. Note that when a particular λ_{j+1} is complex, and assuming that $Im(\lambda_{j+1}) > 0$ (This is always possible from the complex conjugate pairs and the modified Leja ordering), the complex arithmetic is avoided by writing the first part of Equation 6.16 as

$$\begin{aligned} k_{j+1} &= 1/\sigma_{j+1}(B - Re(\lambda_{j+1}I)k_j \\ \sigma_{j+2}k_{j+2} &= (B - \lambda_{j+1}I)(B - \bar{\lambda}_{j+1}I)k_j = (B - Re(\lambda_{j+1}I)k_{j+1} + 1/\sigma_{j+1}Im(\lambda_{j+1})^2 k_j. \end{aligned}$$

In this case, the matrix $\bar{T} \in \mathbb{R}^{s+1 \times s}$ of Equations 6.13 and 6.30 is tridiagonal. So far, the scalars $\sigma_j, j = 1, \ldots, m$ in Equation 6.13 are used to control the growth of the vectors $\{k_j\}_{j=1}^{j=m}$. The common choice is to take $\sigma_j = \|k_j\|$. The parallelism inside this step is through the preconditioning and the parallel matrix-vector operations $(AM^{-1} - \lambda I)k \equiv A(M^{-1}k) - \lambda_j k$. When $\sigma_j = \|k_j\|$, then there are $(m + r)$ global communications, which are far less than the $\frac{1}{2}(m^2+3m)$ global communications in the Arnoldi process. With some particular cases, this norm can be computed distributedly. When using for instance the explicit formulation of multiplicative Schwarz, the basis vectors are computed in a pipeline accross all the subdomains. Each process is thus able to compute its own contribution to the norm and the basis vectors are normalized *a posteriori* [11, 101]. When the size of the basis is small enough, the rows and columns of the matrix can be equilibrated and no

scaling, thus no global communication, should be needed during the computation of the basis vectors [79].

6.3.2.3 Orthogonalization of the basis

After the basis vectors are computed, they should be orthogonalized between each other at step 6 of Algorithm 6.3.2 to produce the orthogonal system V_{s+1}. At the end of the step 5, the vectors K_{s+1} are distributed on all processors as a contiguous blocks of rows which is equivalent to the classical $1D$ rowwise partitioning for the matrix-vector products. Any algorithm for the parallel dense QR factorization can now be used to orthogonalize the system K_{s+1}. In our implementation, we use the RODDEC algorithm described in [118, section 4.2]. This method performs first a Householder orthogonalization on each block of rows. This is done in a perfect parallel phase by all the processes having the rows. After that, the Givens rotations are used to annihilate the blocks below the first one. During this second step, the processors are placed on a ring topology and each process sends the required data on this ring. This step requires $\mathcal{O}(m^2)$ point-to-point messages and the average message length is $(m+1)/2$ double precision number. The TSQR algorithm of Demmel *et al* [42], which is a divide-and-conquer approach, can be used as well at this step. It requires $\mathcal{O}(log(P))$ MPI messages where P is the total number of MPI processes sharing the system K_{s+1}.

6.3.2.4 Updating the current approximation

At the end of the QR factorization, the triangular matrix $\mathtt{R}_s$ of Equation 6.19 is usually available on one process. In the RODDEC algorithm, it is available in the last process. It can be broadcasted to all other processes such that the steps 7 and 8 are done by all the processes. When the number of MPI processes gets large, it is more efficient to perform these steps on the last process and to broadcast only the result of the least-squares problem at step 8. In our implementation, we choose to send a copy of the matrix since it is required by all processes to update the eigenvectors, see section 6.3.2.5. So far, the Hessenberg matrix $\bar{\mathtt{H}}_s$ is assembled from $\mathtt{R}_s$ and $\bar{T}_s$ using a modification of Algorithm 1.2 in [118]. The modification allows to take into account the scaling factors of the augmented vectors in the basis. A QR factorization is perfomed on the output Hessenberg matrix to solve the least-squares problem in the minimization step. The LAPACK routine *dgeqrf* is used for this purpose. The output solution is used to compute the new approximate solution at step 9. Note that since we are using a right preconditioning, we can obtain an estimate of the true residual norm without explicitly computing the residual vector r_s. Nevertheless, at the time of restart, we need r_s for the new search direction.

6.3.2.5 Updating the eigenvectors

When the iterative process starts at line 13 of Algorithm 6.3.2, the eigencomponents (u_j, λ_j) of $B \equiv AM^{-1}$ are approximated from the first GMRES(m) cycle with a standard projection technique as follows :

$$V_m^T(B - \lambda_j I)V_m g_j = 0 \tag{6.38}$$

leading to the eigenvalue problem

$$H_m g_j = \lambda_j g_j. \tag{6.39}$$

The Ritz values $\lambda_j, j = 1, \ldots, m$ are used as shifts for the Newton basis and the vectors $u_j = V_m g_j, j = 1, \ldots, r$ corresponding to the r smallest eigenvalues are used to augment the

Newton basis. Then to update the vectors U_r at step 18, we use a Rayleigh-Ritz procedure. Indeed, as advised by the previous studies [34, 98], this procedure does better at finding eigenvalues nearest zero.

Using the augmented subspace $\mathcal{C}_s$, each extracted eigenvector u is thus expressed as $u = \mathtt{W}_s g_i$. Using the bases $B\mathtt{W}_s$ and $\mathtt{W}_s$, the Galerkin condition writes :

$$(B\mathtt{W}_s)^T(B - \lambda_j I)\mathtt{W}_s g_j = 0. \tag{6.40}$$

It comes with the relation 6.13 that,

$$\underbrace{\bar{\mathtt{H}}_s^T \bar{\mathtt{H}}_s}_{\mathtt{G}_s} g_j = \lambda_j \underbrace{\bar{\mathtt{H}}_s^T \mathtt{V}_{s+1}^T \mathtt{W}_s}_{\mathtt{F}_s} g_j. \tag{6.41}$$

We thus obtain a dense generalized eigenvalue problem of size $s \times s$ where $(\lambda_j, \mathtt{W}_s g_j)$ gives a Ritz pair of B. Multiplying $\mathtt{F}_s$ and $\mathtt{G}_s$ by $\mathtt{H}_s^{-T}$, we get

$$\begin{aligned} H_s^{-T}\mathtt{G}_s &= \mathtt{H}_s^{-T} \begin{bmatrix} \mathtt{H}_s^T & \alpha e_s \end{bmatrix} \begin{bmatrix} \mathtt{H}_s \\ \alpha e_m^T \end{bmatrix} \\ &= \mathtt{H}_s + \mathtt{h}_{s+1,s}^2 \mathtt{H}_s^{-T} e_s e_s^T \end{aligned} \tag{6.42}$$

$$H_s^{-T}\mathtt{F}_s = \begin{bmatrix} I_s & \mathtt{h}_{s+1,s}\mathtt{H}_s^{-T} e_s \end{bmatrix} \mathtt{V}_{s+1}^T \mathtt{W}_s. \tag{6.43}$$

The drawback here is the computational work required to form $\mathtt{F}_s$ as it induces s scalar products of size n. Nevertheless, the numerical experiments show in most test cases that when the convergence is accelerated by deflation, the time to update the eigenvectors is negligible compared to the total time saved without the deflation. Moreover, the adaptive strategy proposed next sets off deflation only if convergence is too slow.

6.3.2.6 Adaptive strategy

When the desired accuracy is not achieved, the method restarts and r new approximate eigenvectors (corresponding to the eigenvalues to deflate) are extracted from the s-dimensional subspace $\mathcal{C}s$. This process may become expensive and not beneficial if the convergence rate is not improved enough. We thus propose an adaptive strategy which detects if the deflation process will be beneficial to speedup the convergence or to avoid stagnation. This approach is based upon the work by Sosonkina *et al.* [124] which has been used successfully in another formulation of deflated GMRES[104]. At lines 13, based on the convergence rate already achieved, we estimate the remaining number of steps ($Iter$) needed to reach the desired accuracy ϵ. We use a small multiple (smv) of the remaining number of steps to detect some insufficient reduction in the residual norm. If it is greater than a small multiple (smv) of the number of steps allowed ($itmax$), then we switch to the deflation. We use a large multiple (bgv) of $itmax$ to detect a near-stagnation in the iterative process. In this case, the number of eigenvectors to augment is increased by a fixed (small) value. Clearly with the parameters $r, l, Iter, smv, bgv$, the adaptive strategy can be sketched as follows :

- If $Iter \leq smv * itmax$, the convergence rate is good enough and no more update should be done on the eigenvectors already computed.
- If $smv * itmax < Iter \leq bgv * itmax$, there is an insufficient reduction in the residual norm and the r eigenvectors are updated for the next cycles of AGMRES.

- If $its > bgv * itmax$, a stagnation may have occurred and we increase the number of eigenvalues to extract/update by a fixed number l. Typically, $l = 1$ in all our test cases.

Note that there are more sophisticated methods to ensure that for some given values of m, GMRES(m) (and thus AGMRES(m)) will not stagnate; see for instance [120, 122]. Our current stagnation test is computed *a posteriori* and should be mostly used to detect a very slow reduction in the residual norm. Although the proposed parameters are problem-dependent, they can be useful to avoid the stagnation if there are some previous knowledge in the convergence behaviour for the problems under study. Some numerical results are given in this sense in the next section.

6.4 Numerical experiments

This section presents some numerical results to show the parallel efficiency and the numerical robustness of the proposed approach. We first present the template for all the numerical tests in section 6.4.1 and the test cases in section 6.4.2.

6.4.1 Test routines and implementation notes

Implementations are done using the PETSc routines and data structures [18, 19]. The algorithm 6.3.2 has been implemented by a *KSP* module called AGMRES using a locally modified version of PETSc revision 3.1.p8. It uses the routines for matrix-vector product, the application of the preconditioner and the other parallel linear algebra functions. It can be used transparently with any preconditioner implemented in the package including the domain decomposition preconditioners. We use so far the Restricted Additive Schwarz (RAS) method [30] applied as a right preconditioner in all our tests. The main steps are outlined in Algorithm 6. Note from the step 7 of our test routine that we compare AGM-

Algorithm 6 Test routine for the parallel computation of the system 6.2 using restricted additive Schwarz method and GMRES-based accelerator.

1: Read the matrix from a binary file and store it in a distributed CSR format. Read the right-hand side vector and store it accordingly.
2: Perform a parallel iterative row and column scaling on the matrix and the right-hand side vector [6].
3: Partition the weighted graph of the matrix in parallel with PARMETIS.
4: Redistribute the matrix and right-hand-side according to the PARMETIS partitioning.
5: Define the overlap between the submatrices for the additive Schwarz preconditioner.
6: Setup the submatrices (ILU or LU factorization using MUMPS [4]).
7: Solve iteratively the system using either the KSP AGMRES (Algorithm 6.3.2) or the PETSc built-in KSP GMRES [114, Algorithm 4].
8: Write the solution vector to a binary file.

RES with the classical implementation of GMRES. As stated earlier, either the classical Gram-Schmidt or the modified Gram-Schmidt can be used for the Arnoldi process. The main advantage of CGS over MGS is the number of MPI messages, the amount of MPI reductions and the granularity in the computational kernel. However, a practical implementation of CGS includes a possible refinement strategy to be as stable as MGS. During our numerical experiments however, this refinement has not be used in CGS and we did not

notice any difference between GMRES-MGS and GMRES-CGS. We therefore give the results of GMRES with CGS. Unless stated, the stopping criterion of GMRES and AGMRES is $\frac{\|b-Ax\|}{\|b\|} < 10^{-10}$ and the maximum number of iterations is 1,000. In AGMRES, the residual norm is computed only at each outer iteration. In GMRES, it is available during each inner iteration. Note that since we are using a right preconditioner, this residual norm is obtained cheaply from the Givens rotations that are used to transform the Hessenberg matrix in Equation 6.7 into a triangular matrix.

In the following, since the right preconditioning is used, the number of iterations is understood as the total number of matrix-vectors products and preconditioning steps. Hence in GMRES(m), it is equivalent to the counts of $A(M^{-1}k)$. In AGMRES, it is equal to the size of the augmented basis times the restart cycles. So far, AGMRES(m) refers to the algorithm 6.3.2 without the deflation (i.e $r = 0, l = 0$); In AGMRES(m, r), r vectors corresponding to the smallest harmonic Ritz values are added to the basis and updated at each restart; In AGMRES(m, r, l), r is adaptively increased by l until r_{max} at each restart.

6.4.2 Test problems

The matrices of tests arise from industrial applications in fluid dynamics and from convection-diffusion problems. The main characteristics are listed in Table 6.1

Table 6.1: Characteristics of test matrices, N:Number of rows/columns, NNZ:Nonzero entries

Matrix	N	NNZ	geometry
IM07R	261,465	26,872,530	3D
VV11R	277,095	30,000,952	3D
RM07R	272,635	37,355,908	3D
3DCONSKY_121	1,7771,561	50,178,241	3D
3DCONSKY_161	4,173,281	118,645,121	3D

The problems *IM07R, VV11R* and *RM07R* arise from design optimization in computational fluid dynamics simulations. They are provided by the FLUOREM company, a CFD software editor[§]. Table 6.1 lists the coefficient matrices with their main characteristics. The physical equations are the Reynolds-Averaged Navier-Stokes for compressible flows discretized using the finite volume methods as presented by [106]. The resulting matrix is formed of $b \times b$ blocks where b is the number of fluid conservative variables (density, velocity, energy and turbulent variables). The matrix RM07R is available online in the University of Florida sparse matrix collection (see [39]) in the FLUOREM directory. The matrix is structurally symmetric in blocks. Regarding the values, the matrix is nonsymmetric and indefinite. In [103, 106], preliminary studies show that hybrid solvers based on GMRES and Schwarz-based preconditioners offer robust approaches to solve efficiently these systems. As pointed in [106], we avoid the ILU factorization in the subdomain matrices because of its unpredictable behavior. We therefore rely on a direct solver (MUMPS) within each subdomain.

The test cases *3DCONSKY_121* and *3DCONSKY_161* correspond to the *convective SkyScraper* problem in [1, 89]. The physical equation is given by the boundary value

[§]www.fluorem.com/en/softwares/optimization/turb-opty-cfd

problem

$$\eta(x)u + div(a(x)u) - div(\kappa(x)\nabla u) = f \text{ in } \mu \tag{6.44}$$

$$u = 0 \text{ on } \partial\mu_D \tag{6.45}$$

$$\frac{\partial u}{\partial n} = 0 \text{ on } \partial\mu_N \tag{6.46}$$

where $\mu = [0,1]^3$, $\partial\mu_N = \partial\mu \backslash \partial\mu_D$. The tensor κ is isotropic and discontinuous. The domain contains many zones of high permeability which are isolated from each other. Let $[x]$ denote the integer value of x then κ is given in 3D by

$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1), & if\ [10 * x_i] = 0\ mod(2), i = 1,2,3, \\ 1, & otherwise \end{cases} \tag{6.47}$$

The velocity field $a = (1000, 1000, 1000)^T$ and $f = x_1^2 + x_2^2 + x_3^2$. The discretization is done using P2-type finite element methods in the Freefm++[¶] package. We consider a uniform grid with $n \times n \times n$ nodes and we choose $n = 121$ and 161. During our numerical experiments, we rely on the $ILU(1)$ factorization to approximate the solutions on the subdomains induced by the additive Schwarz method.

6.4.3 Platform of tests

Experiments are done on a distributed memory supercomputer *Vargas*[‖] which has 3,584 Power6 CPUs. Each Power6 CPU is a dual-core 2-way SMT with a peak frequency at 4.7 GHz. The computer is made of 112 nodes connected through an Infiniband network. Each node has 32 Power6 CPUs that access 128GB of local memory in a non-uniform way (hardware NUMA nodes). The memory accessed by a single MPI process is limited to 3.2GB for the data and 0.5GB for the stack.

6.4.4 Analysis of convergence

In this section, we first compare GMRES and AGMRES without deflation. The goal is to confirm that the two methods have the same convergence behavior for a reasonable restart length. After that, we show the benefits of using the deflation when the restart length in AGMRES is small and when the number of subdomains increase. We finish this section by giving the benefits of using an adaptive strategy.

We consider the large test case *RM07R* from the FLUOREM collection. In Figure 6.4.1, we give the convergence of GMRES(m) and AGMRES(m) with three restart lengths, $m = 32, 48$ and 64. The number of subdomains is 32 and the LU factorization is used within the subdomains. The first remark from Figure 6.4.1 is that there is no real difference between the residual norm obtained from the two strategies. Secondly, the convergence curve of GMRES(m) indicates a periodic stagnation in the iterative process. These ticks occur at the time of restart and are more visible when m is small, hence the larger number of iterations. These ticks suggest that some information is lost at the time of restart and that the augmented basis could be beneficial to improve the convergence rate on these cases. The other test cases give similar behaviours.

Now we show the impact of deflation by augmenting the basis. In Figure 6.4.2, we give the convergence history of GMRES(m) and AGMRES(m, r) with $m = 32, 48$ and $r = 2$,

[¶] http://www.freefem.org/ff++/index.htm
[‖] http://www.idris.fr/su/Scalaire/vargas/hw-vargas.html

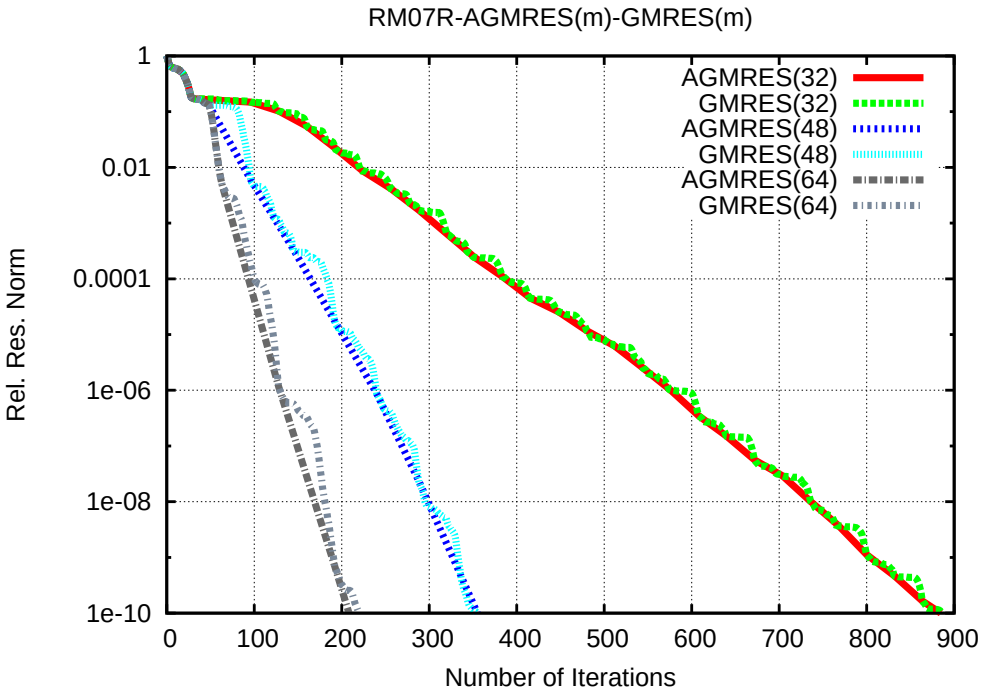


Figure 6.4.1: RM07R : Influence of the restart length in AGMRES and GMRES, 32 subdomains

that is we compute two approximate eigenvectors at each restart and we use a basis of size $s = m + 2$. The number of subdomains is still 32. The adaptive strategy is not used at this point. It can be clearly noticed that adding only two eigenvectors in the basis is sufficient to speedup the convergence in AGMRES. For instance, GMRES(32) requires 886 iterations while AGMRES(32,2) needs almost 272 iterations. When we increase the restart length to 48, GMRES benefits greatly from that and requires almost 355 iterations to reach the desired accuracy while AGMRES(48,2) needs 250 iterations. The general notice here is that AGMRES(32,2) and AGMRES(48,2) give close convergence rate while GMRES is more sensitive to the restart length. This is more visible when the number of subdomains vary.

The robustness of Schwarz preconditioners decreases as the number of subdomains increases. GMRES will thus require more and more iterations, particularly if the restart length is fixed. We show this behaviour in Figure 6.4.3, where the restart length is fixed and the number of subdomains is increased. Clearly, as expected, the number of iterations in GMRES increases as we add more subdomains. For instance, GMRES(32) requires 886 iterations with 32 subdomains. With 64 subdomains, this number reaches 1000 iterations without reaching the prescribed tolerance of 10^{-10}. In AGMRES(m, r), there is no such difference. As we increase the number of subdomains, we observe that the convergence rates remain quite close. Indeed, AGMRES(32,2) requires respectively 272 and 311 iterations for 32 and 64 subdomains. The fact that the number of iterations increases only slightly when increasing D has a great impact to the scalability of AGMRES. We give the timing results in the next section.

