LABORATOIRE D'ANALYSE NUMERIQUE ET D'OPTIMISATION

Numéro d'ordre : 2460

THÈSE

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR EN MATHÉMATIQUES

par

Azeddine ESSAI

MÉTHODE HYBRIDE PARALLÈLE HÉTÉROGÈNE ET MÉTHODES PONDÉRÉES POUR LA RÉSOLUTION DES SYSTÈMES LINÉAIRES

soutenue le 21 Janvier 1999, devant la commission d'examen

Membres du jury

Président	:	C. BREZINSKI	Professeur, Université de Lille I
Rapporteurs	:	G. ALLAIRE	Professeur, Université de Paris 6
		B. PHILIPPE	Directeur de recherche IRISA
Examinateurs	:	S. PETITON	Professeur, Université de Lille I
		H. SADOK	Professeur, Université du Littoral

Laboratoire d'Analyse Numérique et d'Optimisation, UFR IEEA - M3, USTL,
F-59655 Villeneuve d'Ascq Cedex, France.

À ma famille
et à tous ceux que j'aime.

Remerciments

Je tiens tout d'abord à remercier vivement Mr Claude BREZINSKI, Professeur à l'Université des Sciences et Technologies de Lille et Directeur du Laboratoire d'Analyse Numérique et d'optimisation, pour m'avoir accueilli dans son équipe et d'avoir accepté de diriger ma thèse.

Je remercie également Mr Serge PETITON, Professeur à l'Université des Sciences et Technologies de Lille, d'avoir accepté d'être co-directeur de ma thèse. Il m'a apporté de nombreux conseils qui m'ont permis d'approfondir ma reflexion sur le coté parallélisme de ma thèse.

Je tiens à remercier Mr Grégoire ALLAIRE, Professeur à l'Université de Paris 6, ainsi que Mr Bernard PHILIPPE, Directeur de recherche IRISA, d'avoir accepté d'être rapporteurs de ma thèse.

Je voudrais aussi remercier Mr Hassane SADOK, Professeur à l'Université du littoral, pour l'intérêt qu'il a manifesté en acceptant d'examiner ce travail.

Je tiens à remercier Mr Guy BERGÈRE de l'équipe MAP (Méthodologie et Algorithmique Parallèles), pour sa collaboration dans l'étude de la méthode hybride parallèle que nous avons programmée ensemble. Je le remercie également pour sa gentillesse, sa patience et les discussions fructueuses que nous avons échangées.

Je remercie particulièrement mon ami Mohammed HEYOUNI, Maître de Conférences à l'Université du Littoral, qui m'a toujours épaulé. Je le remercie encore pour la relecture de ma thèse et ses precieux conseils.

Ma gratitude s'exprime aussi envers Nathalie REVOL et Bernard GERMAIN-BONNE d'avoir lu si soigneusement ce manuscrit.

Je remercie l'Etablissement Technique Central de l'Armement d'avoir mis à notre disposition Guy et moi la machine CM-5, pour nos expérimentations.

Je souhaite également remercier toutes les personnes des Laboratoires ANO et LIFL qui m'ont aidé, Carloine LE CALVEZ avec qui j'ai partagé le même bureau, pour les discussions fructueuses que nous avons échangées, Bernhard Beckermann pour son aide et les discussions constructives toujours avec la même rigueur scientifique, El Hassan AYACHOUR, Hassan ZOURHLAL, Ahmed ELGUENNOUNI, Driss BENCHIBOUN, Abderrahim MESSAOUDI, Khalide JBILOU pour leur amitié et leur soutient moral.

Je remercie aussi Mr Edouard DUSZYNSKI, chef du département informatique à l'IUT "A" de Lille, et Mr Pierre DESPLANQUES, directeur du département IMA de l'EUDIL, pour m'avoir bien accueilli dans leurs départements lors de l'exercice de mes fonctions d'ATER.

Je remercie également tous mes amis, Said, Yassine, Abderrahim, Fouad, Yasser, Mohammed^n, Abdellah, M'hammed, Jaafar, Jamaï, Hussein, Rabih et tous les autres.

Finalement, je veux remercier vivement mes parents, mes frères, mes soeurs, ma femme et ma belle famille qui ont contribué à mon succès par leur encouragement moral et materiel. Pour cela, je leur dédie cette thèse.

Résumé

Dans ce travail, nous avons décrit et étudié la parallélisation du procédé de Gram-Schmidt. Les deux versions de ce procédé (classique et modifiée) ont été implantées en data-parallèle sur des machines massivement parallèles. Nous avons aussi étudié la parallélisation du processus d'Arnoldi qui n'est autre que le processus de Gram-Schmidt appliqué à une famille de vecteurs générateurs de sous-espaces de Krylov.

Nous avons introduit et étudié une nouvelle méthode hybride parallèle, nommée GMRES(m_1)/LS(k,l)-Arnoldi(m_2). Cette méthode permet de résoudre des systèmes linéaires non symétriques creux et de grande taille. Elle combine trois méthodes de Krylov, la méthode GMRES, la méthode Moindres Carrés (LS(k,l)) et la méthode d'Arnoldi (pour le calcul des valeurs propres). Cette méthode hybride nous permet d'accélérer la convergence et d'augmenter le degré de parallélisme à gros grain dans la méthode GMRES. L'hétérogénéité de cette méthode à la fois du point de vue algorithmique et du point de vue parallélisme, nous permet de l'implanter sur des réseaux hétérogènes composés de machines parallèles et séquentielles.

Dans la dernière partie de cette thèse, nous avons introduit les méthodes FOM et GMRES pondérées pour la résolution des systèmes linéaires. Ces méthodes sont obtenues à partir des méthodes FOM et GMRES en changeant le produit scalaire euclidien par un autre associé à une matrice diagonale. Ce changement a pour but d'accélérer la convergence en essayant de faire tendre les composantes du résidu vers zéro d'une manière uniforme. Nous avons également établi des relations entre ces méthodes et les méthodes d'origine.

Mots clefs

Méthodes de Krylov, systèmes linéaires, matrices creuses, parallélisme, processus de Gram-Schmidt, processus d'Arnoldi, méthodes hybrides, méthodes pondérées.

Abstract

In this work, we describe and study the parallelization of the Gram-Schmidt process. The two versions of this process (classic and modified) were implemented in data-parallel on massively parallel machines. We have also studied the parallelization of the Arnoldi process, which is the Gram-Schmidt process applied to a family of vectors generating Krylov subspaces.

We have introduced and studied a new hybrid parallel method, named GMRES(m_1)/LS(k,l)-Arnoldi(m_2). This method allows us to solve nonsymmetric, sparse and large linear systems, it combines three methods, the GMRES method, the Least Squares method (LS(k,l)) and the Arnoldi method (for solving eigenvalue problems). This hybrid method accelerates the convergence and increases coarse-grain parallelism degree in the GMRES method. The heterogeneousness of this method, at the same time, in the algorithmic point of view and in the parallelism point of view, allows us to implement it on a heterogeneous network composed of parallel and sequential machines.

In the last part of this thesis, we have introduced the weighted FOM and GMRES methods for solving linear systems. These methods are obtained from FOM and GMRES methods using, instead of the euclidean inner product, a different one associated to a diagonal matrix. The aim of this change is accelerating the convergence by trying to vanish the components of the residual in an uniform way. We have also established relationships between these methods and the initial ones.

Key words

Krylov methods, linear systems, sparse matrices, parallelism, Gram-Schmidt process, Arnoldi process, hybrid methods, weighted methods.

Table des matières

Introduction générale

La résolution des systèmes linéaires est l'un des problèmes que rencontrent fréquement les scientifiques dans de nombreuses disciplines. Citons par exemple: la mécanique des fluides, l'électromagnétisme, les prévisions météorologiques et la pollution atmosphérique. En général, ces phénomènes physiques sont modélisés mathématiquement à l'aide d'équations aux dérivées partielles (EDP). La discrétisation de ces équations aux dérivées partielles aboutit à résoudre un système linéaire représenté par l'équation algébrique

$$Ax = b. \tag{0.1}$$

Nous supposons dans la suite de ce mémoire que la matrice A est réelle de taille n, inversible et que $b \in \mathbb{R}^n$.

Pour résoudre de tels systèmes deux grandes classes de méthodes existent: les méthodes directes et les méthodes itératives.

Les méthodes directes procèdent à la factorisation de la matrice A sous forme d'un produit de facteurs. Ces facteurs sont en général plus simples à manipuler que la matrice A elle même. Les factorisations LU et QR, sont les plus souvent utilisées et les plus connues. Ces factorisations transforment le système linéaire initial en un système triangulaire qui est, bien sur, plus facile à résoudre. Ces méthodes sont très coûteuses du point de vue nombre d'opérations et du point de vue place mémoire. En arithmétique exacte, ces méthodes délivrent la solution exacte du système au bout de n itérations. Cependant, en précision finie, elles risquent de ne pas trouver la solution exacte du système et ce à cause des erreurs d'arrondi.

Les méthodes itératives génèrent, à partir d'une solution de départ x_0, une suite $\{x_k\}$ d'estimations qui tend vers x^* la solution exacte du système (0.1). Le coût de ces méthodes du point de vue place mémoire et nombre d'opérations, est en général, plus faible que celui des méthodes directes.

Pour certains problèmes, la précision imposée sur le calcul d'une approximation

de la solution exacte du système n'est pas très grande, ce qui permet de résoudre ces problèmes avec un faible coût algorithmique à l'aide des méthodes itératives.

Souvent la matrice A du système (0.1) est de grande taille et est creuse (i.e. A contient peu d'éléments non nuls).

En général, les méthodes itératives n'utilisent la matrice A que pour effectuer des produits matrice-vecteur. Pour cette raison, ces méthodes profitent de la structure creuse des matrices en les stockant à l'aide de formats économiques et adaptés à ce genre de produit (CSR, Ellpack-Itpack, SGP,...). Notons que cet avantage est propre aux méthodes itératives, ce qui les rend plus attractives que les méthodes directes [11] puisqu'elles sont capables de résoudre des problèmes de grande taille.

La classe des méthodes itératives contient une sous-classe de méthodes basées sur la décomposition de la matrice A sous la forme $A = M - N$ où M est une matrice non singulière et facile à inverser. Ces méthodes sont appelées méthodes stationnaires ou méthodes de relaxation car les itérés de ces méthodes vérifient la relation $x_{k+1} = M^{-1}Nx_k + M^{-1}b$. Parmi ces méthodes citons Jacobi, Gauss-Seidel et SOR (*Successive OverRelaxation*) [74, 76, 30].

Une autre sous-classe des méthodes itératives est la classe des méthodes de Krylov. Les itérés de ces méthodes s'écrivent sous la forme $x_k = x_0 + P(A)r_0$ où x_0 est une solution de départ, $r_0 = b - Ax_0$ le résidu associé à x_0 et P un polynôme de degré $k-1$. Chaque méthode de Krylov construit implicitement le polynôme P de façon à faire converger la suite $\{x_k\}$ vers x^*.

Les méthodes de Krylov sont nombreuses. Parmi les plus connues, nous citons.

- La méthode CG (*Conjugate Gradient*) [31] due à Hestenes et Stiefel utilisée dans le cas des matrices symétriques définies positives.
- La méthode BCG (*Bi-Conjugate gradient*) [39] due à Lanczos. Cette méthode est une généralisation de le méthode CG pour les matrices non symétriques.
- La méthode FOM (*Full Orthogonalization Method*) [61] due à Saad et est basée sur le processus d'Arnoldi. Notons que dans cette méthode peut survenir des divisions par zéro (*Breakdown*).
- La méthode GMRES (*Generalized Minimum RESidual*) [66] a été proposée par Saad et Schultz pour remédier au problème des divisions par zéro dans FOM.
- La méthode QMR (*Quasi Minimal Residual*) [28], due à Freund et Nachtigal, est basée sur le processus de Lanczos.

- La méthode TFQMR (*Transpose-Free Quasi-Minimal Residual*) [27] a été proposée par Freund pour éviter l'utilisation de la matrice transposée dans la méthode QMR.

Plusieurs méthodes numériques (FOM, GMRES,...) ont besoin d'orthonormaliser une famille de vecteurs. Le procédé de Gram-Schmidt est l'un des plus connus, sa mise en œuvre peut être effectuée en deux façons, l'une s'appelle Gram-Schmidt Classique (GSC) et l'autre s'appelle Gram-Schmidt modifié (GSM). Le processus d'Arnoldi constitue un cas particulier du processus de Gram-Schmidt pour la famille des vecteurs générateurs des sous-espace de Krylov. Le premier thème traité dans cette thèse est l'étude de la parallélisation des processus de Gram-Schmidt et d'Arnoldi sur des machine massivement parallèles.

Le développement technologique a poussé les scientifiques à résoudre des systèmes linéaires de plus en plus difficiles et de tailles plus grandes. Pour ce faire, ils ont été amenés à utiliser des machines parallèles pour la résolution de ces systèmes. Ces machines, de différentes architectures, permettent le traitement de beaucoup de données dans un temps faible par rapport aux machines séquentielles. L'efficacité de l'implantation d'une méthode dépend de la nature des calculs à effectuer, des données traitées et de l'architecture de la machine cible. Notons qu'une méthode efficace en séquentiel n'est pas nécessairement efficace en parallèle. Avec l'utilisation du parallélisme, des méthodes anciennes qui sont trop coûteuses en séquentiel sont utilisées maintenant pour l'efficacité de leurs versions parallèles et de nouvelles méthodes ont été conçues spécialement pour le parallélisme.

L'implantation sur machine cible de certaines méthodes est confrontée à la diversité des natures des calculs nécessités. Ces méthodes sont algorithmiquement hétérogènes puisqu'elles sont composées de parties parallèles et d'autres séquentielles. Les méthodes hybrides qui nécessitent le calcul de certaines valeurs propres font partie de ces méthodes. Parmi ces méthodes, nous citons la méthode hybride GMRES/Moindres Carrés introduite par Saad [63]. Nous allons adapter cette méthode au parallélisme en séparant la méthode d'Arnoldi pour le calcul des valeurs propres de la méthode GMRES. Nous profitons de l'hétérogénité algorithmique de la méthode obtenue pour l'implanter sur des réseaux hétérogènes de machines.

Dans cette thèse, nous introduisons aussi les méthodes pondérées FOM et GMRES. Ces méthodes sont basées sur le processus d'Arnoldi pondéré qui n'est autre que le processus d'Arnoldi utilisant un autre produit scalaire. Nous montrons qu'il existe des relations entre ces méthodes et les méthodes d'origine.

Cette thèse est composée de cinq chapitres et est présentée selon le plan détaillé suivant.

Le **Chapitre 1** est constitué de quelques rappels sur l'analyse numérique et le parallélisme. Nous essayons au début de montrer le lien entre analyse numérique et parallélisme. Nous présentons ensuite une classification des machines parallèles à mémoire distribuée et de quelques machines particulières que nous allons utiliser dans les implantations de nos algorithmes. Nous présentons aussi une brève description de la parallélisation des algorithmes sur les machines parallèles. Nous terminons ce chapitre par une description des matrices creuses et de quelques formats de stockage de ces matrices.

Dans le **Chapitre 2**, nous étudions la parallélisation des deux versions (classique (GSC) et modifiée (GSM)) du processus de Gram-Schmidt, en parallélisme de données sur des machines massivement parallèles. Nous proposons ensuite une comparaison des deux versions sur des machines parallèles dont les processeurs sont disposés sous forme d'une grille. Les tests numériques sont effectués sur la MasPar MP-1 (16384 processeurs). Nous étudions ensuite le cas particulier du processus d'Arnoldi et sa parallélisation en data-parallèle.

Au **Chapitre 3**, nous rappelons les méthodes GMRES pour la résolution des systèmes linéaires, Arnoldi pour le calcul des valeurs et vecteurs propres et la méthode Moindres Carrés (LS), utilisée pour trouver une approximation polynomiale de l'inverse d'une matrice, et qui peut servir aussi comme préconditionnement polynomial. Nous décrivons aussi l'implantation en mode MIMD et SIMD de ces méthodes sur des machines parallèles à mémoire distribuée.

Dans le **Chapitre 4**, nous présentons la méthode hybride GMRES/LS qui combine les méthodes GMRES et Moindres Carrés et nous analysons sa convergence. Nous introduisons ensuite la méthode hybride parallèle GMRES(m_1)/LS(k,l) -Arnoldi(m_2) que nous implantons sur des réseaux hétérogènes de machines. Après, nous décrivons une implantation asynchrone de cette méthode utilisant deux machines parallèles et trois machines séquentielles. Nous présentons des résultats expérimentaux de cette implantation effectués sur la ferme d'Alpha (16 nœuds), la MasPar MP-1 et des stations SUN. Nous décrivons aussi une implantation entrelacée de la méthode hybride parallèle utilisant une seule machine parallèle et trois machines séquentielles. Nous présentons également des résultats expérimentaux de cette implantation effectués sur la CM-5 et des stations SUN. Ce travail a été réalisé en collaboration avec des chercheurs du Laboratoire d'Informatique Fondamental de Lille (LIFL), en particulier Guy Bergère [6, 7, 24] qui prépare une thèse sur l'aspect plus informatique de ce travail.

Finalement, dans le **Chapitre 5**, nous introduisons les méthodes FOM et GMRES pondérées pour la résolution des systèmes linéaires. Au début nous commençons par introduire le processus d'Arnoldi pondéré sur lequel sont basées ces

deux méthodes. Nous établissons ensuite des relations entre le processus d'Arnoldi pondéré et le processus d'Arnoldi. Après avoir défini les méthodes WFOM et WGMRES redémarrées, nous établisons la relation liant les méthodes WFOM et FOM et celle liant les méthodes WGMRES et GMRES. Nous terminons ce chapitre par une comparaison numérique des différentes méthodes.

Une conclusion de ce travail suivi de quelques perspectives termineront ce manuscript.

Chapitre 1

Analyse numérique et parallélisme

Ce chapitre est consacré au parallélisme et à son utilisation en analyse numérique. Nous donnerons une brève description du parallélisme et des deux classes de machines parallèles SIMD et MIMD, en particulier des machines que nous utiliserons dans l'implantation des algorithmes parallèles : la MasPar, la ferme d'Alpha et la *Connection Machine* CM-5. Nous présenterons ensuite quelques éléments essentiels pour paralléliser des algorithmes sur des machines parallèles. À la fin, nous présenterons les matrices creuses et nous décriverons quelques formats de stockage. Ces formats permettent notamment une économie de place mémoire et une facilité d'utilisation de ces matrices en programmation.

1.1 Parallélisme en analyse numérique

Dans beaucoup de domaines scientifiques ou industriels, la demande d'outils et de méthodes d'analyse numérique est importante, notamment pour la résolution des systèmes linéaires et l'étude de problèmes aux valeurs propres. Souvent ces problèmes proviennent de discrétisation d'équations aux dérivées partielles (EDP) et nécessitent le traitement de matrices de très grande taille. L'importance du nombre de données pose des problèmes de stockage et de temps de calcul qui sont exorbitants sur des machines séquentielles (dites de Von Neumann), d'où la nécessité d'utiliser des machines parallèles pour surmonter ces limitations. Ces machines font coopérer plusieurs processeurs et possèdent une grande capacité mémoire. La minimisation du temps de calcul parallèle provient de la réalisation de plusieurs opérations simultanément. La façon dont les processeurs sont connectés, leurs composantes et leur mode de contrôle conduit à l'obtention de plusieurs sortes d'architectures.

1.2 Machines parallèles

Les machines parallèles existantes sont réparties selon l'accès à la mémoire en deux grandes classes :

Les machines à mémoire partagée : Tous les processeurs de ces machines accèdent aux données dans une même mémoire. Dans ce cas l'utilisateur n'a pas besoin de connaître le lieu de stockage des données ni de gérer les communications entre les processeurs. Pour cette raison, la conception de programme sur ces machines est très simple. Mais, le problème de l'accès à la mémoire limite le nombre de processeurs.

Les machines à mémoire distribuée : Chaque processeur de ces machines dispose de sa propre mémoire et de ses propres données. Sachant que ces processeurs travaillent sur la même application dont les données sont réparties, il est évidemment nécessaire qu'ils communiquent entre eux pour échanger leurs données au cours de l'exécution de l'application.

Par rapport à la première classe de machines parallèles, la deuxième classe permet d'utiliser des tailles mémoire très grandes. Cependant, l'efficacité des programmes sur les machines à mémoire distribuée ne peut être obtenue qu'en plaçant judicieusement les données, afin d'exploiter la localité des données pour minimiser le coût des communications entre les processeurs et les mémoires. Dans notre thèse nous n'utilisons que des machines parallèles à mémoire distribuée qui sont de plus en plus répandues.

Flynn [25] se base sur la façon dont sont manipulés le flot d'instructions et le flot de données pour classer et comparer les machines parallèles. Selon cette taxinomie, deux classes de machines parallèles à mémoire distribuée sont distinguées : SIMD et MIMD [14], [16]. Avant de passer à la description de ces deux classes de machines parallèles, notons qu'un processeur est formé de trois composantes : l'unité de contrôle (décode les instructions), l'unité de traitement (effectue les opérations) et la mémoire (stocke les données).

1.2.1 Machines SIMD

SIMD signifie **S***imple* **I***nstruction* **M***ultiple* **D***ata*; il s'agit d'un ensemble de processeurs supervisés par la même unité de contrôle, qui exécutent tous la même instruction en même temps, chacun sur ses propres données contenues dans sa propre mémoire locale. Un ordinateur appelé frontal (*FE, Front End*) auquel sont reliés tous

les processeurs à travers l'unité de contrôle permet l'accès à la machine parallèle. En général les processeurs de ces machines sont très nombreux, mais ils ne sont pas puissants. Ce type de machines conduit souvent à une programmation basée sur le parallélisme de données. Sur ces machines, nous procédons à la parallélisation des données des applications dont le parallélisme potentiel est maximal. L'architecture de ces machines est représentée dans la figure (1.1).

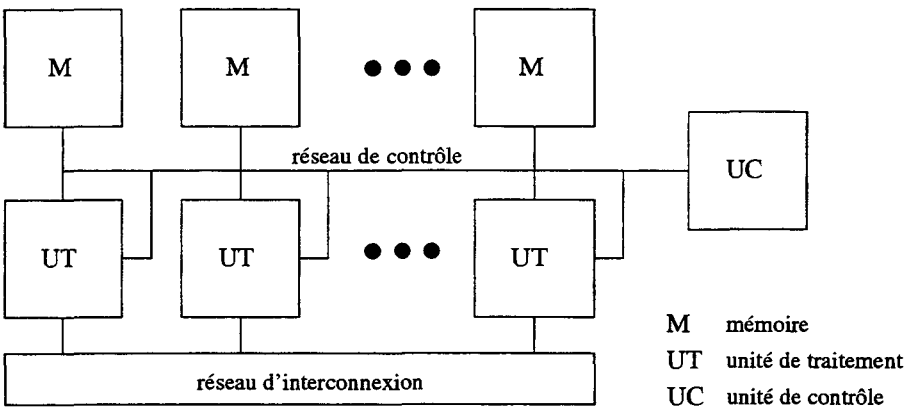

FIG. 1.1 - *L'architecture des machines SIMD.*

1.2.2 Machines MIMD

MIMD signifie **M***ultiple* **I***nstruction* **M***ultiple* **D***ata*; il s'agit d'un ensemble de processeurs dont chacun d'entre eux a sa propre mémoire contenant des données locales sur lesquelles s'exécute son propre programme. Chaque processeur peut donc évoluer indépendamment des autres processeurs. Ces machines ne disposent pas de frontal, mais chacun de ses processeurs peut jouer ce rôle. En général les processeurs de ces machines ne sont pas très nombreux, mais ils sont très puissants. Ce type de machines convient au deux modèles de programmation : parallélisme de données et parallélisme de tâches. Les machines MIMD fonctionnent en mode SPMD (**S***ingle* **P***rogramme* **M***ultiple* **D***ata*) lorsqu'un même programme s'exécute sur tous les processeurs. L'architecture des machines MIMD est représentée dans la figure (1.2).