In GMRES, a better convergence rate can be obtained if the restart length is increased as a function of the number of subdomains. We show in Table 6.2 that in such case, it is still beneficial to have an augmented basis in AGMRES. These results can be divided into

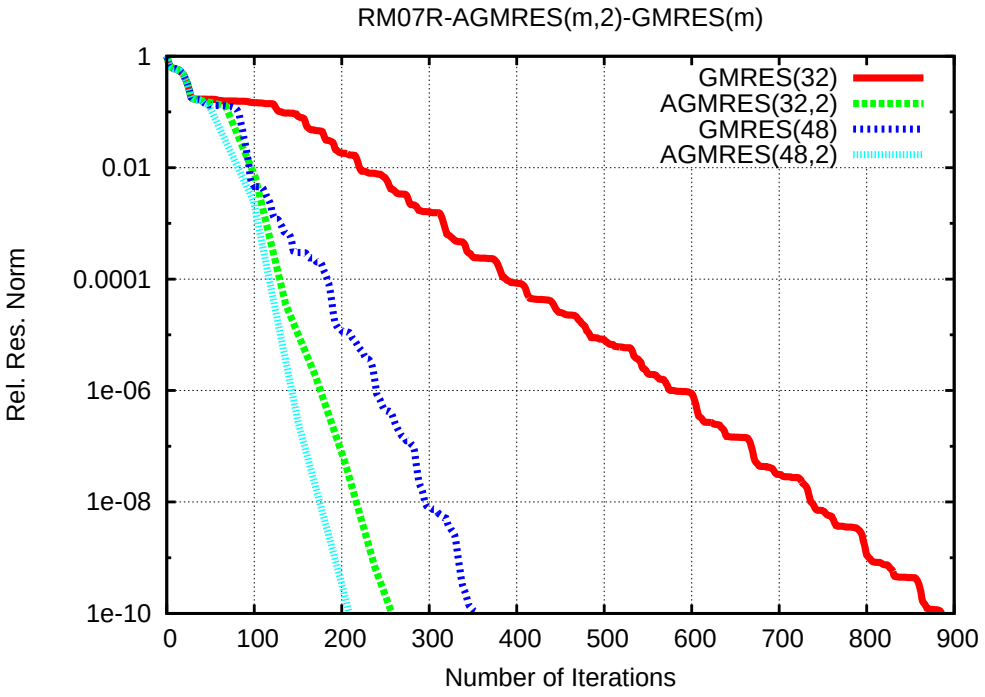


Figure 6.4.2: RM07R: Influence of the augmented basis in AGMRES over GMRES, 32 subdomains

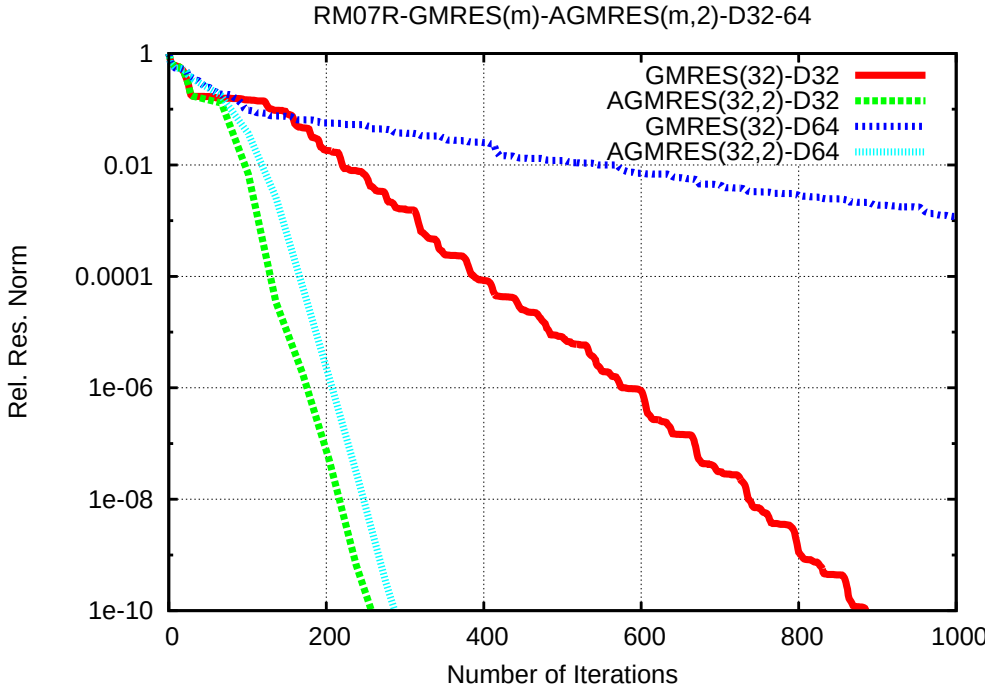


Figure 6.4.3: RM07R: Influence of the number of subdomains in the convergence of GMRES. The restart length is fixed and the benefits of the augmented basis in AGMRES is given.

KSP	GMRES(m)			AGMRES($m,2$)			AGMRES($m,4$)		
D \ m	32	48	64	32	48	64	32	48	64
8	93	70	57	100	98	57	105	100	57
16	254	169	123	169	148	130	177	153	132
32	886	355	220	272	250	196	212	205	200
64	-	702	445	311	303	265	287	258	270

Table 6.2: RM07R: Number of iterations in GMRES(m), AGMRES($m,2$) and AGMRES($m,4$) as a function of the number of subdomains in the restricted additive Schwarz

three parts:

1. With 8 subdomains, GMRES needs less iterations than AGMRES for all values of the restart length. Note that this difference is mainly due to the fact that the stopping test is computed only at each outer cycle in AGMRES. The accuracy achieved in AGMRES for these cases is always better. Typically, AGMRES gives an accuracy of 10^{-14} in the computed residual while it is 10^{-11} in GMRES.

2. For the same reasons, AGMRES needs more iterations than GMRES for 16 subdomains and large value of m. However for small values of m, AGMRES is cleary better than GMRES.

3. From 32 subdomains, AGMRES needs less iterations than GMRES for all restart lengths. The dash in GMRES(32) for 64 subdomains denotes that the desired accuracy has not be reached within the 1,000 iterations allowed. On the contrary, it requires almost 300 iterations with AGMRES(m,r) to converge.

   Thus, the main empirical conclusion from these experiments and others not reported here is that AGMRES is less sensitive to the restart length and the number of subdomains than GMRES. On the other hand, AGMRES is rather sensitive to the number of extracted eigenvectors. As for the basis length, it is difficult indeed to know how many vectors should be added to the basis to improve the convergence. If r is very large, the process of updating the eigenvectors could add more overhead. If r is small, the deflation could not be beneficial. The proposed adaptive strategy provides a trade-off between these two bounds.

If some information about the convergence behaviour has been collected before, then it can be used to define the *smv* and *bgv* parameters in the adaptive strategy. Our goal is to show that this technique can be used to speedup the convergence by adaptively adjusting the frequency and the number of extracted eigenvalues. We take the smallest restart length $m = 32$, the largest number of subdomains and the smallest number of eigenvectors $r = 1$. Yet, we know from $D = 16$ that GMRES(32) and thus AGMRES(32) needs roughly 254 iterations. From the adaptive strategy, we still set the maximum number of iterations $itmax = 1,000$ but now we set $smv = 0.1$ and $bgv = 0.2$. As explained in section 6.3.2.6, $smv \times itmax$ defines the lower bound below which it is not beneficial to use an augmented basis, and $bgv \times itmax$ defines the upper bound beyond which a slow convergence rate is expected and some action should be done. In this last case, we increase r by a fixed value l. We take $l = 2$ in this case. Figure 6.4.4 gives the convergence history of AGMRES($m,1$) with $m = 24$ and $m = 32$. It can be seen that when $r = 1$ and without

adaptive strategy, the augmented basis does not contain enough spectral information to speed up the convergence. When r is adaptively increased, the basis recovers more and more spectral information and the convergence rate gets better. The actual limitation of the proposed adaptive strategy is the choice of the right values of smv and bgv. It is heuristic and problem-dependent. Nevertheless, if there are some experimental knowledge about the convergence of GMRES on similar problems, a good interval can be set with smv and bgv around $itmax$ to detect a near-stagnation and switch to the augmented basis.

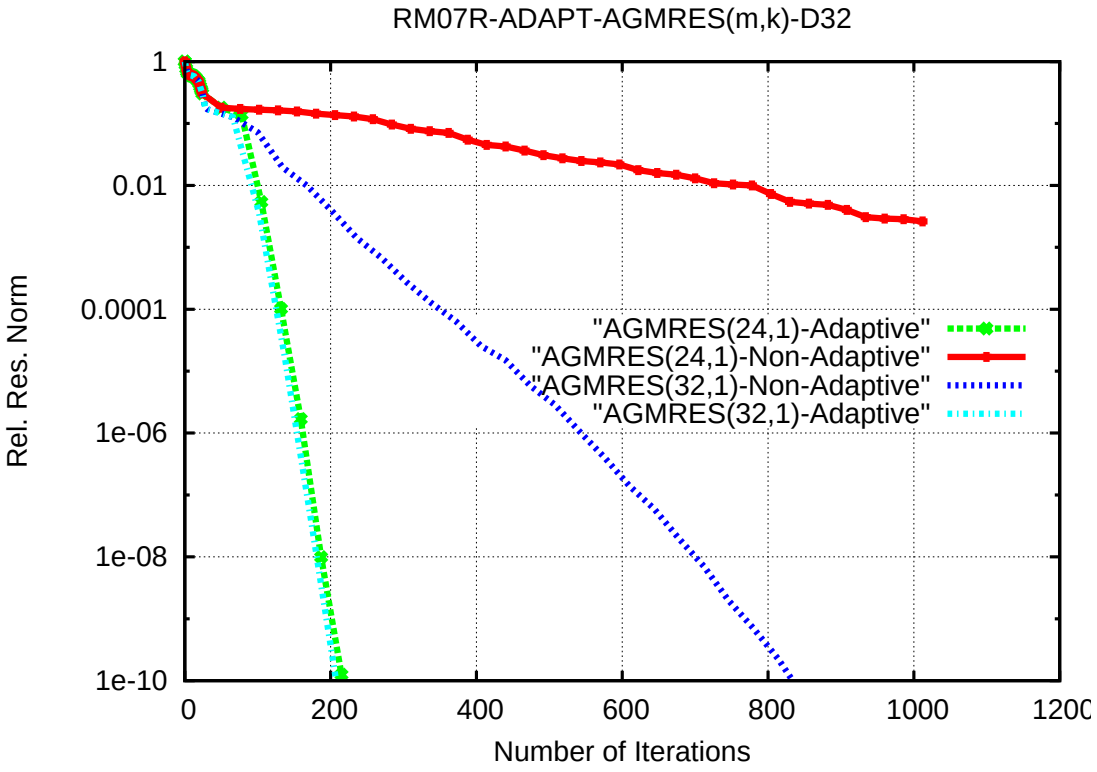


Figure 6.4.4: RM07R : Benefits of the adaptive deflation strategy, Restart=24 and 32, 32 subdomains

We end this section by reporting the number of iterations for the remaining test cases. In Table 6.3, we consider the best value of r which gives the smallest number of iterations for the test cases *IM07R* and *VV11R*. This value is typically less than 3. As noted before, we see that for a fixed value of m, the number of iterations increases as D increases. It increases faster in GMRES than AGMRES. We note here again that deflation is needed to reach a good accuracy for a large D. With *IM07R* test case for instance and for 32 sudomains in the additive Schwarz, neither GMRES(32) nor GMRES(48) can produce the desired accuracy while AGMRES(32) requires 724 iterations to converge. In Table 6.4, we give the number of matrix-vectors for the convection-diffusion problems. Unlike the previous test cases, GMRES here is less sensitive to the restart parameter and the variation of subdomains. Hence the augmented basis is not as beneficial to the convergence as in the previous cases. Nevertheless, AGMRES is still faster than GMRES if we consider the parallel efficiency. This is the aim of the next sections.

KSP	GMRES(m)			AGMRES(m,r)		
D \ m	24	32	48	24	32	48
VV11R						
8	251	191	147	248	172	146
16	499	458	288	492	304	207
32	-	957	670	641	541	516
IM07R						
8	240	235	189	249	203	195
16	695	623	521	378	370	316
24	927	913	759	492	444	408
32	-	-	833	724	629	579

Table 6.3: VV11R & IM07R: Number of iterations in GMRES(m), AGMRES(m,r) as a function of the number of subdomains in the restricted additive Schwarz, r is the best value of ≤ 3 which gives the smallest number of iterations in AGMRES

Matrix	3DCONSKY_121		3DCONSKY_161	
D \ m	GMRES(16)	AGMRES(16,1)	GMRES(16)	AGMRES(16,1)
16	158	169	229	177
32	164	141	251	177
64	170	141	261	177
128	180	141	262	177
256	202	159	266	195

Table 6.4: Number of matrix-vector products in GMRES and AGMRES for the test problems 3DCONSKY_121 and 3DCONSKY_161

6.4.5 Analysis of the CPU time

To better show the benefits of using an augmented subspace approach with the Newton basis, we analyze in this section the timing results. The paramount goal when showing these results is that, as we increase the number of subdomains, we should be able to get a decreasing time during the iterative time. In GMRES(m) and AGMRES(m), the best way is undoubtedly to increase the restart length as well. Even then, the time will not decrease efficiently because of the negative effects of the restarting procedure and the weakness of one-level Schwarz preconditioner. In AGMRES(m, r), only a few extracted Ritz vectors are sufficient to get a decreasing time and obtain a significant efficiency.

D	Algo.	Total Time	Iter. Time	Time/Iter	MSG ($\times 10^4$)
8	GMRES(32)	427.3	327.33	3.52	1.74
	GMRES(48)	386.1	291.64	4.166	1.41
	GMRES(64)	358.1	264.58	4.64	1.03
	AGMRES(32)	358.5	263.08	2.74	1.3
	AGMRES(48)	369.1	271.9	2.832	1.45
	AGMRES(64)	329.4	236.76	4.228	1.23
	AGMRES(32,r)	347.4	257.11	2.624	1.32
	AGMRES(48,r)	373.1	277.98	2.837	1.48
	AGMRES(64,r)	329.4	236.76	4.228	1.23
16	GMRES(32)	379.3	349.97	1.378	13.1
	GMRES(48)	333.1	302.66	1.791	9.05
	GMRES(64)	286.8	257.03	2.09	6.88
	AGMRES(32)	305.8	276.1	1.079	8.1
	AGMRES(48)	263.0	230.5	1.201	6.78
	AGMRES(64)	256.8	227.82	1.78	5.56
	AGMRES(32,r)	224.1	193.39	1.316	9.84
	AGMRES(48,r)	240.9	210.56	1.376	10.01
	AGMRES(64,r)	231.4	201.05	1.547	5.66
32	GMRES(32)	573.4	557.13	0.629	96.25
	GMRES(48)	239.5	223.54	0.63	39.74
	GMRES(64)	158.4	139.2	0.633	25.38
	AGMRES(32)	273.0	256.91	0.287	54.97
	AGMRES(48)	167.1	150.84	0.393	25.93
	AGMRES(64)	131.4	114.83	0.449	19.42
	AGMRES(32,r)	91.41	75.23	0.357	31.83
	AGMRES(48,r)	94.79	79.028	0.38	33.9
	AGMRES(64,r)	99.45	83.148	0.406	32.24
64	GMRES(32)	-	-	-	-
	GMRES(48)	214.8	204.16	0.291	227.02
	GMRES(64)	165.6	156.44	0.352	145.69
	AGMRES(32)	-	-	-	-
	AGMRES(48)	167.0	157.72	0.219	132.42
	AGMRES(64)	97.87	86.066	0.192	88.67
	AGMRES(32,r)	62.39	52.839	0.202	101.53
	AGMRES(48,r)	67.0	57.733	0.22	110.99
	AGMRES(64,r)	63.15	53.788	0.203	116.08

Table 6.5: Timing statistics for RM07R; D: Number of subdomains and number of MPI processes. Total Time: CPU elapsed time in seconds, Iter. Time: CPU time in the iterative phase. Time/Iter : average time spent in each iteration (matrix-vector product and preconditioning step). MSG: MPI messages and reductions. r : best value between 2 and 6 which gives the minimum number of iterations in AGMRES(m, r).

In Table 6.5, We compare GMRES(m), AGMRES(m) and AGMRES(m, r) on the test case *RM07R* by varying the number of subdomains D, the restart length and we choose a best value of r between 2 and 6. The number of MPI processes is equal to the number of subdomains. The total time is the CPU time required to perform all the steps in Algorithm 6. The iterative time is the time spent in the step 7. The setup time is the difference between the two times. It is independent of the method and of m. It decreases when D increases because the subdomains become smaller and the LU factorizations are faster. Thus, we concentrate from now on the iterative time. The time per iteration is the time of one cycle divided by the number of matrix-vectors products in the cycle, which is m or $m+r$. It includes the time to compute the orthonormal basis (with Arnoldi GMRES or the QR factorization for AGMRES) and the time to update the eigenvectors U for AGMRES(m, r). The iterative time is thus the product of the time per iteration by the number of iterations. The behaviours of both GMRES(m) and AGMRES(m) are similar. Increasing m has two opposite effects: it decreases the number of iterations (in some cases, the number of cycles remain the same for AGMRES) and increases the time per iteration, because of the orthogonalization steps. Thus, in most cases, there is an optimal value of m, which depends on D, with a minimal iterative time. Increasing D has also two opposite effects, but in the reverse way: it increases the number of iterations and decreases the time per iteration, thus there is in general an optimal value of D, which depends on m. Even though their behavior are similar, AGMRES(m) clearly performs faster than GMRES(m), for all but one configurations. This is mainly due to a faster time per iteration thanks to a more efficient parallel algorithm . This is explored in the next section by analyzing the communication volume.

The objective of deflation in AGMRES(m, r) is two-fold: to get an algorithm less sensitive to m and to increase the number of subdomains (thus the number of MPI processes). For D fixed, there is still an optimal value of m but it is smaller. The iterative time decreases from $D = 8$ until $D = 64$. Thus our method allows to choose a small value of m and to reduce the CPU time with a large number of subdomains. We get indeed a more efficient parallelism because the number of iterations does not inflate. Clearly, AGMRES(m, r) gives the smallest CPU time. These results are confirmed with other test cases as shown in Table 6.6 and Figures 6.4.5 and 6.4.6.

It is better for GMRES(m) to choose a small number of subdomains D and a large restart m. On the contrary, it is more efficient to choose a large number of subdomains D and a small restart m with our method AGMRES(m, r). Clearly, AGMRES(m, r) is faster than GMRES(m). In order to compare the methods with similar memory requirements, we choose $m = 24$ for AGMRES and $m = 48$ for GMRES, since AGMRES needs to store the two systems W_s and V_s. For all but one values of D, AGMRES($24, r$) is faster than GMRES(48), for both matrices VV11R and IM07R. It is also true for AGMRES($32, r$) compared with GMRES(64) for the matrix RM07R.

6.4.6 Analysis of parallelism

The other advantage of AGMRES over GMRES is the communication volume. In Tables 6.5 and 6.6 and in Figure 6.4.7, we have reported the number of MPI messages exchanged. The counts are done on the Send/receive routines as well as the collective communications (Reduce and broadcast). We do not take into account the MPI message lengths. It appears first that the number of messages is a function of the number of subdomains. This is generally reduced by allowing many subdomains to be assigned to a unique CPU. In the problems under study, this is not feasible in practise as it will induce more iterations as

D \ m	24		32		48		
	Iter. Time	MSG	Iter. Time	MSG	Iter. Time	MSG	
GMRES(m)							VV11R
8	92.84	2.05	68.95	1.69	77.7	1.47	
16	101.1	12.27	89.37	11.47	63.2	7.66	
32	-	-	31.2	22.5	29.7	18.54	
AGMRES(m,r)							
8	52.8	1.28	38.5	1.02	40.5	1.05	
16	51.8	7.4	34.5	4.91	28.08	3.87	
32	38.3	25.6	31.2	22.5	29.7	18.5	
GMRES(m)							IM07R
8	76.219	2.6	73.3	2.63	63.669	2.31	
16	111.74	20.06	96.246	18.25	83.583	15.76	
32	-	-	-	-	77.066	59.87	
AGMRES(m,r)							
8	45.781	1.65	40.905	5.48	40.85	1.52	
16	36.492	21.65	34.803	24.12	33.65	23.64	
32	33.262	94.54	27.837	93.27	27.109	105.35	

Table 6.6: Timing statistics in GMRES and AGMRES for test cases *VV11R* and *IM07R*. D: Number of subdomains and number of MPI processes. Iter. Time : time spent in the iterative phase. MSG: MPI messages and reductions. r : best value between 2 and 6 which gives the minimum number of iterations in AGMRES(m,r)

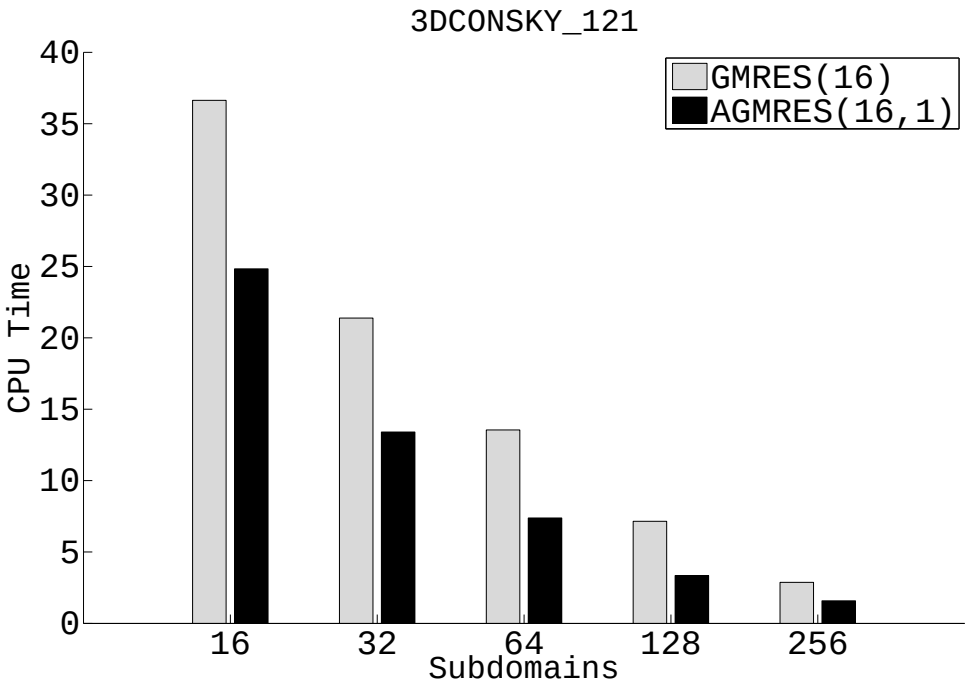


Figure 6.4.5: CPU Time in the iterative phase for the 3D $121 \times 121 \times 121$ convective SkyScraper problem (Matrix size 1,771,561; Nonzeros 50,178,241);16 to 256 subdomains, ILU(1) in subdomains; $m = 16$; $r = 1$

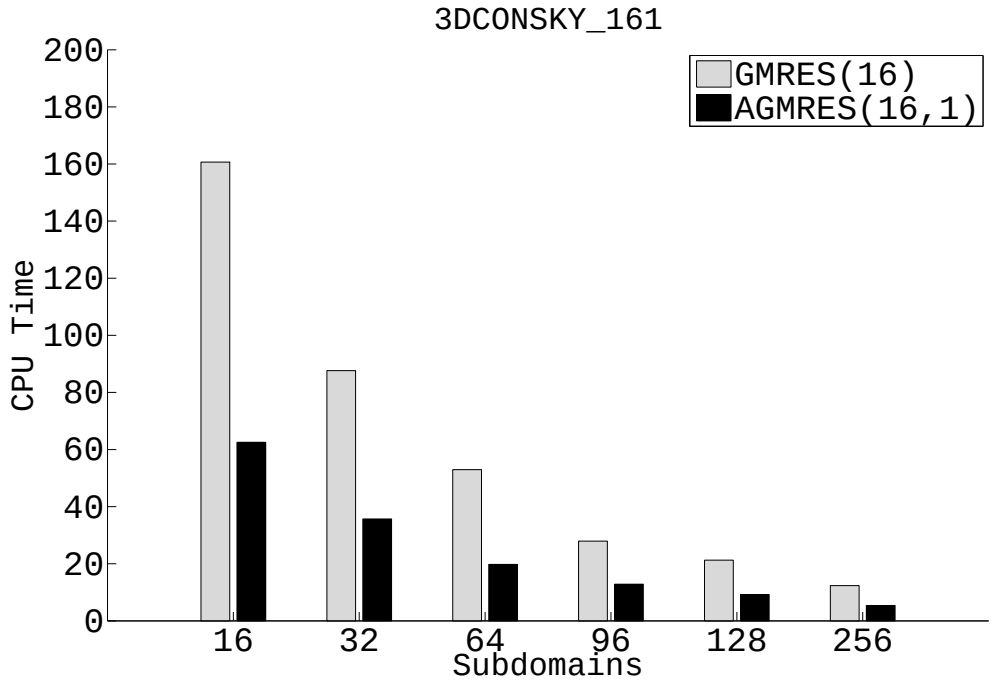


Figure 6.4.6: CPU Time in the iterative phase of GMRES and AGMRES for the 3D $161 \times 161 \times 161$ convective SkyScraper problem (Matrix size 4,173,281; Nonzeros 118,645,121); 16 to 256 subdomains, ILU(1) in subdomains; $m = 16$; $r = 1$

the subdomains increase. The communication volume is obviously proportional to the number of iterations as well. The second observation is that AGMRES communicates less than GMRES for the same number of subdomains and the same basis length. As more subdomains are used, the gap between the two methods increase. For instance, in Table 6.5, GMRES on 64 subdomains produces nearly a ratio of 1.5 more messages than AGMRES. In the augmented basis, the situation is different. At each cycle, AGMRES(m, r) communicates more than AGMRES(m) because of the computation of the eigenvectors. However, since a substantial number of iterations is saved by using the augmented basis, we observe actually a better communication in AGMRES(m, r). Now between GMRES and AGMRES(m, r), the previous analysis holds as well but there are two situations: when the restart length is very close to the number of subdomains, then the communication for the computation of eigenvectors may dominate if there is no substantial acceleration in the convergence rate of AGMRES(m, r). This is observed in Table 6.6 for *VV11R* and *IM07R*. With a substantial gain in the convergence rate as in Table 6.5, AGMRES(m, r) benefits from that and the communication volume decreases proportionally to the number of iterations. The second situation is when the number of subdomains is very large with respect to the basis length. Even if there is no substantial acceleration in AGMRES(m, k), the kernel computations of AGMRES will produce less communication volume than that in GMRES. This is observed in Figure 6.4.7. As the number of subdomains increase, the difference between the two methods are more and more distincts. Between the two situations, a fine-tuned adaptive strategy is still required to determine whether or not to augment the basis.

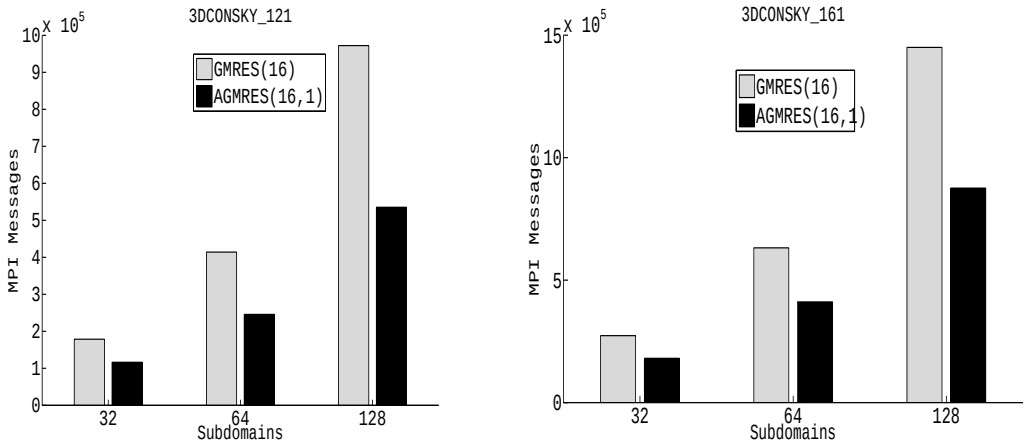


Figure 6.4.7: Amount of MPI Messages in the iterative phase of GMRES and AGMRES for the convective skyscraper problems on the 3D $121 \times 121 \times 121$ and $161 \times 161 \times 161$ grids

6.5 Concluding remarks

We have proposed the AGMRES(m, k, l) implementation, which combines the Newton basis GMRES implementation with the augmented subspace technique. This approach benefits from the high level of parallelism during the kernel computation of the Krylov basis. The proposed augmented basis reduces the negative effects due to restarting and to a large number of subdomains.

The numerical results on the VARGAS supercomputer (IBM Power 6 processors) confirm that AGMRES communicates less than GMRES and produces a faster solution of large linear systems. Moreover, on the proposed test cases, AGMRES gives a fairly good convergence rate when few eigenvectors are added to the Krylov basis. The proposed implementation is done in the PETSc package. It thus benefits from the optimized routines for the usual linear algebra operations on matrices and vectors. Its object-oriented interface allows to use transparently any parallel preconditioner implemented in the package, based on algebraic domain decomposition methods or multilevel methods. It can be used indeed as a smoother for algebraic multigrid methods [49].

Although the proposed augmented basis behaves well on the proposed test cases, there are some cases where it may not be useful and thus expensive to use. Hence a good analysis is still needed in the adaptive strategy to avoid the computation of eigenvectors on such cases.

Acknowledgments

This work was funded by the French National Agency of Research under the contract ANR-TLOG07-011-03 LIBRAERO. Numerical experiments have been done on the VARGAS supercomputer from GENCI-IDRIS (Grand Equipement National de Calcul Intensif - Institut du Développement et des Ressources en Informatique Scientifique). Preliminary tests have been carried out using the GRID'5000 experimental testbed (https://www.grid5000.fr). We are very grateful to Bernard PHILIPPE and François PACULL for many suggestions and helpful discussions during this work. We would like to thank Guy A. Atenekeng for providing us the test matrices of the Convection-diffusion equation. We are grateful to Roger B. Sidje for giving us the implementation of the RODDEC method.

CHAPTER 7

Overview of the parallelism and robustness in Krylov subspace methods with Schwarz preconditioners

7.1 Introduction

In this chapter, we give an overview of the main steps of the hybrid direct/iterative solver that have been used throughout this report. The goal is to present in a unified way some improvements that have been proposed at different steps. The focus is made on the parallel performance aspects and on the numerical robustness. The study is more experimental than theoretical, thus we give many illustrations with various linear systems arising from different applications.

The hybrid direct/iterative approaches used in this report are based on the following steps :

1. Perform a partitioning from the adjacency graph of the input sparse matrix.
2. Define the subdomains by dividing the variables into overlapping sets.
3. Setup the submatrices associated to each subdomain.
4. Solve iteratively the system using a Krylov subspace method with a Schwarz preconditioner.

We assume that a distributed-memory computer is used with a message passing programming model. The goal of the matrix partitioning is to distribute efficiently the data to different compute nodes and to define suitable subdomains for the Schwarz method. In section 7.2, we begin by showing how the partitioning of the input matrix is derived. From the matrix graph divided into (overlapping) subgraphs to form the subdomains, we show in section 7.3 how a preconditioner is formulated for the initial linear system. This preconditioner is based on the algebraic Schwarz methods. We describe especially the case of Restricted Additive Schwarz [30] and the explicit formulation of the Multiplicative Schwarz method [14]. We study the impact of the number of subdomains on the convergence. We also analyze the impact of the size of overlap on the convergence and the MPI communication. Schwarz methods are barely used as stand-alone solvers as their convergence is

guaranteed only for a class of matrices such as M-matrices. They are therefore always accelerated by Krylov subspace methods. This part is the heart of our study. In section 7.6, we review some ways to improve the parallelism of the global scheme. Section 7.6.4 presents especially the potential parallelism that is obtained by defining multiple levels of communications accross and inside the subdomains. We show in Section 7.7 how the restarted version of GMRES is accelerated by deflating eigenvalues in the preconditioned operator.

7.2 Graph Partitioning in iterative methods

Let A be a $n \times n$ sparse matrix with a structural symmetric pattern and let $G = (V, E)$ be its adjacency graph where $V = \{v_1, \ldots, v_n\}$ is the list of vertices and $E = \{(v_k, v_j), 1 \leq k, j \leq n, a_{kj} \neq 0\}$ is the list of edges. When the structure of A is nonsymmetric, we consider the graph of the matrix $A + A^T$. G is thus a non-oriented graph.

7.2.1 Non overlapping partitioning and matrix-vector product

A graph partitioning algorithm divides V in D non overlapping sets V_k, $k = 1, \ldots, D$ such that $V = \bigcup_{k=1}^{D} V_k$ and $V_k \bigcap V_j = \emptyset, 1 \leq k < J \leq D$.

A graph partitioning algorithm should produce partitions that have equal number of vertices i.e. $|V_k| \approx |V_j|, k \neq J$. Moreover, on distributed memory computers, the number of edges cut between different subdomains should be small i.e each set $V_j^E, 1 \leq j \leq D$ defined by

$$V_j^E = \{v_j \in V_j, \exists v_k \in V_k, 1 \leq k \leq D, k \neq j / (v_j, v_k) \in E\}, j = 1, \ldots, D \tag{7.1}$$

is minimal. When data are distributed to computers from this partitioning, these two goals ensure that the computational load is well balanced and that the volume of data exchanged is minimized. Note that the problem of finding balanced partitions that minimize edge cut is NP-complete, see [26]. Heuristics are often applied to obtain good sub-optimal solutions. Many public software tools are available to obtain such partitions, see for instance METIS [84], Scotch [35] and their parallel implementations PARMETIS and PT-SCOTCH.

From such partitioning, a parallel implementation of a parallel matrix-vector product $y = Ax$ is derived as in Algorithm 7, see [50, 113]. A single Program Multiple Data (SPMD) programming model is used. Each block is assigned to a unique process and the total number of processes P is equal to D. Based on the partitioning of equation 7.1, the matrix is reordered as a stride of rows such that each stride A_k is formed of a diagonal block matrix A_k^I corresponding to the nodes V_k and off-diagonal matrices A_k^E corresponding to nodes V_k^E. Each block row is then assigned to a process k. This is known as 1D rowwise partitioning. The vectors x and y are partitioned accordingly, that is the process k owns the block vector x_k. The steps 1-3 allow communication and computation to be overlapped.

Algorithm 7 Matrix-vector product kernel $y = Ax$

Require: x_k, A_k^I, P_E = processes that hold A_k^E

1: Scatter x_k to processes P_E
2: Compute $y_k = A_k^I x_k$
3: Gather x_k^E from processes P_E
4: Compute $y_k = y_k^I + A_k^E x_k^E$
5: **return** y_k

So far, the computational complexity is $\mathcal{O}(n \cdot nzr/P)$ where nzr is the average number of non-zero elements in each row. Since the entire row is assigned to a process, the bound of the number of MPI messages exchanged is $P \cdot (P-1)$. The total communication volume in this worst case is thus $n \cdot (P-1)$. This happens when there is at least one non-zero entry in the off-diagonal blocks A_k^E for each process. Note that this bound is reduced by assigning more than one block to a single process. It is further reduced if the matrix is permuted such that the off-diagonal non-zero entries in A_k^E are column-aligned. This is achieved using hypergraph partitioning algorithms such as PATOH, see [33].

7.2.2 Overlapping partitions

Until now, we have dealt with the effect of the partitioning on the matrix-vector product. A chosen partitioning algorithm should be able to produce a partitioning for the preconditioning operation as well. Given the right-preconditioned step $y \leftarrow AM^{-1}x$ of an iterative method, it is therefore expected that there is as little data movement as possible. This is known as the bipartite graph partitioning in [74] or the simultaneous partitioning problem in [129]; If the partitioning for $z \leftarrow M^{-1}x$ is different to that of $y \leftarrow Az$, then a reordering should occur in between the two operations. For many problems such as those arising from partial differential equations, the same algorithm is used efficiently for the two tasks. For the matrix-vector products, the partitioning algorithm should preserve the locality of data on structured computational domains such that the communication occur only between neighboring processes. In addition, the Schwarz preconditioners, which are the focus of this study, should require to have overlapping blocks to improve the convergence of the iterative method; see the discussion in [23] in the case of elliptic problems.

From the disjoint partitions $V_k, k = 1, \ldots, D$ described above, a δ-overlap partition ($\delta \geq 0$) is defined recursively such that $W_k^0 = V_k$ and $W_k^{\delta+1} \supset W_k^\delta$. A partition $W_k^{\delta+1}$ is defined from W_k^δ by adding a well-defined set of vertices O_k^δ :

$$W_k^{\delta+1} = W_k^\delta \cup O_k^\delta \text{ where } O_k^\delta = \{v_k \in V / (v_j, v_k) \in E, v_j \in W_k^\delta\}. \tag{7.2}$$

Various partitioning algorithms for Schwarz methods differ in the way of building the initial partitions V_k and the recursive overlapping partitions W_k^δ. The common case is to take the partitioning V_k for the matrix-vector product. Later on, the sets O_k^δ are defined by taking recursively the vertices that are adjacent to the partitions W_k^δ, see Equation (7.1).

The drawback when taking all the adjacent nodes to grow a partition is that, the new overlapping partitions may be very large. Moreover, depending on some criteria, not all adjacent nodes are 'good' nodes. In [37], Fritzsche, Frommer and Szyld propose, from any initial non-overlapping partition (produced by METIS or SCOTCH for instance), to combine several criteria for selecting nodes to be included in the existing non overlapping partitions. These criteria are formulated for instance by adding some weights to the nodes and/or the edges of the matrices. They are chosen for instance such that the new partitions produce well-conditioned subdomain matrices.

7.2.3 Weighted partitions

The weights can also be used for the non-overlapping partitioning. In this case, a multi-constrained partitioning algorithm should be used with the goals of minimizing the communication (the edgecuts) for the matrix-vector product and to equally distribute the weights among the subdomains. For instance, the PARMETIS [84] package which is used throughout this report includes a multi-constrained multilevel recursive partitioning algorithm.

We use for instance this functionnality in the multi-coupled system-matrix as given by the FLUOREM company (see Appendix A). The matrix is written as

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,N} \\ A_{2,1} & A_{2,2} & & A_{2,N} \\ \dots & & \ddots & \dots \\ A_{N,1} & A_{N,2} & \dots & A_{N,N} \end{bmatrix} \tag{7.3}$$

where $A_{I,J} \in \mathbb{R}^{b \times b}$, for $1 \leq I, J, \leq N$, and $N = n/b$. b is the number of physical variables at each grid point. Now, the partitioning is done on the mesh points (represented by the blocks A_{IJ} instead of the variables). In this case, each weight is computed as the Frobenius norm of the block matrix A_{IJ}; this constraint keeps the underlying coupling between the variables and ensures that the strongly connected mesh points are kept in the same subdomain. The advantages of such partitioning is presented in chapter 5.

7.3 Formulation of algebraical Schwarz preconditioners

From this point on, we assume that a partitioning algorithm has been applied to a sparse matrix A to produce overlapping blocks defined by the sets of vertices W_k^0 and W_k^δ, $1 \leq k \leq D$. Let R_k^0 (resp. R_k^δ) be the restriction operator from $\mathbb{R}^n$ to the vector subspace spanned by the indices in W_k^0 (resp. W_k^δ). Their transpose $(R_k^0)^T$ and $(R_k^\delta)^T$ are the respective prolongation operators. Therefore, the matrix associated to each subdomain k is defined as

$$A_k^\delta = R_k^\delta A (R_k^\delta)^T. \tag{7.4}$$

Note that there is no guarantee that A_k^δ is nonsingular even if A is, except if A is an M-matrix. In this situation, if some gaussiam elimination should be use to invert the matrix A_k^δ, the diagonal entries can be modified such that the block becomes diagonally dominant [11] and [58].

7.3.1 Additive Schwarz

With the restriction operator R_k^δ and the submatrices A_k^δ, $(k = 1, \dots, D)$ in mind, the preconditioner associated to the additive Schwarz method (ASM) is written as (see [59, 123, 127])

$$M_{ASM}^{-1} = \omega \sum_{i=1}^{D} (R_k^\delta)^T (A_k^\delta)^{-1} R_k^\delta \tag{7.5}$$

where $\omega > 0$ is a damping factor. We assign the block matrices A_k^δ and A_k^0 to a process P_k. We assume that the non-overlapping blocks A_k^0 are used for the matrix-vector product. Hence each vector follows the partitioning described in Section 7.2.1. Algorithm 8 computes one step of the preconditioned iterative method $y = M^{-1}x$. As previously stated, the code is executed by the process which owns the subdomain k. In a practical implementation, the operator R_k^δ is an index set which says for each row the process having that element. Thus the restriction operation at line 1 is a MPI send/receive where each process receives the off-process values and sends the local values needed by the other processes. The length of MPI messages is thus proportional to the size of the overlap. After that, a linear system is solved with the subdomain matrices. We discuss about this step in Section 7.3.4. Then, the updated off-process values are sent back to the neighboring processes.