Les échanges de données entre les différents processeurs sur ces machines sont effectués à l'aide d'une bibliothèque de communications, comme par exemple la bibliothèque PVM [4](**P***arallel* **V***irtual* **M***achine*) et la bibliothèque MPI (**M***essage* **P***assing* **I***nterface*) [50].

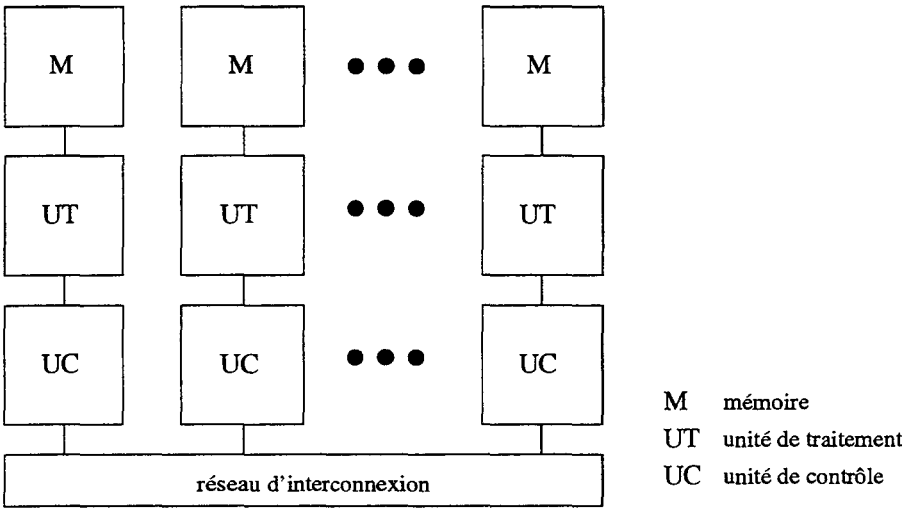

FIG. 1.2 - *L'architecture des machines MIMD.*

1.3 Description de quelques machines parallèles

1.3.1 La MasPar MP-1

C'est une machine SIMD composée de 16384 (128 x 128) processeurs disposés en une grille torique, chaque processeur est relié à ses 8 processeurs voisins [8], [16]. Ces processeurs 4 bits fonctionnent avec une fréquence d'horloge de 12.5 MHz et sont équipés d'une mémoire locale de 64 Ko.

Chaque processeur est connecté à une unité de contrôle (*ACU, Array Control Unit*) ou séquenceur. Cette unité de contrôle est un processeur 4 bits qui reçoit le code exécutable et chargé est d'envoyer ce code et les données aux processeurs pendant l'exécution.

L'accès et le contrôle de la MasPar sont effectués par l'intermédiaire du frontal (*FE, Front End*). C'est une station DEC 5000/240 chargé de la compilation et l'exécution des calculs séquentiels.

Les communications entre les processeurs de la MasPar sont composées de deux types :

* Les communications de voisinage *X-Net* permettant à chaque processeur de communiquer avec tous les processeurs se trouvant à une distance donnée, dans une des huit directions de la rose des vents (N, E, W, S, NE, NW, SE et

SW).

* Les communications générales *Global Router* permettant à deux processeurs quelconques de communiquer. Ce mécanisme de communication est réalisé grace à trois niveaux de réseaux appelés *Crossbars*. Ces communications générales ne sont pas parfaitement parallélisables à cause des conflits qui peuvent survenir entre les données. La durée des communications générales est donc plus élevée que celle relative aux *X-Net*.

1.3.2 La ferme d'Alpha

La ferme d'Alpha est une machine parallèle MIMD composée de 16 processeurs DEC Alpha. Ces processeurs fonctionnent avec une fréquence d'horloge de 133 MHz et sont équipés d'une mémoire locale de 64 Mo. L'accès à la ferme peut être effectué à partir de l'une de ses machines qui joue dans ce cas le rôle du frontal.

Les machines de la ferme d'Alpha sont reliées par un *Crossbar* ayant un très grand débit appelé le GIGAswitch à base de fibres optiques FDDI (**F**iber **D**istributed **D**ata **I**nterface) pouvant atteindre une crête de 3,6 Gb/s[1]. Ce réseau assure les communications entre ces machines. En plus du port GIGAswitch, les machines de la ferme sont équipées d'un port Ethernet ayant un débit de 10 Mb/s[2].

Le *Crossbar* est un réseau dynamique qui permet la connection de tous les nœuds entre eux, c'est une matrice de points de croisement.

1.3.3 La *Connection Machine* CM-5

C'est une machine MIMD de *Thinking Machines Corporation*, prévue pour interconnecter de 32 à 2048 processeurs [72],[16]. Les processeurs élémentaires sont constitués d'un processeur Sparc, d'une mémoire de 32 Mo et de quatre unités de calcul flottant de 32 Mflops chacune.

La CM-5 possède trois réseaux : un réseau de données (*Data Network*) qui gère les communications entre processeurs, un réseau de contrôle (*Control Network*) qui permet des opérations globales (telles que les diffusions, réductions, synchronisations, etc.), et enfin un réseau de diagnostic (*Diagnostic Network*) qui transmet les

1. Giga bits par seconde : 10^9 b/s
2. Méga bits par seconde : 10^6 b/s

messages d'erreur.

La topologie adoptée pour le réseau de données est un *fat-tree* une topologie complexe défini par Leiserson dans [43]. Une telle topologie offre à cette machine une bonne résistance aux pannes, en permettant de doubler les liens entre les processeurs.

1.4 Parallélisation d'algorithmes

La parallélisation d'une méthode passe par plusieurs étapes, afin qu'elle soit la meilleure possible sur une machine parallèle donnée. Notons que cette machine parallèle doit être adaptée aux calculs nécessités par cette méthode. Au début, il faut explorer les différentes versions de cette méthode pour en choisir la meilleure à paralléliser. La bonne version n'est pas obligatoirement celle dont l'algorithme séquentiel est optimal. Cette raison a conduit à la réévaluation de beaucoup d'algorithmes en fonction des nouveaux critères du parallélisme, même ceux qui étaient délaissé depuis plusieurs années.

Après avoir choisi la meilleure version à paralléliser, il faut procéder au partitionnement de ses calculs en sous-tâches en exploitant le parallélisme potentiel et à définir ensuite le graphe de précédence. Ce graphe montre les dépendances entre les tâches et donne l'ordre de leur exécution. L'aspect du graphe de précédence et la nature des tâches (granularité) nous permettent de choisir l'architecture de la machine parallèle la mieux adaptée pour implanter cette méthode.

Une notion très importante en parallélisme est la granularité, cette notion caractérise la nature des tâches affectées aux processeurs. La granularité du parallélisme est dite fine (ou parallélisme à grains fins) quand le parallélisme potentiel est maximal. Les tâches sont réduites à des opérations élémentaires, Le nombre de processeurs virtuels est donc supérieur ou égale au nombre de données.
A l'inverse, la granularité du parallélisme est dite grossière (ou parallélisme à grains grossiers) quand les tâches regroupent plusieurs instructions. Le nombre de processeurs est donc inférieur au nombre de données. Cette granularité est généralement choisie dans le but de diminuer le nombre de communications.

Il n'est pas toujours possible de paralléliser certaines applications, à cause du nombre excessif de dépendences entre ses données dans leur calcul. Un exemple simple de ce cas est le calcul des itérés d'une suite récurrente $u_{n+1} = f(u_n)$, avec $u_0 \in \mathbb{R}$, $n \geq 0$ et f une fonction réelle donnée.
Chaque itéré dépend de l'itéré précédent, ce qui rend impossible de faire des calculs

en parallèle. Par contre, si nous arrivons à trouver le terme général de cette suite numérique et si ce terme ne dépend que de n, nous obtenons alors un parallélisme potentiel maximal puisque chaque processeur calcule un terme de la suite indépendament des autres.
Le processus d'Arnoldi, que nous allons étudier dans le chapitre suivant, constitue lui aussi un bon exemple de ce genre d'applications. Selon la version utilisée pour l'implantation de ce processus, les dépendences entre les données changent et influent considérablement sur sa parallélisation.

Beaucoup d'algorithmes en analyse numérique peuvent être parallélisés, puisqu'ils nécessitent des opérations matricielles possédant un parallélisme intrinsèque. Nous citons parmi ces opérations, l'addition et la multiplication de matrices ainsi que le produit matrice-vecteur.

1.5 Matrices creuses

La parallélisation des méthodes d'analyse numérique nécessite la connaissance des données traitées et de leurs caractéristiques. Ces données sont en général sous forme de matrices. Dans cette section, nous présontons la classe la plus courante et quelques formats de stockage de ces matrices.

Les matrices à traiter en analyse numérique proviennent souvent de discrétisation d'équations aux dérivées partielles (EDP) sur un maillage de nœuds. La discrétisation peut être effectuée soit par différences finis [34] soit par éléments finis [37]. Chaque équation obtenue par la discrétisation ne lie qu'un nombre limité de nœuds, puisque chaque nœud interagit seulement avec les nœuds voisins. Les matrices ainsi obtenues contiennent beaucoup de zéros. Par exemple, une EDP du second ordre discrétisée sur un carré donne une matrice ayant au maximum 5 éléments non nuls par ligne quelle que soit la taille de la matrice.

Les matrices contenant beaucoup de zéros sont appelées *matrices creuses*. Cependant, cette appellation ne peut être atribuée à une matrice que si des techniques spéciales sont utilisées pour profiter de l'abondance d'éléments nuls et de leurs positions [59].

Dans le but d'obtenir une bonne approximation de la solution d'une EDP sur un domaine donné, la discrétisation doit être la plus fine possible. Or, la taille de la matrice obtenue correspond au nombre de nœuds du maillage, ces matrices sont donc en général de très grande taille.

Afin d'éviter le stockage de la totalité des composantes nulles, les matrices creuses sont stockées sous des formats différents de celui des matrices denses. Plusieurs formats existent, chacun est adapté à une classe de matrices ou à une architecture de machine ou à un type d'opération de matrices. Parmi ces formats citons : CSR, CSC, Ellpack-Itpack et SGP que nous décrivons brièvement ci-après. Dans la suite, n désigne la taille de la matrice A et nnz le nombre d'éléments non nuls de A.

Le format CSR *Compressed* **Sparse Row**:

La matrice est stockée dans trois vecteurs : $Aval$, JA et IA.
Le vecteur Aval contient les nnz éléments non nuls de A stockés ligne par ligne.
Le vecteur entier JA contient les indices des colonnes des éléments stockés dans $Aval$. Ce vecteur est aussi de taille nnz.
Le vecteur IA de taille $n+1$ contient les positions dans $Aval$ des premiers éléments non nuls de chaque ligne de A, la dernière composante contient la valeur $nnz+1$.
Dans la figure 1.3, nous présentons la compression CSR d'une matrice de taille 5.

$$A = \begin{pmatrix} 0 & 0 & \mathbf{2} & -1 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 9 \\ 0 & \mathbf{-3} & 2 & 0 & 6 \\ \mathbf{-5} & 4 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{8} & -7 & 0 \end{pmatrix}$$

$Aval =$

2	−1	**1**	9	**−3**	2	6	**−5**	4	**8**	−7

$JA =$

3	4	1	5	2	3	5	1	2	3	4

$IA =$

1	3	5	8	10

FIG. 1.3 - *Exemple de compression CSR d'une matrice*

Ce format de stockage est bien adaptée pour effectuer les produits matrice-vecteur sur des machines séquentielles ou MIMD. Sur les machines MIMD la matrice est découpée en sous-matrices formées d'un certain nombre de lignes. Chaque sous-matrice est stockée sous le format CSR sur un processeur. Ainsi, chaque processeur calcule une partie du produit matrice-vecteur, correspondante aux lignes compressées dont dispose ce processeur.

Ils existe plusieurs formats semblables au format CSR, l'un d'eux procède au stockage des colonnes au lieu des lignes et est appelé format **CSC** (*Compressed* **Sparse Column**).

Le format Ellpack-Itpack :

Ce format procède aussi à une compression de la matrice A en lignes. Comme nous l'avons vu auparavant, les matrices provenant de discrétisation des EDP possèdent un nombre limité d'éléments non nuls par ligne et par colonne, négligable par rapport à n. Notons C_{max} le nombre maximal d'éléments non nuls par ligne. Le format Ellpack-Itpack stocke la matrice dans deux matrices : *Aval* et JA, de taille $n \times C_{max}$, chacune.
La matrice *Aval* contient la matrice A compressée par lignes, le reste des lignes étant complété par des zéros.
Dans la matrice d'entiers JA sont stockés les numéros de colonnes de chaque élément de *Aval*.
Dans la figure 1.4, nous présentons la compression CSR d'une matrice de taille 5.

$$A = \begin{pmatrix} 0 & 0 & 2 & -1 & 0 \\ 1 & 0 & 0 & 0 & 9 \\ 0 & -3 & 2 & 0 & 6 \\ 5 & 4 & 0 & 0 & 0 \\ 0 & 0 & 8 & -7 & 0 \end{pmatrix} \quad Aval = \begin{array}{|c|c|c|} \hline 2 & -1 & \\ \hline 1 & 9 & \\ \hline -3 & 2 & 6 \\ \hline -5 & 4 & \\ \hline 8 & -7 & \\ \hline \end{array} \quad JA = \begin{array}{|c|c|c|} \hline 3 & 4 & \\ \hline 1 & 5 & \\ \hline 2 & 3 & 5 \\ \hline 1 & 2 & \\ \hline 3 & 4 & \\ \hline \end{array}$$

FIG. 1.4 - *Exemple de compression Ellpack-Itpack d'une matrice*

Le format Ellpack-Itpack est plus coûteux que le format CSR en mémoire, mais il est bien adapté au calcul du produit matrice-vecteur sur les machines d'architecture MIMD. En ce qui concerne les machines SIMD, le format SGP est bien adapté pour le calcul des produits matrice-vecteur sur ces machines. Le format SGP est basé sur le format Ellpack-Itpack et est décrit ci-dessous.

Le format SGP *S*parse *G*eneral *P*attern :

Ce format procède à une compression de la matrice A en colonnes [19]. Le format SGP stocke la matrice A dans troix matrices : *Aval*, IA et JC, de taille $C_{max} \times n$, chacune.
La matrice *Aval* contient la matrice A compressée par colonnes, le reste des colonnes étant complété par des zéros.
Dans la matrice d'entiers IA sont stockés les numéros de lignes de chaque élément de *Aval*.
Dans la matrice d'entiers JC sont stockés les numéros de colonnes dans les lignes compressées, correspondants à chaque élément de *Aval*.

Dans la figure 1.5, nous présentons la compression SGP d'une matrice de taille 5. Le tableau situé à droite de la matrice A représente la compression des lignes de A, utilisé pour construire le tableau JC.

$$A = \begin{pmatrix} 0 & 0 & 2 & -1 & 0 \\ 1 & 0 & 0 & 0 & 9 \\ 0 & -3 & 2 & 0 & 6 \\ -5 & 4 & 0 & 0 & 0 \\ 0 & 0 & 8 & -7 & 0 \end{pmatrix}$$

2	−1	
1	9	
−3	2	6
−5	4	
8	−7	

$Aval =$

1	−3	2	−1	9
−5	4	2	−7	6
		8		

$JC =$

1	1	1	2	2
1	2	2	2	3
		1		

$IA =$

2	3	1	1	2
4	4	3	5	3
		5		

FIG. 1.5 - *Exemple de compression SGP d'une matrice*

Le tableau JC est nécessaire pour calculer facilement la compression SGP de la transposée de la matrice A. Posons $Aval\_T$, IA_T et JC_T les tableaux de taille $C_{max} \times n$ correspondants à la compression SGP de A^T.

Les tableaux $Aval\_T$, IA_T et JC_T sont définis comme suit :
pour tout $i \in \{1, \ldots, n\}$ et pour tout $j \in \{1, \ldots, C_{max}\}$ tels que $IA(i,j) \neq 0$ nous avons

$$\begin{aligned} Aval\_T(JC(i,j), IA(i,j)) &= Aval(i,j) \\ IA\_T(JC(i,j), IA(i,j)) &= j \\ JC\_T(JC(i,j), IA(i,j)) &= i \end{aligned}$$

Cette transposition est nécessaire quand nous calculons le produit de A par un vecteur puisque nous n'avons pas l'alignement dans la matrice $Aval$ des composantes de chaque ligne de A. Or, le tableau $Aval\_T$ représente la transposée de la matrice A compressée par ligne, ce qui facilitera le calcul de la somme des composantes d'une ligne.

Afin de calculer $y = Ax$ nous commençons d'abord par multiplier chaque composante du vecteur x par la colonne correspondante de $Aval$, nous obtenons une

matrice intemédiaire $Mval$. Les matrices $Mval$, IA et JC correspondent au format SGP d'une matrice M ayant la même structure que A. Ensuite, nous calculons la matrice $Mval\_T$ en transposant le format SGP de la matrice M. Les colonnes de $Mval\_T$ ne sont autres que les lignes de M compressées. Les composantes du vecteur y s'obtiennent donc en faisant la somme des composantes des colonnes de $Mval\_T$. Le calcul de produit d'une matrice représentée par le format SGP par un vecteur est décrit par l'algorithme suivant

Algorithme 1.5.1 *: Produit matrice-vecteur en SGP ($y = Ax$)*

1. *Calcul de la matrice intermèdiaire $Mval$:*
   Pour $j = 1, \ldots, n$, calculer le vecteur $Mval_j = x_j * Aval_j$,

2. *Transposer la matrice $Mval$ en utilisant les matrices IA et JC pour obtenir la matrice $Mval\_T \in \mathbb{R}^{C_{max} \times n}$.*

3. *Calcul de y le vecteur produit:*
   Pour $j = 1, \ldots, n$, calculer $y_j = \sum_{i=1}^{C_{max}} Mval\_T_{i,j}$,

Cet algorithme génére des communications globales sur les machines parallèles SIMD, toutefois le format de stockage SGP est le mieux adapté à ces machines.

1.6 Conclusion

Dans ce chapitre, nous avons essayé d'expliquer l'importance du parallélisme dans le calcul numérique. Nous avons également décrit quelques classes de machines parallèles.

La classification des machines parallèles à mémoire distribuée que nous avons présentée dans ce chapitre n'est pas la seule qui existe, mais c'est la plus utilisée. La classification de ces machines est très difficile puisque chaque machine parallèle est très particulière et souvent adaptée à un certain genre d'applications scientifiques.

Le traitement des matrices creuses de très grandes tailles en parallélisme nécessite en même temps un bon choix du format de stockage facilitant les opérations demandées et un bon choix de l'architecture de la machine parallèle à utiliser.

En plus des machines parallèles que nous avons cité dans ce chapitre, des machines parallèles dites hétérogènes peuvent être obtenues en rassemblant des machines parallèles de différentes architectures et/ou des machines séquentielles à l'aide du réseau Ethernet. Ces machines parallèles sont très prometeuses pour l'implantation d'applications de nature hybride. Nous allons voir un exemple de telles applications et deux de ses implantations possibles sur deux réseaux différents de machines dans le Chapitre 4.

Chapitre 2

Le processus de Gram-Schmidt en parallélisme de données

2.1 Introduction

Dans ce chapitre, nous nous intéressons à l'étude d'un algorithme particulier pour l'orthonormalisation d'une famille de vecteurs. Les algorithmes d'orthonormalisation tels que Housholder, Givens et Gram-Schmidt [29] sont utilisés dans différentes méthodes de résolution de systèmes linéaires ou de calcul de valeurs propres. En ce qui nous concerne nous allons particulièrement nous intéresser au processus de Gram-Schmidt et à sa parallélisation en data parallèle. À la fin de ce chapitre nous allons étudier le processus d'Arnoldi qui est un cas particulier du processus de Gram-Schmidt.

Nous donnons d'abord les notations que nous utilisons dans ce chapitre. Les matrices seront notées par des lettres majuscules (A, Q, R, ...), les colonnes de matrices seront notées par des lettres minuscules ayant un seul indice (a_k désigne la $k^{\text{ième}}$ colonne de la matrice A) et les composantes d'une matrice seront notées par des lettres minuscules ayant deux indices ($a_{i,j}$ désigne la $i^{\text{ième}}$ composante de la $j^{\text{ième}}$ colonne de la matrice A).
Dans les algorithmes parallèles nous utilisons les notations *array section* utilisées dans le livre *Matrix Computations* de G. H. Golub et C. F. Van Loan [29], utilisées aussi dans les langages de programmation Matlab, Fortran 90 et ses extensions parallèles :
$A(m:n, p:q)$ désigne la sous-matrice $(a_{i,j})_{m \leq i \leq n,\ p \leq j \leq q}$.
$A(m:n, k)$ désigne les composantes $(a_{i,k})_{m \leq i \leq n}$ de la colonne k.

$A(k, p : q)$ désigne les composantes $(a_{k,j})_{p \leq j \leq q}$ de la ligne k.

Étant donnée une matrice réelle A de taille $m \times n$, la factorisation QR consiste à factoriser la matrice A sous la forme $A = QR$ où $Q \in \mathbb{R}^{m \times n}$ est une matrice orthogonale, et $R \in \mathbb{R}^{n \times n}$ une matrice triangulaire supérieure.

Le processus de Gram-Schmidt effectue la factorisation QR en se basant sur la relation suivante

$$r_{k,k} q_k = a_k - \sum_{i=1}^{k-1} r_{i,k} q_i \qquad k = 1, \ldots, n \tag{2.1}$$

avec $r_{i,k} = q_i^T a_k \quad i = 1, \ldots, k-1$, et $r_{k,k} = \|q_k\|_2$.
q_k est formé en soustrayant à a_k sa projection sur les vecteurs $q_1, q_2, \ldots, q_{k-1}$ et en normalisant le vecteur obtenu.

Dans la pratique, il existe deux versions du processus de Gram-Schmidt ; la première est celle appelée méthode de Gram-Schmidt classique (GSC), la seconde est appelée méthode de Gram-Schmidt modifiée (GSM). Ces deux versions demandent le même nombre d'opérations et sont mathématiquement équivalentes (i.e. en arithmétique exacte), mais non numériquement puisque les calculs ne se font pas de la même façon.

Des versions de la factorisation QR par bloc sont étudiées par W. Jalby et B. Philippe [35] et sont implantées sur deux machines parallèles à mémoire partagée hiérarchique: CRAY-2 et ALLIANT FX80. L'algorithme proposé (B2GS) se comporte numériquement comme celui de GSM et il a l'avantage d'être plus parallèle que GSM sur ce type de machines parallèles puisqu'il tire profit de leurs spécificités. Ces résultats ne peuvent pas être généralisés sur toutes les machines parallèles qui ont des caractéristiques différentes. En particulier pour les machines data-parallèles à mémoire distribuée.

La parallélisation de ces deux méthodes que nous allons présenter est en parallélisme de données sur des machines d'architecture SIMD massivement parallèles dont les processeurs sont disposés sous forme d'une grille. Cette étude est effectuée sur un modèle virtuel de machine, c'est-à-dire que nous supposons que nous disposons d'autant de processeurs physiques que de processeurs virtuels.

Les essais numériques de ce chapitre sont réalisés à l'aide du MasPar Fortran ([49], [48]), sur la MasPar MP-1 (voir § 1.3.1) du laboratoire LIFL (Laboratoire d'Informatique Fondamentale de Lille).

Dans ce chapitre, les Sections 2.2 et 2.3 présentent respectivement les versions

classique et modifiée du processus de Gram-Schmidt avec leurs parallélisation en parallélisme de données. Une brève comparaison de ces deux versions parallèles est présentée dans la Section 2.4. Dans les Sections 2.5 et 2.6, nous implantons GSC et GSM sur la MasPar MP-1 et nous présentons quelques résultats expérimentaux. Avant de conclure, nous traîtons dans la Section 2.7 le cas particulier du processus d'Arnoldi.

2.2 Le processus de Gram-Schmidt Classique

2.2.1 Description

La mise en œuvre du processus de Gram-Schmidt classique se déroule de la façon suivante :
l'étape 1 consiste à normaliser a_1, nous obtenons $q_1 = a_1/r_{1,1}$, avec $r_{1,1} = \|a_1\|_2$.