Algorithm 8 Iterative step $y = M^{-1}x$ with ASM (Subdomain k)

Require: A_k^δ, x and R_k^δ
1: Compute $z = R_k^\delta x$ *(Full Restriction).*
2: Solve $A_k^\delta t_k = z$ for t_k *(Local solution with the subdomain matrix)*
3: Compute $u_k = (R_k^\delta)^T t_k$ *(Full interpolation)*
4: **return** $y = \omega \sum_{k=1}^{D} u_k$

7.3.2 Restricted Additive Schwarz

While working with ASM, Cai and Sarkis [30] found that if the interpolation step in Algorithm 8 is removed, then the 'new' ASM method converges faster. Moreover, the communication is reduced by half. It is formulated as follows.

$$M_{RAS}^{-1} = \omega \sum_{i=1}^{D} (R_k^0)^T (A_k^\delta)^{-1} R_k^\delta \tag{7.6}$$

This is known as the Restricted Additive Schwarz (RAS). It is used as the default preconditioner in many parallel libraries [18]. A convergence theory can be found in [47]. We give in Algorithm 9 the procedure to compute one step of a preconditioned iterative method. It is very similar to that in Algorithm 8. The only difference is at step 3. It requires only local data movements if the ordering of the output vector $z_k^{(3)}$ given by the local solver is different from the one required by the routine for the matrix-vector product.

Algorithm 9 Iterative step $y = M^{-1}x$ with RAS (Subdomain k)

Require: A_k^δ, x and R_k^δ
1: Compute $z = R_k^\delta x$ *(Full restriction).*
2: Solve $A_k^\delta t_k = z$ for t_k *(Local solution with the subdomain matrix)*
3: Compute $u_k = (R_k^0)^T t_k$ *(Local interpolation)*
4: **return** $y = \omega \sum_{k=1}^{D} u_k$

7.3.3 Multiplicative Schwarz

7.3.3.1 Classical formulation

The classical way to introduce the multiplicative Schwarz method is at the continuous level from the classical alternating Schwarz method; At the discrete level, it is equivalent to the block Gauss-Seidel relaxation method on an extended system (without overlap). Let the matrix A be partitioned in D subdomains with overlap as given in Section 7.2 and let $y = M^{-1}x$ where M^{-1} is the action of applying one iteration of this method. Starting with $y^{(0)} = 0$, the methods writes (see [123, 22]):

$$\begin{cases} r^{(k)} = x - Ay^{(k-1)} \\ y^{(k)} = y^{(k-1)} + (R_k^\delta)^T A_k^{-1} R_k^\delta r^{(k)}, k = 1, \ldots, D \end{cases} \tag{7.7}$$

and thus $y = y^{(D)}$. It follows that the matrix iterating on $r^{(0)}$ is given by

$$B = \prod_{k=1}^{D} \left(I - A(R_k^\delta)^T A_k^{-1} R_k^\delta \right) \tag{7.8}$$

It thus comes from $r^{(D)} = Br^{(0)}$ and $r^{(D)} = r^{(0)} - Ay(D)$ that $y^{(D)} = A^{-1}(I - B)r^{(0)}$. An explicit form of the preconditioner associated to the multiplicative Schwarz method is given by

$$M^{-1} = A^{-1}\left(I - \left((I - A(R_D^\delta)^T A_D^{-1} R_D^\delta) \cdots (I - A(R_0^\delta)^T A_0^{-1} R_0^\delta)\right)\right) \tag{7.9}$$

Note that this formulation cannot be used in practise. The procedure in Algorithm 10 is used instead. The vector y is corrected through all the subdomains. Compared to the additive version, it is thus less used in a parallel iterative method.

Algorithm 10 Iterative step $y = M^{-1}x$ with MSM (Subdomain k)

Require: D, A_k^δ, A, x and R_k^δ
1: $y \leftarrow$ *zero vector*
2: **if** $k > 0$ **then**
3: *Receive* $y^{(k-1)}$ *from the process with rank* $k-1$
4: **end if**
5: $r^{(k)} = x - Ay^{(k-1)}$
6: $y^{(k)} = y^{(k-1)} + R_k^\delta A^{-1} R_k^\delta r^{(k)}$
7: **if** $myrank < D$ **then**
8: Send $y^{(K)}$ to process with rank $k+1$
9: **else**
10: **return** $y = y^{(D)}$
11: **end if**

A good parallelism can be extracted from the algorithm through the coloring of subdomains (see [123]). For each subdomain, a color is associated such that neighboring subdomains have different colors. Let Q be the number of colors arranged in increasing order, let $\mathcal{C}_q = \{k, color_k = q\}$ be the set of processes having the color $q, 1 \leq q \leq Q$. Algorithm 10 is rewritten as in Algorithm 11. Now the size of the recursion is the number Q of colors. For each color, the current residual is corrected additively. Hence the fewer the number of colors, the better is the parallelism.

Algorithm 11 Iterative step $y = M^{-1}x$ with MSM through coloring of subdomains

Require: A_k^δ, A, x and $R_k^\delta, q, \mathcal{C}_q$,
1: $y \leftarrow$ *zero vector*
2: **if** $q > 0$ **then**
3: *Receive* $y^{(q-1)}$ *from the processes with color* $(q-1)$
4: **end if**
5: $r^{(q)} = x - Ay^{(q-1)}$
6: $z^{(k)} = R_k^\delta A_k^{-1} R_k^\delta r^{(q)}$, $k \in \mathcal{C}_{\text{II}}$
7: $y^{(q)} = y^{(q-1)} + \sum_{k \in \mathcal{C}_q} z^{(k)}$ *(Gather* $z^{(k)}$ *from processes* $k \in \mathcal{C}_q$*)*
8: **if** $q < Q$ **then**
9: Send $y^{(q)}$ to processes $k \in \mathcal{C}_{q+1}$
10: **else**
11: **return** $y^{(Q)}$
12: **end if**

The drawback with the multiplicative Schwarz preconditioner as presented here is that when the vector y is corrected, a residual vector $r = x - Ay$ is build at the same time wich

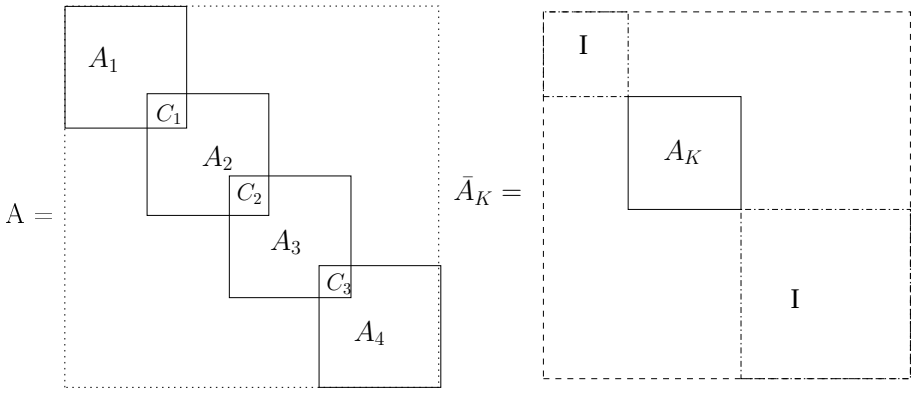
add an extra cost because of the product with the matrix A. Moreover, the parallelism is limited by the number of colors that can be extracted from the domain decomposition. The next section presents an explicit formulation of this preconditioner. It has several advantages over the previous formulations: there is no product with A; the explicit formulation can be easily used in an existing iterative method; there is also a more substantial parallelism; it requires however a specific partitioning.

7.3.3.2 Explicit formulation with a block diagonal partitioning

Here, the input matrix A is partitioned in a block-diagonal form as in Figure 7.3.1.(a) Such partitioning is obtained from the adjacency graph of the matrix by means of level sets and iterative refinement of partitions (see [12]). The main difference with other graph partitioning algorithms is that each subdomain should share its elements with at most two other subdomains. It is equivalent to say that when $|i-j|>1$ then $W_i \cup W_j = \emptyset$. With the matrices $\bar{A}_k$ and $\bar{C}_k$ completed by the identity matrix to the size of A (see Figure 7.3.1.(b)), the multiplicative Schwarz preconditioner is explicitly formulated as (see [14]) :

$$M_{MSM}^{-1} = \bar{A_D}^{-1}\bar{C}_{D-1}\bar{A}_{D-1}^{-1}\bar{C}_{D-2}\ldots\bar{A_2}^{-1}\bar{C}_1\bar{A_1}^{-1} \tag{7.10}$$

where C_k is completed as A_k by the identity matrix to the size of A.



(a) : block-diagonal form of A (b): Completed submatrix

Figure 7.3.1: Block diagonal partitioning of a matrix

The algorithm to compute the preconditioning step $y = M^{-1}x$ is derived as in Algorithm 12. In a practical implementation, the whole vector y need not be exchanged between the neighboring processes. Only the overlapped regions are needed. In the linear system to be solved at step 5, only the local matrix A_k is needed as well. A more detailed parallel algorithm is given in Algorithm 2 of chapter 3. Compared to the previous formulation, this explicit formulation do not require to compute the residual vector during the operation $y \leftarrow M^{-1}x$. Moreover, it provides a less expensive procedure that is applied to a sequence of vectors, for instance the preconditioned vectors of the Krylov basis (see the section 7.6).

7.3.4 Subdomain solver

During each application of the Schwarz preconditioner, a linear system is solved at the subdomain level. As a general linear system, its solution can be found using any of state-of-the-art solver. We discuss here on the general issues when choosing the type of the

Algorithm 12 Iterative step $y = M^{-1}x$ with the explicit formulation of MSM

Require: $C_k, \bar{A}_k^{\delta}$, x, R_k and O_k
1: $y^{(0)} \leftarrow x$
2: **if** $k > 0$ **then**
3: *Receive* $y^{(k-1)}$ *from the process with rank* $k-1$
4: **end if**
5: Solve $\bar{A}_k z = y^{(k-1)}$ for z
6: **if** $MyRank < D$ **then**
7: $y^{(k)} = \bar{C}_k z$
8: *Send* $y^{(k)}$ *to process* $(k+1)$
9: **else**
10: **return** $y(D)$ as y.
11: **end if**

solvers at the subdomain level. The sparse linear systems at the subdomain level are in the scope of the preconditioner operator M^{-1}. Therefore, the solutions of these systems are obtained more or less accurately:

1. An accurate solution, with a complete gaussian elimination for instance, leads to a robust preconditioner, thus accelerating the convergence of the global iterative method. The drawback here is the significant memory needed in each subdomain. Indeed, not only the local submatrix should be stored but also the L and U matrices from its LU factorization. With the presence of fill-in during numerical factorization, the size of those factors will also increase significantly with respect to the size of the original submatrix.

2. The problem of memory is overcome by using an incomplete (ILU) factorization of local matrices, thus producing fast solutions in subdomains with fair accuracy. In this case, the preconditioner looses its robustness and the global iterative method is more likely to stagnate.

3. A third alternative is to use an iterative method inside each subdomain. This approach could be useful if the GMRES method is used with deflation. In this case, the spectral information is retained from one linear system to another. Some promising results have been given in this sense with the DGMRES method presented in chapter 5. Note that when the local systems are solved this way, the expression of the preconditioner changes from one iteration to another. In this situation, a flexible variant of the Krylov methods such as FGMRES [112] should be used at the global level to guarantee a decreasing residual norm.

7.3.5 Scalable Schwarz preconditioners

In general, one-level Schwarz preconditioners tend to be weak as the number of subdomains increase. The construction of optimal parallel preconditioners which are independent of the number of subdomains require to build coarse-space approximations of the matrix. This is the basis of the multilevel domain decomposition preconditioners. As one-level Schwarz methods act locally on each subdomain to remove the low frequencies in the error solution, it may take longer to propagate the information from one subdomain to another. The multilevel preconditioners provide smoothers at the coarsest level that act globally on all subdomains to remove the high frequencies in the error solution. The main step

in multilevel Schwarz methods is to build successive coarse spaces that will carry global information to all the subdomains. Such spaces are obtained with less effort when the coefficient matrix arises from a structured PDE discretization [123]. If the matrix only is available, theory from algebraic multigrid technique is used to build the interpolation and the restriction operators between the different spaces. We refer the reader to [110] as an introductory material on these techniques. As an alternative to the multilevel methods, we propose in section 7.7 the deflation-based krylov methods to enhance the robustness in the Schwarz preconditioners. We first give in the next section a brief overview of the Krylov methods.

7.4 Krylov subspaces accelerators

The Krylov subspace methods compute the approximate solutions of the preconditioned system

$$Bx = b \text{ with } AM^{-1} \tag{7.11}$$

as

$$x_m = V_m y_m, \quad V_m \in \mathbb{R}^{n\times m}, y_m \in \mathbb{R}^m \tag{7.12}$$

where $V_m = [v_1, \ldots, v_m]$ form a basis of the Krylov subspace $\mathcal{K}_m = span(b, Bb, \ldots, B^{m-1}b)$. The basis is constructed using the Arnoldi or the Lanczos process. The Arnoldi process builds the basis V_m together with a Hessenberg matrix $\bar{H}_m \in \mathbb{R}^{m+1\times m}$ such that

$$BV_m = V_{m+1}\bar{H}_m \tag{7.13}$$

In the GMRES method [114], the vector y_m is computed such that the norm of the residual vector

$$r_m = b - Ax_m = V_{m+1}(e_1 - \bar{H}_m y_m) \tag{7.14}$$

is minimized, e_1 is the first canonical vector. The Lanczos process works with short recurrences to compute a basis V_m of $\mathcal{K}_m$ with respect to B and a second basis $W_m = [w_1, \ldots, w_m]$ with respect to B^T such that

$$W_{m+1}^T A V_m = D_{m+1}\bar{H}_m \tag{7.15}$$

where $\bar{H}_m$ is a block tridiagonal matrix. The Lanczos vectors v_j and w_j satisfy a biorthogonality condition related to the diagonal matrix D_m. The BiCG method [90] is derived using the Galerkin condition which requires the residual vector to be orthogonal to the subspace spanned by W_m. The FOM method [111] is derived as well from the Galerkin condition using the Arnoldi process and the subspace spanned by V_m. The QMR method [57] uses the Lanczos process and the minimal residual condition to compute the vector y_m. GMRES/FOM and QMR/BiCG differ in their storage requirements and the amount of work at each iteration. BiCG and QMR require matrix-vector product by B and B^T but require minimal storage requirements. BiCGSTAB(m) [130] is based upon BiCG and GMRES(m) and TFQMR [56] is based upon QMR. BiCGSTAB and TFQMR are transpose-free. They do not require the product with B^T. GMRES and FOM access all the previously-generated Arnoldi at each iteration but require only B. GMRES(m) uses the restarting procedure to limit the storage and the computational work required at each iteration. In GMRES, the recursions can break down only if at some stage k of the Arnoldi process, $h_{k+1,k} = 0$ which means that the exact solution is found, see [113, Proposition 5.6]. This is called a *happy breakdown*. In BiCG, the recursions break down if a term $w^T v = 0$ for some $w \neq 0$ and $v \neq 0$. This means that for some k, $\hat{r}_k^T r_k = 0$ but $\hat{r}_k^T \neq 0$ and $r_k \neq 0$ where $\hat{r}_k^T$ is the

residual row vector associated to B^T. See [24, 52, 113, 96, 121] for an overview of these methods and [78] for a unified analysis on the error and residual bounds.

We focus here on the GMRES(m) method. It is generally used for its monotonic convergence property. In the next section, we give some illustrations of this method as accelerator to Schwarz methods. In Section 7.6, we analyze different strategies to derive the Krylov basis needed in GMRES(m) and we discuss on the data dependency with Schwarz preconditioners. In Section 7.7, we show how the deflation of eigenvalues is used to improve the convergence and the robustness of the method.

7.5 Numerical behavior of Schwarz preconditioners with GMRES

The aim of this section is to give some illustrations on the difference between the Schwarz methods as preconditioners for the GMRES(m) method.

7.5.1 Additive Schwarz and Restricted additive Schwarz

For a given number of subdomains, the convergence of an iterative algorithm depends heavily on the size of the overlap. When it increases, it is expected to have a better convergence since the preconditioner gives a better approximation of the inverse of A. The trivial case is for two subdomains. We can notice indeed from Equation (7.2) that for a maximum value of δ, $W_1^\delta = W_2^\delta = V$. In this section, we first analyze the effect of the overlap in the additive Schwarz preconditioner, then we compare the additive Schwarz to the restricted variant in terms of the number of iterations and the MPI messages exchanged.

We consider the test problem IJ02R from our database test where $n = 7,000$ and $nnz = 753,500$. The matrix is partitioned into 4 submatrices using the PARMETIS package. The PETSc [18, 19] implementation of the additive Schwarz is used for all tests. The Krylov basis is built with the Arnoldi process using the modified Gram-Schmidt orthogonalization; The size of the Krylov basis is 32 (restart parameter m). The influence of this parameter is analyzed in section 7.7. The relative norm of the residual vector (i.e $\|b - Ax\|/\|b\|$) at the convergence is less than 10^{-8}. A right preconditioning is used and the local solutions are solved at the machine precision using a direct solver (MUMPS [4] in this case). The 0-overlap corresponds to a block Jacobi. As explained in Section 7.2, a δ-overlap is defined by taking recursively the adjacent rows/columns of a given subdomain; adjacency is given here in terms of the graph of the sparse matrix.

From the convergence history in Figure 7.5.1, we note that the method converges very fast as the overlap increases. This has a cost. We see in Table 7.1 that the size of subdomain matrices A_k^δ increases very fast with the overlap. When the overlap is 5, the maximum size overall subdomain matrices is almost equal to that of the global matrix. Moreover, The volume of exhanged MPI messages will increase as well. This is reduced using the restricted additive Schwarz.

Now, we give in Table 7.2 the MPI messages and the number of GMRES iterations when the restricted Additive Schwarz (RAS) is used. As expected, the number of iterations is better than that in ASM. Moreover, the number of MPI messages is divided by two since there is no communication for the local interpolation at step 3 of Algorithm 9. Note that the statistics related to the MPI messages are given only for the preconditioning step i.e. the operation $z \leftarrow M^{-1}x$.

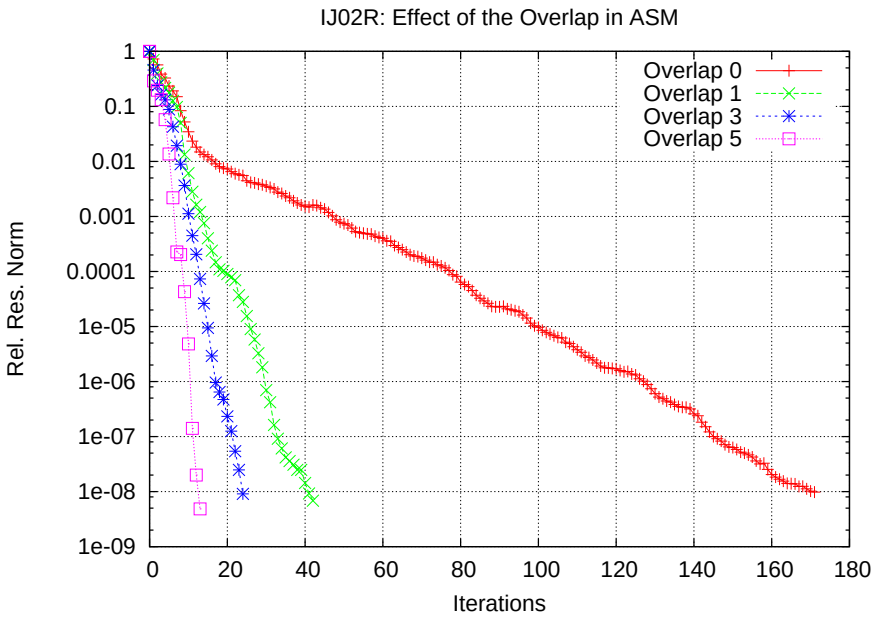


Figure 7.5.1: Effect of the overlap in the additive Schware preconditioner with four subdomains

Table 7.1: *IJ02R* : Statistics of the ASM preconditioner with respect to the overlap; NLoc: Maximum size overall subdomain matrices ; ITS: Number of GMRES iterations; MPI Messages for $z \leftarrow M^{-1}x$; Counts and Length: Average length of MPI messages; Counts/ITS : Number of MPI Messages per GMRES iteration

Overlap	NLoc	NNZ	ITS	MPI Messages		
				Counts	Length	Counts/ITS
0	1775	186,250	171	0	0	0
1	2900	313,300	42	900	2700	21.4
3	5225	565,575	24	520	8300	21.6
5	6475	696,727	13	280	12000	21.5

Table 7.2: *IJ02R* (size=7,000, entries=753,500): Statistics of the RAS preconditioner with respect to the overlap; NLoc and NNZ: Maximum size and entries overall subdomain matrices; ITS: Number of GMRES iterations; MPI Messages for $z \leftarrow M^{-1}x$; Length: Average length of MPI messages; Counts/ITS : Number of MPI Messages per GMRES iteration

Overlap	NLoc	NNZ	ITS	MPI Messages		
				Counts	Length	Counts/ITS
0	1775	186,250	171	0	0	0.00
1	2900	313,300	34	370	2700	10.88
3	5225	565,575	11	120	8300	10.91
5	6475	696,725	9	100	12000	11.11

7.5.2 Restricted additive Schwarz and Multiplicative Schwarz

As a well-known theory, the multiplicative Schwarz method should converge faster than the additive variant for the same partitioning. To illustrate this property, we consider the same matrix IJ02R. A block-diagonal permutation is performed to produce 4 partitions. The main characteristics are given in Table 7.3. In Figure 7.5.2, we give the convergence

Table 7.3: Block diagonal partitioning of IJ02R $size = 7,000$, $entries = 753,500$

Block #	1	2	3	4
Size	2,155	2,170	2,125	2,130
Entries	220,900	253,675	241,000	214,925
Size of overlap	525	550	505	0

of GMRES(16) and we note that ASM requires twice iterations than MSM. Here the same

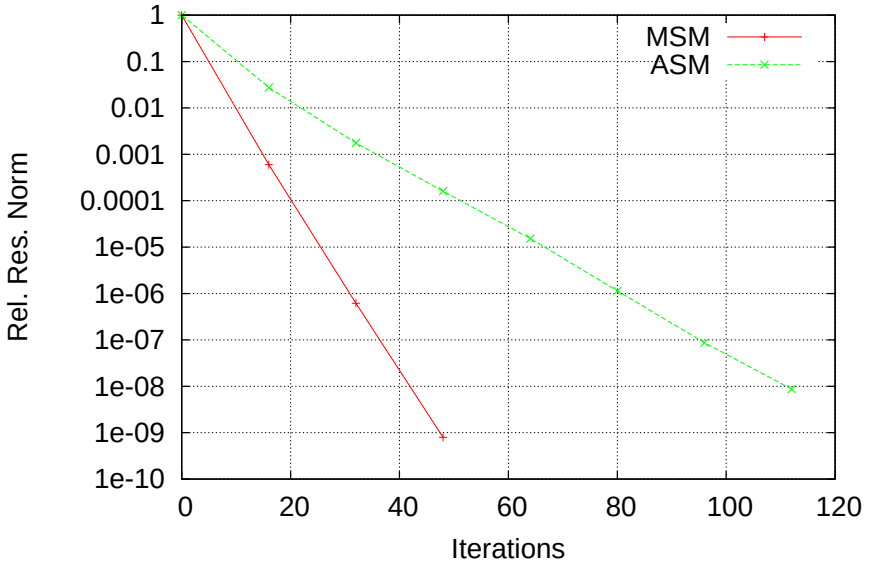


Figure 7.5.2: IJ02R:GMRES(16) with Multiplicative Schwarz and additive Schwarz on a block diagonal partitioning

partitioning is used to formulate the two Schwarz preconditioners. In this case, the additive Schwarz method is formulated as follows :

$$M_{ASM}^{-1} = \mathfrak{D} \sum_{k=1}^{D} \bar{A}_k^{-1}. \tag{7.16}$$

$\mathfrak{D}$ is a $(n \times n)$ diagonal matrix where the coefficients are the damping factors for the overlapping regions. In a more general situation, the block-diagonal partitioning is not required to formulate the additive Schwarz method. We therefore consider in the following experiments the additive Schwarz method with the nested dissection partitioning from the PARMETIS package as provided by the PETSc library.

To better understand the robustness of the multiplicative Schwarz preconditioner over the restricted additive Schwarz (RAS) method in an algebraic point of view, we consider an ill-conditioned Helmholtz problem on a 2D cartesian grid as presented in [44]. The

partitioning is not done on the physical grid but on the adjacency graph of the input matrix, either with the PARMETIS package for the RAS preconditioner or the block diagonal partitioning with overlap (BDO) for MSM. The main characteristics of the subdomain matrices with 4 and 8 subdomains are presented in Tables 7.4 and 7.5. In BDO, the partitioning is done so that to equilibrate the size of the submatrices and the overlap between neighboring subdomains. In PARMETIS, disjoint partitions are created such that to minimize the edge cuts. After that, overlapping submatrices are created by including the connected rows/columns. This is equivalent to an overlap of length 1. It can be seen from Tables 7.4 and 7.5 that the two partitioning schemes produce submatrices with similar size and number of nonzero entries.

Block #	1	2	3	4	Total
Block Diagonal Partitioning with Overlap					
Size	6,899	6,744	6,746	6,905	27,294
Entries	33,579	33,336	33,338	33585	133,838
Overlap	117	164	117	0	
PARMETIS partitioning and extended Overlap of size 1.					
Size	6924	7154	6967	6962	28,007
Entries	33095	34669	33543	33682	134,989

Table 7.4: Block diagonal partitioning and PARMETIS partitioning on an 2D Helmholtz problem into 4 overlapped subdomains. initial global size = 26,896, initial number of nonzeros entries = 131,872

Block #	1	2	3	4	5	6	7	8
Block Diagonal Partitioning with Overlap								
Size	3,482	3,383	3,483	3,519	3,521	3,482	3,383	3,488
Entries	16,766	16,643	17,199	17,411	17,413	17,199	16,643	16,772
Overlap	83	116	142	164	142	116	83	0
PARMETIS partitioning and extended Overlap of size 1.								
Size	3,589	3,517	3,551	3,500	3,744	3,602	3,625	3,637
Entries	17,099	16,552	17,030	16,568	17,986	17,363	17,631	17,374

Table 7.5: Block diagonal partitioning and PARMETIS partitioning on an 2D Helmholtz problem into 8 overlapped subdomains; size = 26,896, entries = 131,872

Now we consider the GMRES method preconditioned by the restricted additive Schwarz or the explicit formulation of the multiplicative Schwarz (MSM). Two types of solvers are used to compute the solution in the subdomains. We first consider an approximate solver based on an incomplete gaussian elimination in the subdomain matrices. Figure 7.5.3 gives the reduction in the norm of the residual vector as a function of the number of iterations in GMRES. It is seen that either with 4 or 8 subdomains, GMRES(16)+MSM requires less than 200 iterations to reach the accuracy of 10^{-8}. With RAS however, GMRES starts to converge but after almost 32 iterations, the method experiences a slow reduction in the residual norm and the desired accuracy could not be reached after 500 iterations. In Figure 7.5.4 now, we consider an exact solver based on the LU factorization of the submatrices. As an immediate effect of the solver in the subdomains, RAS performs better than in the previous case. MSM performs better as well in the case of 4 subdomains. Surprisingly with 8 subdomains, MSM+GMRES requires more iterations than that in the previous case. We

note a periodic stagnation in the convergence curve. This is mainly due to the restarting procedure as explained in section 7.7.

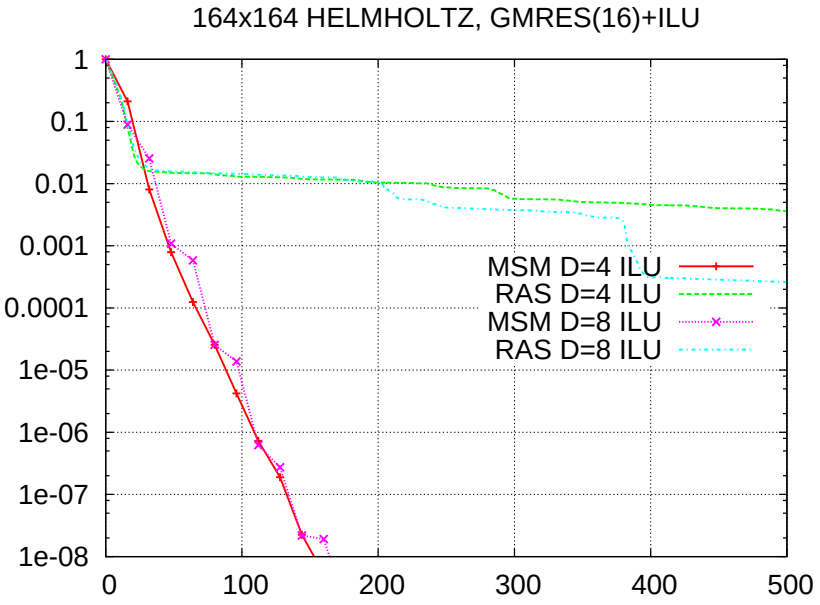


Figure 7.5.3: 2D Helmholtz problem on a 164×164 grid : Comparing the restricted additive Schwarz (with ParMETIS partitioning) and multiplicative Schwarz (with block diagonal partitioning). GMRES(16) as the Krylov accelerator, ILU solver in subdomains

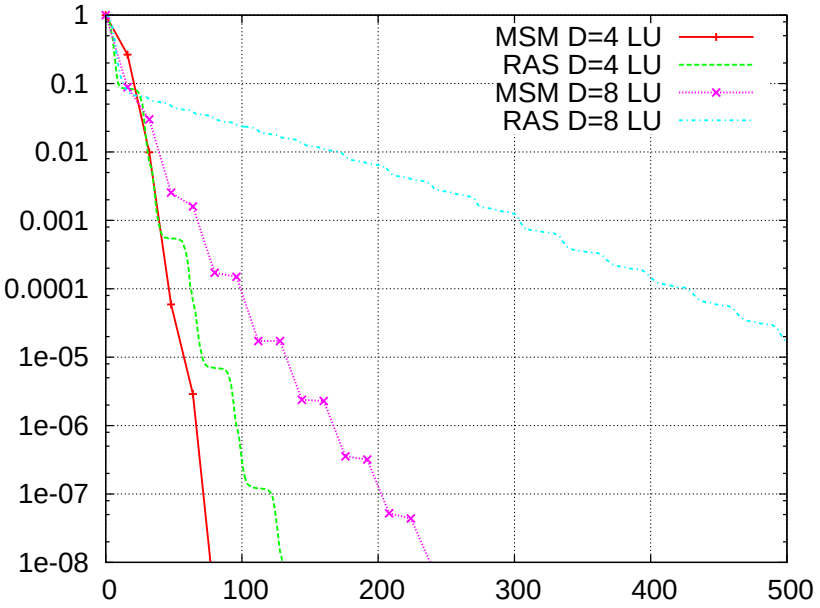


Figure 7.5.4: 2D Helmholtz problem on a 164×164 grid : Comparing the restricted additive Schwarz (with ParMETIS partitioning) and multiplicative Schwarz (with block diagonal partitioning). GMRES(16) as the Krylov accelerator, LU solver in subdomains

A general observation from these test cases and others is that MSM is more robust

than ASM for all number of subdomains. However, since MSM is less parallel than RAS, it will require more CPU time. In the next section, we give the parallelism that is exploited in the explicit formulation of the multiplicative Schwarz and we show in section 7.6.4 how this parallelism can be enhanced.

7.6 Improving the parallelism

In this section, we analyze some ways to improve the parallelism in the GMRES method with Schwarz-based preconditioners. We start by showing how to compute the Krylov basis, then we illustrate the effect of the data dependency during the construction of this basis. We end this section by showing how to improve the parallelism at the subdomain level of the Schwarz preconditioners.

7.6.1 Deriving the Krylov basis

As stated in section 7.4, the formulation of the GMRES method relies on the computation of an orthogonal basis V_m of a m-dimensional Krylov subspace. It is a two-steps process:

1. Computation of the basis vectors.
2. Orthogonalization of the basis vectors.

These two steps can be interleaved or separated to give different performance in terms of communication volume, data dependency and robustness. The first strategy is based on the Arnoldi process which compute the basis vectors and orthogonalized them at the same time. The second strategy generate first the basis vectors with the Newton polynomials for instance and orthogonalize them with a dense QR factorization. We refer to the second strategy as the *Newton basis process*. In the Arnoldi process, the subspace is spanned by the following vectors

$$K_m = [b, Bb, \ldots, B^m b] \tag{7.17}$$

while the Krylov subspace obtained from the Newton basis process is spanned by the vectors

$$K_m = \left[b, (B - \lambda_1 I)b, \ldots, \prod_{k=1}^{m} (B - \lambda_k I)b \right]. \tag{7.18}$$

where the scalars $\{\lambda_k\}_{k=1}^{m}$ are used to get a well-conditioned basis, see [17]. The two processes generate a basis for the same Krylov subspace $\mathcal{K}_m$.

A practical parallel implementation of the Arnoldi process uses the Gram-Schmidt method. For references purposes, we give in Algorithm 13 the classical Gram-Schmidt (CGS) and the modified Gram-Schmidt (MGS) as found in [113]. The Newton basis process to build and orthogonalize the basis vectors is given in Algorithms 14 and 15 as respectively found in [17] and [118]. The communication volume* for the Arnoldi-MGS is $\mathcal{O}(m^2 log(P)$ messages to compute the $(m^2 + 3m)/2$ inner products where P is the total number of MPI processes. The maximum message length here is 1 double precision number plus the overhead. In CGS, the reduce-scatter operation of MPI is used on a group of vectors during the computation of the inner products to limit the number of MPI messages. Hence the communication volume for CGS is $\mathcal{O}(m.log(P))$ with an average message length of $m/2$ double precision numbers. In the implementation of the Newton basis process, the

*Here, we do not consider the MPI communication during the matrix-vector product and the application of the preconditioner

Algorithm 13 Arnoldi process to compute the Krylov basis V_m, at left is the classical Gram-Schmidt(CGS), at right is the modified Gram-Schmidt(MGS)

Require: m, A, b
1: $v_1 \leftarrow b/\|b\|_2$
2: **for** $j = 1 : m$ **do**
3: $w_j \leftarrow AM^{-1}v_j$
4: **for** $i = 1 : j$ **do**
5: $h_{i,j} \leftarrow v_i^T w_j$
6: **end for**
7: $w_j \leftarrow w_j - \sum_{i=1}^{j} h_{ij} v_i$
8: $h_{j+1,j} = \|w_j\|$
9: $v_{j+1} = w_j / h_{j+1,j}$
10: **end for**
11: **return** $V_{m+1} \in \mathbb{R}^{n \times m+1}$ orthonormal basis of $\mathcal{K}_m$, $\bar{H}_m \in \mathbb{R}^{m+1 \times m}$ upper Hessenberg matrix.

Require: m, A, b
1: $v_1 \leftarrow b/\|b\|_2$
2: **for** $j = 1 : m$ **do**
3: $w_j \leftarrow AM^{-1}v_j$
4: **for** $i = 1 : j$ **do**
5: $h_{i,j} \leftarrow v_i^T w_j$
6: $w_j \leftarrow w_j - h_{ij} v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|$
9: $v_{j+1} = w_j / h_{j+1,j}$
10: **end for**
11: **return** $V_{m+1} \in \mathbb{R}^{n \times m+1}$ orthonormal basis of $\mathcal{K}_m$, $\bar{H}_m \in \mathbb{R}^{m+1 \times m}$ upper Hessenberg matrix.

Algorithm 14 Newton basis process to compute the Krylov basis W_{m+1}

1: m, A, b, Shifts $\{\lambda_k\}_{k=1}^{m}$ ordered with the modified Leja ordering [17].
2: $w_1 \leftarrow b/\|b\|$
3: $d_1 = 1/\|w_1\|_2$, $j = 1$
4: **while** $j \leq m$ **do**
5: **if** $Im(\lambda_j) = 0$ **then**
6: $w_{j+1} = d_j(AM^{-1} - \lambda_j I)w_j$
7: $d_{j+1} = 1/\|w_{j+1}\|_2$
8: **else if** $Im(\lambda_j > 0)$ **then**
9: $w_{j+1} = d_j(AM^{-1} - Re(\lambda_j)I)w_j$
10: $w_{j+2} = (AM^{-1} - Re(\lambda_j)I)w_{j+1} + d_j Im(\lambda_j)^2 w_j$
11: $d_{j+1} = 1/\|w_{j+1}\|$
12: $d_{j+2} = 1/\|w_{j+2}\|$
13: **end if**
14: **end while**
15: **return** $W_{m+1} \in \mathbb{R}^{n \times m+1}$ (non orthogonal) basis of $\mathcal{K}_m$, $D_{m+1} \in \mathbb{R}^{m \times m}$ diagonal matrix

Algorithm 15 RODDEC algorithm to compute the QR factorization of W_{m+1}

Require: $m, nbproc, myproc, myleft, myright, W_{Loc}$ Local part of the basis W_{m+1}

1: $V_{loc}, R_{Loc} = QR(W_{Loc})$
2: **for** $j = 1 : m + 1$ **do**
3: **if** $myproc = 0$ **then**
4: Send row $R_{LOC}(j, j : m + 1)$ to $myright$
5: **else**
6: Receive row$(d : m + 1)$ from $myleft$
7: Compute the Givens rotation to annihilate element $R_{Loc}(1, d)$
8: **if** $myproc \neq nbproc - 1$ **then**
9: send the updated row row$(d : m)$ to $myright$
10: **end if**
11: Create rotations to annihilate the d^{th} diagonal of R_{Loc}.
12: **end if**
13: **end for**
14: **return** V_{m+1}, $R \in \mathbb{R}^{(m+1)\times(m+1)}$ triangular matrix.

computation of the basis requires $\mathcal{O}(m.log(P))$ messages for the m inner products and $\mathcal{O}(m^2)$ point-to-point messages during the orthogonalization. The average message length is $(m+1)/2$ double precision number. Other parallel optimal strategies exist to perform a dense QR on the Newton basis vectors while producing less communication volume. The recently proposed *tall skinny* QR (TSQR) of Demmel *et al*[42] requires $\mathcal{O}(log(P))$ MPI messages with an average message length of $(m + 1)/2$. The communication volume is not the only point of concern when choosing how to build the basis. The efficiency in the computation of the basis is also determined by the data dependency between the processes sharing the computation of the basis vectors. We illustrate next two types of dependencies.

7.6.2 Illustration of data dependency between the Krylov basis vectors

The first dependency is between the computation of the successive basis vectors. It is mostly determined by the strategy to generate those vectors. With the Arnoldi process, one basis vector should be entirely computed before the next vector starts. As presented in Algorithm 14, the Newton basis process has the same effect because of the presence of the inner products which introduce global synchronizations between all the processes. These inner products can be removed in the case of the Newton basis process for a moderate size of the basis. It is equivalent in Algorithm 14 to set $d_j = 1, 1 \leq j \leq m$. We refer to this version as *Newton-NoNorm*. To avoid the rapid growth of the basis vectors, the rows and columns of the matrix are equilibrated using the parallel iterative algorithm presented in [6]. Now, we have four strategies : the classical Gram-Schmidt (CGS), the modified Gram-Schmidt(MGS), the Newton basis process ($Newton$) of Algorithm 14 together with the orthogonalization in Algorithm 15 and the Newton basis process without global communications *Newton-NoNorm*.

To illustrate the effect of these dependencies in the GMRES method, we consider a simple 2D Poisson problem discretized in a 512×512 unit square with finite difference methods. The restricted additive Schwarz method is used with a 1-overlap; the incomplete LU (ILU(0)) is used as the local solver. Figure 7.6.1 gives the CPU time with respect to the number of processes for the parallel iterative time of GMRES(12) and GMRES(24). We note globally that either with CGS or MGS, the Arnoldi process requires more CPU time than the Newton basis process. As we expected, there is a huge difference between

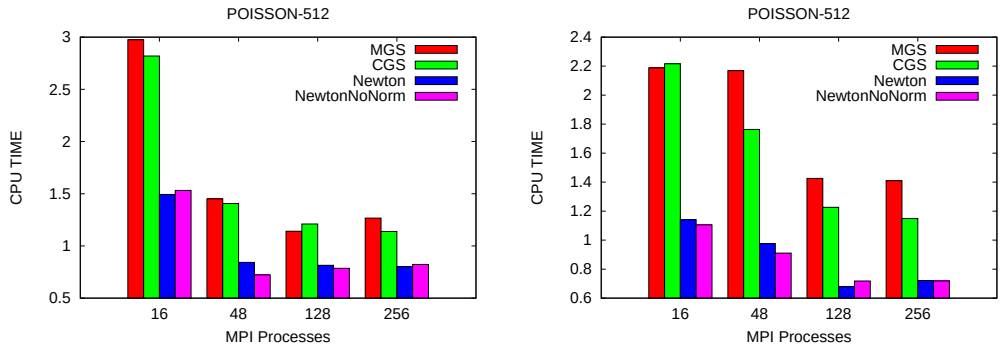


Figure 7.6.1: POISSON problem on a 512×512 unit grid: CPU Time of the parallel iterative loop of GMRES(12) (left figure) and GMRES (24) (right figure), RAS preconditioner

the four strategies if we consider the total number of MPI reductions as plotted in Figure 7.6.2. However, this does not affect the CPU time between CGS and MGS and between Newton and Newton-NoNorm. The main reason is because the processes are connected through an Infiniband network which has a very low latency. On Gigabit ethernet cluster, we would see a difference as the number of processes increase. For a broader analysis of the efficiency of the Newton basis process over the standard parallel Arnoldi process, see the chapter 6 and [50].