À l'étape k, les vecteurs $\{q_1, \ldots, q_{k-1}\}$ étant déjà calculés, nous désirons former le vecteur suivant q_k et la colonne $(r_{i,k})_{i=1,k}$. Nous devons alors orthogonaliser le vecteur a_k par rapport à la famille de vecteurs $\{q_1, \ldots, q_{k-1}\}$. Pour cela nous soustrayons à a_k des multiples des vecteurs de cette famille, de telle sorte que le vecteur résultant $\hat{q} = a_k - \sum_{i=1}^{k-1} r_{i,k} q_i$ soit orthogonal à tous les vecteurs q_i. Les coefficients $r_{i,k}$ sont donc donnés par la formule

$$r_{i,k} = (a_k, q_i) \qquad i = 1, \ldots, k-1. \tag{2.2}$$

Ensuite, nous obtenons le vecteur q_k qui n'est autre que le vecteur $\hat{q}$ normalisé.

En posant $r_{k,k} = \|\hat{q}\|_2$, nous avons alors la relation

$$r_{k,k} q_k = a_k - \sum_{i=1}^{k-1} r_{i,k} q_i \qquad k = 1, \ldots, n \tag{2.3}$$

Nous notons que le calcul des n^2 éléments de la matrice R par la relation (2.2) caractérise la méthode de Gram-Schmidt classique dont l'algorithme séquentiel est décrit ci-dessous

Algorithme 2.2.1 : *GSC Séquentiel.*

$r_{1,1} = \|a_1\|_2$;
$q_1 = a_1/r_{1,1}$;
pour $k = 2, \ldots, n$
    pour $i = 1, \ldots, k-1 \quad r_{i,k} = (a_k, q_i)$;
    $\hat{q} = a_k - \sum_{i=1}^{k-1} r_{i,k} q_i$;
    $r_{k,k} = \|\hat{q}\|_2$;
    $q_k = \hat{q}/r_{k,k}$;
fin pour.

2.2.2 Algorithme GSC parallèle

Nous observons que le calcul d'un élément $r_{i,k}$ ne nécessite pas la connaissance des autres éléments $r_{j,k}$ $(j \neq i)$ non nuls de la colonne k. Les quantités $r_{i,k}$ $(i = 1, \ldots, k-1)$ sont donc indépendantes. Cette indépendance nous permet de les calculer simultanément et donc le processus de Gram-Schmidt se prête facilement au parallélisme de données. L'algorithme ci-dessous décrit le processus GSC parallèle.

Algorithme 2.2.2 : *GSC Parallèle.*

Pour $k = 1, \ldots, n$
1 $R(1:k-1,k) = Q(1:m,1:k-1)^T A(1:m,k)$;
2 $Q(1:m,k) = A(1:m,k) - Q(1:m,1:k-1)R(1:k-1,k)$;
3 $R(k,k) = \|Q(1:m,k)\|_2$;
4 $Q(1:m,k) = Q(1:m,k)/R(k,k)$;
fin pour.

Cet algorithme génére des communications générales à cause des instructions 1 et 2 puisque la colonne $R(1:k-1,k)$ est calculée et ensuite utilisée comme une ligne.
La machine utilisée est une grille de processeurs torique sur la quelle les communications générales sont très coûteuses par rapport aux communications de voisinage. Pour éviter ces communications, nous stockons les colonnes de R horizontalement à

la place des lignes, c'est-à-dire nous stockons la matrice R dans sa transposée.

2.2.3 Déroulement des calculs à une étape k de l'algorithme GSC parallèle

Nous supposons que nous disposons d'une machine parallèle fromée d'une grille de $m \times n$ processeurs virtuels. Dans ce cas, chaque processeur de la machine dispose d'un élément de la matrice A. Nous supposons que le réseau d'interconnexion permet d'effectuer les réductions avec additions en un temps logarithmique.

À l'étape k, nous avons déjà calculé les vecteurs $q_1, \ldots, q_{k-1}$. Les données sont donc distribuées comme il est indiqué sur la figure 2.1. Les calculs de l'algorithme GSC à cette étape se déroulent comme suit :

$q_{1,1}$		$q_{1,k-1}$	$a_{1,k}$	$a_{1,k+1}$		$a_{1,n}$
$q_{m,1}$		$q_{m,k-1}$	$a_{m,k}$	$a_{m,k+1}$		$a_{m,n}$

FIG. 2.1 -

L'instruction 1 : nous commençons par diffuser la colonne k de la matrice A sur ses $(k-1)$ premières colonnes (voir figure 2.2).

$a_{1,k}$ $q_{1,1}$		$a_{1,k}$ $q_{1,k-1}$	$a_{1,k}$	$a_{1,k+1}$		$a_{1,n}$
$a_{m,k}$ $q_{m,1}$		$a_{m,k}$ $q_{m,k-1}$	$a_{m,k}$	$a_{m,k+1}$		$a_{m,n}$

FIG. 2.2 -

Dans chaque processeur (i,j) où $(1 \leq i \leq m, 1 \leq j \leq k-1)$ nous effectuons en une opération data-parallèle le produit composante par composante suivant :

$a_{i,j} \longleftarrow a_{i,k} * q_{i,j}$.
Nous additionnons alors les lignes des $(k-1)$ premières colonnes de A pour obtenir les produits scalaires $r_{k,j} =< a_k, q_j >$ (pour $1 \leq j \leq k-1$) que nous stockons dans la ligne k de la matrice R (voir figure 2.3). Le coût de cette dernière opération est de l'ordre de $log_2(m)$ opérations data-parallèles (réductions avec additions) .

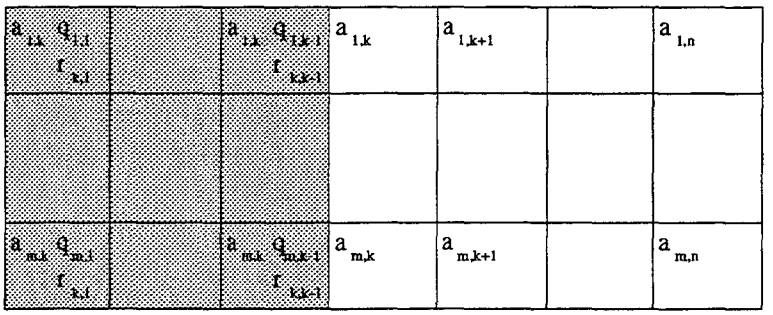

FIG. 2.3 -

L'instruction 2: nous diffusons la $k^{\text{ème}}$ ligne de R sur toutes les lignes pour obtenir en une opération data-parallèle les produits $a_j \longleftarrow r_{k,j} * q_j$ (pour $1 \leq j \leq k-1$), nous stockons ensuite dans la colonne q_k la somme des $(k-1)$ premières colonnes de A retranchée de a_k (voir figure 2.4), cette opération est effectuée en $log_2(k)$ opérations data-parallèles.

L'instruction 3: pour former $r_{k,k}$ la norme de q_k, nous calculons le carré des composantes de q_k, ensuite nous sommons ces quantités, ainsi $r_{k,k}$ est la racine du résultat obtenu et est placé sur chaque processeur de la colonne k. En total la norme nécessite $2 + log_2(m)$ opérations data-parallèles.

L'instruction 4: nous divisons q_k par $r_{k,k}$, ce qui nécessite une opération data-parallèle.

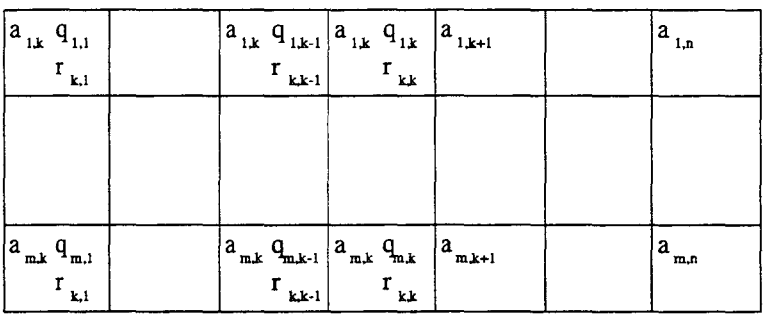

FIG. 2.4 -

Après avoir décrit le déroulement de l'algorithme de GSC, dans la section suivante, nous étudions la complexité de cet algorithme.

2.2.4 Complexité de l'algorithme

Dans le tableau ci-dessous, nous présentons le nombre d'opérations data-parallèles, ainsi que le nombre de communications élémentaires de voisinage nécessité par chaque instruction à l'étape k de l'algorithme GSC parallèle.
Notons d'abord que la somme de s éléments sur s processeurs alignés horizontalement ou verticalement est réalisée avec une complexité de $\lceil log_2(s) \rceil$ (i.e. la partie entière supérieure de $log_2(s)$) additions et s communications élémentaires. Dans la suite, nous approchons la quantité $\lceil log_2(s) \rceil$ par $log_2(s)$ pour faciliter les calculs de complexité.

instruction	opérations	communications
1	$1 + log_2(m)$	$k - 1 + log_2(m)$
2	$1 + log_2(k)$	$m + log_2(k)$
3	$2 + log_2(m)$	$log_2(m)$
4	1	1
Total	$5 + 2log_2(m) + log_2(k)$	$k + m + 2log_2(m) + log_2(k)$

Ainsi le nombre total d'opérations data-parallèles nécessaires pour n étapes est

$$n_{op}(GSC) = 5n + 2nlog_2(m) + log_2(n!),$$

et le nombre total de communications voisinage est

$$n_{com}(GSC) = \frac{n(n+1)}{2} + mn + 2nlog_2(m) + log_2(n!).$$

Notons que la quantité $log_2(n!)$ est équivalent à $nlog_2(n)$ quand n tend vers l'infini. Notons aussi que le processus de Gram-Schmidt en séquentiel nécessite $2n^2m$ opérations.

2.3 Le processus de Gram-Schmidt Modifié

2.3.1 Description

Les exemples numériques effectués pour orthonormaliser une famille de vecteurs à l'aide du processus de GSC montrent que ce processus présente une perte d'orthogonalité entre les vecteurs calculés. C'est pour cette raison que le processus de

Gram-Schmidt modifié a été proposé, vu qu'il est plus stable numériquement [29]. La mise en œuvre de cet algorithme se fait comme suit :

L'étape 1 consiste à normaliser a_1, nous avons donc $q_1 = a_1/r_{1,1}$, avec $r_{1,1} = \|a_1\|_2$.
À l'étape k, les vecteurs $\{q_1, \cdots, q_{k-1}\}$ étant déjà calculés, nous désirons former le vecteur suivant q_k et la colonne $(r_{i,k})_{i=1,k}$. Si nous supposons que nous avons déjà calculé $(r_{j,k})_{j=1,i-1}$, alors pour calculer $r_{i,k}$ nous orthogonalisons le vecteur $a_k - \sum_{j=1}^{i-1} r_{j,k}q_j$ par rapport à q_i, en lui soustrayant un multiple de q_i, de telle sorte que le vecteur résultant $a_k - \sum_{j=1}^{i} r_{j,k}q_j$ soit orthogonal à q_i. Ainsi, les coefficients $r_{i,k}$ sont donnés par la relation suivante

$$r_{i,k} = (a_k - \sum_{j=1}^{i-1} r_{j,k}q_j, q_i). \tag{2.4}$$

Il s'en suit que le vecteur $\hat{q} = a_k - \sum_{i=1}^{k-1} r_{i,k}q_i$ est orthogonal à tous les vecteurs $q_1, \cdots, q_{k-1}$, et donc nous obtenons le vecteur unitaire q_k en normalisant $\hat{q}$ (i.e. $\hat{q} = r_{k,k}q_k$).
Remarquons que nous retrouvons la relation (2.3), ce qui montre que les processus GSC et GSM sont mathématiquement équivalents.

Le calcul des n^2 éléments de la matrice R par la relation (2.4) caractérise la méthode de Gram-Schmidt modifiée, l'algorithme séquentiel de ce processus est décrit ci-dessous

Algorithme 2.3.1 : *GSM Séquentiel.*

$r_{1,1} = \|a_1\|_2$;
$q_1 = a_1/r_{1,1}$;
pour $k = 2, \ldots, n$
  $\hat{q} = a_k$;
  pour $i = 1, \ldots, k-1$;
    $r_{i,k} = (\hat{q}, q_i)$;
    $\hat{q} = \hat{q} - r_{i,k}q_i$;
  fin pour i ;
  $r_{k,k} = \|\hat{q}\|_2$;
  $q_k = \hat{q}/r_{k,k}$;
fin pour k.

2.3.2 Algorithme GSM parallèle

Le processus de GSM décrit par l'algorithme précédent ne peut pas être parallélisé directement comme l'algorithme GSC. Cela est dû au fait que les coefficients $r_{i,k}$ (pour $i = 1, \ldots, k-1$) sont calculés l'un après l'autre puisque chaque élément $r_{i+1,k}$ dépend de l'élément $r_{i,k}$. Nous allons alors chercher une version de GSM plus parallèle. Nous avons

$$r_{i,k} = (a_k - \sum_{j=1}^{i-1} r_{j,k} q_j, q_i). \tag{2.5}$$

Posons alors

$$\begin{cases} a_k^{(1)} = a_k & 1 \leq k \leq n \\ a_k^{(i)} = a_k - \sum_{j=1}^{i-1} r_{j,k} q_j & 2 \leq i \leq n \text{ et } i+1 \leq k \leq n, \end{cases}$$

nous obtenons donc

$$\begin{aligned} a_k^{(i+1)} &= a_k - \sum_{j=1}^{i} r_{j,k} q_j \quad 1 \leq i \leq n \text{ et } i+2 \leq k \leq n \\ &= a_k^{(i)} - r_{i,k} q_i. \end{aligned}$$

Notons $A^{(i)}(1:m, i:n)$ la matrice dont les colonnes sont les vecteurs $a_i^{(i)}, a_{i+1}^{(i)}, \ldots, a_n^{(i)}$.

La dernière relation s'écrit matriciellement sous la forme

$$A^{(i+1)}(1:m, i+1:n) = A^{(i)}(1:m, i+1:n) - Q(1:m, i) R(i, i+1:n). \tag{2.6}$$

A l'aide des nouvelles notations, nous pouvons écrire

$$r_{i,k} = (a_k^{(i)}, q_i) \quad k = i+1, \ldots, n,$$

d'où

$$R(i, i+1:n) = Q(1:m, i)^T A^{(i)}(1:m, i+1:n). \tag{2.7}$$

Et comme $r_{i,i} = \|a_i - \sum_{j=1}^{i-1} r_{j,i} q_j\|_2$ alors $r_{i,i} = \|a_i^{(i)}\|_2$, et donc

$$R(i,i) = \|A^{(i)}(1:m, i)\|_2 \tag{2.8}$$

Les relations (2.6), (2.7) et (2.8) nous permettent d'obtenir l'algorithme parallèle du processus de GSM qui est décrit comme suit

Algorithme 2.3.2 : *GSM Parallèle.*

Pour $k = 1, \ldots, n$

1	$R(k,k) = \|A(1:m,k)\|_2$;
2	$Q(1:m,k) = A(1:m,k)/R(k,k)$;
3	$R(k,k+1:n) = Q(1:m,k)^T A(1:m,k+1:n)$;
4	$A(1:m,k+1:n) = A(1:m,k+1:n) - Q(1:m,k)R(k,k+1:n)$;

fin pour.

2.3.3 Déroulement des calculs à une étape k de l'algorithme GSM parallèle

Nous avons la même distribution de données que dans l'algorithme GSC. Alors, les figures (2.5), (2.6), (2.7) et (2.8) décrivent le déroulement de l'étape k de l'algorithme GSM.

À l'étape k, nous avons déjà calculé $q_1, \ldots, q_{k-1}$ et nous avons mis à jour pendant les étapes précédentes les colones $k, k+1, \ldots, n$ de la matrice A (voir figure 2.5).

$q_{1,1}$		$q_{1,k-1}$	$a^{(k)}_{1,k}$	$a^{(k)}_{1,k+1}$		$a^{(k)}_{1,n}$
$q_{m,1}$		$q_{m,k-1}$	$a^{(k)}_{m,k}$	$a^{(k)}_{m,k+1}$		$a^{(k)}_{m,n}$

FIG. 2.5 -

L'instruction 1 : Pour former la norme $r_{k,k}$ de $a_k^{(k)}$, nous calculons le carré des composantes de $a_k^{(k)}$, ensuite nous sommons ces quantités, $r_{k,k}$ est alors la racine du résultat obtenu (voir figure 2.6), en total la norme nécessite $2 + log_2(m)$ opérations data-parallèles.

L'instruction 2 : q_k est obtenu en divisant $a_k^{(k)}$ par $r_{k,k}$, en une opération data-parallèle.

L'instruction 3 : Nous commençons par diffuser la colonne k de la matrice Q sur ses $(n-k)$ dernières colonnes. Dans chaque processeur (i,j) où $(1 \leq i \leq m,\ k+1 \leq$

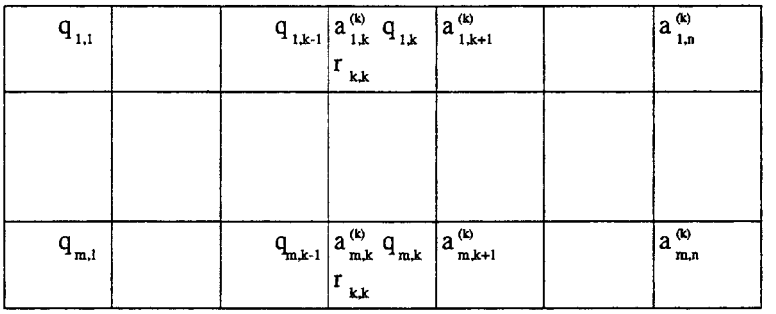

FIG. 2.6 -

$j \leq n$) nous effectuons le produit suivant: $q_{i,j} \longleftarrow q_{i,k} * a_{i,j}$, en une opération data-parallèle (voir figure 2.7).
Puis, nous additionnons les lignes de cette sous matrice de Q pour obtenir les produits scalaires $r_{k,j} =< q_k, a_j > \quad k+1 \leq j \leq n$ que nous stockons dans la matrice R, ce qui nécessite un coût de $log_2(m)$ opérations data-parallèles.

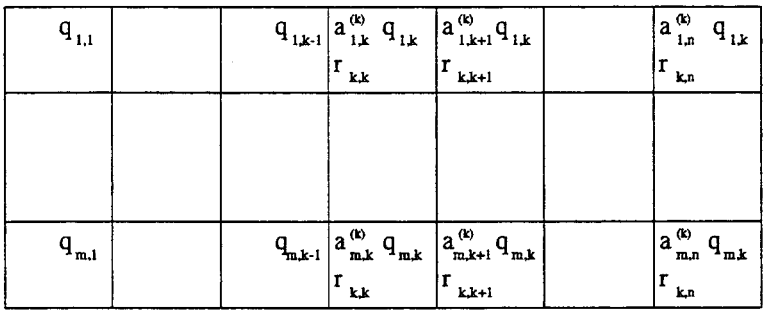

FIG. 2.7 -

L'instruction 4: Nous diffusons la ligne k de R verticalement, et comme la colonne q_k est déjà diffusée sur les $(n-k)$ dernières colonnes, nous obtenons alors les produits $q_j \longleftarrow r_{k,j} * q_k$ où $k+1 \leq j \leq n$ (voir figure 2.8), ce qui se fait en une opération data-parallèle. Ensuite nous retranchons des a_j les q_j ainsi calculés, nous stockons alors le nouveau résultat dans a_j, le coût est d'une opération data-parallèle.

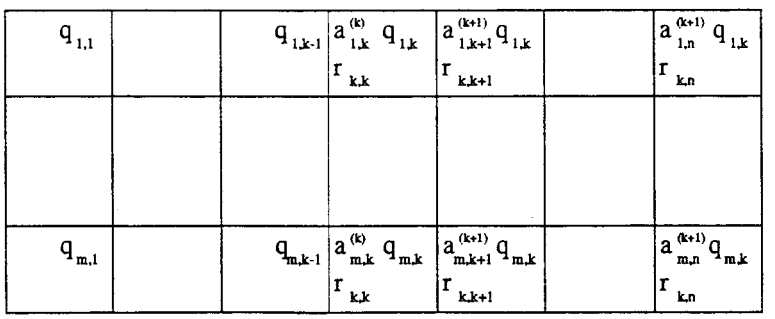

FIG. 2.8 -

Après avoir décrit le déroulement de l'algorithme de GSM, dans la section suivante, nous étudions la complexité de cet algorithme.

2.3.4 Complexité de l'algorithme

Dans le tableau ci-dessous, nous présentons le nombre d'opérations data-parallèles, ainsi que le nombre de communications élémentaires de voisinage nécessité par chaque instruction à l'étape k de l'algorithme GSM parallèle.

instruction	opérations	communications
1	$2 + log_2(m)$	$log_2(m)$
2	1	1
3	$1 + log_2(m)$	$n - k + log_2(m)$
4	2	m
total	$6 + 2log_2(m)$	$1 + m + n - k + 2log_2(m)$

Le nombre total d'opérations data-parallèles nécessaires pour n étapes est

$$n_{op}(GSM) = 6n + 2nlog_2(m),$$

et le nombre total de communications est

$$n_{com}(GSM) = \frac{n(n+1)}{2} + mn + 2nlog_2(m).$$

2.4 Comparaison entre les algorithmes GSC et GSM parallèles

Si nous comparons le nombre d'opérations data-parallèles de GSC et GSM, nous obtenons

$$n_{op}(GSC) - n_{op}(GSM) = log_2(n!) - n > 0.$$

Et si nous comparons le nombre de communications de GSC et GSM, nous obtenons

$$n_{com}(GSC) - n_{com}(GSM) = log_2(n!) > 0.$$

Nous observons donc que les nombres d'opérations et de communications de l'algorithme GSC sont supérieurs à ceux de l'algorithme GSM. Nous en concluons que GSM est plus parallèle que GSC en parallélisme de données.

2.5 Implantation de GSC et GSM sur la MasPar MP-1

Les essais numériques effectués sur la MasPar MP-1 en Fortran parallèle des deux algorithmes GSC et GSM ont donné un résultat opposé de celui de la section précédente. En effet, nous trouvons que le temps d'exécution de l'algorithme GSM est plus grand que celui de l'algorithme GSC. Notons que les communications sur la MasPar ne sont pas effectuées en un temps logarithmique de la distance entre les processeurs, mais en un temps linéaire.

Des tests sur la MasPar à l'aide MasPar Fortran et aussi en MPL (extention data-parallèle du langage C), nous ont permis de remarquer que les communications de gauche à droite sont plus coûteuses que celles de droite à gauche, et que les communications du haut vers le bas sont plus coûteuses que les communications du bas vers le haut. Cela n'est pas dû au problème de placement des données sur les processeurs physiques puisque nous avons fait des tests seulement avec des vecteurs de taille 128. Notons que ce résultat ne figure dans aucun document décrivant la MasPar MP-1.

Dans le processus GSM, les communications se font de la gauche vers la droite, ce qui est coûteux. Pour éviter cela il faut changer le sens des communications, cela se fait en appliquant l'algorithme GSM à la matrice A en inversant l'ordre des colonnes. Ainsi l'algorithme devient

Algorithme 2.5.1 : *GSM Parallèle.*

Pour $k = n, \ldots, 1$, *pas* $= -1$
1 $R(k,k) = \|A(1:m,k)\|_2$;
2 $Q(1:m,k) = A(1:m,k)/R(k,k)$;
3 $R(k,1:k-1) = Q(1:m,k)^T A(1:m,1:k-1)$;
4 $A(1:m,1:k-1) = A(1:m,1:k-1) - Q(1:m,k)R(k,1:k-1)$;
fin pour.

La remarque ci-dessus n'étant pas vrai pour toutes les machines SIMD, cet

algorithme reste particulièrement valable sur la MasPar. Notons que de telle optimisation d'algorithme sur une machine parallèle dépend énormément de l'architecture et du réseau de la machine cible.

2.6 Résultats expérimentaux

Nous prenons comme matrice test la matrice de Hilbert. Nous considérons les deux choix $m = 128$ et $m = 256$, pour cela nous utilisons tous les processeurs de la MasPar: 128×128. Dans le cas où $m = 128$, nous n'avons pas de problème de placement de données puisque chaque élément de la matrice est placé sur un processeur de la machine. Par contre, dans le cas où $m = 256$ la matrice est répartie cycliquement et chaque processeur prend en charge quatre éléments de la matrice A (par exemple, dans le processeur (1,1) il y a les quatre composantes suivantes de A: $a_{1,1}$, $a_{1,129}$, $a_{129,1}$, et $a_{129,129}$).

Les courbes montrant l'évolution du temps CPU en fonction du nombre de colonnes sont presentées dans les figures 2.9 et 2.10.

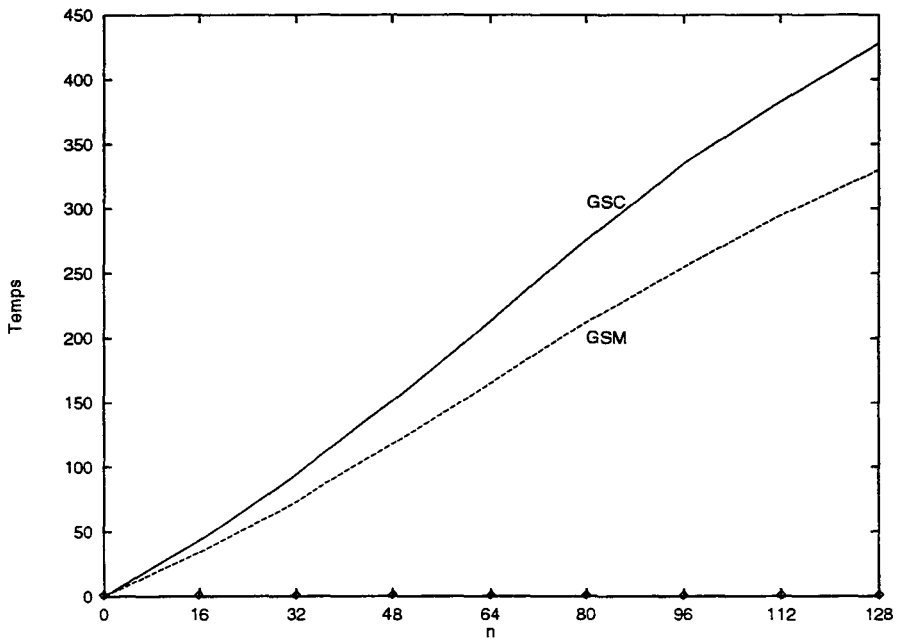

FIG. 2.9 - *L'évolution du temps CPU en fonction du nombre de colonnes, dans le cas où le nombre de lignes est 128*

Les deux courbes montrent que l'algorithme GSM est plus rapide que l'algo-

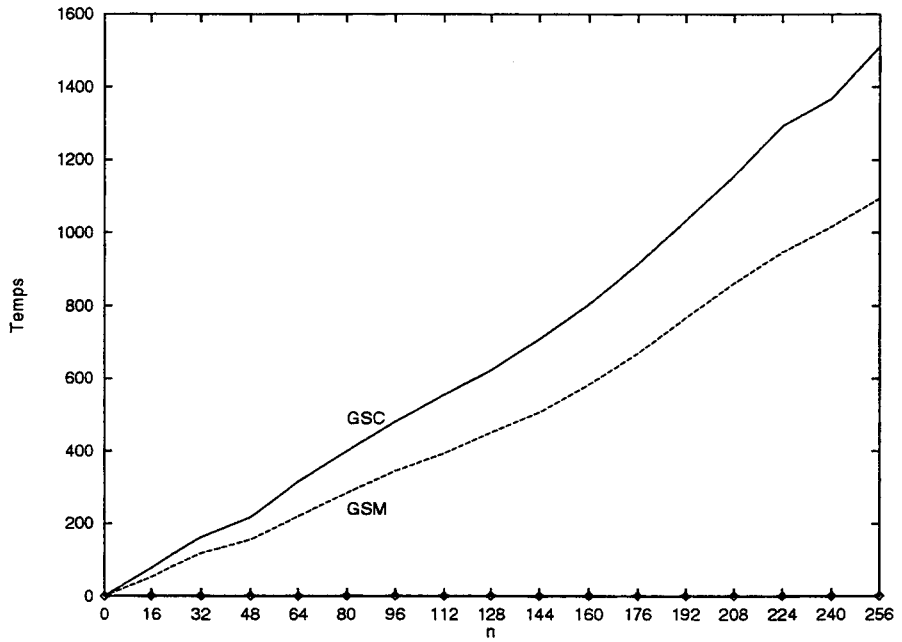

FIG. 2.10 - *L'évolution du temps CPU en fonction du nombre de colonnes, dans le cas où le nombre de lignes est 256*

rithme GSC en temps CPU. Le processus GSM est donc plus parallèle que GSC en data-parallèle et aussi plus stable numériquement. Dans la section suivante nous étudions le processus d'Arnoldi et nous examinons ses parallélisations en utilisant GSC et GSM.

2.7 Le processus d'Arnoldi

Le processus d'Arnoldi est l'un des processus que l'on retrouve souvent dans des méthodes de projection sur des sous-espaces de Krylov pour la résolution de systèmes linéaires, ou de calcul de valeurs propres. Ce processus construit une base orthonormale $V_m = [v_1, v_2, \ldots, v_m]$ du sous-espace de Krylov

$$K_m(A, v) = \text{vect}(v, Av, \ldots, A^{m-1}v),$$

où $A \in \mathbb{R}^{n \times n}$ et $v \in \mathbb{R}^n$.
L'algorithme d'Arnoldi résulte de l'application récursive du processus de Gram-Schmidt sur les colonnes de la matrice $[v_1, Av_1, \ldots, Av_{m-1}]$, où v_1 est le vecteur v normalisé.
Pour un certain j la relation de Gram-Schmidt qui construit le vecteur v_{j+1} à partir

des vecteurs $v_1, \ldots, v_j$ et de Av_j peut s'écrire

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j}v_i, \qquad j = 1, \ldots, m, \tag{2.9}$$

Notons $\overline{H}_m$ la matrice de Hessenberg supérieure de taille $(m+1) \times m$ dont les composantes non nulles sont les coefficients $h_{i,j}$.

La relation (2.9) s'écrit sous la forme matricielle suivante

$$AV_m = V_{m+1}\overline{H}_m. \tag{2.10}$$

Notons par H_m la matrice de Hessenberg supérieure carrée de taille $m \times m$ qui n'est autre que la matrice $\overline{H}_m$ privée de sa dernière ligne, nous avons alors

$$H_m = V_m^T AV_m, \tag{2.11}$$

et la relation (2.10) peut être réécrite sous la forme

$$AV_m = V_m H_m + h_{m+1,m}v_{m+1}e_m^T, \tag{2.12}$$

où e_m est le $m^{\text{ème}}$ vecteur de la base canonique de $\mathbb{R}^m$.

Il existe deux versions du processus d'Arnoldi, selon le processus de Gram-Schmidt utilisé classique (GSC) ou modifé (GSM). Comme nous l'avons vu auparavant, le processus de Gram-Schmidt modifié (GSM) est plus stable et se prête mieux au parallélisme que GSC. Mais, dans le cas du processus d'Arnoldi, les vecteurs à orthonormaliser se calculent au fur et à mesure, ce qui rend impossible l'implantation parallèle de ce processus en utilisant l'algorithme de GSM (algorithme 2.5.1). À l'étape k, cet algorithme calcule le vecteur q_k et met à jour les colonnes $a_{k+1}^{(k)}, \ldots, a_n^{(k)}$ de la matrice A. Ces colonnes ne sont pas encore formées dans le cas du processus d'Arnoldi.

L'algorithme parallèle du processus d'Arnoldi utilisant Gram-Schmidt classique est décrit ci-dessous.

Algorithme 2.7.1 : *Arnoldi Parallèle.*

$\beta = \|V(1:n)\|_2$;
$V(1:n,1) = V(1:n)/\beta$;
Pour $k = 1, \ldots, m$
1 $V(1:n,k+1) = A(1:n,1:n)V(1:n,k)$;
2 $H(1:k,k) = V(1:n,1:k)^T V(1:n,k+1)$;
3 $V(1:n,k+1) = V(1:n,k+1) - V(1:n,1:k)H(1:k,k)$;
4 $H(k+1,k) = \|V(1:n,k+1)\|_2$;
5 $V(1:n,k+1) = V(1:n,k+1)/H(k+1,k)$;
fin pour.

Réorthogonalisation :

Nous pouvons remédier au problème de stabilité de GSC par la technique de réorthogonalisation. Si une perte d'orthogonalité est détectée à l'itération j on réorthogonalise le vecteur v_{j+1}, que nous venons juste de calculer, par rapport à $v_1, v_2, \ldots, v_j$. Ce qui revient à calculer le vecteur v_{j+1}^r tel que

$$h_{j+1,j} v_{j+1}^r = v_{j+1} - \sum_{i=1}^{j} h'_{i,j} v_i$$

où $h'_{i,j} = v_{j+1}^T v_i$ et en posant

$$h_{i,j} = h_{i,j} - h'_{i,j} \quad \text{pour} \quad i = 1, \ldots, j \quad \text{et} \quad h_{j+1,j} = \|v_{j+1}^r\|_2.$$

Remarquons que nous n'avons pas besoin de calculer un autre produit matrice-vecteur et que le vecteur v_{j+1} à réorthogonaliser peut ne pas être normalisé.

Il existe plusieurs façons de détecter une perte d'orthogonalité. Notamment, le test adopté par Kelley dans [38] est fondé sur la condition de Brown/Hindmarch, où l'on réorthogonalise si l'égalité suivante

$$\|Av_j\|_2 + \delta h_{j+1,j} = \|Av_j\|_2$$

est vérifiée en précision machine, avec $0 \ll \delta < 1$. Kelley propose de prendre la valeur de δ égale à 10^{-3}, cette valeur est obtenue numériquement, mais ce choix n'est pas démentré théoriquement.

Le test proposé par Bennani dans sa thèse [5] consiste a choisir un réel ϵ tel que $0 \ll \epsilon < 1$ et l'on réorthogonalise si $\|Av_j\|_2 < \epsilon h_{j+1,j}$.

Nous pouvons également réorthogonaliser à chaque itération en se passant d'un test de perte d'orthogonalité, ce qui est relativement coûteux, mais l'algorithme obtenu reste plus performant que GSM.

Dans l'exemple ci-dessous nous comparons l'algorithme d'Arnoldi utilisant Gram-Schmidt classique avec sa version réorthogonalisée.

Exemple numérique :

Nous prenons comme matrice test la matrice de Pascal définie par

$$a_{i+1,j+1} = a_{i+1,j} + a_{i,j+1}; \quad i,j \in \{1,\ldots,n-1\},$$

avec l'initialisation $a_{i,1} = a_{1,i} = 1; \quad i \in \{1,\ldots,n\}$
où n est la taille de la matrice. Cette matrice est très mal conditionnée (la fonction *cond* de Matlab estime le conditionnement par l'infini). Dans cet exemple nous prenons $n = 128$ et le vecteur v_1 est aléatoire.

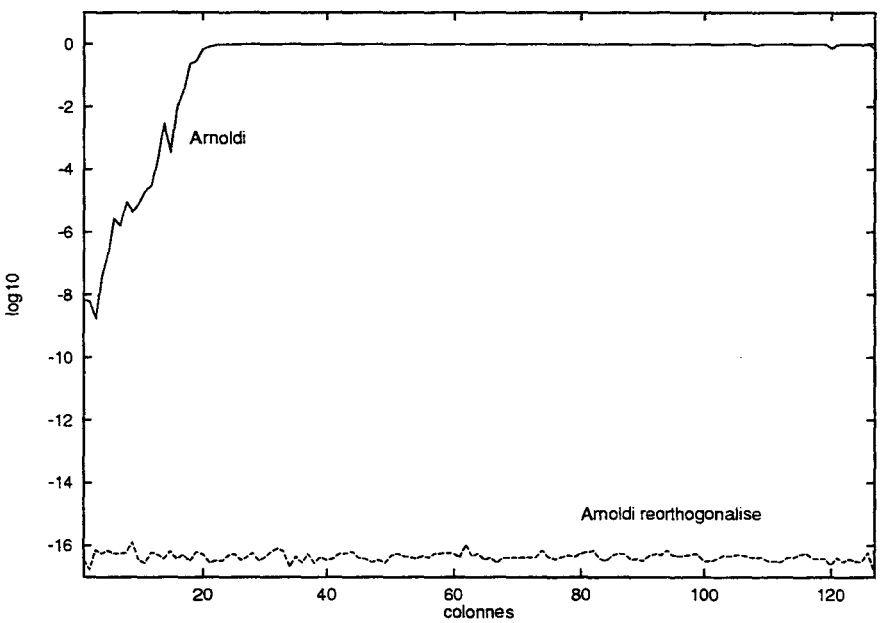

FIG. 2.11 - *La courbe représente le maximum des valeurs absolues des produits scalaires de chaque colonne v_j de V_m avec les colonnes précédentes ($\max_{i=1}^{j-1} |v_j^T v_i|$ pour la colonne j avec $j \in \{2,\ldots,n\}$).*

Dans la figure 2.11, nous remarquons que la réorthogonalisation du processus d'Arnoldi est très efficace pour cet exemple qui présente un grande perte d'orthogonalité.

2.8 Conclusion

Nous avons donc montrer qu'en parallélisme de données, la version modifiée du processus de Gram-Schmidt GSM est plus parallèle que sa version classique GSC. Ce résultat est vrai seulement dans le cas où nous connaissons dès le début les vecteurs à orthonormaliser, ce n'est pas le cas pour le processus d'Arnoldi.

Le processus d'Arnoldi étant beaucoup plus parallélisable en utilisant Gram-Schmidt classique qui est moins stable que Gram-Schmidt modifié, la réorthogonalisation de son algorithme est nécessaire afin de remédier à cet incovénient. Cette réorthogonalisation peut doublé le nombre d'opérations de cet algorithme, sans compter les opérations correspondantes au produits matrice-vecteur, si un test de perte d'orthogonalité n'est pas utilisé.

Ce résultat obtenu pour Arnoldi sur les machines parallèles SIMD reste valable sur les machines parallèles MIMD. Nous utilisons ce résultat pour l'implantation de GMRES et Arnoldi dans les deux chapitre suivants.

Aux deux chapitre suivants, nous implantons en parallélisme de données les méthodes GMRES et Arnoldi en utilisant le processus de Gram-Schmidt classique, quand il y a risque de perte d'orthogonalité, cet algorithme est réorthogonalisé.

Une autre technique basée sur l'idée dévelopée dans [2], [12] et [57] par Reichel et al est très intéressante puisqu'elle perment d'obtenir des algorithmes très stables et plus parallèles construisant une base orthonormale d'un sous-espace de Krylov. Cette technique utilise les points de Leja introduits par Edrei [21] et Leja [44].
Des implantations sur la Paragon d'Intel (machine parallèle à mémoire distribuée) de certains algorithmes basés sur la même approche sont étudiées par B. Philippe et R. B. Sidje dans [71] et [56].

Chapitre 3

Description et parallélisation des méthodes utilisées

Dans ce chapitre nous allons décrire brièvement les méthodes: GMRES (résolution de système linéaire), Arnoldi (calcul des valeurs propres d'une matrice) et la méthode Moindres Carrés (approximation de l'inverse d'une matrice). Nous traitons aussi dans ce chapitre la parallélisation de ces méthodes en respectant les natures des calculs dans leurs algorithmes. Dans le chapitre suivant, ces méthodes seront combinées pour obtenir une méthode hybride pour la résolution de systèmes linéaires.

3.1 GMRES

3.1.1 Description

La méthode GMRES (Generalized Minimum RESidual) a été proposée par Saad et Schultz en 1986 [66] comme une méthode de Krylov pour la résolution de systèmes linéaires non symétriques. Le $m^{\text{ième}}$ $(m \geq 1)$ itéré x_m de la méthode GMRES est celui qui minimise le résidu $r = b - Ax$ sur l'ensemble des x appartenant à l'espace affine $x_0 + K_m(A, r_0)$. L'itéré x_m est donc solution du problème aux moindres carrés suivant :

$$\text{minimiser}_{x \in x_0 + K_m(A,r_0)} \|b - Ax\|_2. \tag{3.1}$$

La méthode GMRES utilise le processus d'Arnoldi pour construire une base or-

thonormale V_m du sous-espace de Krylov $K_m(A, r_0)$. Les matrices $V_{m+1} = [V_m, v_{m+1}]$, et $\overline{H}_m$ construites par le processus d'Arnoldi satisfont la relation (2.10).

Comme $K_m(A, r_0) = \text{vect}(v_1, v_2, \ldots, v_m)$, l'itéré x_m s'écrit sous la forme

$$x_m = x_0 + V_m y_m,$$

où $y_m \in \mathbb{R}^m$ et le résidu $r_m = b - Ax_m$ associé à x_m s'écrit comme suit :

$$\begin{aligned} r_m &= b - A(x_0 + V_m y_m) \\ &= r_0 - AV_m y_m \\ &= V_{m+1}(\beta e_1 - \overline{H}_m y_m), \end{aligned}$$

où e_1 est le premier vecteur de la base canonique de $\mathbb{R}^{m+1}$.

Comme la matrice V_{m+1} est orthonormale, la norme de r_m s'écrit :

$$\begin{aligned} \|r_m\|_2 &= \|V_{m+1}(\beta e_1 - \overline{H}_m y_m)\|_2 \\ &= \|\beta e_1 - \overline{H}_m y_m\|_2. \end{aligned}$$

Ainsi, y_m est solution du problème aux moindres carrés suivant

$$\text{minimise}_{y \in \mathbb{R}^m} \|\beta e_1 - \overline{H}_m y\|_2. \tag{3.2}$$

Pour résoudre ce problème d'optimisation, nous utilisons la factorisation QR [29] basée soit sur les transformations de Householder, qui est très stable numériquement, soit sur les rotations de Givens, qui a l'avantage de donner des estimation des normes des résidus, sans avoir à calculer des produits matrice-vecteur.

La méthode GMRES est une amélioration de la méthode *Full Orthogonalization Method* notée FOM [61] proposée par Saad. La méthode FOM calcule l'itéré x_m^F appartenant au sous-espace affine $x_0 + K_m(A, r_0)$ en imposant la condition de Galerkin suivante

$$b - Ax_m^F \perp K_m(A, r_0).$$

Cet itéré s'écrit donc sous la forme $x_m^F = x_0 + V_m y_m^F$ où y_m^F est la solution du système linéaire $H_m y = \beta e_1$.

La propriété de minimisation permet à la méthode GMRES d'obtenir des itérés dont les normes du résidus sont décroissantes, ce n'est pas le cas de la méthode FOM.

Des relations ont été établies entre FOM et GMRES dans [10] et [15]. En particulier, les itérés de la méthode GMRES peuvent être obtenus à partir de ceux de la méthode FOM par la technique de lissage MRS (Minimal Residual Smoothing technique) [77].

Dans l'algorithme de la méthode GMRES le nombre de vecteurs à stocker croît en fonction de m la dimension du sous-espace de Krylov. D'où la nécessité d'utiliser l'algorithme itérativement, en le redémarrant après un nombre fixé m d'itérations et en prenant x_m comme nouveau itéré de départ.

L'algorithme ainsi obtenu est appelé méthode GMRES redémarrée et est noté GMRES(m).

Algorithme 3.1.1 : *GMRES(m).*

1. *Initialisation : choisir* x_0, m *la dimension du sous-espace de Krylov et* ε *la tolérance,*
   calculer $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ *et* $v_1 = r_0/\beta$.

2. *Processus d'Arnoldi (Gram-Schmidt classique)*
   Pour $j = 1, \ldots, m$
   $w = Av_j$
   pour $i = 1, \ldots, j$ $h_{i,j} = (w, v_i)$,
   $w = w - \sum_{i=1}^{j} h_{i,j} v_i$,
   calculer $h_{j+1,j} = \|w\|_2$ *et* $v_{j+1} = w/h_{j+1,j}$
   fin pour.

3. *Calculer* $y_m = \arg\min_{y \in \mathbb{R}^m} \|\beta e_1 - \overline{H}_m y\|_2$ *par la factorisation QR de* $\overline{H}_m$
   et poser $x_m = x_0 + V_m y_m$, $r_m = b - Ax_m$.

4. *Redémarrage : si* $\|r_m\|_2 \leq \varepsilon$ *stop*
   sinon poser $x_0 = x_m$, $r_0 = r_m$, $\beta = \|r_0\|_2$ *et* $v_1 = r_0/\beta$, *et aller en 2.*

Dans le cas où pour un certain j la quantité $h_{j+1,j}$ est nulle, l'algorithme doit être arrêté car la solution exacte du système a été obtenue à l'itération j et cette solution est donnée par $x^* = x_0 + V_j y_j$ où $y_j \in \mathbb{R}^j$ est la solution du système linéaire de taille m suivant

$$H_j y_j = \beta e_1.$$

Dans le cas où la quantité $h_{j+1,j}$ n'est pas nulle mais très proche de zéro (ce phénomène est dit *near-breakdown*), une approximation de la solution est donnée par la même formule ci-dessus.

La version du processus d'Arnoldi que nous utilisons est celle basée sur Gram-Schmidt classique GSC, bien qu'elle soit moins stable que celle basée sur Gram-Schmidt modifié CGM. Comme nous l'avons vu au Chapitre 2, la première version est plus parallèle et nous pouvons la rendre plus stable à l'aide de la technique de réorthogonalisation.

Dans [66], Saad et Schultz ont établi un résultat de convergence de la méthode GMRES. Ce résultat est rappelé dans la proposition suivante.

Proposition 3.1.1
Supposons que la matrice A est diagonalisable et posons $A = X\Lambda X^{-1}$, où Λ est la matrice diagonale des valeurs propres ($\Lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$).
Posons

$$\varepsilon^{(m)} = \min_{p \in \mathcal{P}_m,\, p(0)=1} \max_{\lambda \in \sigma(A)} |p(\lambda)|.$$

Alors, le résidu à l'étape m de GMRES vérifie l'inégalité suivante :

$$\|r_m\|_2 \leq \kappa(X)\varepsilon^{(m)}\|r_0\|_2,$$

où $\kappa(X) = \|X\|_2\|X^{-1}\|_2$ est le conditionnement de la matrice X.

La convergence superliéaire de GMRES a été prouvée par Van der Vorst [73] qui a observé à travers des résultats expérimentaux que la convergence de GMRES est liée à la convergence des valeurs de Ritz vers les valeurs propres exactes de A. Notons aussi que Sadok [67], dans un travail récent, a lié la convergence de GMRES à la convergence des valeurs singulières des matrices H_m et $\overline{H}_m$ vers celles de A.

Remarque 3.1.1

À partir de la proposition 3.1.1 nous pouvons déduire un résultat important sur la convergence de GMRES(m). Quand une valeur propre λ_e est bien approximée par une valeur de Ritz à une étape de GMRES(m), la composante de son vecteur propre associé s'annule dans l'expression du résidu. À l'étape suivante, dans l'expression de $\varepsilon^{(m)}$ la minimisation ne se fait que sur $\sigma(A) - \{\lambda_e\}$, ce qui accélère la convergence de GMRES(m). Ce résultat est exploité par la méthode Moindres Carrés que nous allons décrire dans la Section 3.3.

3.1.2 Méthode GMRES(m) parallèle

La méthode GMRES redémarré est composée de trois parties : le processus d'Arnoldi, la minimisation et le calcul de la solution approchée x_m.
La minimisation est complètement séquentielle puisqu'en général la valeur de m n'est pas trés grande donc le nombre de données traitées n'est pas très important. Tandis que les deux autres parties sont parallèles. Nous présentons ci-après les implantations de la méthode GMRES(m) en parallélisme de données sur les deux architectures de machines parallèles : MIMD et SIMD.