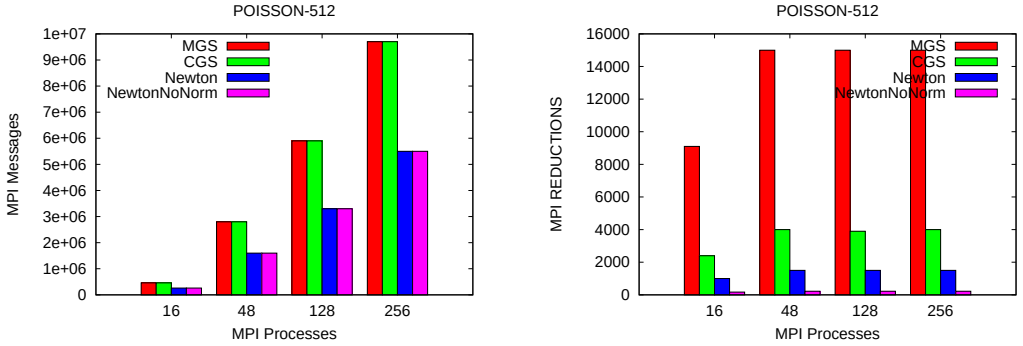
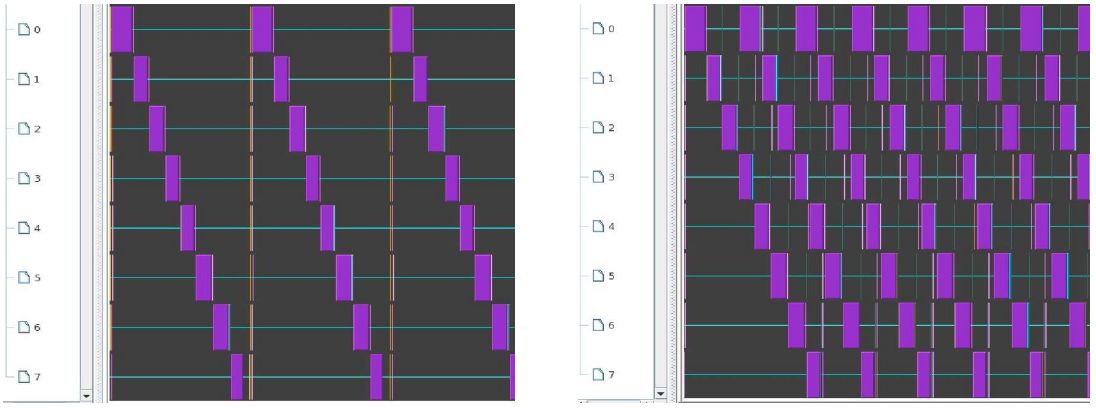


Figure 7.6.2: POISSON problem on a 512×512 unit grid: MPI messages (left figure) and reductions (right figure) in the parallel iterative time of GMRES

7.6.3 Illustration of data dependency with the multiplicative Schwarz

The second type of dependency arises when the Schwarz preconditioner is applied to build a single basis vector. In either the additive or the multiplicative Schwarz, this data dependency is between the processes having neighboring subdomains. However, the multiplicative Schwarz produces more synchronizations because of the recursive update of the basis vector through all the subdomains. For instance, when the explicit formulation of the Multiplicative Schwarz as presented in section 7.3.3.2 is applied to a basis vector and if one subdomain is assigned to one process, then only one process is working at a time. Thus the two dependencies (between the basis vectors and between the subvectors owned by each process) would produce a sequential algorithm across the subdomains if the Arnoldi process is used to build the basis vectors. This is caused by the global communications needed to orthogonalize the basis vectors. In [11, 125], the authors use the Newton basis process to

produce a substantial parallelism during the construction of the basis vectors. Althoug the dependency between the subvectors are not avoided, the sequence of basis vectors can be computed as in a pipeline. We show for instance in Figure 7.6.3 a MPI profiling during the computation of the basis vectors by 8 processes. It is seen that the Newton basis process relax the dependencies between the successive basis vectors by allowing next vectors to be computed before the previous are available.



(Arnoldi process) (Newton basis process)

Figure 7.6.3: Parallelism in the construction of the Krylov basis with the multiplicative Schwarz

In [101], we have investigated on the efficiency that are obtained with this process and we have shown that it is limited by the number of subdomains and the number of basis vectors that are built at once. In theory, increasing the length of the basis vectors built in one pass should enhance the efficiency of the approach. In practise, the basis vectors grow very fast as the length increases and the condition number of the basis increases as well. The extreme case is when the basis is rank-deficient. A first solution is to equilibrate the rows and the columns of the input matrix. This is efficient until some size of the basis. A second solution is to incrementally estimate the rank of the basis and discard the other vectors in the basis [11]. This affects the performance since the time to build the discarded search directions is wasted. In chapter 6, we suggests to use the augmented subspace approaches to limit the basis length. In section 7.7, we show that it can be applied to produce more stable Krylov basis for GMRES with the multiplicative Schwarz preconditioner.

7.6.4 Improving the parallelism through the subdomain solvers

Until now, we have assumed that one or several subdomains to a unique process. In terms of numerical convergence and parallel performance, we have seen that this approach has some limitations. For instance, the number of iterations tends to increase with the number of subdomains. It is therefore essential to keep this number of subdomains small in order to provide a robust iterative method. Depending on the initial size of the matrix, the subdomain linear systems are getting large and cannot be solved efficiently with only one process in a subdomain. It has been found natural [65, 92, 101, 102] to use a parallel linear solver to solve these subdomain systems. On today's supercomputers with hierarchical architectures from SMT cores to nodes and from nodes to clusters, this approach exploit efficiently the available compute ressources. Assume for instance a SMP node as depicted

in Figure 7.6.4.a. When the subdomain matrix is very large, the available memory per core

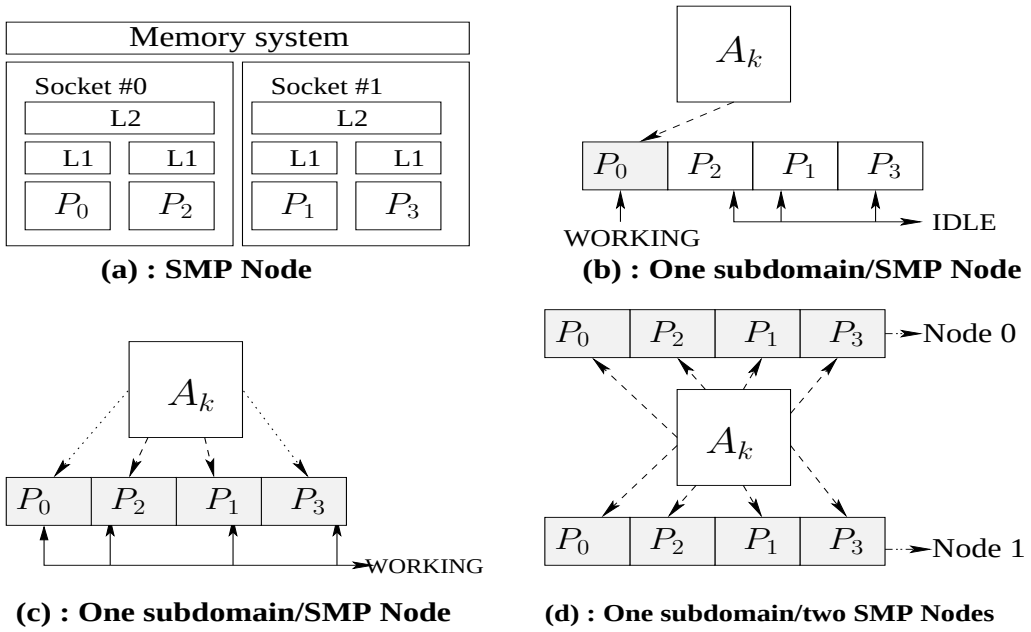


Figure 7.6.4: Distribution of subdomains on a dual-socket dual-CPU SMP node : This figure assume that all processes have equal access to the global memory, however, most of the computers are rather based on a non-uniform access strategy (NUMA architectures).

may not be large enough to store this matrix hence the whole cores of the node should be scheduled for a unique subdomain. In this situation, with the one level of parallelism, only one processor will be working (the processor P_0 for instance in Figure 7.6.4.b). However, if a new level of parallelism is introduced inside each scheduled node, all the processors could contribute to the treatment of the data stored into the memory of the node(Figure 7.6.4.c). Moreover, with this approach, data associated to a particular subdomain could be distributed on more than one SMP node; In this case, the new level of parallelism is defined across all the nodes responsible for a subdomain (Figure 7.6.4.d).

Now, with a parallelism across and inside the subdomains, a discussion on the underlying parallel programming paradigm is of order. In the previous discussion, we assume that we use MPI everywhere, that is MPI for the parallelism across and inside the subdomains communications. The main motivation of this approach is the availability of performant intranode communication channels in many MPI implementation; see for instance [27]. If the intranode MPI processes do not communicate very much and if the exchanged messages are of small size, a good data transfer is obtained from these channels, see for instance the analysis in [108]. In a general case, the speed of the computation will mostly be determined by the per-CPU memory bandwith. The common advise is therefore to use fewer MPI processes than available cores and then to bind processes to separate sockets. Hence, each process use its own memory bus. Now, if the *LU* factorization should be performed, a multithreaded BLAS implementation can be used to keep busy the remaining cores of the socket. This could lead however to weird performance if the threads are not correctly binded to the corresponding cores. Indeed, although current MPI implementations provide interfaces to bind MPI processes to cores, it is not straightforward from the end-user view to manage at the same time a process-to-core and thread-to-core binding. We thus stick in this work to MPI everywhere or threads everywhere inside the subdomains' computations.

7.6.5 Illustration of two levels of parallelism with multiplicative Schwarz

In Table 7.6, we give the benefits of these two levels of parallelism on a public matrix *ATMOSMODL* taken from the UFL collection [39]. The size is 1,489,752 and the number of nonzero entries is 10,319,760. The solution is computed with the Newton basis GMRES preconditioned by the multiplicative Schwarz preconditioner. The size of the basis is 16 and the relative residual norm at the convergence is 10^{-09}. We give the CPU time of the most-consuming time steps in this algorithm, that is the time to perform the numerical factorization of the subdomain matrices and the time in the iterative phase. We first assign one subdomain to one computing node and we consider two types of parallelism inside the subdomains/nodes. In the distributed *LU*, we use a parallel version of the *LU* factorization as implemented in the MUMPS [4] package; hence *Pr* indicates the number of MPI processes assigned to a subdomain times the number of subdomains. In the threaded BLAS *LU*, a sequential MUMPS is linked to the multithreaded implementation of the BLAS operations as provided by the GotoBLAS [68] package. In this case, *Pr* gives the number of subdomains times the number of concurrent threads to be used during the calls to BLAS functions. The compute nodes are part of the *parapide* cluster

D	Pr	Distributed *LU*			Threaded BLAS *LU*		
		Setup	Solve	Time/Iter	Setup	Solve	Time/Iter
4	4	78.7	148.2	4.6	78.7	148.2	4.6
	8	55.2	111.7	3.4	45.9	170.7	5.3
	16	30.0	60.8	1.9	29.1	156.1	4.8
	32	21.0	48.6	1.5			
8	8	23.9	107.3	3.3	23.9	107.3	3.3
	16	16.9	79.1	2.4	14.8	146.4	4.5
	32	11.1	43.2	1.3	10.1	141.1	4.4
	64	6.8	33.0	1.0	10.2	140.6	4.3
16	16	7.1	130.1	2.7	7.1	130.1	2.7
	32	4.5	94.9	1.9	4.8	203.2	4.2
	64	3.3	56.56	1.1	3.7	202.6	4.2
	128	2.4	38.9	0.8	3.6	200.9	4.1

Table 7.6: ATMOSMODL N=1,489,752, NNZ=10,319,760: Using two levels of parallelism in the Newton basis GMRES with a multiplicative Schwarz, MUMPS is used as a local solver with GotoBLAS, D: Number of subdomains, Pr: Total Number of processes (MPI processes or MPI+Pthreads processes), setup: time in the factorization phase, solve: Time in the iterative phase, Time/Iter : Average time spent in each iteration.

of the Grid'5000 testbed[†]. Each node is composed of dual-sockets where each socket has a quad-core Intel Nehalem with a frequency at 2.93GHz. As a general observation, we note for all number of subdomains that the CPU time in the setup phase decreases as we add more processes/threads. This is true with the two types of parallelism. With 8 subdomains for instance, the setup time goes from 23 s. with one process/thread per subdomain to 6.8s. and 10.2s respectively with 8 MPI processes and 8 threads per subdomain. Note however that in this last case, the setup does not decrease significantly when the number of active threads is beyond 4 processes. This could be explained by the overhead to create and destroy the threads compared to the real amount of computation. In a general point of view, the overall performance in this phase is explained by the good kernel of computations. Indeed, the numerical factorization involves many calls to BLAS-3 operations. The situation is different in the iterative phase. Now we see that the threaded BLAS gives a poor

[†]https://www.grid5000.fr

performance which is probably due to the low-grained computation in this phase. Indeed the forward and backward substitution with the L and U triangular factors obtained from the previous phase will involve at most BLAS-2 operations. Moreover, the performance of these operations are determined by the size of the small dense matrices in the factors (frontal matrices). It is thus necessary to have a trade-off between the time to create the concurrent threads and the acceleration obtained from these parallelized kernels. With MPI inside the nodes, the tradeoff should be found between the computation and the time to communicate. Since the MPI processes are scheduled inside the same physical node, the communications are not very expensive. We thus observe a substantial acceleration in the iterative phase as we add more processes inside the node. This acceleration is noticeable with all number of subdomains. Although the efficiency of this approach is not very high, it helps to keep busy all the cores inside the subdomains, because these cores would be idle otherwise. Lastly, it is also worth mentioning the availability of direct solvers that fully use the threads model to parallelize the factorization. Spooles [9] and PaStiX [76] are among these categories. Due to the configuration problems, we did not use these solvers in our hybrid approach.

7.7 Improving robustness with deflation

In the previous chapters, we have shown the benefits of deflating the eigenvalues in a GMRES method. The main motivation of these approaches is to reduce the effects of the restarting and in some extent to choose an appropriate restart length. Indeed, it has been proved that in some situations, the convergence of Ritz and Harmonic Ritz values exhibit the potential of a superlinear convergence in the GMRES method [132]. Restarting may thus have the disadvantage to discard the associated Ritz vectors that form the approximate solution before their convergence [28]. A major difficulty in the restarted GMRES is thus how to choose the appropriate restart length m. If it is too small, then a stagnation may occur. When m is too large, the cost of storing the basis vectors may become prohibitive. In the case of the Newton basis GMRES, a large restart length may also produce a non exploitable basis, in terms of stability. On another side, when the Schwarz methods are used as preconditioners for GMRES, the Krylov basis length is also a function of the number of subdomains. With a fixed value of m, the contribution we have done in this part resides in the ability of the proposed algorithms to work well when the number of subdomains increase in the Schwarz preconditioners whereas the restarted GMRES fails to converge. Two strategies have been considered : either by building a second preconditioner for the deflation or by augmenting the Krylov basis. Now with the two strategies to generate the basis, namely the Arnoldi process and the Newton basis process, we thus have four variants of the deflated GMRES.

	Preconditioning	Augmenting
Arnoldi basis	DGMRES	GMRES-E
Newton basis	PGMRES	AGMRES

Table 7.7: Different strategies to build the Krylov basis and to deflate the eigenvalues

We give in the next sections the main algorithmic differences between these methods. The strategy behind DGMRES has been introduced by Erhel *et al* [53] and Burrage and Erhel [28]. A parallel implementation with an adaptive strategy has been described in chapters 4 and 6. A similar approach can be formulated in the case of the Newton basis process. We call this strateggy PGMRES. A short algorithm has been outlined for this

strategy in [51]. We give in section 7.7.1 a brief derivation. GMRES-E has been proposed by Morgan[98]. AGMRES is described in chapter 6. We give in section 7.7.2 a short description and the benefits of using AGMRES with the explicit formulation of the multiplicative Schwarz method.

7.7.1 Deflation by preconditioning

Here a preconditioner M_D is built from some basis U of the invariant subspace associated to the eigenvalues to deflate. Hence, instead of solving the system 7.11, the following system is considered for the right preconditioning:

$$AM^{-1}M_D^{-1}\tilde{x} = b, \qquad x = M^{-1}M_D^{-1}\tilde{x}. \tag{7.19}$$

Here M^{-1} is a Schwarz preconditioner but it can be any type of preconditioning. Starting with $M_D^{-1} = I$ and $U = [\,]$, M_D^{-1} is built along the restart cycles of GMRES as

$$M_D^{-1} = I + U(|\lambda_n|T_r^{-1})U^T, T_r = U^T AM^{-1}U \tag{7.20}$$

At the beginning of GMRES, U contains no vector and $M_D^{-1} = I$. At each restart, U is updated with new values collected from the current cycle of GMRES. r is the size of U at a certain stage and $|\lambda_n|$ is an estimation of the largest eigenvalue of B. So far, the basis U is approximated differently depending on how the Krylov basis is generated.

The Arnoldi process produces orthogonal basis vectors $V_{m+1} = [v_1, v_2, \ldots, v_{m+1}]$ and a Hessenberg matrix H_m at each cycle of GMRES such that

$$BV_m = V_{m+1}\bar{H}_m = V_m H_m + h_{\{m+1,m\}}v_{m+1}e_m^T \tag{7.21}$$

where $B = AM^{-1}M_D^{-1}$. A simple orthogonal projection with the basis V_m produces the following eigenvalue problem

$$H_m y = \lambda y \tag{7.22}$$

$(\lambda, V_m y)$ is thus an approximate eigenpair of B where $y \in \mathbb{R}^m$. Assuming that the Schur form of H_m has been computed with the eigenvalues ordered in increasing order and X is the set of Schur vectors corresponding to the selected eigenvalues to deflate (usually the smallest ones), then $V_m X$ is used to increase the approximate basis U of the invariant subspace associated to the selected eigenvalues.

In the Newton basis process, the orthogonal basis V_m for the m-dimensional subspace is derived such that :

$$BW_m = W_{m+1}\bar{T}_m \tag{7.23}$$

and

$$W_{m+1} = V_{m+1}R_{m+1} \tag{7.24}$$

where the columns of $W_m \in \mathbb{R}^{n\times m}$ span the Krylov subspace $\mathcal{K}_m$. They are generated by a three-term reccurence which produces the coefficients for the tridiagonal matrix $\bar{T}_m \in \mathbb{R}^{(m+1)\times m}$. A dense QR factorization is then performed to give the orthogonal matrix $V_{m+1} \in \mathbb{R}^{n\times(m+1)}$ and the triangular matrix $R_{m+1} \in \mathbb{R}^{(m+1)\times(m+1)}$. A relation similar to that in Equation (7.13) follows :

$$BV_m = V_{m+1}R_{m+1}\bar{T}_m R_m^{-1} \tag{7.25}$$

Multiplying the two terms of the equation by V_m^T, we get

$$V_m^T BV_m = \underbrace{(R_m T_m + R_{m+1}e_{m+1}e_{m+1}^T\bar{T}_m)}_{C_m} R_m^{-1}. \tag{7.26}$$

The standard orthogonal projection technique produces the eigenvalue problem $C_m R_m^{-1} y = \lambda y$ which can be rewritten as a generalized eigenvalue problem

$$C_m g = \lambda R_m g, \quad y = R_m g \tag{7.27}$$

Hence the approximated eigenvectors of B extracted from $span\{V_m\}$ are given by

$$u = V_m y = V_m R_m g = W_m g \tag{7.28}$$

To sum up, estimating the Schur vectors of B requires a generalized eigenvalue problem in the Newton basis process; the basis V_m is not needed, the basis W_m can be used instead. In the Arnoldi process, a standard eigenvalue problem is solved; In the two approaches so far, U is stored and BU should be kept as well to reduce the matrix-vector products.

We call DGMRES $(m, k, rmax)$ the parallel implementation of this strategy of deflation with the Arnoldi process. At each restart cycle of GMRES, k Schur vectors are extracted to augment the basis U. When the size of U reaches $rmax$, there is no more estimation. It is however possible to update the Schur vectors in U by solving a generalized eigenvalue problem [28] but it has a cost. In Chapters 4 and 5, we introduce an adaptive strategy which implements a deflation on demand. It is based on the same approach but the estimation or the update of the Schur vectors are done only if there is a slow convergence in GMRES. This approach has been used to solve large systems from CFD applications.

We call PGMRES the parallel implementation of the deflation by preconditioning in the Newton basis process as just derived. It is not used in practise considering that one of the motivation of the Newton basis process is to limit the synchronizations points. Indeed, the application of the preconditioner M_D^{-1} for the deflation involves the products such as $U^T z$ where the vector z is distributed between all processes, hence the global communications. Using for instance PGMRES with the multiplicative Schwarz will not produce the pipeline parallelism as presented in section 7.6.3. We therefore rely on the augmented basis approach which is presented next.

7.7.2 Deflation by augmenting the basis

In this strategy, the smallest eigenvalues that slow down the convergence of GMRES are approximated from the harmonic Ritz values of the preconditioned matrix and the associated eigenvectors are used to augment the basis. Hence at each cycle of GMRES, the residual vector is minimized by using a projection onto the augmented subspace defined as

$$\mathcal{C}_s = \mathcal{K}_m(B, r_0) + \mathcal{U}_r \tag{7.29}$$

where $\mathcal{U}_r$ is a r-dimensional invariant subspace associated to the eigenvalues that should be deflated and $s = m + r$. Just as in the previous section, the basis of $\mathcal{K}_m(B, r_0)$ can be generated with the Arnoldi process or the Newton basis process; Let W_m be the basis of that subspace then $W_s = \begin{bmatrix} W_m & U \end{bmatrix}$ is the basis of the augmented subspace $\mathcal{C}_s$. Any approximate eigenvector can be used to augment the Krylov basis. It has been noted [34] that the augmented basis approach works much better with harmonic Ritz vectors than regular Ritz vectors. This is because the augmented Krylov subspace has certain properties; in the case of deflation with thick restarting [99], the generated subspace is itself a Krylov subspace.

So far, the estimation of the (harmonic) ritz vectors rely on the following relation

$$BW_s = V_{s+1} \bar{H}_s. \tag{7.30}$$

When the Arnoldi process is used, the whole basis W_s need not be saved since $W_m = V_m$, only U and V_{s+1} are needed. In the Newton basis process however, W_s should be stored. Now, using a projection technique onto the subspace $span\{BW_s\}$, one gets

$$(BW_s)^T(B - \lambda I)W_s y = 0 \tag{7.31}$$

which produces the following generalized eigenvalue problem

$$\bar{H}_s^T \bar{H}_s y = \lambda \bar{H}_s^T V_{s+1} W_s y. \tag{7.32}$$

In GMRES-E(m, k) proposed by Morgan [98], k vectors are extracted from the augmented subspace $span\{W_s\}$. These vectors give better and better approximation of the harmonic Ritz vectors extracted from the previous restart cycle of GMRES. The basis is built with the Arnoldi process. In AGMRES(m, k, l) as proposed in chapter 6, k vectors are still computed at each restart but an adaptive strategy is used to increase k by l if the convergence rate is not 'good enough'. Compared to GMRES-E(m, k), AGMRES(m, k, l) will thus require more memory to store the increasing basis W_s. Nevertheless, AGMRES benefits from a better parallelism during the generation of the basis vectors through the Newton basis process. Moreover, with the adaptive strategy, the convergence rate is less sensitive to the basis length, particularly when this basis length is fixed and the number of subdomains in the Schwarz preconditioners increases. We provide a brief illustration in the next section. Another benefit of formulating the deflation with the Newton basis process is that the pipeline parallelism is kept when the multiplicative Schwarz preconditioner is used. We briefly give as well a benefit of this approach in the next section.

7.7.3 Benefits of the deflation in GMRES with Schwarz preconditioners

We consider the test case RM07R from the FLUOREM matrix collection (see Appendix A. We first compare DGMRES and AGMRES over GMRES in terms of the numerical convergence. The restricted additive Schwarz preconditioner is used with a LU factorization of subdomain matrices.

Figure 7.7.1 presents the convergence history of GMRES where a restart length m is fixed at 32 and the number of subdomains is 16. In DGMRES$(m, k, rmax)$, k Schur vectors corresponding to the smallest eigenvalues of B are extracted at each restart. $rmax$ gives the size of the basis U at the convergence, that is the number of vectors extracted so far. This is different in AGMRES(m, k) where k Schur vectors are computed and updated at each restart. Hence the number of vectors extracted are different in the two approaches. This may explain the difference in the number of GMRES iterations as shown in Figure 7.7.1. However, when k is increased in AGMRES, the convergence rate is greatly improved. The general observation is still that the two strategies perform better than the approach with no deflation. This is more noticeable when the number of subdomains increases.

We consider indeed the same test case with 32 subdomains in the restricted additive Schwarz preconditioner. Now, an adaptive strategy is used in DGMRES and AGMRES. From Figure 7.7.1, we have an estimation of the number of iterations when $D = 16$. When increasing D to 32, we want to be able to have the same convergence behavior. We thus use an adaptive strategy in AGMRES and DGMRES. We fix a maximum number of iterations $maxit$ at 1000 and we use two relaxation parameters $smv = 0.2$ and $bgv = 0.4$. As explained in chapters 4 and 6, the adaptive strategy computes the convergence rate between two restart cycles and estimates the remaining number of iterations to reach the desired accuracy. If it is greater than a certain maximum number of iterations $smv \cdot maxit$, then a deflation is carried out. In AGMRES, when this bound is greater than $bgv \cdot maxit$,

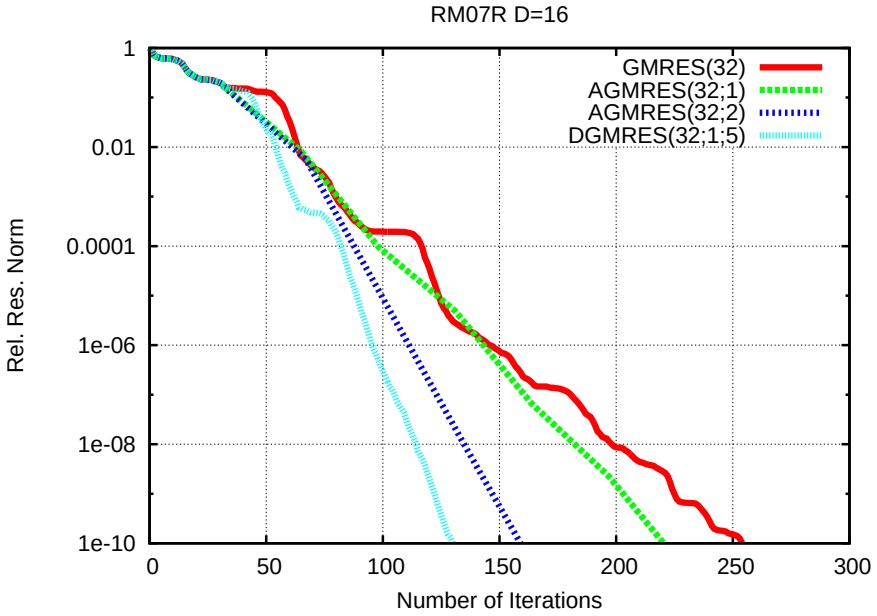


Figure 7.7.1: RM07R, 16 subdomains : Comparing different strategies for deflation in GMRES; AGMRES : GMRES With Newton basis and augmented basis; DGMRES: GMRES with Arnoldi process and deflation as preconditioner

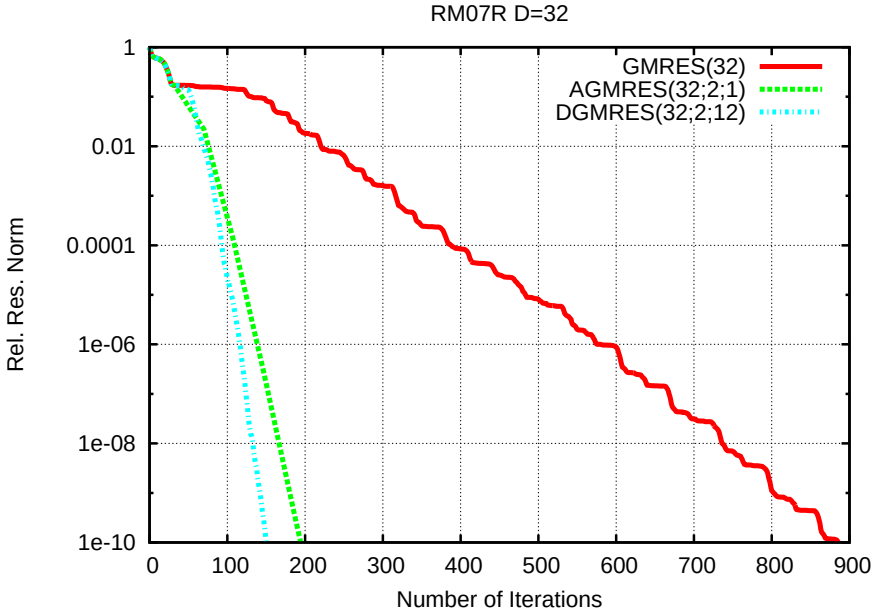


Figure 7.7.2: RM07R, 32 subdomains : Comparing the adaptive deflation in GMRES; AGMRES : GMRES With Newton basis and augmented basis; DGMRES: GMRES with Arnoldi process and deflation as preconditioner

then a slow convergence is declared and the number k of extracted vectors is increased by l. Figure 7.7.2 gives the convergence curve of GMRES with these approaches. It can be seen that the number of iterations in GMRES is almost 900 iterations which is very large compared to that with 16 subdomains. The deflation in DGMRES and AGMRES is very effective to reduce this number of iterations. The adaptive strategy gives close efficiency in the two strategies in terms of the number of iterations. Regarding the memory requirements, DGMRES($m = 32, k = 2, rmax = 12$) stores the basis U and also BU which size are 12. AGMRES($m = 32, k = 2, l = 1$) stores the basis W_s. At the beginning, $k = 2$ but this is increased by l in the adaptive strategy. In this case, k is 6 at the convergence. Hence the size of W_s is $s = m + k = 38$. In practise, AGMRES will store more basis vectors than DGMRES but the parallelism is different in the two methods. AGMRES has the same computational kernel in the generation of the basis vectors and the augmentation step, hence the motivation of using it with the multplicative Schwarz.

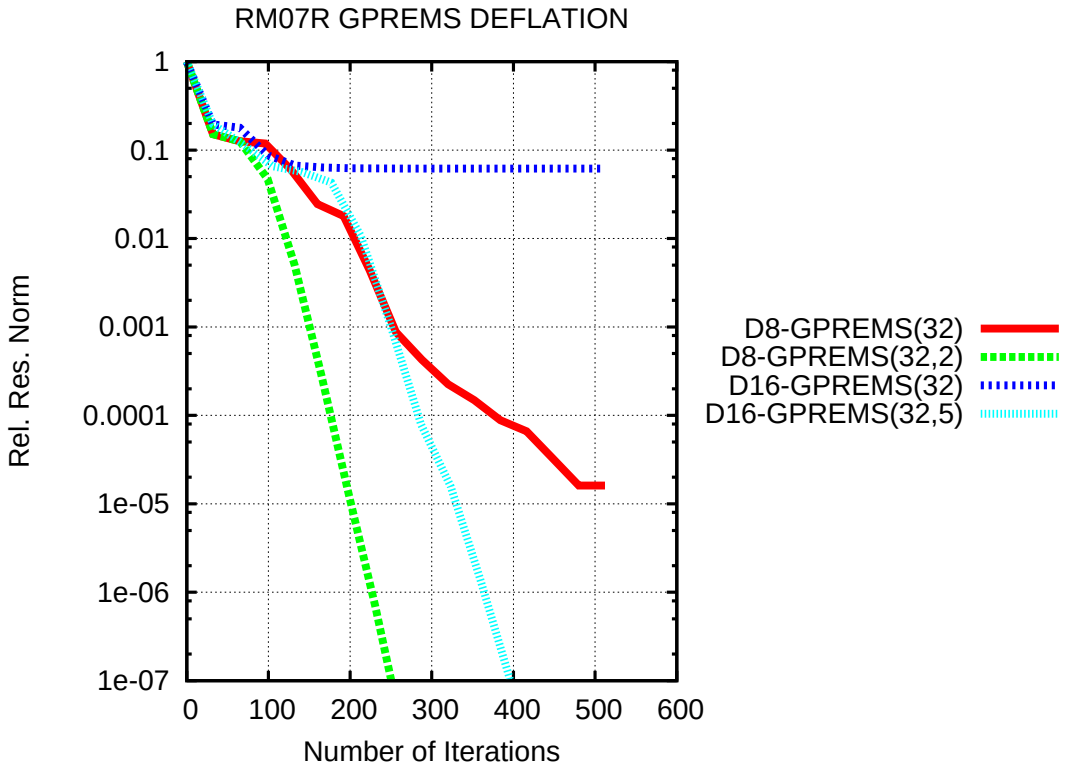


Figure 7.7.3: RM07R, 8 & 16 subdomains: Benefits of the deflation in the GMRES with the Newton basis and the multiplicative Schwarz preconditioner.

We consider indeed the GMRES with multiplicative Schwarz on 8 and 16 subdomains. Unlike the previous section where the partitioning is done with a nested dissection technique, the subdomain matrices are obtained here from a block-diagonal partitioning aprach; as noticed before, DGMRES is not considered here because of the product by U^T which induces global synchronizations points between all the processes sharing the computation of the basis vectors. In Figure 7.7.3 thus, GPREMS(m, k) stands for AGMRES(m, k) with the multiplicative Schwarz. A general notice is that either with 8 and 16 subdomains and without deflation, GPREMS(32) converges slowly. It tends to stagnate when the number of subdomains increases. Deflation helps to keep suitable information at each restart

and to improve convergence rate.

7.8 Conclusion

In this chapter, we have presented the main steps of a hybrid direct/iterative solver based on the Schwarz methods as preconditioners for the Krylov subspace accelerators. Our presentation include the improvements that have been proposed throughout this work to enhance the parallelism and the robustness of the overall algorithm. In terms of robustness, we have proposed to deflate few eigenvalues along the iterations of the hybrid solver. One other approach presented in this chapter is related to the Newton basis GMRES with the explicit formulation of the multiplicative Schwarz preconditioner. In this part, we have shown that using two levels of parallelism improve the efficiency of the method and enable a good usage of the allocated ressources. More work need to be done however to provide a fully robust and efficient general solver. The main weakness of this approach is the partitioning algorithm to obtain the matrix in block diagonal form. An improvement could be to introduce the weights in the adjacency graph and to use the recent technique proposed in [58] to define equally distributed and well-conditioned partitions with minimal overlap.

CHAPTER 8

Conclusion

Les systèmes linéaires suviennent fréquemment dans plusieurs disciplines scientifiques et d'ingénierie. Leur résolution est généralement dans la boucle la plus interne des simulations à grande échelle de plusieurs phénomènes physiques. Notre objectif dans cette thèse était de proposer un ensemble de méthodes robustes pour des systèmes de grande taille provenant de la dynamique des fluides.

De façon globale, les approches que nous avons proposées sont basées sur un schéma hybride direct/iteratif utilisant une décomposition de domaine algébrique. Tel que données par l'étude comparative au chapitre 2, les motivations de ce choix partent de plusieurs constats : les méthodes directes requierent de plus en plus de mémoire lorsque la taille du système croît; de plus leur implémentation parallèle ne permet pas dans certains cas d'obtenir la même robustesse qui fait leur atout. Pour les systèmes de très grande taille, les méthodes itératives sont préferées; elle sont moins sujettes au problème de mémoire mais ont neanmoins besoin de préconditionneurs parallèles pour être robustes; Les préconditionneurs utilisant les techniques de décomposition de domaine permettent donc naturellement d'allier les points positifs des méthodes directes et itératives. Nous nous sommes attelés à ameliorer la robustesse, la consommation mémoire et le parallélisme dans une telle approche.

Nous avons tout d'abord revisité le parallélisme dans une approche hybride utilisant un préconditionneur basé sur Schwarz multiplicatif. Nous avons proposé de définir deux niveaux d'opérations parallèles dans le but de dissocier le nombre de sous-domaines du nombre total de processeurs à utiliser. En plus des opérations parallèles liées à la méthode itérative globale, nous avons fait appel à un solveur direct parallèle pour des sous-systèmes liés à l'application du préconditionneur. Notre motivation aussi était de mieux utiliser les differentes unités de calcul disponible sur les noeuds SMP (*symmetric multiprocessing*). Ces deux niveaux de parallélisme s'adaptent donc naturellement à ces architectures.

Nous nous sommes ensuite interessés à la convergence de GMRES utilisé comme méthode itérative globale dans le schéma hybride. Le réel challenge dans cette méthode est souvent la sélection d'une valeur appropriée de la taille de la base de Krylov. Nous avons étudié l'effet de ce paramètre sur la convergence et nous avons montré experimentalement qu'il était difficile, lorsque le nombre de sous-domaines varie dans le schéma hybride, d'obtenir une valeur appropriée. Nous avons proposé d'utiliser des techniques adaptatives de déflation pour limiter l'effet de cette taille de la base de Krylov sur la convergence. Les résultats sur plusieurs cas-tests ont montré que ces méthodes de déflation, non seule-

ment réduisent considérablement la mémoire, mais permettent d'obtenir un bon taux de convergence.

Concernant le parallélisme dans l'accélérateur GMRES, nous avons aussi proposés une implémentation qui combine une base de Newton avec la déflation. Nous avons montré, avec des cas-tests de taille variable, que cette implémentation est efficace en terme de temps CPU, et de trafic de données (entre les processeurs et la mémoire et entre les différents processeurs). De plus, l'approche de sous-espace augmentée reprend la robustesse liée aux techniques de déflation et permet à la méthode d'être moins influencée par la taille du redemarrage dans GMRES.

De façon générale, on est partagé entre plusieurs critères de performances lors du choix d'un solveur adequat pour les méthodes de résolution des systèmes linéaires. Les critères qui nous ont le plus interessés ici sont l'efficacité parallèle et la robustesse, et dans une non moindre mesure, la consommation mémoire et le mouvement de données entre differents unités de calcul. Au final, il est important pour nous de noter ici le réel challenge qu'il y a à atteindre ces critères en même temps:

- Le preconditionneur de Schwarz multiplicatif est connu pour sa robustesse par rapport à Schwarz additif; nous avons illustré cet aspect sur certains cas-tests aux chapitres 3 et 7. Cependant, il resssort de l'implémentation parallèle que son efficacité est limitée, eu égard à celle obtenue par une approche additive. Les deux niveaux de parallélisme essaient donc dans un certain sens d'améliorer de réduire cet ecart.

- Les deux méthodes de déflation proposées, en l'occurence AGMRES et DGMRES, sont basées sur deux techniques différentes pour la construction de la base de Krylov. DGMRES utilise une base d'Arnoldi qui est plus stable numériquement que AGMRES qui, elle, utilise une base de Newton. Cependant, AGMRES requiert moins de trafic de données que DGMRES et ceci peut avoir un effet si l'architecture ne permet pas un échange rapide de données (fréquence du bus d'accès mémoire, latence du réseau); L'une ou l'autre approche peut donc être préférée pour un meilleur parallélisme ou une meilleure robustesse.

Dans le developpement des solveurs à usage général, les implémentations proposées fournissent cependant des modules réutilisables destinés à ameliorer la robustesse ou le parallélisme dans d'autres schémas: DGMRES ou AGMRES par exemple peuvent être utilisés comme accélérateurs pour les méthodes basées sur le complément de Schur, ou comme des lisseurs pour les méthodes multigrilles; DGMRES peut être tout aussi utilisé comme solveur pour les sous-systèmes produits par la décomposition de domaine en exploitant le fait que la même matrice est utilisée plusieurs fois dans un système avec plusieurs second membres. Nous avons donc mis un accent particulier à proposer des implémentations dans une suite logicielle largement utilisée, en l'occurence PETSc [18, 19].

Il y a cependant plusieurs améliorations possibles à ce travail: la robustesse du préconditionneur de Schwarz multiplicatif est fortement liée au partitionnement bloc-diagonal produit. Un tel partitionnement peut être amélioré en ajoutant des poids au graphe de la matrice comme dans le cas de PARMETIS décrit au chapitre 5. À partir de ces poids, des critères peuvent donc être définis comme dans [37]. Dans le cas de AGMRES, la robustesse peut être améliorée en générant la base de Newton en plusieurs étapes [79] et en associant la déflation. Nous avons proposé une étape de dérivation de cette combinaison au chapitre 6.

APPENDIX A

Test cases

Name 1	Name 2	Physical Info				General Info	
		Nvar	NPts	Nb Blocs	Total Entries	Size	Explicit Nonzeros
CASE_02	IJ02R	5	1,400	30,140	753,500	7,000	359,223
CASE_04	GT01R	5	1,596	20,622	14,575,246	7,980	430,909
CASE_05	FR02R	7	23,010	297,454	14,575,246	161,070	5,066,996
CASE_07	GC13R	7	33,398	459,920	22,536,080	233,786	11,762,405
CASE_09	VV11R	7	39,585	848,719	41,587,231	277,095	30,000,952
CASE_10	IM07R	5	52,293	1,090,477	27,261,925	261,465	26,872,530
CASE_17	RM07R	7	54,527	1,623,991	79,575,559	381,689	37,464,962
CASE_18	HV15R	7	288,167	8,064,431	395,157,119	2,017,169	283,073,458

Table A.1: FLUOREM matrix collection

Name	Conditioning			Row Density			Values	
	1-Norm	F-Norm	I-Norm	MinNnz	MaxNnz	AvgNnz	Min	Max
CASE_02	8.25e+04	1.28+06	1.46e+05	1	118	51	-4.75e+04	5.13e+04
CASE_04	1.97e+05	5.85e+05	3.90e+05	1	90	53	-1.30e+05	1.30e+05
CASE_05	5.51E+5	3.79E+6	5.43E+5	1	72	31	-1.15E+5	1.89E+5
CASE_07	7.79E+2	3.31E+3	1.28E+3	1	145	50	-2.74E+2	3.87E+2
CASE_09	3.34E+3	5.41E+3	3.59E+3	1	300	108	-2.92E+3	4.81E+2
CASE_10	5.29E+4	7.56E+5	5.45E+4	50	230	102	-3.46E+4	3.58E+4
CASE_17	3.21E+6	6.26E+6	3.29E+6	1	295	98	-8.67E+5	7.25E+5
CASE_18	1.24E+5	1.71E+6	1.70E+5	1	484	140	-6.02E+4	3.85E+4

Table A.2: Additional Characteristics of the FLUOREM matrices; F-Norm: Frobenius Norm; I-Norm: Infinity Norm, MinNnz: Minimum nnz per row, MaxNnz: Maximum nnz per row, AvgNnz: Average nnz per row

BIBLIOGRAPHY

[1] Y. Achdou and F. Nataf, *Low frequency tangential filtering decomposition*, Numerical Linear Algebra with Applications, 14 (2007), pp. 129–147.