Implantation en mode MIMD

Nous implantons GMRES(m) en parallélisme de tâches sur une architecture MIMD. Un processus père lance p processus identiques sur p stations et distribue la matrice et le second membre sur ces processus (voir figure 3.1). Chaque processeur reçoit n/p lignes de la matrice A et exécute la partie de GMRES correspondante à sa propre sous-matrice. Pendant l'exécution, le processus père se charge des communications entre les processeurs.

La projection étant parallèle, chaque processeur exécute les calculs de cette partie liés à ses propres données et obtient des résultats qu'il stocke dans sa mémoire. A la fin de cette partie, nous obtenons la matrice $\overline{H}_m$ sur tous les processeurs, et la matrice V_{m+1} qui a la même répartition que la matrice A sur les processeurs. En vue de minimiser les communications, la partie séquentielle (calcul de y_m) s'effectue de façon redondante sur tous les processeurs. Ainsi, le vecteur y_m se trouvant sur tous les processeurs permettra de calculer chaque partie de l'itéré x_m sur un processeur. Tous les processeurs exécutent donc le même programme sur leurs propres données sauf le père. Les processeurs fils fonctionnent donc selon le modèle de programmation SPMD.

Degré de parallélisme dans GMRES(m)

Le temps total pour atteindre la convergence (d'un système linéaire donné) est la somme du temps de calcul et des temps de communications qui dépendent du nombre de processeurs utilisés ainsi que du type et de la puissance du réseau d'interconnexion utilisé. Le temps de calcul est décroissant en fonction du nombre de processeurs, puisque en augmentant le nombre de processeurs le nombre de lignes distribuées sur les processeurs diminue, les caluls sont donc effectués plus rapidement.

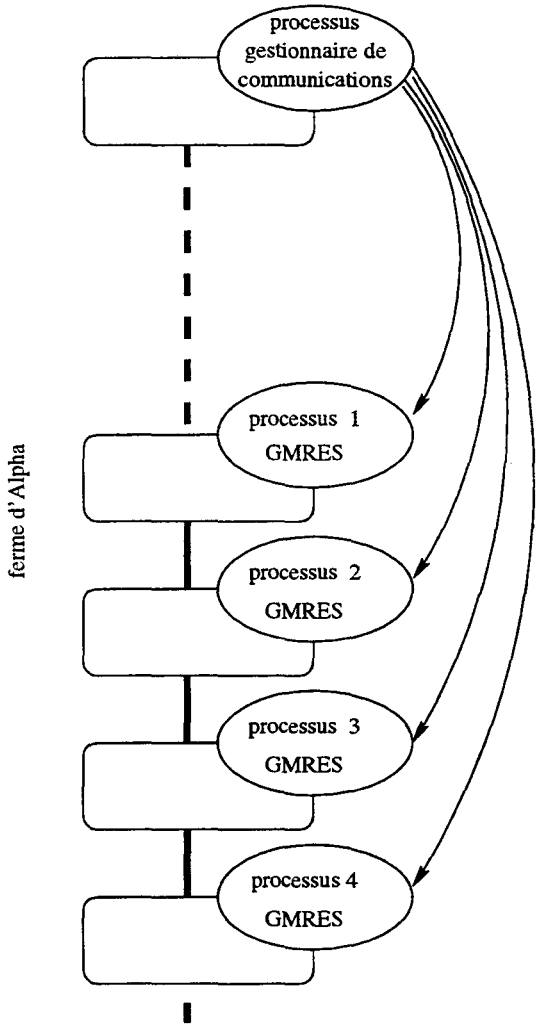

FIG. 3.1 - *Le schéma général de l'implantation de GMRES(m)*

Au contraire, en augmentant le nombre de processeurs le nombre des communications augmente considérablement. Dans ce cas, les données transférées sont moins importantes, mais le temps total des communications effectuées augmente. Le temps des communications est donc croissant en fonction du nombre de processeurs.
Donc, il existe un nombre de processeurs optimal p_{opt} qui donne un temps total minimal d'exécution (voir figure 3.2). Le nombre p_{opt} dépend de la taille et de la structure de la matrice A, ainsi que de la taille du sous-espace de Krylov m, ce qui veut dire qu'augmenter le nombre de processeurs au-dessus de p_{opt} handicape l'algorithme. Nous en déduisons donc que le degré de parallélisme est limité dans la méthode de GMRES(m).

Dans la figure 3.2, nous présentons la courbe représentant le temps total de-

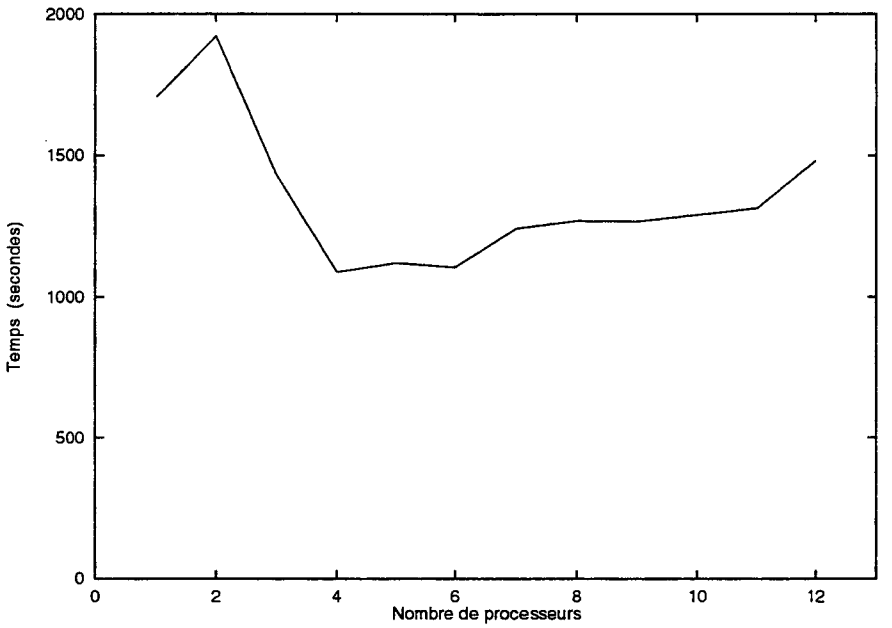

FIG. 3.2 - *Le temps total de GMRES(400) pour atteindre la convergence tracé en fonction du nombre de processeurs (la matrice "utm1700a").*

mandé pour atteindre la convergence de GMRES(m) en fonction du nombre de processeurs. Ces résultats sont obtenus sur la ferme d'Alpha 1.3.2 en utilisant le langage Fortran 77 et la bibliothèque PVM pour effectuer les communications. Nous avons pris la matrice de test "utm1700a" de dimensions 1700×1700 et $m = 400$ comme taille du sous-espace de Krylov de projection. Le nombre optimal de processeurs correspondant à cet exemple est donc $p_{opt} = 4$, nous remarquons que cette valeur est très limitée.
Nous remarquons aussi que pour $p = 2$ le temps des communications nécissité n'est pas compensé par le gain du temps de calcul dû au rajout d'un processeur, le programme exécuté en séquentiel est donc meilleur que ce cas particulier. À partir du nombre de processeurs $p = 3$ la parallèlisation de GMRES(m) est utile dans cet exemple.

Implantation en mode SIMD

Nous l'avons vu auparavant, l'algorithme de la méthode GMRES nécessite la résolution du problème de minimisation qui est complètement séquentielle, son exécution doit obligatoirement être effectuée sur une machine séquentielle. Ainsi, nous pouvons confier cette partie à la frontal de la machine parallèle cible ou à une station puissante (Sun par exemple). La partie parallèle est bien sûr exécutée sur l'unité

data parallèle de la machine SIMD.

La matrice A est distribuée de telle sorte que son format de stockage soit adapté à l'architecture de la machine parallèle pour faciliter les multiplications matrice-vecteur. Le format que nous utilisons est le format SGP décrit dans la Section 1.5.

3.2 Méthode d'Arnoldi pour le calcul des valeurs propres

Nous procédons maintenant à la description de la méthode d'Arnoldi pour le calcul des valeurs propres d'une matrice réelle. Cette méthode admet beaucoup de similitudes avec la méthode GMRES que nous venons de décrire. Ces deux méthodes procèdent au calcul des approximations voulues à partir de la projection sur un sous-espace de Krylov.

3.2.1 Description

La méthode d'Arnoldi est l'une des méthodes les plus connues et utilisées pour le calcul des valeurs propres des matrices creuses de très grande taille. Cette méthode a été initialement proposée par Arnoldi en 1951 [1].

Cette méthode calcule des approximations des valeurs propres de A dans le sous-espace de Krylov $K_m(A, v)$. Le processus d'Arnoldi réduit la matrice A à une matrice de Hessenberg supérieure H_m sur $K_m(A, v)$. Notons $(\lambda_i, y_i)_{1 \leq i \leq m}$ les paires respectivement des valeurs propres et vecteurs propres de H_m. Ces paires sont calculées par la méthode de décomposition puisque la valeur de m est en général assez petite. Les valeurs λ_i sont des approximations des valeurs propres de A et sont appelées les valeurs de Ritz de A. Les vecteurs $u_i = V_m y_i$ sont des approximations des vecteurs propres de A et sont appelés les vecteurs de Ritz de A.

D'une part la convergence de cette méthode ne peut pas être atteinte facilement pour une valeur assez petite de m et d'autre part le nombre de vecteurs à stocker croît avec m, donc la valeur de m ne doit pas dépasser un certain seuil. Ce problème est surmonté par le redémarrage de l'algorithme après un certain nombre m d'itérations avec un vecteur v combinaison linéaire des parties réelles des vecteurs de Ritz u_i

associés aux d valeurs propres cherchées

$$v = \sum_{i=1}^{d} \rho_i \Re(u_i).$$

Divers choix des coefficients ρ_i sont possibles. Le choix proposé par Saad [60] est de prendre les normes des résidus des valeurs propres ($\rho_i = \|\lambda_i u_i - A u_i\|_2$).

Ainsi, la méthode d'Arnoldi pour le calcul de d valeurs propres dominantes ainsi que de leurs vecteurs propres associés peut être décrite par l'algorithme suivant

Algorithme 3.2.1 : *La méthode d'Arnoldi*

1. *Initialisation : choisir v un vecteur initial,*
   m la dimension des sous-espaces de Krylov,
   d le nombre de valeurs propres dominantes voulues
   et ε la précision demandée.

2. *Appliquer le processus d'Arnoldi à $K_m(A, v)$.*

3. *Calculer les valeurs propres (λ_i, $1 \leq i \leq d$) et les vecteurs propres associés (y_i, $1 \leq i \leq d$) de H_m.*

4. *Calculer les vecteurs de Ritz $u_i = V_m y_i$, pour $i = 1, \ldots, d$.*

5. *Calculer $\rho_i = \|\lambda_i u_i - A u_i\|_2$, $1 \leq i \leq d$.*

6. *Redémarrage : si $\max_{i=1,d} |\rho_i| < \varepsilon$ stop*
   sinon poser $v = \sum_{i=1}^{d} \rho_i \Re(u_i)$ et aller en 2.

Dans le cas où l'on voudrait calculer un plus grand nombre de valeurs propres, nous pouvons soit déflater la matrice A [58], soit redémarrer avec un vecteur v orthogonal aux vecteurs propres déjà calculés, voir [60].

3.2.2 Méthode d'Arnoldi parallèle

Les algorithmes des deux méthodes GMRES et Arnoldi sont constitués de plusieurs parties qui ont les mêmes natures. Ces deux algorithmes débutent par la

projection d'Arnoldi qui est parallèle, ensuite il y a une partie séquentielle, la minimisation pour GMRES et le calcul des valeurs et vecteurs propres de la matrice H_m pour Arnoldi. Cela est dû au fait que les nombres des données traitées (les matrices $\overline{H}_m$ et H_m) par ces deux parties ne sont pas très grands, la majorité du temps d'un traitement parallèle de ces calculs passera dans le très grand nombre des communications générées. Notons que la partie séquentielle d'Arnoldi est plus coûteuse que celle de GMRES.
Après ces parties séquentielles, il y a ensuite le calcul de la solution x_m pour GMRES et le calcul des vecteurs de Ritz pour Arnoldi. À la fin, les deux méthodes procédent au redémarrage si la convergence n'est pas atteinte. Notons que les deux dernières parties des deux méthodes peuvent être effectuées en parallèle.
Nous remarquons donc que les parallélisations de la méthode d'Arnoldi sur les deux architectures de machines parallèles sont semblables à celles de GMRES décrites précédement.

La version du processus d'Arnoldi que nous utilisons pour implanter la méthode d'Arnoldi est celle basée sur Gram-Schmidt classique GSC rendu plus stable à l'aide de la technique de réorthogonalisation (voir Section 2.7).

3.3 Méthode Moindres Carrés

La méthode Moindres Carrés (*Least Squares*) que nous notons LS, est utilisée essentiellement pour calculer le préconditionnement polynômial ou hybrider (combiner deux méthodes) une méthode de résolution de systèmes linéaires. Comme la méthode de Tchebyshev [45], la méthode LS tente d'approcher l'inverse d'une matrice A par $P(A)$, où P est un polynôme minimisant une certaine norme de la matrice $I - AP(A)$. Dans la suite nous allons commencer par une brève description de la méthode de Tchebyshev, ensuite nous décrirons la méthode LS.

Notons $\mathcal{P}_j$ l'ensemble des polynômes de degré j. La méthode de Tchebyshev détermine le polynôme $P_k \in \mathcal{P}_{k-1}$ solution du problème suivant

$$\min_{P_k \in \mathcal{P}_{k-1}} \max_{\lambda \in \mathcal{E}(c,d,a)} |1 - \lambda P_k(\lambda)|, \tag{3.3}$$

où $\mathcal{E}(c,d,a)$ est une ellipse contenant les approximations des valeurs propres de la matrice A.

Le polynôme résiduel $R_k(\lambda) = 1 - \lambda P_k(\lambda)$ n'est autre que le polynôme de Tche-

byshev suivant

$$1 - \lambda P_k(\lambda) = T_k\left(\frac{c-\lambda}{d}\right) / T_k\left(\frac{c}{d}\right),$$

où T_k est un polynôme de Tchebyshev du premier type et de degré k.

Le facteur de convergence de la méthode de Tchebyshev est donné par le rapport suivant

$$\frac{(c-\lambda) + \sqrt{((c-\lambda)^2 - d^2)}}{c + \sqrt{(c^2 - d^2)}}.$$

Ce facteur dépend des paramètres de l'ellipse $\mathcal{E}(c, d, a)$. Afin d'améliorer la convergence de cette méthode l'ellipse $\mathcal{E}(c, d, a)$ est donc calculée de telle sorte qu'elle minimise ce facteur de convergence. Un algorithme calculant cette ellipse optimale est donné dans [45] et [58].

Nous remarquons que si l'origine est à l'intérieur de l'ellipse, le minimum du problème (3.3) est au moins égal à 1, tandis que nous voulons tendre ce minimum vers zéro, ce qui donne un mauvais polynôme moindres carrés. Pour éviter cet inconvénient de la méthode Tchebyshev, la méthode LS prend comme domaine contenant l'approximation du spectre l'enveloppe convexe de cette approximation. Dans ce cas, nous avons une bonne minimisation du polynôme résiduel sur un domaine contenant les valeurs propres sans l'origine, comme nous allons le voir dans la suite.

3.3.1 Description

Comme dans toutes les méthodes de Krylov, l'itéré de la méthode LS s'écrit sous la forme

$$\tilde{x} = x_0 + P_k(A) r_0 \tag{3.4}$$

où x_0 est une approximation initiale de la solution x^* du système linéaire, r_0 le résidu correspondant et P_k un polynôme de degré $k-1$.

Soit $\mathcal{P}_k^1$ le sous-espace des polynômes réels défini par

$$\mathcal{P}_k^1 = \{p \in \mathcal{P}_k, \text{ tel que } p(0) = 1\},$$

et soit $R_k \in \mathcal{P}_k^1$ le polynôme défini par $R_k(z) = 1 - zP_k(z)$.

Le résidu de l'itéré $\tilde{x}$ s'écrit alors

$$\tilde{r} = R_k(A) r_0.$$

Supposons que A soit diagonalisable et notons $\sigma(A) = \{\lambda_1, \cdots, \lambda_n\}$ son spectre associé à la base formée des vecteurs propres $\{u_1, u_2, \cdots, u_n\}$.

Le résidu initial r_0 peut être exprimé dans cette base sous la forme

$$r_0 = \sum_{i=1}^{n} \rho_i u_i,$$

et ainsi $\tilde{r}$ peut être réécrit comme suit

$$\tilde{r} = \sum_{i=1}^{n} \rho_i R_k(\lambda_i) u_i.$$

Cette dernière égalité nous permet d'obtenir une majoration de la norme euclidienne de $\tilde{r}$ donnée par

$$\|\tilde{r}\|_2 \leq \|r_0\|_2 \max_{\lambda \in \sigma(A)} |R_k(\lambda)|. \tag{3.5}$$

Cette inégalité permet de voir que pour minimiser $\|\tilde{r}\|_2$, un choix naturel de R_k est le polynôme qui minimise le terme de droite de cette inégalité (3.5) majorant de $\|\tilde{r}\|_2$. Ainsi nous considérons R_k comme solution du problème min-max suivant

$$\min_{R_k \in \mathcal{P}_k^1} \max_{\lambda \in \sigma(A)} |R_k(\lambda)|. \tag{3.6}$$

Sachant qu'en général, nous ne disposons pas du spectre complet de A, mais seulement de quelques estimations de certaines valeurs propres, nous nous rendons compte que nous ne pouvons pas résoudre le problème (3.6). Cependant, nous pouvons considérer le problème suivant

$$\min_{R_k \in \mathcal{P}_k^1} \max_{\lambda \in H} |R_k(\lambda)|, \tag{3.7}$$

où H est l'enveloppe convexe contenant les estimations des valeurs propres calculées par la méthode d'Arnoldi (ou par une autre méthode quelconque). Dans le cas où nous prenons pour H une ellipse, nous obtenons la méthode de Tchebyshev, comme nous l'avons vu précédemment.
Sachant que la matrice A est réelle, les valeurs propres sont conjuguées, H est donc symétrique par rapport à l'axe des abscisses.

Afin d'éviter d'inclure l'origine dans le domaine H, nous construisons donc H comme réunion de deux convexes H_1 et H_2, chacun est situé dans un demi-plan complexe (voir figure 3.3). Le domaine H_1 (respectivement H_2) est le convexe contenant les valeurs propres de partie réelle négative (respectivement positive).

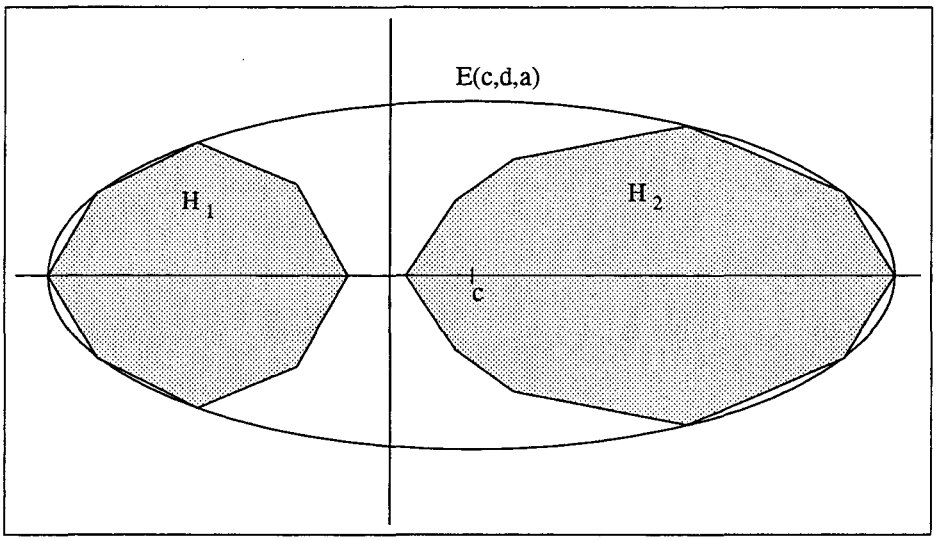

FIG. 3.3 - *Le convexe et l'ellipse contenant le spectre*

Remarquons de plus que la valeur de $\max_{\lambda \in H} |R_k(\lambda)|$ est atteinte sur la frontière ∂H de H et ainsi (3.7) devient

$$\min_{R_k \in \mathcal{P}_k^1} \max_{\lambda \in \partial H} |R_k(\lambda)|. \tag{3.8}$$

Ce problème min-max étant difficile à résoudre, Smolarski et Saylor [69] ont proposé de remplacer la norme uniforme (i.e. $\max_{\lambda \in H} |R_k(\lambda)|$) par la norme L_2 $\|.\|_w$ sur l'espace des polynômes réels, affectée d'une fonction poids w bien appropriée. Nous obtenons ainsi le problème aux moindres carrés suivant

$$\min_{R_k \in \mathcal{P}_k^1} \|R_k\|_w. \tag{3.9}$$

Notons ∂H^+ la partie supérieure de ∂H et E_ν ($\nu = 1, \cdots, \mu$) ses côtés, ainsi nous avons $\partial H^+ = \cup_{\nu=1}^{\mu} E_\nu$. Soit c_ν et d_ν respectivement le centre et la demi-longueur du côté E_ν.
Le produit scalaire $(.,.)_w$ sur l'espace des polynômes réels associé à la norme $\|.\|_w$ est défini après simplification par

$$(p,q)_w = 2\Re\left(\sum_{\nu=1}^{\mu} \int_{E_\nu} p(\lambda)\overline{q(\lambda)} w_\nu(\lambda) d\lambda\right), \tag{3.10}$$

où w_ν est la restriction de w sur le côté E_ν, notons que w_ν est défini par

$$w_\nu(\lambda) = \frac{2}{\pi} |d_\nu^2 - (\lambda - c_\nu)^2|^{-1/2}. \tag{3.11}$$

Cette fonction est la fonction poids généralisée sur un segment (E_ν) complexe associée à la base des polynômes de Tchebyshev $T_i\left(\frac{\lambda-c_\nu}{d_\nu}\right)$. Cette propriété facilite le calcul du produit scalaire. Exprimons p et q dans cette base sur le segment E_ν par

$$p(\lambda) = \sum_{i=0}^{n} \xi_i^{(\nu)} T_i\left(\frac{\lambda - c_\nu}{d_\nu}\right) \qquad q(\lambda) = \sum_{i=0}^{n} \zeta_i^{(\nu)} T_i\left(\frac{\lambda - c_\nu}{d_\nu}\right), \tag{3.12}$$

nous avons alors

$$(p,q)_w = 2\Re\left[\sum_{\nu=1}^{\mu}\left(2\xi_0^{(\nu)}\overline{\zeta_0^{(\nu)}} + \sum_{i=1}^{n}\xi_i^{(\nu)}\overline{\zeta_i^{(\nu)}}\right)\right]. \tag{3.13}$$

Soit $(t_j)_{j\geq 0}$ la base de Tchebyshev sur l'ellipse $\mathcal{E}(c,d,a)$ définie comme suit

$$t_j(\lambda) = T_j\left(\frac{\lambda - c}{d}\right) / T_j\left(\frac{a}{d}\right) \qquad j = 0, 1, \cdots \tag{3.14}$$

Cette base de polynômes est la meilleure base sur l'ellipse $\mathcal{E}(c,d,a)$ de surface minimale contenant H (voir figure 3.3), pour une raison que nous allons citer ultérieurement. Un algorithme calculant cette ellipse optimale peut être inspiré de celui donné par Manteuffel [45].

De plus, cette base vérifie une relation de récurrence à trois termes de la forme

$$\beta_{i+1} t_{i+1}(z) = (z - \alpha_i) t_i(z) - \delta_i t_{i-1}(z) \qquad i = 0, 1, \cdots \tag{3.15}$$

L'utilisation de la base des $(t_j)_{j\geq 0}$ définie par (3.14) nous permet d'obtenir une matrice $M_k = (m_{i,j})$ bien conditionnée [62] appelée matrice de Gram modifiée. Les composantes de M_k sont définies par

$$m_{i,j} = (t_{i-1}, t_{j-1})_w \qquad i,j \in \{1, \cdots, k+1\}. \tag{3.16}$$

Le produit scalaire (3.13) s'écrit alors

$$(p,q)_w = (M_k \eta, \theta). \tag{3.17}$$

où $\eta = (\eta_1, \ldots, \eta_k)$ et $\theta = (\theta_1, \ldots, \theta_k)$ sont les vecteurs des coordonnées des polynômes p et q dans la base des t_i.

Procédons maintenant au calcul de P_k et exprimons ce polynôme dans la base des Tchebyshev comme suit

$$P_k = \sum_{i=0}^{k-1} \eta_i t_i,$$

alors

$$R_k(\lambda) = 1 - \lambda P_k(\lambda) = 1 - \sum_{i=0}^{k-1} \eta_i \lambda t_i(\lambda).$$

Or, d'après la relation de récurrence (3.15) et sachant que $t_0 = 1$, nous pouvons écrire

$$\begin{aligned} R_k(\lambda) &= t_0 - \sum_{i=0}^{k-1} \eta_i(\beta_{i+1}t_{i+1} + \alpha_i t_i + \delta_i t_{i-1}), \\ &= t_0 - \sum_{i=0}^{k}(\beta_i\eta_{i-1} + \alpha_i\eta_i + \delta_{i+1}\eta_{i+1})t_i. \end{aligned}$$

Ainsi, R_k a pour coordonnées $(e_1 - T_k\eta)$ dans la base des polynômes t_i, où T_k est la matrice de taille $(k+1) \times k$ définie par

$$T_k = \begin{bmatrix} \alpha_0 & \delta_1 & & & & \\ \beta_1 & \alpha_1 & \delta_2 & & & \\ & \beta_2 & \alpha_2 & . & & \\ & & . & . & . & \\ & & & . & . & \delta_{k-1} \\ & & & & \beta_{k-1} & \alpha_{k-1} \\ & & & & & \beta_k \end{bmatrix},$$

et alors la fonctionnelle à minimiser s'écrit

$$\|R_k\|_w = (R_k, R_k)_w = (M_k(e_1 - T_k\eta), (e_1 - T_k\eta)).$$

La matrice M_k étant symétrique, nous pouvons la décomposer sous la forme $M_k = L_k L_k^T$, nous obtenons alors

$$\begin{aligned} \|R_k\|_w &= (L_k^T(e_1 - T_k\eta), L_k^T(e_1 - T_k\eta)) && (3.18) \\ &= \|L_k^T(e_1 - T_k\eta)\|_2 && (3.19) \\ &= \|l_{1,1}e_1 - F_k\eta\|_2, && (3.20) \\ & && (3.21) \end{aligned}$$

où $F_k = L_k^T T_k$ est une matrice Hessenberg supérieure de taille $(k+1) \times k$.

Ainsi, les coefficients η_i sont solution du problème de minimisation suivant

$$\min_{\eta \in \mathbb{R}^k} \|l_{1,1}e_1 - F_k\eta\|_2.$$

C'est un problème similaire au problème (3.2) rencontré dans la méthode GMRES, que nous résolvons à l'aide de la factorisation QR.

3.3.2 Calcul des coefficients de la relation de récurrence à trois termes

Les polynômes t_i de Tchebyshev sur l'ellipse $\mathrm{E}(c,d,a)$ sont définis par

$$t_i(\lambda) = T_i\left(\frac{\lambda - c}{d}\right) / T_i\left(\frac{a}{d}\right), \tag{3.22}$$

où les T_i sont les polynômes de Tchebyshev de première espèce vérifiant la relation de récurrence

$$T_{i+1}(\xi) = 2\xi T_i(\xi) - T_{i-1}(\xi). \tag{3.23}$$

A l'aide du changement de variable $\xi = \dfrac{\lambda - c}{d}$, nous obtenons

$$T_{i+1}(\xi) = 2\left(\frac{\lambda - c}{d}\right) T_i(\xi) - T_{i-1}(\xi), \tag{3.24}$$

et d'après (3.22) nous avons

$$T_{i+1}(a/d)t_{i+1}(\xi) = 2\left(\frac{\lambda - c}{d}\right) T_i(a/d)t_i(\xi) - T_{i-1}(a/d)t_{i-1}(\xi), \tag{3.25}$$

d'où nous obtenons la relation de récurrence à trois termes

$$\beta_{i+1}t_{i+1}(\xi) = (\lambda - \alpha_i)t_i(\lambda) - \delta_i t_{i-1}(\lambda), \tag{3.26}$$

avec

$$\begin{cases} \alpha_i &= c \\ \beta_{i+1} &= \dfrac{d}{2}\dfrac{T_{i+1}(a/d)}{T_i(a/d)} \\ \delta_i &= \dfrac{d}{2}\dfrac{T_{i-1}(a/d)}{T_i(a/d)} \end{cases}.$$

En remplaçant ξ dans (3.23) par la valeur a/d nous obtenons

$$T_{i+1}(a/d) = 2\frac{a}{d}T_i(a/d) - T_{i-1}(a/d), \tag{3.27}$$

et nous en déduisons les relations

$$\begin{cases} \delta_i &= \dfrac{d^2}{4\beta_i}, \\ \beta_{i+1} &= a - \delta_i, \end{cases}$$

avec les initialisations $\delta_0 = 0$ et $\beta_1 = a/2$.
D'après ces relations, nous remarquons que les quantités β_{i+1} et δ_i sont réelles même si d est complexe, ce qui facilite le calcul de ces quantités.

3.3.3 Calcul de l'itéré de la méthode LS

Nous nous proposons de calculer l'itéré $\tilde{x} = x_0 + P_k(A)r_0$, où P_k est le polynôme LS de degré $k-1$ qui s'écrit dans la base des polynômes t_i $(i = 0, \ldots, k-1)$ sous la forme $P_k = \sum_{i=0}^{k-1} \eta_i t_i$.

Les polynômes t_i vérifient la relation de récurrence à trois termes suivante

$$t_{i+1}(\lambda) = \frac{1}{\beta_{i+1}}[\lambda t_i(\lambda) - \alpha_i t_i(\lambda) - \delta_i t_{i-1}(\lambda)]. \tag{3.28}$$

Définissons les vecteurs $w_i \in \mathbb{R}^n$ par $w_i = t_i(A)r_0$; la relation ci-dessus induit la relation de récurrence liant ces vecteurs

$$w_{i+1} = \frac{1}{\beta_{i+1}}(Aw_i - \alpha_i w_i - \delta_i w_{i-1}). \tag{3.29}$$

Or, l'itéré $\tilde{x}$ s'écrit en fonction des vecteurs w_i sous la forme

$$\begin{aligned} \tilde{x} &= x_0 + P_k(A)r_0, \\ &= x_0 + \sum_{i=0}^{k-1} \eta_i w_i, \end{aligned}$$

nous obtenons donc l'algorithme suivant

Algorithme 3.3.1 :
$r_0 = b - Ax_0$,
$w_{-1} = 0, \quad w_0 = r_0, \quad \delta_0 = 0, \quad \tilde{x} = x_0 + \eta_0 w_0$.
pour $i = 0, \; k-2$
$w_{i+1} = \frac{1}{\beta_{i+1}}(Aw_i - \alpha_i w_i - \delta_i w_{i-1})$,
$\tilde{x} = \tilde{x} + \eta_{i+1} w_{i+1}$,
fin pour.

Remarque 3.3.1

Notons que la méthode LS est composée de deux parties, la première est le calcul des paramètres LS (α_i, β_i, δ_i et η_i), cette partie est complètement séquentielle. La deuxième partie est le calcul de l'itéré de LS, ce calcul est complètement parallèle

puisqu'il consiste à appliquer un polynôme matriciel à un vecteur.
Cette méthode donc peut être implantée sur un réseau hétérogène composé d'une machine parallèle et une autre séquentielle.

3.4 Conclusion

Dans ce chapitre, nous avons décrit brièvement les méthodes GMRES, Arnoldi et Moindres Carrés. Nous avons également étudié les parallélisations de ces méthodes sur les machines parallèles MIMD et SIMD.

Dans la méthode GMRES, d'une itération à l'autre les valeurs de Ritz s'approchent de plus en plus vers les valeurs propres de A correspondantes. Ainsi, dans l'expression du résidu les composantes des vecteurs propres correspondants à ces valeurs propres tendent vers zéro. La méthode Moindres Carrés est basée sur ce même principe, la combinaison de cette méthode avec GMRES conduit donc à l'accélération de la convergence de GMRES vers x^* la solution du système linéaire.

Nous avons vu que le degré de parallélisme de gros grains dans la méthode GMRES(m) est limité. Dans le chapitre suivant nous allons hybrider cette méthode par la méthode Moindres Carrés, en vue d'obtenir une méthode hybride dont le degré de parallélisme est plus grand que celui de GMRES(m) et qui converge plus vite. Les approximations des valeurs propres nécessitées par la méthode Moindres Carrés seront calculées par la méthode d'Arnoldi redémarrée.

Chapitre 4

La méthode hybride parallèle hétérogène GMRES(m_1)/LS(k, l)-Arnoldi(m_2)

Nombreuses sont les applications scientifiques nécessitant la résolution de systèmes linéaires de la forme

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

où A est une matrice réelle de dimensions $n \times n$, $b \in \mathbb{R}^n$ un vecteur réel et posons x^* la solution exacte du système.

Pour résoudre de tels systèmes on a souvent recours à des méthodes itératives parallèles telles que la méthode GMRES(m) [66] décrite précédemment dans la Section 3.1. Comme nous l'avons déjà signalé, la convergence de cette méthode peut être très lente lorsque la valeur de m est très petite et parfois la méthode peut même stagner. Rappelons aussi (voir Section 3.1.2), que le degré de parallélisme dans GMRES(m) est limité ; même si nous disposons de plus de processeurs qu'un certain seuil p_{opt}, nous ne pouvons pas diminuer le temps d'exécution de GMRES(m). Afin de remédier à ces problèmes et d'accélérer la convergence de GMRES, nous pouvons soit préconditionner la méthode GMRES (préconditionnement basé sur la décomposition LU incomplète, Préconditionnement polynomial, ...) [59] soit lui associer d'autres méthodes qui peuvent être éventuellement parallèles telles que les méthodes Tchebyshev ou Moindres Carrés.

Elman, Saad et Saylor ont proposé une méthode hybride GMRES/Tchebyshev [22]. Dans cette méthode, une itération GMRES(m) est effectuée pour obtenir une approximation de la solution et la matrice H_m (construite par le processus d'Arnoldi

appliqué au sous-espace de Krylov $K_m(A, r_0)$) est utilisée pour déterminer des approximations des valeurs propres de A (voir la Section 3.2). Ces valeurs sont utilisées pour effectuer une itération Tchebyshev et obtenir ainsi une nouvelle approximation de la solution x^*. La méthode hybride est redémarrée tant que la précision imposée n'est pas atteinte.

Plus tard, Saad [63] a proposé de remplacer la méthode Tchebyshev par la méthode Moindres Carrés (voir Section 3.3) qui est plus précise que la précédente. La méthode LS n'a pas l'inconvénient de la méthode Tchebyshev quand les valeurs propres sont situées dans les deux demi-plans complexes séparés par l'axe des imaginaires purs. Cette méthode évite cet handicap en prenant comme domaine de minimisation la réunion de deux convexes, chacun englobant les valeurs propres situées dans un demi-plan complexe. Dans la suite nous nous intéresserons particulièrement à la méthode hybride utilisant la méthode Moindres Carrés.

Nous allons adapter cette méthode GMRES(m)/LS(k,l) pour obtenir une version dans laquelle l'algorithme d'Arnoldi est rendu indépendant de GMRES(m), afin de calculer les valeurs propres avec une bonne précision, que nous notons GMRES(m_1)/LS(k,l)-Arnoldi(m_2). Cette version, étant constituée de parties ayant moins de dépendances entre elles, a l'avantage d'être plus adaptée au parallélisme.

La nature hybride de cette association d'algorithmes, qui est plus coûteuse que GMRES tout seul, demande l'utilisation intensive du parallélisme, ce qui nous permet d'augmenter le degré de parallélisme de gros grains qui est limité dans GMRES. La diversité de la nature du parallélisme des différentes parties de la méthode hybride nous permet de répartir les calculs sur des machines de différentes architectures, ces calculs sont exécutés - quand c'est possible - d'une façon asynchrone. Nous obtenons donc un parallélisme hétérogène [13] qui nous permet de tirer profit des ressources informatiques séquentielles ou parallèles disponibles et ainsi accélérer considérablement la convergence vers la solution x^*.

Nous commencerons ce chapitre par une description et une analyse de la convergence de la méthode hybride GMRES(m)/LS(k,l) dans la Section 4.1. Ensuite, nous présenterons dans la Section 4.2 la méthode hybride parallèle GMRES(m_1)/LS(k,l)-Arnoldi(m_2) [7], [24]. Nous avons réalisé et étudié deux implantations de cette méthode. La première implantation est asynchrone (voir Section 4.3.3), tandis que la seconde est entrelacée (voir Section 4.4). Ces implantations de la méthode hybride parallèle seront faites sur deux réseaux hétérogènes de machines parallèles. Chaque implantation sera suivie d'une série d'expérimentations numériques pour montrer ses bonnes performances par rapport à la méthode GMRES(m_1) implantée toute seule sur le même réseau.

4.1 La méthode hybride GMRES(m)/LS(k, l)

4.1.1 Description

Cette méthode hybride combine les méthodes GMRES(m) et Moindres Carrés notée LS(k, l). L'entier naturel k est tel que le polynôme des moindres carrés P_k soit de degré $k-1$ et l représente le nombre de fois que nous appliquons cette méthode itérativement.

Au début, GMRES(m) calcule l'itéré x_m qui est une solution approchée du système (4.1). Or, la méthode LS nécessite le calcul de quelques valeurs propres de la matrice A. La projection d'Arnoldi dans GMRES(m) est donc exploitée pour obtenir des approximations de ces valeurs.

La méthode LS procède au calcul du convexe H et de l'ellipse E(c, d, a) optimale qui contiennent toutes les approximations des valeurs propres déjà calculées. Ces paramètres servent pour construire le polynôme des moindres carrés de degré $k-1$. Ce polynôme est exprimé dans la base des Tchebyshev sur l'ellipse E(c, d, a) (voir Section 3.3).

L'algorithme (3.3.1) décrit dans le chapitre précédent est utilisé pour calculer l'itéré de la méthode LS, en prenant comme point initial la dernière solution approchée x_m obtenue par GMRES(m). La méthode hybride est ensuite redémarrée itérativement tant que la précision voulue n'est pas atteinte.

Ainsi, l'algorithme suivant décrit la méthode hybride GMRES(m)/LS(k, l) séquentielle.

Algorithme 4.1.1 : *GMRES(m)/LS(k,l) séquentiel*

1. *Initialisation : choisir x_0, m la dimension du sous-espace de Krylov,*
   k le degré du polynôme des moindres carrés,
   ε la tolérance et l le nombre d'itérations Moindres Carrés appliquées successivement,
   calculer $r_0 = b - Ax_0$.

2. *Effectuer m itérations du processus d'Arnoldi en démarrant avec r_0,*

3. *Calculer x_m, le $m^{ième}$ itéré de GMRES en prenant x_0 comme point initial, si $\|b - Ax_m\|_2 < \varepsilon$ Stop sinon poser $x_0 = x_m$ et $r_0 = b - Ax_0$.*

4. *Calculer les valeurs propres de H_m.*

5. *Calculer le polynôme des moindres carrés P_k sur la frontière de H l'enveloppe convexe de toutes les valeurs propres calculées.*

6. *Pour $j = 1, \cdots, l$*
   Calculer $\tilde{x} = x_0 + P_k(A)r_0$, et poser $x_0 = \tilde{x}$, $r_0 = b - Ax_0$.
   fin pour

7. *Redémarrage : si $\|r_0\|_2 < \varepsilon$ Stop sinon aller en 2.*

4.1.2 Analyse de la convergence

Afin de faciliter l'étude du comportement de la méthode hybride, nous supposons que la matrice A est diagonalisable de spectre $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$. Nous supposons aussi que le convexe H actuel ne contient que les valeurs propres suivantes : $\lambda_1, \ldots, \lambda_s$.

Rappelons la définition du polynôme résiduel R_k de la méthode LS de degré k, pour tout $\lambda \in \mathbb{R}$, $R_k(\lambda) = 1 - \lambda P_k(\lambda)$.

Le résidu de l'itéré de la méthode LS(k,l) s'écrit dans la base $\{u_1, u_2, \ldots, u_n\}$ des vecteurs propres de A comme suit

$$\tilde{r} = (R_k(A))^l r_0,$$

$$= \sum_{i=1}^{n} \rho_i (R_k(\lambda_i))^l u_i,$$

en supposant que le vecteur résidu initial s'exprime sous la forme $r_0 = \sum_{i=1}^{n} \rho_i u_i$.

Nous décomposons cette somme pour séparer les valeurs propres selon leur position par rapport à H

$$\tilde{r} = \sum_{i=1}^{s} \rho_i (R_k(\lambda_i))^l u_i + \sum_{i=s+1}^{n} \rho_i (R_k(\lambda_i))^l u_i. \tag{4.2}$$

Nous avons vu dans la Section 3.3 que R_k est déterminé par la méthode des moindres carrés de telle sorte qu'il minimise $|R_k(\lambda)|$ sur H selon la L_2-norme de fonction poids w. Ainsi, le premier terme du résidu $\tilde{r}$ est très petit par rapport au second. Pour cette raison les vecteurs propres associés aux valeurs propres situées à l'extérieur du convexe H deviennent dominants dans l'expression du résidu $\tilde{r}$. Le nombre de vecteurs propres composant le résidu $\tilde{r}$ diminue au fur et à mesure que le convexe H courant se rapproche du convexe exacte (le convexe englobant les points du spectre $\sigma(A)$ de la matrice A).

Nous remarquons aussi, que lorsque l croît, le premier terme tend vers zéro, mais le second terme devient de plus en plus grand ; cela induit que la norme du résidu $\tilde{r}$ peut être très grande par rapport à celle du résidu initial r_0.

Ainsi, GMRES(m_1) redémarré avec l'itéré de la méthode Moindres Carrés, dont le résidu $\tilde{r}$ est combinaison d'un petit nombre de vecteurs propres, converge plus vite même si la norme du résidu $\tilde{r}$ est très grande.

4.2 La méthode hybride parallèle

Afin de paralléliser efficacement une application, la démarche la plus courante consiste d'abord à choisir une version telle que son algorithme soit constitué en plusieurs tâches le plus possible indépendantes entre elles. Dans ce cas, les calculs seront exécutés en parallèle sur plusieurs machines avec un coût minimal des communications entre les différentes tâches.

Dans la méthode hybride, nous avons besoin de calculer des valeurs propres de la matrice A. Si nous voulons des valeurs propres calculées avec une précision donnée,

le processus d'Arnoldi doit être redémarré avec un vecteur approprié. Cela n'est pas possible dans la version séquentielle décrite par l'algorithme 4.1.1 puisque cet algorithme procède au calcul des valeurs propres à partir de la projection d'Arnoldi de l'algorithme GMRES. Nous devons donc calculer les valeurs propres à l'aide de la méthode d'Arnoldi avec sa propre projection d'Arnoldi. Afin de contrôler le calcul des valeurs propres et d'avoir une méthode hybride plus souple, nous prenons une dimension m_2 du sous-espace de projection de la méthode d'Arnoldi qui peut être différente de m_1 celle de GMRES.

Le calcul de l'itéré de la méthode LS se fait à partir de celui de GMRES et il est complètement parallèle puisqu'il ne nécessite que des produits matrice-vecteur. Cette tâche doit donc s'exécuter alternativement avec l'algorithme GMRES(m_1).

La méthode hybride comprend trois grands groupes de tâches, indépendants entre eux, mais qui ont besoin de communiquer après un certain temps. Ces grands groupes de tâches, sont les suivants

1. GMRES(m_1) : contient la projection et les calculs des solutions x_{m_1} itéré de GMRES(m_1) et $\tilde{x}$ itéré de LS(k,l), qui sont parallèles, et la résolution du problème de minimisation à l'aide de la factorisation QR qui est séquentielle,

2. Arnoldi(m_2) : contient la projection, le calcul des valeurs et vecteurs propres de la matrice H_{m_2} par la méthode de décomposition qui est séquentielle et le calcul des vecteurs de Ritz qui est parallèle,

3. le calcul du convexe et de l'ellipse contenant les estimations des valeurs propres et ensuite le calcul des coefficients du polynôme des moindres carrés ; ce processus est complètement séquentiel.

Nous remarquons que cette méthode hybride est composée de tâches parallèles et d'autres séquentielles, d'où la possibilité de l'utilisation du parallélisme hétérogène pour implanter cette méthode. Les parties parallèles de GMRES(m_1) (avec le calcul des itérés LS) et Arnoldi(m_2) s'exécuteront sur des machines parallèles. Les parties séquentielles de ces deux méthodes, ainsi que le calcul des paramètres LS s'exécuteront sur des machines séquentielles [19, 53].

Plusieurs organisations de la méthode hybride parallèle sont possibles. Nous allons étudier dans ce qui suit deux versions de cette méthode. Ces deux versions diffèrent dans l'exécution des projections d'Arnoldi des méthodes GMRES et Arnoldi. La première est caractérisée par l'exécution des projections sur deux machines parallèles ; cette version est dite asynchrone. La deuxième est caractérisée par l'exécution

des projections alternativement sur une même machine parallèle; cette version est dite entrelacée.

Avant de passer à la description des deux versions de la méthode hybride, il est nécessaire de préciser certains points de leur fonctionnement

- dès la réception des valeurs propres de l'algorithme d'Arnoldi(m_2), le nouveau convexe est calculé à partir de l'ancien et de ces nouvelles valeurs. Cela nous évite de stocker toutes les valeurs reçues et nous permet de minimiser le temps de calcul du convexe, parce que le temps de mise à jour du convexe est plus faible,

- le résidu de la méthode LS étant riche en informations obtenues à partir des vecteurs propres associés aux valeurs propres situées à l'extérieur du convexe H, le calcul de ces valeurs propres permet d'améliorer l'allure du convexe H pour approcher de mieux en mieux le convexe englobant le spectre complet de A. Pour cela, le résidu de la méthode LS constitue un bon vecteur de départ pour l'itération suivante de la méthode d'Arnoldi. Ainsi ce résidu est envoyé à l'algorithme d'Arnoldi après chaque itération LS pour en constituer un vecteur initial. Cette technique est inspirée de celle utilisée dans les méthodes Tchebyshev-Arnoldi [58] et Least Squares-Arnoldi [58] pour accélérer le calcul des valeurs propres de parties réelles dominantes.

- les valeurs propres calculées avec la précision voulue, notée ϵ_{val_p}, sont accumulées. Ces valeurs sont envoyées au processus LS quand leur nombre dépasse un seuil nvp_{min} fixé au début. Dans ce cas, l'algorithme Arnoldi(m_2) est redémarré avec un autre vecteur initial que nous choisissons aléatoirement si un nouveau résidu de la méthode LS n'a pas été reçu au moment du redémarrage.

4.3 La méthode hybride GMRES(m_1)/LS(k,l)-Arnoldi(m_2) parallèle asynchrone

4.3.1 Description

Cette version de la méthode GMRES(m_1)/LS(k,l)-Arnoldi(m_2) [7], [24] est caractérisée par la séparation complète des algorithmes GMRES et Arnoldi. Chacune des deux méthodes est exécutée sur une machine parallèle. De cette façon nous obtenons moins de dépendance entre les différentes parties de la méthode hybride.