[2] E. Agullo, A. Guermouche, and J.-Y. L'Excellent, *A parallel out-of-core multifrontal method: storage of factors on disk and analysis of models for an out-of-core active memory*, Parallel Comput., 34 (2008), pp. 296–317.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[4] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[5] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li, *Analysis and comparison of two general sparse solvers for distributed memory computers*, ACM Transactions on Mathematical Software, 27 (2000), p. 2001.

[6] P. R. Amestoy, I. S. Duff, D. Ruiz, and B. Uçar, *A parallel matrix scaling algorithm*, in High Performance Computing for Computational Science - VECPAR 2008, J. M. Palma, P. R. Amestoy, M. Daydé, M. Mattoso, and J. a. C. Lopes, eds., Berlin, Heidelberg, 2008, Springer-Verlag, pp. 301–313.

[7] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing, 32 (2006), pp. 136–156.

[8] P. R. Amestoy, X. S. Li, and E. G. Ng, *Diagonal Markowitz scheme with local symmetrization*, SIAM J. Matrix Anal. Appl., 29 (2006/07), pp. 228–244 (electronic).

[9] C. Ashcraft and R. Grimes, *Spooles: An object-oriented sparse matrix library*, in Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.

[10] C. Ashcraft and J. W. H. Liu, *Using domain decomposition to find graph bisectors*, BIT, 37 (1997), pp. 506–534. Direct methods, linear algebra in optimization, iterative methods (Toulouse, 1995/1996).

[11] G.-A. Atenekeng-Kahou, *Parallélisation de GMRES préconditionné par une itération de Schwarz multiplicatif*, PhD thesis, University of Rennes 1 and University of Yaounde 1, 2008. ftp://ftp.irisa.fr/techreports/theses/2008/atenekeng.pdf.

[12] G.-A. Atenekeng-Kahou, L. Grigori, and M. Sosonkina, *A partitioning algorithm for block-diagonal matrices with overlap*, Parallel Computing, 34 (2008), pp. 332–344.

[13] G.-A. Atenekeng-Kahou, E. Kamgnia, and B. Philippe, *Parallel implementation of an explicit formulation of the multiplicative schwarz preconditioner*, in CDroms Proceedings of IMACS05, 2005.

[14] ——, *An explicit formulation of the multiplicative Schwarz preconditioner*, Applied Numerical Mathematics, 57 (2007), pp. 1197 – 1213.

[15] S. Aubert, J. Tournier, M. Rochette, J. Blanche, M. NDiaye, S. Melen, M. Till, and P. Ferrand, *Optimisation of a gas mixer using a new parametric flow solver*, in Proceedings of the European Conference on Computational Fluid Dynamics ECCOMAS CFD, Swansea, Wales, UK, The Institute of Mathematics and its Applications, ed., September 2001.

[16] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, *Adaptively preconditioned GMRES algorithms*, SIAM J. Sci. Comput., 20 (1998), pp. 243–269.

[17] Z. Bai, D. Hu, and L. Reichel, *A Newton basis GMRES implementation*, IMA J Numer Anal, 14 (1994), pp. 563–581.

[18] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 3.2.0, Argonne National Laboratory, 2011.

[19] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*, 2011. http://www.mcs.anl.gov/petsc.

[20] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[21] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[22] M. Benzi, A. Frommer, R. Nabben, and D. B. Szyld, *Algebraic theory of multiplicative Schwarz methods*, Numer. Math., 89 (2001), pp. 605–639.

[23] P. E. Bjørstad and O. B. Widlund, *To overlap or not to overlap: a note on a domain decomposition method for elliptic problems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1053–1061.

[24] C. Brezinski and H. Sadok, *Lanczos-type algorithms for solving systems of linear equations*, Appl. Numer. Math., 11 (1993), pp. 443–473.

[25] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second ed., 2000.

[26] R. A. Brualdi, *Introductory Combinatorics*, Prentice Hall, 5th ed., 2009.

[27] D. Buntinas, G. Mercier, and W. Gropp, *Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem*, Parallel Computing, 33 (2007), pp. 634 – 644. Selected Papers from EuroPVM/MPI 2006.

[28] K. Burrage and J. Erhel, *On the performance of various adaptive preconditioned GMRES strategies*, Numerical Linear Algebra with Applications, 5 (1998), pp. 101–121.

[29] X.-C. Cai and Y. Saad, *Overlapping domain decomposition algorithms for general sparse matrices*, Numerical Linear Algebra with Applications, 3 (1996), pp. 221–237.

[30] X.-C. Cai and M. Sarkis, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 21 (1999), pp. 792–797 (electronic).

[31] E. Canot, C. De Dieuleveult, and J. Erhel, *A parallel software for a saltwater intrusion problem*, vol. 33 of Parallel Computing: Current and Future Issues of High-End Computing, NIC, 2006, pp. 399–406.

[32] L. Carvalho, L. Giraud, and G. Meurant, *Local preconditioners for two-level non-overlapping domain decomposition methods*, Numerical Linear Algebra with Applications, 8 (2001), pp. 207–227.

[33] U. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Syst., 10 (1999), pp. 673–693.

[34] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numerical Linear Algebra with Applications, 4 (1997), pp. 43–66.

[35] C. Chevalier and F. Pellegrini, *PT-Scotch: a tool for efficient parallel graph ordering*, Parallel Comput., 34 (2008), pp. 318–331.

[36] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[37] A. F. David Fritzsche and D. B. Szyld, *Overlapping blocks by growing a partition with applications to preconditioning*, Tech. Rep. 10-07-26, Department of Mathematics, Temple University, July 2010.

[38] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, *Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 377–380.

[39] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011).

[40] E. de Sturler, *A parallel variant of GMRES(m)*, in Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics, Dublin, Ireland, J. J. H. Miller and R. Vichnevetsky, eds., Criterion Press, 1991.

[41] E. De Sturler, *Iterative Methods on Distributed Memory Computers*, PhD thesis, Delft University of Technology, Delft, The Netherlands, October 1994.

[42] J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM journal on Scientific Computing (to appear), (2011).

[43] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[44] T. Dufaud and D. Tromeur-Dervout, *Aitken's acceleration of the restricted additive schwarz preconditioning using coarse approximations on the interface*, Comptes Rendus Mathematique, 348 (2010), pp. 821 – 824.

[45] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices*, Monographs on Numerical Analysis, The Clarendon Press Oxford University Press, New York, second ed., 1989. Oxford Science Publications.

[46] I. S. Duff and J. K. Reid, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 633–641.

[47] E. Efstathiou and M. Gander, *Why restricted additive schwarz converges faster than additive schwarz*, BIT Numerical Mathematics, 43 (2003), pp. 945–959. 10.1023/B:BITN.0000014563.33622.1d.

[48] M. Eiermann, O. G. Ernst, and O. Schneider, *Analysis of acceleration strategies for restarted minimal residual methods*, J. Comput. Appl. Math., 123 (2000), pp. 261–292. Numerical analysis 2000, Vol. III. Linear algebra.

[49] H. C. Elman, O. G. Ernst, and D. P. O'Leary, *A multigrid method enhanced by Krylov subspace iteration for discrete Helmhotz equations*, SIAM J. Sci. Comput., 23 (2001), pp. 1291–1315 (electronic).

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

[51] ——, *A parallel preconditioned GMRES algorithm for sparse matrices*, in The mathematics of numerical analysis, vol. 32 of Lectures in Appl. Math., AMS, Providence, RI, 1996, pp. 345–355.

[52] J. Erhel, *Some properties of Krylov projection methods for large linear systems*, vol. 3 of Computational Technology Reviews, Saxe-Coburg Publications, 2011, pp. 41–70.

[53] J. Erhel, K. Burrage, and B. Pohl, *Restarted GMRES preconditioned by deflation*, Journal of Computational and Applied Mathematics, 69 (1996), pp. 303–318.

[54] R. Falgout and U. Yang, *Hypre: a library of high performance preconditioners.*, in Lectures Notes in Computer Science, vol. 2331, Springer-Verlag, 2002, pp. 632–641.

[55] FLUOREM, *The FLUOREM matrix collection*, 2009. LIB0721 2.0 / FP-SA http://www.fluorem.com.

[56] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.

[57] R. W. Freund and N. M. Nachtigal, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.

[58] D. Fritzsche, A. Frommer, and D. B. Szyld, *Extensions of certain graph-based algorithms for preconditioning*, SIAM J. Sci. Comput., 29 (2007), pp. 2144–2161 (electronic).

[59] A. Frommer and D. B. Szyld, *Weighted max norms, splittings, and overlapping additive Schwarz iterations*, Numer. Math., 83 (1999), pp. 259–278.

[60] J. Gaidamour, *Conception d'un solveur linéaire creux paralléle hybride direct-itératif*, PhD thesis, Universite de Bordeaux I, 2009.

[61] K. Gallivan, A. Sameh, and Z. Zlatev, *A parallel hybrid sparse linear system solver*, Computing Systems in Engineering, 1 (1990), pp. 183 – 195. Computational Technology for Flight Vehicles.

[62] M. J. Gander, *Optimized Schwarz methods*, SIAM J. Numer. Anal., 44 (2006), pp. 699–731 (electronic).

[63] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363. Collection of articles dedicated to the memory of George E. Forsythe.

[64] L. Giraud, S. Gratton, X. Pinel, and X. Vasseur, *Flexible GMRES with deflated restarting*, SIAM Journal on Scientific Computing, 32 (2010), pp. 1858–1878.

[65] L. Giraud, A. Haidar, and S. Pralet, *Using multiple levels of parallelism to enhance the performance of domain decomposition solvers*, Parallel Computing, In Press (2010).

[66] L. Giraud, A. Haidar, and L. T. Watson, *Parallel scalability study of hybrid preconditioners in three dimensions*, Parallel Comput., 34 (2008), pp. 363–379.

[67] H. H. Goldstine, *A history of numerical analysis from the 16th through the 19th century*, Springer-Verlag, New York, 1977. Studies in the History of Mathematics and Physical Sciences, Vol. 2.

[68] K. Goto and R. Van De Geijn, *High-performance implementation of the level-3 BLAS*, ACM Trans. Math. Softw., 35 (2008), pp. 4:1–4:14.

[69] L. Grigori, E. G. Boman, S. Donfack, and T. A. Davis, *Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization*, SIAM J. Sci. Comput., 32 (2010), pp. 3426–3446.

[70] L. Grigori, J. W. Demmel, and X. S. Li, *Parallel symbolic factorization for sparse LU with static pivoting*, SIAM J. Sci. Comput., 29 (2007), pp. 1289–1314 (electronic).

[71] L. Grigori, P. Kumar, F. Nataf, and K. Wang, *A class of multilevel parallel preconditioning strategies*, Rapport de recherche RR-7410, INRIA, Oct. 2010.

[72] A. Gupta, *Recent advances in direct methods for solving unsymmetric sparse systems of linear equations*, ACM Trans. Math. Software, 28 (2002), pp. 301–324.

[73] A. Haidar, *On the parallel scalability of hybrid linear solvers for large 3D problems*, PhD thesis, Institut National Polytechnique de Toulouse, 2008.

[74] B. Hendrickson and T. G. Kolda, *Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing*, SIAM J. Sci. Comput., 21 (2000), pp. 2048–2072.

[75] B. Hendrickson and E. Rothberg, *Effective sparse matrix ordering: just around the BEND*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997), Philadelphia, PA, 1997, SIAM, p. 8 pp. (electronic).

[76] P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and Y. Saad, *High Performance Complete and Incomplete Factorizations for Very Large Sparse Systems by using Scotch and PaStiX softwares*, in Eleventh SIAM Conference on Parallel Processing for Scientific Computing, 2004.

[77] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos Project*, ACM Trans. Math. Software, 31 (2005), pp. 397–423.

[78] M. Hochbruck and C. Lubich, *Error analysis of Krylov methods in a nutshell*, SIAM J. Sci. Comput., 19 (1998), pp. 695–701.

[79] M. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD thesis UCB/EECS-2010-37, UC Berkeley, 2010.

[80] D. Hysom and A. Pothen, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM Journal on Scientific Computing, 22 (2000), pp. 2194–2215.

[81] W. Jalby and B. Philippe, *Stability analysis and improvement of the block Gram-Schmidt algorithm*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1058–1073.

[82] A. Jameson, W. Schmidt, and E. Turkel, *Numerical solutions of the euler equations by finite volume methods using runge-kutta time-stepping schemes*, AIAA Paper 81-1259, (1981).

[83] W. Joubert and G. Carey, *Parallelizable restarted iterative methods for nonsymmetric linear systems. part II: Parallel implementation*, Intern. J. Computer Math., 44 (1992), pp. 269–290.

[84] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, J. Parallel Distrib. Comput., 48 (1998), pp. 96–129.

[85] ——, *Parallel multilevel k-way partitioning scheme for irregular graphs*, SIAM Rev., 41 (1999), pp. 278–300 (electronic).

[86] G. Karypis, K. Schloegel, and V. Kumar, *ParMETIS: Parallel graph partitioning and sparse matrix ordering library*, tech. rep., Technical report, University of Minnesota, Department of Computer Science and Engineering, 1997.

[87] S. A. Kharchenko and A. Y. Yeremin, *Eigenvalue translation based preconditioners for the* GMRES(k) *method*, Numer. Linear Algebra Appl., 2 (1995), pp. 51–77.

[88] S. Kim and A. Chronopoulos, *An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices*, International Journal of High Performance Computing Applications, 6 (1992), pp. 407–420.

[89] P. Kumar, L. Grigori, F. Nataf, and Q. Niu, *Combinative preconditioning based on Relaxed Nested Factorization and Tangential Filtering preconditioner*, Research Report RR-6955, INRIA, 2009.

[90] C. Lanczos, *Solution of systems of linear equations by minimized-iterations*, J. Research Nat. Bur. Standards, 49 (1952), pp. 33–53.

[91] X. S. Li and J. W. Demmel, *SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Mathematical Software, 29 (2003), pp. 110–140.

[92] X. S. Li, M. Shao, I. Yamazaki, and E. G. Ng, *Factorization-based sparse solvers and preconditioners*, Journal of Physics: Conference Series, 180 (2009), p. 012015.

[93] Z. Li, Y. Saad, and M. Sosonkina, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Numer. Linear Algebra Appl., 10 (2003), pp. 485–509. Preconditioning, 2001 (Tahoe City, CA).

[94] J. W. H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[95] Y. Liu, Cambridge University Press, 2009.

[96] G. Meurant, *Computer solution of large linear systems*, vol. 28 of Studies in Mathematics and its Applications, North-Holland Publishing Co., Amsterdam, 1999.

[97] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, *Minimizing communication in sparse matrix solvers*, in SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York, NY, USA, 2009, ACM, pp. 1–12.

[98] R. B. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1154–1171.

[99] ——, *GMRES with deflated restarting*, SIAM J. Sci. Comput., 24 (2002), pp. 20–37 (electronic).

[100] ——, *Restarted block-GMRES with deflation of eigenvalues*, Appl. Numer. Math., 54 (2005), pp. 222–236.

[101] D. Nuentsa Wakam and G.-A. Atenekeng Kahou, *Parallel GMRES with a multiplicative Schwarz preconditioner*, ARIMA Rev. Afr. Rech. Inform. Math. Appl. (to appear), (2010). Also Research report INRIA RR-7342.

[102] D. Nuentsa Wakam, J. Erhel, and É. Canot, *Parallélisme à deux niveaux dans GMRES avec un préconditionneur Schwarz multiplicatif*, in CARI 2010 Actes du 10ème Colloque Africain sur la Recherche en Informatique et Mathématiques Appliquées, É. Badouel, A. Sbihi, and I. Lopko, eds., Yamoussoukro, Côte D'Ivoire, 10 2010, INRIA, pp. 189–196.

[103] D. Nuentsa Wakam, J. Erhel, E. Canot, and G.-A. Atenekeng Kahou, *A comparative study of some distributed linear solvers on systems arising from fluid dynamics simulations*, in Parallel Computing: From Multicores and GPU's to Petascale, vol. 19 of Advances in Parallel Computing, IOS Press, 2010, pp. 51–58.

[104] D. Nuentsa Wakam, J. Erhel, and W. D. Gropp, *Parallel adaptive deflated GMRES*, in Proceedings of DD'20, UC San Diego, in revision, 2011.

[105] D. Nuentsa Wakam and F. Pacull, *Memory efficient and robust hybrid algebraic solvers for large CFD linear systems*, Computer and Fluids, submitted (2011). special issue of ParCFD2011.

[106] F. Pacull, S. Aubert, and M. Buisson, *Study of ILU factorization for schwarz preconditioners with application to computational fluid dynamics*, in Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, Stirlingshire, UK, 2011.

[107] B. Philippe and L. Reichel, *On the generation of Krylov subspace bases*, Applied Numerical Mathematics, In Press (2011).

[108] R. Rabenseifner, G. Hager, and G. Jost, *Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes*, Parallel, Distributed, and Network-Based Processing, Euromicro Conference on, 0 (2009), pp. 427–436.

[109] L. Reichel, *Newton interpolation at Leja points*, BIT Numerical Mathematics, 30 (1990), pp. 332–346. 10.1007/BF02017352.

[110] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid methods, vol. 3 of Frontiers Appl. Math., SIAM, Philadelphia, PA, 1987, pp. 73–130.

[111] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105–126.

[112] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM Journal on Scientific Computing, 14 (1993), pp. 461–469.

[113] Y. Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, second ed., 2003.

[114] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[115] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, SIAM Journal on Scientific Computing, 21 (1999), pp. 1337–1356.

[116] A. Sameh, *Solving the linear leastsquares problem on a linear array of processors*, in High Speed Computer and Algorithm Organization, D. L. D. Kuck and A. Sameh, eds., Academic Press, 1977, pp. 207–228.

[117] H. A. Schwarz, *Gesammelte mathematische Abhandlungen. Band I, II*, Chelsea Publishing Co., Bronx, N.Y., 1972. Nachdruck in einem Band der Auflage von 1890.

[118] R. B. Sidje, *Alternatives for parallel Krylov subspace basis computation*, Numerical Linear Algebra with Applications, 4 (1997), pp. 305–331.

[119] R. B. Sidje and B. Philippe, *parallel krylov subspace basis computation*, in CARI'94, 2ème colloque africain sur la recherche en Informatique, 1994.

[120] V. Simoncini, *On a non-stagnation condition for GMRES and application to saddle point matrices*, Electron. Trans. Numer. Anal., 37 (2010), pp. 202–213.

[121] V. Simoncini and D. B. Szyld, *Recent computational developments in Krylov subspace methods for linear systems*, Numer. Linear Algebra Appl., 14 (2007), pp. 1–59.

[122] ——, *New conditions for non-stagnation of minimal residual methods*, Numer. Math., 109 (2008), pp. 477–487.

[123] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[124] M. Sosonkina, L. T. Watson, R. K. Kapania, and H. F. Walker, *A new adaptive GMRES algorithm for achieving high accuracy*, Numer. Linear Algebra Appl., 5 (1998), pp. 275–297.

[125] I. Souopgui, *Parallélisation d'une double recurrence dans GMRES*, Master's thesis, University of Yaounde 1, 2005.

[126] B. Sécher, M. Belliard, and C. Calvin, *Numerical platon: A unified linear equation solver interface by cea for solving open foe scientific applications*, Nuclear Engineering and Design, 239 (2009), pp. 87 – 95.

[127] A. Toselli and O. Widlund, *Domain decomposition methods—algorithms and theory*, vol. 34 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005.

[128] D. Tromeur-Dervout, *Aitken-Schwarz method: acceleration of the convergence of the Schwarz method*, in Domain decomposition methods: theory and applications, vol. 25 of GAKUTO Internat. Ser. Math. Sci. Appl., Gakkōtosho, Tokyo, 2006, pp. 37–64.

[129] B. Uçar and C. Aykanat, *Partitioning sparse matrices for parallel preconditioned iterative methods*, SIAM J. Sci. Comput., 29 (2007), pp. 1683–1709 (electronic).

[130] H. A. van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.

[131] H. A. van der Vorst, *Iterative Krylov methods for large linear systems*, vol. 13 of Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, 2009. Reprint of the 2003 original.

[132] H. A. van der Vorst and C. Vuik, *The superlinear convergence behaviour of GMRES*, J. Comput. Appl. Math., 48 (1993), pp. 327–341.

[133] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 152–163.

[134] D. C. Wilcox, *Reassessment of the scale-determining equation for advanced turbulence models*, AIAA Journal, 26 (1988), pp. 1299–1310.

[135] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 77–79.