Les parties séquentielles de GMRES, d'Arnoldi et de la méthode LS sont exécutées sur des machines séquentielles. Dans le cas où une machine parallèle a une architecture MIMD, la partie séquentielle de la méthode associée à cette machine est exécutée d'une façon redondante sur tous les processeurs, cette machine étant libre lorsque ce calcul doit être effectué.

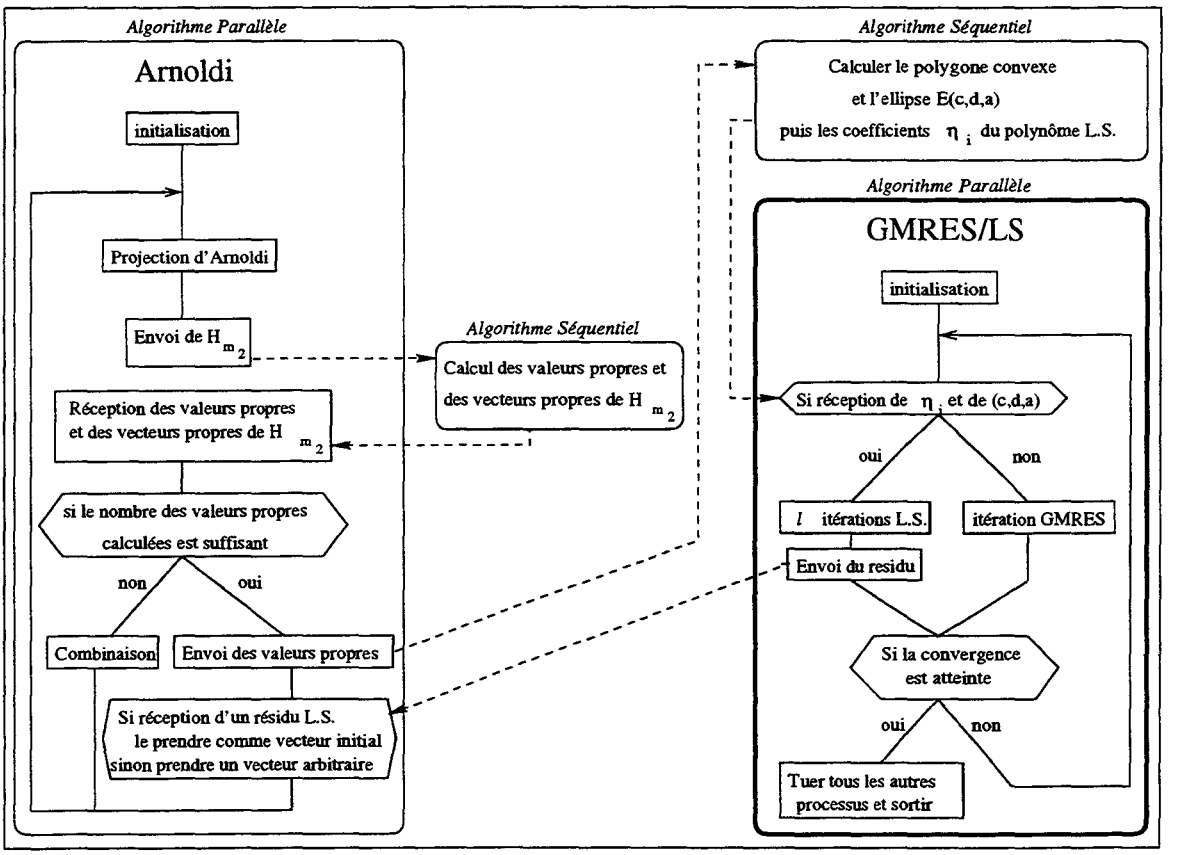

FIG. 4.1 - *Le schéma général de la méthode hybride asynchrone GMRES(m_1)/LS(k,l)-Arnoldi(m_2).*

Dans cette méthode hybride parallèle asynchrone les calculs se déroulent comme suit :
les algorithmes GMRES(m_1) et Arnoldi(m_2) sont lancés en même temps, chacun sur une machine parallèle. Dès que le nombre de valeurs propres calculées - avec la précision ϵ_{val_p} choisie - dépasse le seuil nvp_{min}, ces valeurs sont transmises à la machine séquentielle qui calcule les paramètres du polynôme LS. Ces paramètres sont transmis à la machine exécutant GMRES. À la fin de l'itération courante de GMRES(m_1), cet algorithme est arrêté pour effectuer une itération de la méthode LS(k,l), il est ensuite redémarré (voir figure 4.1).
Dans la suite, nous utilisons le terme «hybridation» pour dire que la méthode LS(k,l) est appliquée après l'arrêt de GMRES(m_1).

4.3.2 Asynchronisation

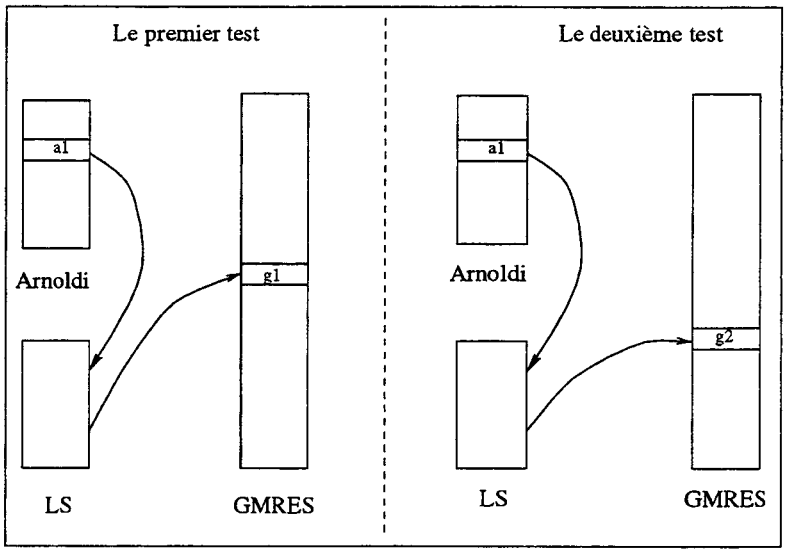

FIG. 4.2 - *L'asynchronisme dans GMRES(m_1)/LS(k,l)-Arnoldi(m_2).*

Les machines travaillent en général en temps partagé (*time-sharing*) avec d'autres utilisateurs, c'est le choix adopté généralement par les centres de calcul. Ce qui induit la variation de la charge au cours du temps. Si un test est lancé deux fois, la première hybridation utilise les valeurs propres de l'itération a_1 d'Arnoldi et l'itéré GMRES de l'itération g_1 pour le premier essai et g_2 pour le deuxième essai. Nous notons que $g_1 \neq g_2$ si un changement de charge a eu lieu, ce qui nous donne des résultats différents (voir figure 4.2). Les moments de l'hybridation dépendent donc de la charge des machines utilisées. Ce non déterminisme de l'algorithme hybride parallèle, dû à la charge des machines, rend difficile la détermination des propriétés de l'algorithme mis en œuvre (voir le tableau 4.1).

Méthode hybride : (*Chaque colonne représente un nouveau test*)

Nombre des itérations GMRES	6	6	6	7	6
Nombre des itérations LS	2	1	2	3	2
Durée totale (en secondes)	574	544	611	734	684

TAB. 4.1 - *La convergence de la méthode hybride avec la matrice "utm1700a" ($n = 1700$, m(GMRES)=400, m(Arnoldi)=256, $k = 15$, $l = 5$).*

En plus de la variabilité de la charge des machines, les moments d'hybridation dépendent du système à résoudre ainsi que des paramètres de l'algorithme d'Arnoldi.

Ces paramètres sont m_2 la dimension du sous-espace de Krylov, le nombre nvp_{min} et ϵ_{val_p} la précision des valeurs propres à calculer.

La vitesse de convergence dépend de l'allure du convexe qui englobe les valeurs propres trouvées, ainsi que de la composition en vecteurs propres des résidus des itérés GMRES(m_1) aux moments de l'hybridation. La convergence est rapide quand la plupart des valeurs propres, correspondantes aux vecteurs propres qui composent le résidu courant, sont situées à l'interieur du convexe courant au moment où la méthode LS(k,l) est appliquée. Ces moments sont donc décisifs pour la rapidité de la convergence de la méthode hybride. La détermination de ces instants est très difficile et reste un problème ouvert.

4.3.3 Implantation de la méthode hybride asynchrone

Pour expérimenter cette version asynchrone nous utilisons les machines parallèles suivantes :

- La ferme d'Alpha : une machine parallèle de 16 nœuds (voir Section 1.3.2).
- La MasPar MP-1 : une machine SIMD de 16k processeurs (voir Section 1.3.1). Sur cette machine nous utilisons son compilateur Fortran data-parallèle, appelé MasPar Fortran.

Nous utilisons en plus deux stations Sun pour exécuter les algorithmes séquentiels. Sur ces stations Sun et sur les machines de la ferme d'Alpha nous utilisons le compilateur Fortran 77. Toutes les communications entre les différentes machines ainsi qu'à l'intérieur de la ferme d'Alpha sont gérées à l'aide de la bibliothèque de passages de messages PVM.

La ferme d'Alpha prend en charge l'algorithme GMRES(m_1). L'implantation utilisée, décrite dans le Chapitre 3, est effectuée sur p processeurs communiquant à l'aide de PVM. Chaque processeur se charge des opérations correspondantes aux n/p lignes de la matrice A dont il dispose. La partie séquentielle GMRES(m_1) est exécutée d'une façon redondante sur les p processeurs de la ferme d'Alpha, puisque cette dernière est une machine MIMD ayant des processeurs puissants.

La MasPar prend en charge la partie parallèle de l'algorithme d'Arnoldi comprenant la projection, le calcul des vecteurs de Ritz, le calcul des résidus et le redémarrage. La partie séquentielle de cet algorithme (le calcul des vecteurs et valeurs propres de la matrice H_{m_2}) est exécutée sur une station Sun.

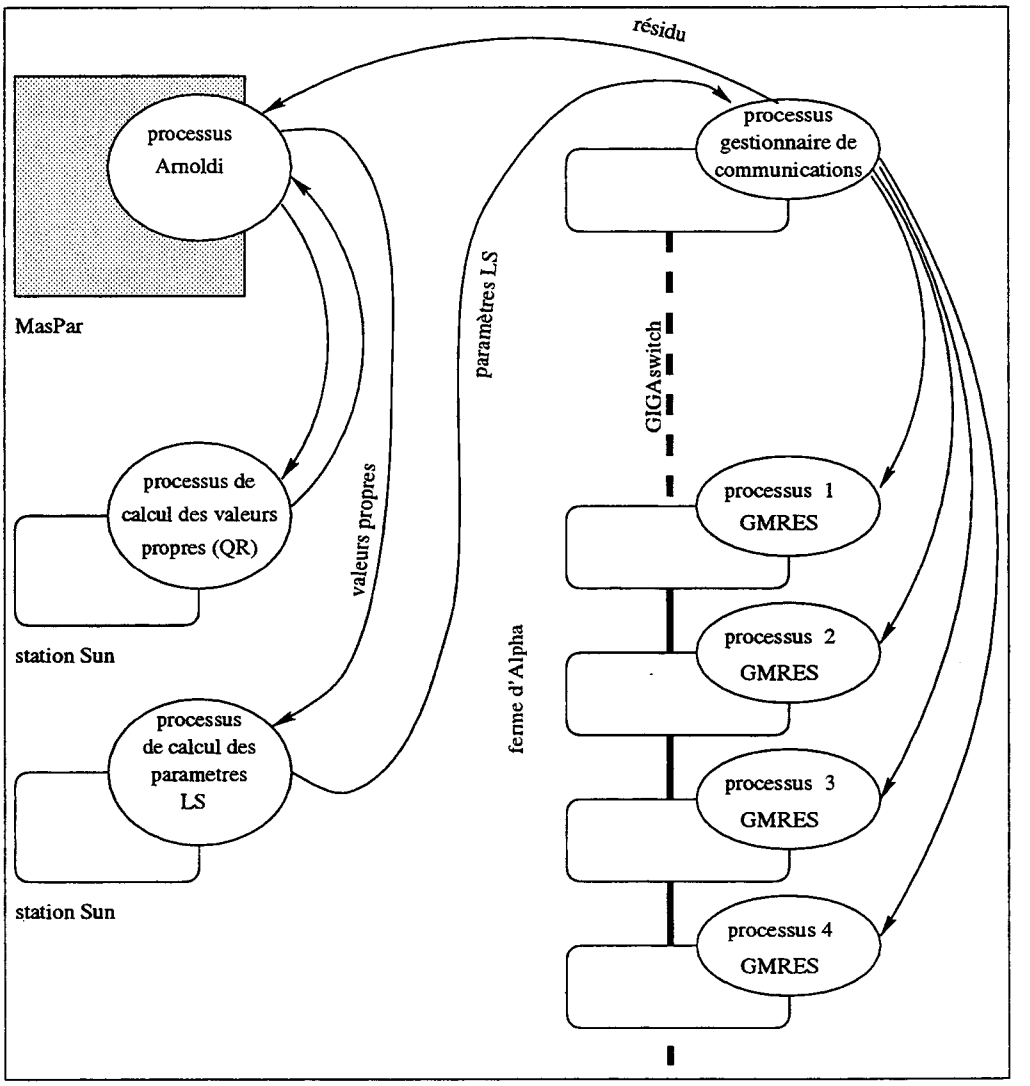

FIG. 4.3 - *Le schéma général de l'implantation asynchrone de GMRES(m_1)/LS(k,l)-Arnoldi(m_2).*

Une autre station calcule les paramètres de la méthode Moindres Carrés (l'enveloppe convexe, l'ellipse et les coefficients du polynôme des moindres carrés). Après leur réception par le processeur père de la ferme d'Alpha à travers le réseau Ethernet, ces paramètres sont diffusés à tous les fils via le GIGAswitch (voir figure 4.3).

À la fin de chaque itération, GMRES vérifie s'il a reçu les paramètres LS. Dans le cas affirmatif, il applique la méthode LS(k,l) au dernier itéré obtenu par GMRES, ce dernier algorithme est ensuite redémarré.

Le résidu de l'itéré LS est envoyé à la MasPar, afin de l'utiliser pour le redémarrage de l'algorithme d'Arnoldi(m_2).

Pour éviter les échanges de données, la matrice est générée ou chargée simultanément sur les deux machines parallèles. La matrice est stockée sous le format CSR (Compressed Sparse Row) [64] sur la ferme et sous le format SGP [19] sur la MasPar, afin de minimiser le coût mémoire et faciliter le calcul des produits matrice-vecteur.

4.3.4 Résultats expérimentaux

Pour comparer la méthode hybride GMRES(m_1)/LS(k,l)-Arnoldi(m_2) avec la méthode GMRES(m_1) tout seule, nous avons testé ces deux méthodes sur quelques systèmes linéaires. Notons que l'implantation de GMRES(m_1) tout seul est pareille à celle adoptée pour l'implanter au sein de la méthode hybride. Dans cette implantation, nous utilisons 8 processeurs de la ferme d'Alpha ($p = 8$).

Nous comparons ces deux méthodes sans préconditionnement afin de mettre en évidence les propriétées de la méthode hybride. Notons aussi qu'un préconditionnement (SSOR, basé sur la décomposition incomplète LU, polynomial,...) n'est pas efficace pour n'importe quelle matrice. L'utilisation d'un préconditionnement peut rendre plus difficile cette comparaison.

Nous avons considéré deux choix le second membre des systèmes linéaires étudiés. Le premier b_1 est calculé de telle façon que la solution exacte du système soit égale à $x^* = (1, 1, \ldots, 1)^T$, cela nous permet de calculer facilement l'erreur $e = x - x^*$. Le second noté b_2 est pris égal à $e_1 = (1, 0, \ldots, 0)^T$. La solution initiale prise pour les deux méthodes comparées est le vecteur $x_0 = (0, 0, \ldots, 0)^T$.

Premier exemple (5-points)

Cet exemple provient de la collection SPARSKIT [64]. La matrice résulte de la discrétisation de l'équation aux dérivées partielles suivante

$$-\Delta u + 10^6 u_x + \frac{\partial}{\partial y}(10^4 e^{-xy} u) = f$$

sur le carré unité $]0,1[^2$) avec les conditions de Dirichlet aux bords ($u = 0$ sur $\partial]0,1[^2$). La dimension n de la matrice est $n = nx \times ny$ où nx est le nombre de points dans la direction x et ny celui dans la direction y. Ici nous prenons $n = 100 \times 100 = 10000$. Dans cet exemple, nous prenons le vecteur b_2 comme second membre.

Deuxième exemple (utm1700a)

Cet exemple provient du site MatrixMarket[1] de matrices. La matrice est générée par l'application TOKAMAK du physique nucléaire (plasmas), de la collection SPARSKIT. Cette matrice de dimensions 1700 x 1700, possède 21313 éléments non nuls et Son conditionnement est estimé à 6.24e+06. Dans cet exemple, nous prenons le vecteur b_1 comme second membre.

Troisième exemple (vp)

Nous considérons la matrice A de dimensions $2q \times 2q$ diagonale par des blocs carrés de taille 2, représentée dans la figure 4.4. Les valeurs propres de cette matrice sont les complexes $a_j \pm ib_j$, où $j \in \{1, \cdots, q\}$.
Cet exemple nous permet donc de choisir les valeurs propres de A. Ici, nous choisissons aléatoirement les valeurs propres dans deux rectangles, R_1 dont les sommets sont $-2.5 \pm 2i$, $-1.5 \pm 2i$, et R_2 dont les sommets sont $1.2 \pm 4i$, $1.8 \pm 4i$. Chaque rectangle contient la moitié des valeurs propres. Dans cet exemple, nous prenons le vecteur b_1 comme second membre.

Les résultats numériques montrent clairement que la méthode hybride GMRES(m_1)/LS(k,l)-Arnoldi(m_2) améliore la convergence de GMRES(m_1), à la fois au point de vue itérations et temps (voir les figures 4.5, 4.6, 4.7, 4.8, 4.9 et 4.10). Notons aussi que parfois la méthode GMRES(m_1) présente une stagnation (voir la figure 4.8), alors que la méthode hybride converge.

Les performances de la méthode hybride varient d'un test à l'autre avec le même exemple et les mêmes paramètres (voir table 4.1). Ce phénomène est dû au non déterminisme de l'algorithme.

1. http://math.nist.gov/MatrixMarket/

$$A = \begin{pmatrix} a_1 & b_1/2 & & & & & \\ 2b_1 & a_1 & & & & & \\ & & a_2 & b_2/2 & & & \\ & & 2b_2 & a_2 & & & \\ & & & & \cdot\ \cdot & & \\ & & & & \cdot\ \cdot & & \\ & & & & & a_q & b_q/2 \\ & & & & & 2b_q & a_q \end{pmatrix}$$

FIG. 4.4 - *La matrice "vp".*

Quand une hybridation a eu lieu, nous remarquons en général que le résidu de l'itéré donné par la méthode LS est plus grand que celui de la dernière itération GMRES(m_1) effectuée. Cependant, pendant les itérations suivantes de GMRES(m_1) le résidu décroît plus vite que lors de la phase précédent l'hybridation (voir l'explication théorique dans la Section 4.1.2), ceci se traduit par des pics dans la courbe représentant la norme du résidu.

Toutefois, des hybridations fréquentes endommagent la convergence parce que GMRES(m_1) n'a pas assez de temps pour faire décroître le résidu. Dans ce cas, la méthode hybride diverge.

Un corollaire qui peut être déduit du paragraphe précédent est que la machine qui exécute Arnoldi pour calculer les valeurs propres n'a pas besoin d'être très performante. Une machine parallèle obsolète ayant une taille mémoire suffisante peut ainsi réduire le temps de calcul de GMRES(m_1) sur une machine performante. Nous pouvons donc nous permettre d'exploiter pour cet effet les vieilles machines parallèles sous-utilisées dans des laboratoires de recherche au sein d'un autre site.

La fréquence des hybridations est contrôlée par le paramètre m_2, le seuil nvp_{min} et ϵ_{val_p} la précision des valeurs propres. Donc pour éloigner les hybridations il suffit d'augmenter l'un de ces paramètres, ce qui retardera l'envoi des valeurs propres calculées par la méthode d'Arnoldi vers le processus LS.

M (Arnoldi)	64	128	192	256
Nombre d'itérations GMRES	8	5	6	6
Durée totale(en secondes)	596	498	612	684

TAB. 4.2 - *Résultats trouvés en variant la valeur de m(Arnoldi) (la matrice "utm1700a", m(GMRES)=400, $k = 15$, $l = 5$).*

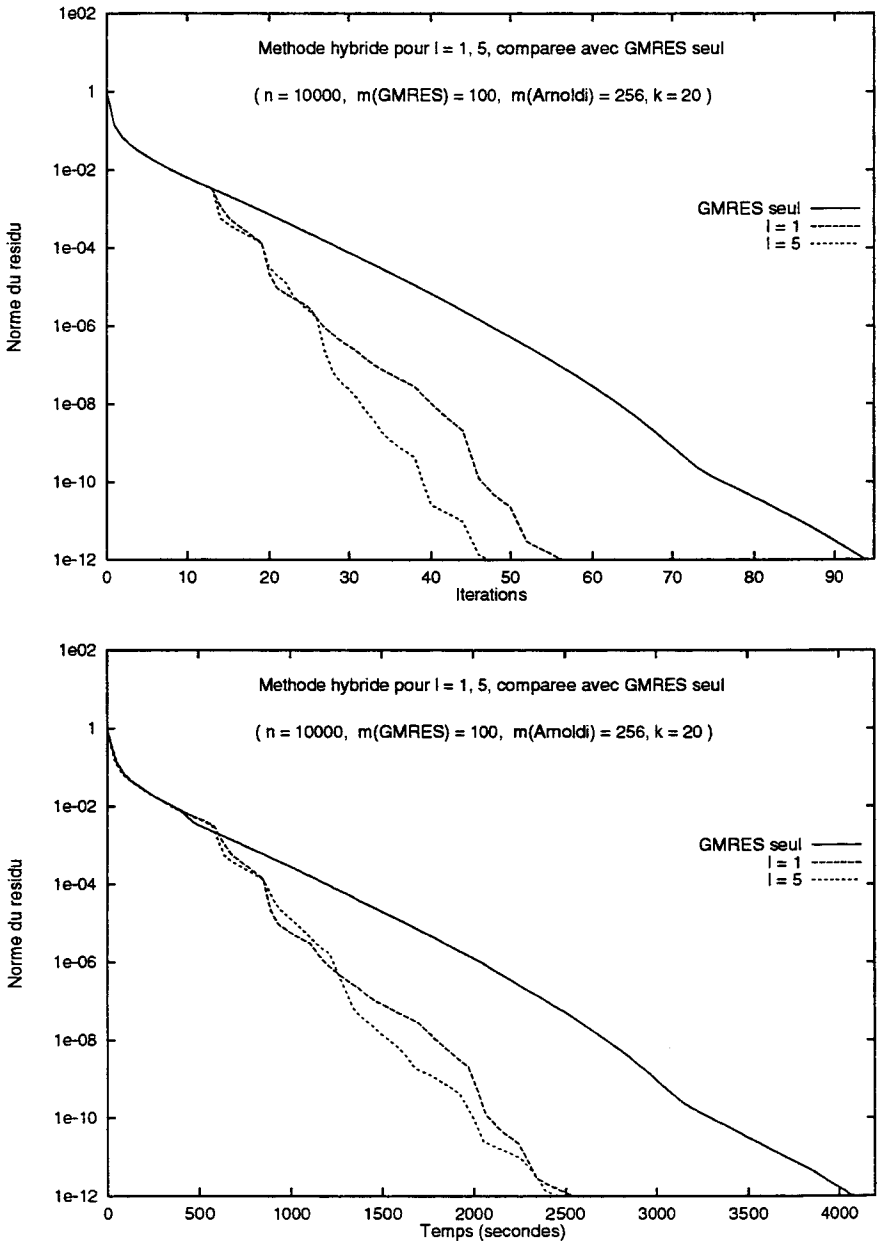

FIG. 4.5 - *L'évolution du résidu de la méthode hybride pour différentes valeurs de l, comparée à celle de GMRES(100), (matrice "5-points").*

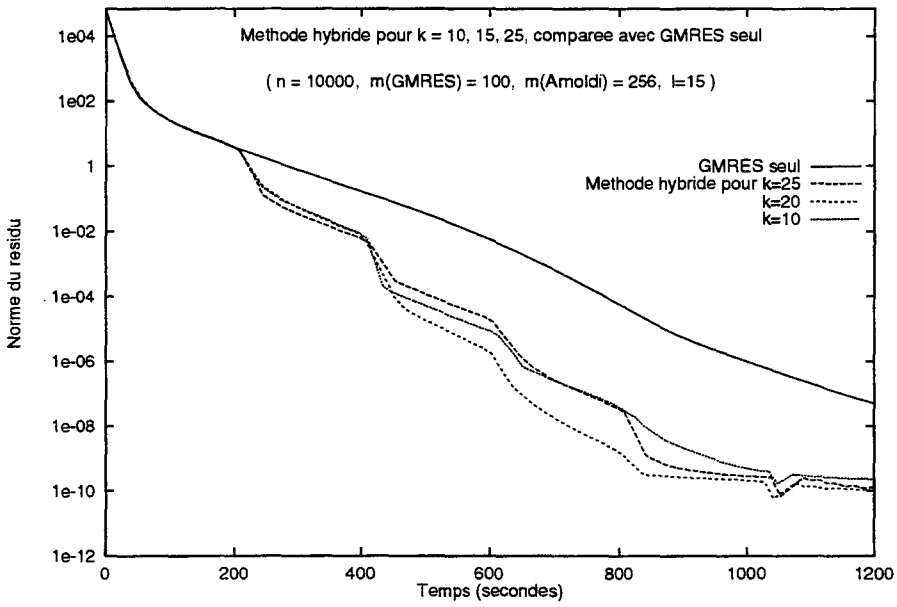

FIG. 4.6 - *L'évolution du résidu de la méthode hybride pour différentes valeurs de k comparée à celle de GMRES(100), (matrice "5-points").*

Pour de petites valeurs de l (notamment $l = 1$) dans certains exemples, la méthode hybride est moins performante que GMRES lui-même (voir figure 4.9). Comme nous l'avons vu dans la Section 4.1.2, nous pouvons améliorer l'efficacité de la méthode LS(k,l) en augmentant la valeur de l. Les pics sont plus grands dans ce cas, mais la convergence est plus rapide.

Pour certains exemples, il est difficile de calculer des approximations des valeurs propres avec une bonne précision ϵ_{val_p}, aucune itération de la méthode LS(k,l) n'est effectuée dans ce cas. Afin d'avoir des prises en compte de cette méthode nous avons à augmenter la valeur du paramètre ϵ_{val_p}.
Les tests sur la matrice "5-points" ont été effectués en prenant $\epsilon_{val\_p} = 10^2$. Bien que les valeurs propres ne sont pas très précises la méthode hybride présente de bonnes performances (voir figures 4.5 et 4.6). Pour la matrice "vp", nous avons pris $\epsilon_{val\_p} = 1$, par contre pour la matrice "utm1700a", nous avons pris une meilleure valeur $\epsilon_{val\_p} = 10^{-3}$.

Étant donné que $k-1$ est le degré du polynôme LS, une itération LS demande $k-1$ produits matrice-vecteur, donc le calcul de l'itéré LS(k,l) demande $(k-1)l$ produits matrice-vecteur. Ceci montre que nous n'avons pas intérêt à prendre des valeurs trop élevées de l et k sinon la méthode LS(k,l) devient très coûteuse. Ainsi, les hybridations seront très rapprochées et dans ce cas, la méthode hybride diverge.

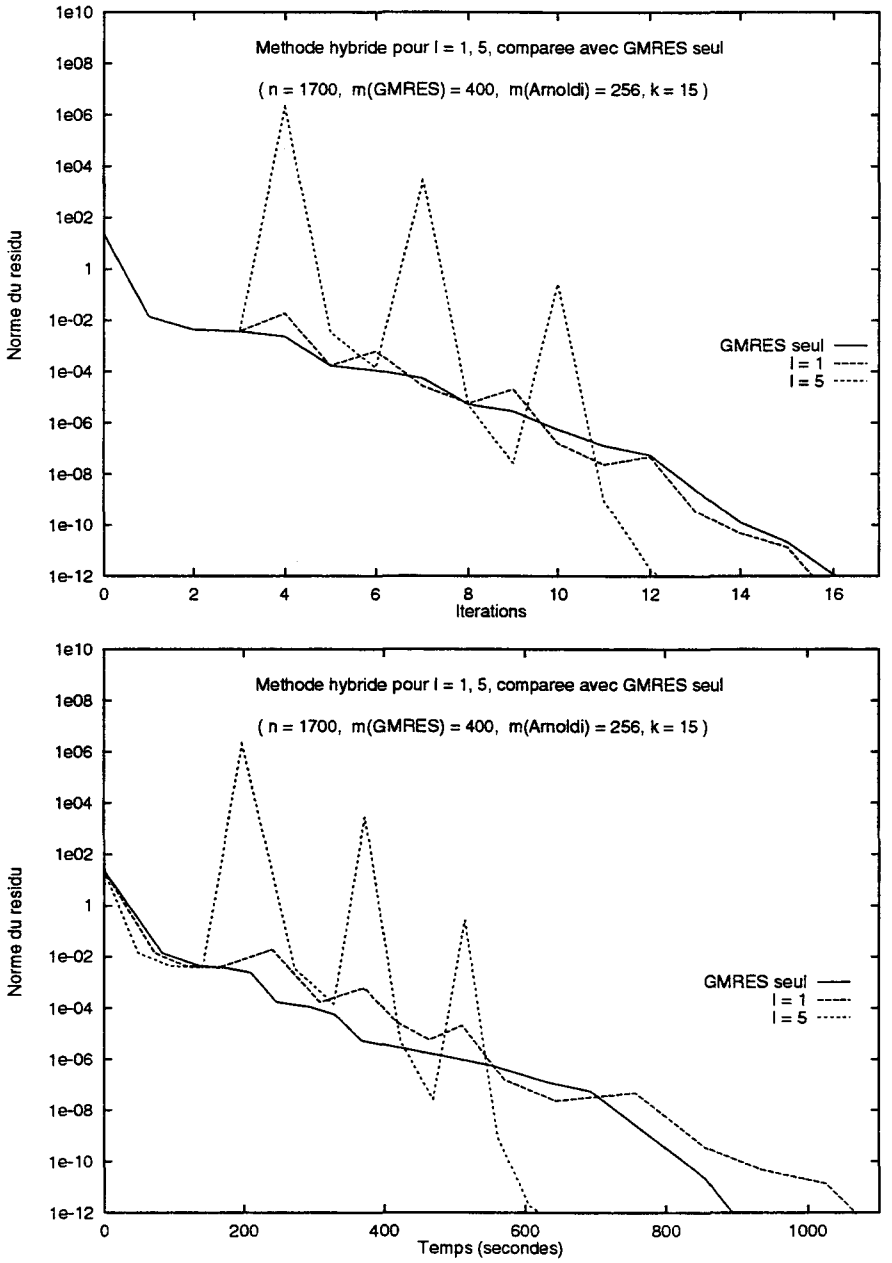

FIG. 4.7 - *L'évolution du résidu de la méthode hybride comparée à GMRES(400), (la matrice "utm1700a").*

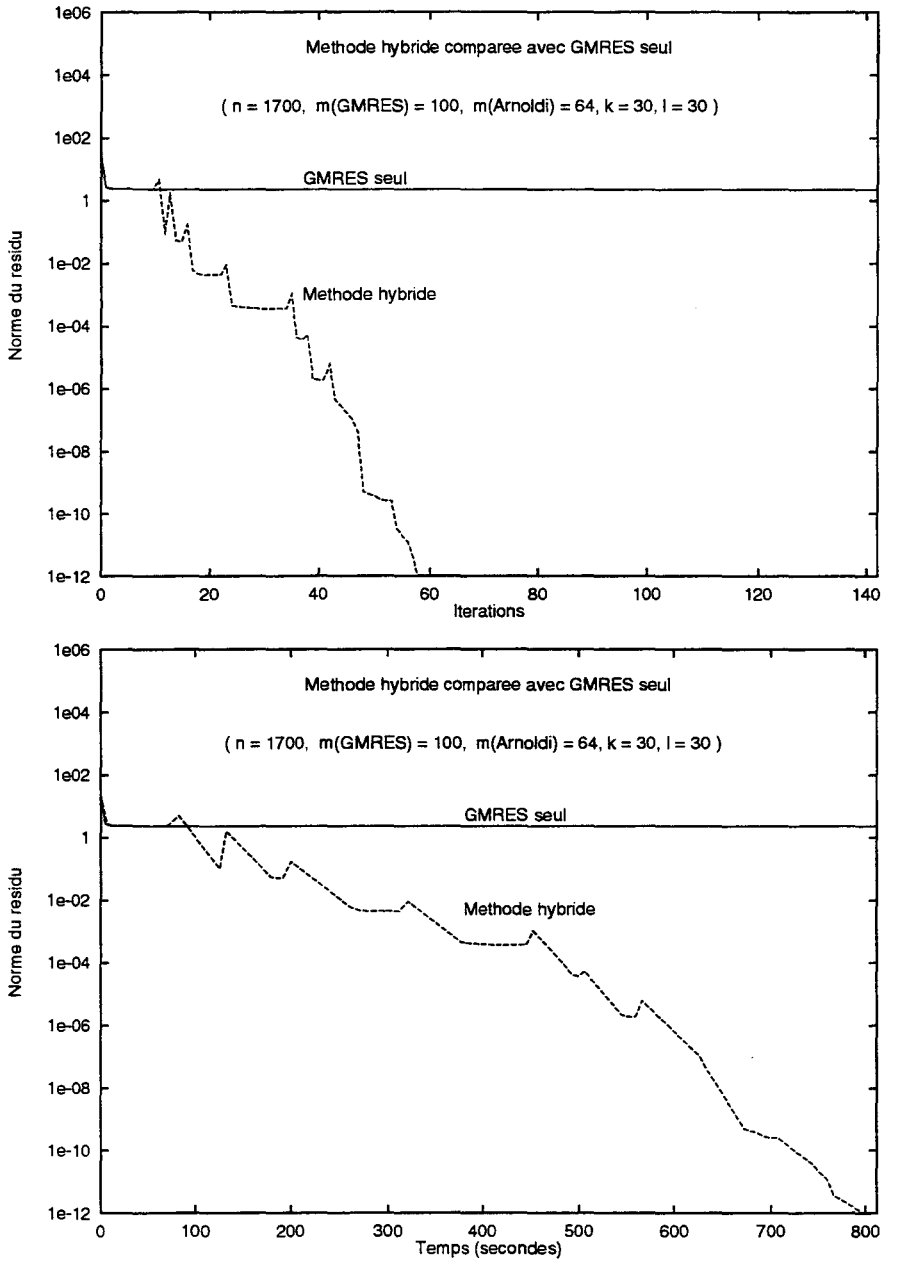

FIG. 4.8 - *L'évolution du résidu de la méthode hybride comparée à GMRES(100), (la matrice "utm1700a").*

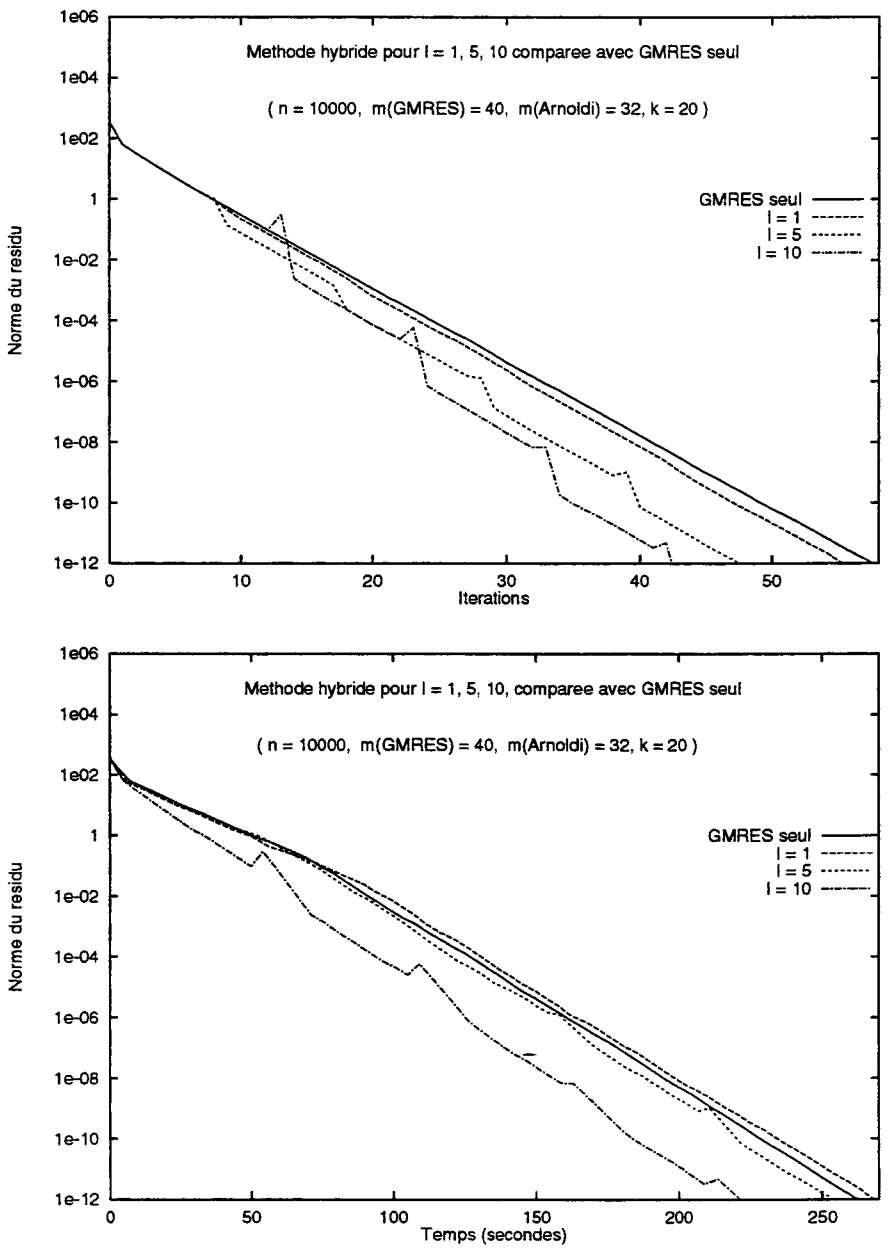

FIG. 4.9 - *L'évolution du résidu de la méthode hybride comparée à GMRES(40), (la matrice "vp").*

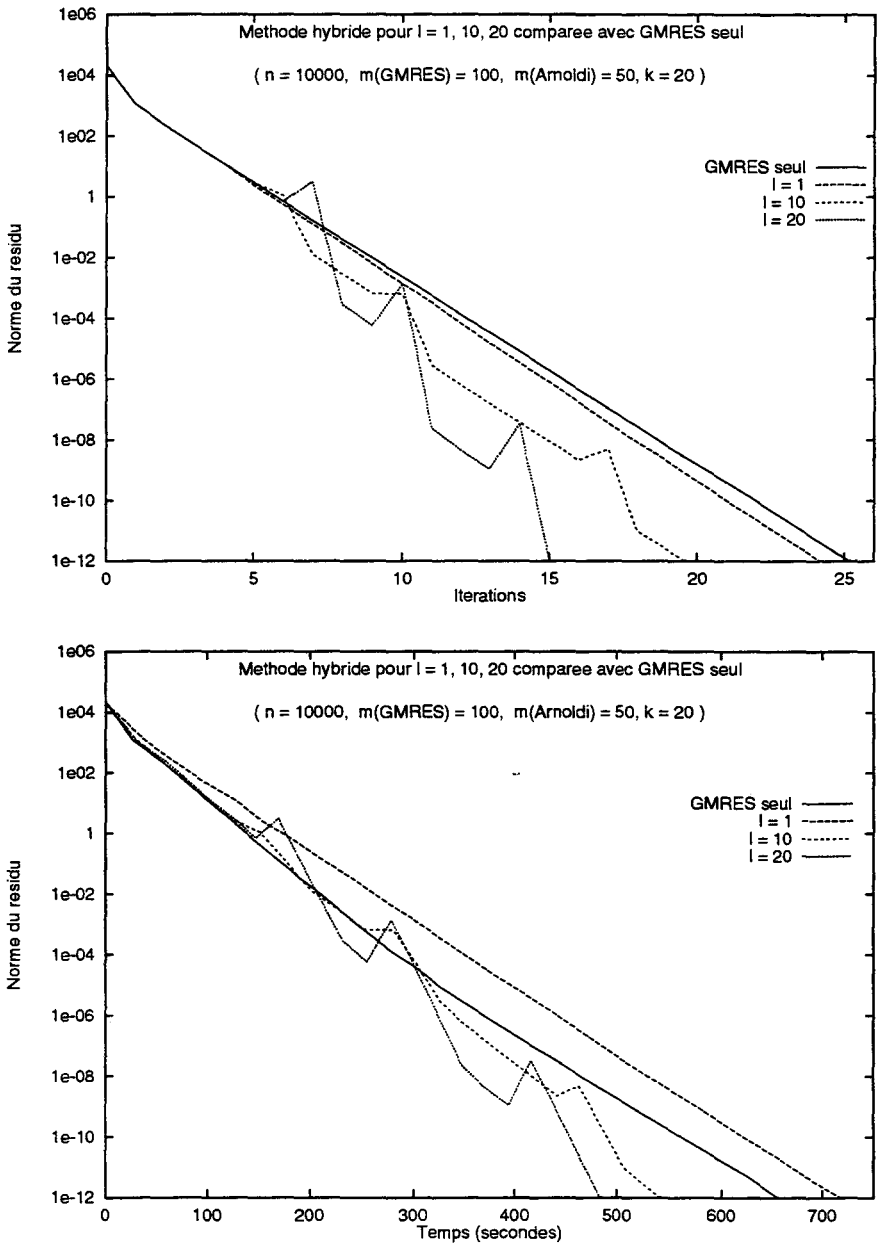

FIG. 4.10 - *L'évolution du résidu de la méthode hybride comparée à GMRES(100), (la matrice "vp").*

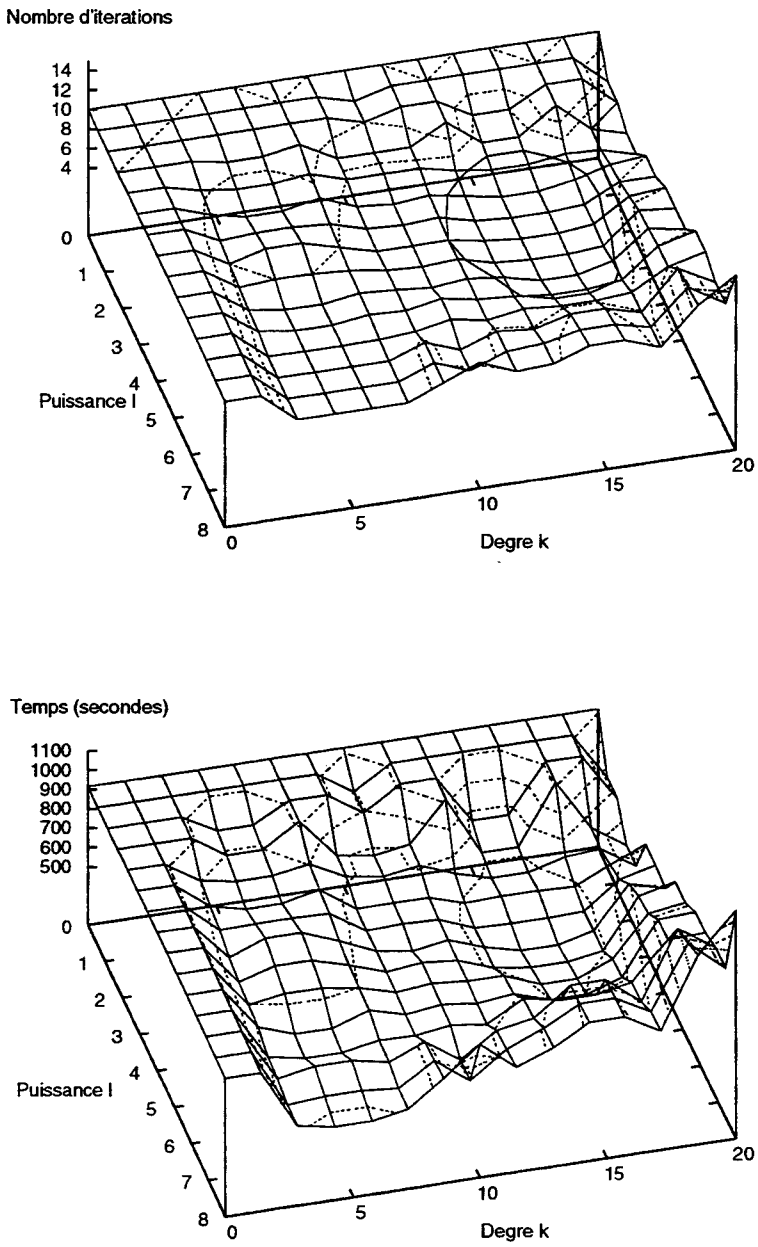

FIG. 4.11 - *L'évolution du nombre d'itérations et du temps de la convergence de la méthode hybride, en fonction des paramètres k et de l (matrice "utm1700a").*

Dans la figure 4.11, nous avons tracé l'évolution du nombre d'itérations et du temps de convergence de la méthode hybride, en fonction des paramètres l et k. Nous remarquons donc l'existence de valeurs optimales de k et de l qui donnent une meilleure convergence de la méthode hybride. La détermination des valeurs optimales de k et l théoriquement est difficile puisqu'elles dépendent de la matrice et de beaucoup d'autres paramètres.

4.4 La méthode hybride GMRES(m_1)/LS(k,l)-Arnoldi(m_2) parallèle entrelacée

4.4.1 Description

Nous savons que chacun des algorithmes de GMRES(m_1) et Arnoldi(m_2) est partagé en deux parties, l'une parallèle et l'autre séquentielle. Nous pouvons tirer parti de cette propriété afin d'entrelacer les parties parallèles de ces deux algorithmes sur une même machine parallèle.

Ainsi, GMRES(m_1) étant l'algorithme le plus important, nous profitons du temps d'exécution de la partie séquentielle de GMRES sur une machine séquentielle pour exécuter la partie parallèle de Arnoldi(m_2). De même, la partie séquentielle de Arnoldi(m_2) est exécutée sur une machine séquentielle pendant l'exécution de la partie parallèle de GMRES(m_1) et le calcul de l'itéré de la méthode LS(k,l). Une telle organisation d'algorithmes a été déjà réalisée pour implanter une méthode d'Arnoldi hybride [19]. La partie séquentielle de la méthode Moindres carrés est prise en charge par une troisième machine séquentielle. La figure 4.12 représente le schéma général de l'implantation de la version entrelacée de la méthode hybride GMRES(m_1)/LS(k,l)-Arnoldi(m_2) [24] sur une seule machine parallèle et trois machines séquentielles.

Afin d'éviter de retarder l'algorithme GMRES par Arnoldi, il faut choisir la valeur de m_2 de telle sorte que le temps de la projection dans la méthode d'Arnoldi ne dépasse pas celui de la résolution du problème d'optimisation dans GMRES(m_1). Cette valeur doit être nécessairement plus petite que m_1, mais avec des valeurs très petites de m_2 le nombre de valeurs propres calculées avec une bonne précision diminue.

Dans cette version, au contraire de la version asynchrone de la méthode parallèle GMRES(m_1)/LS(k,l)-Arnoldi(m_2), il y a plus de synchronisation puisque dans ce cas

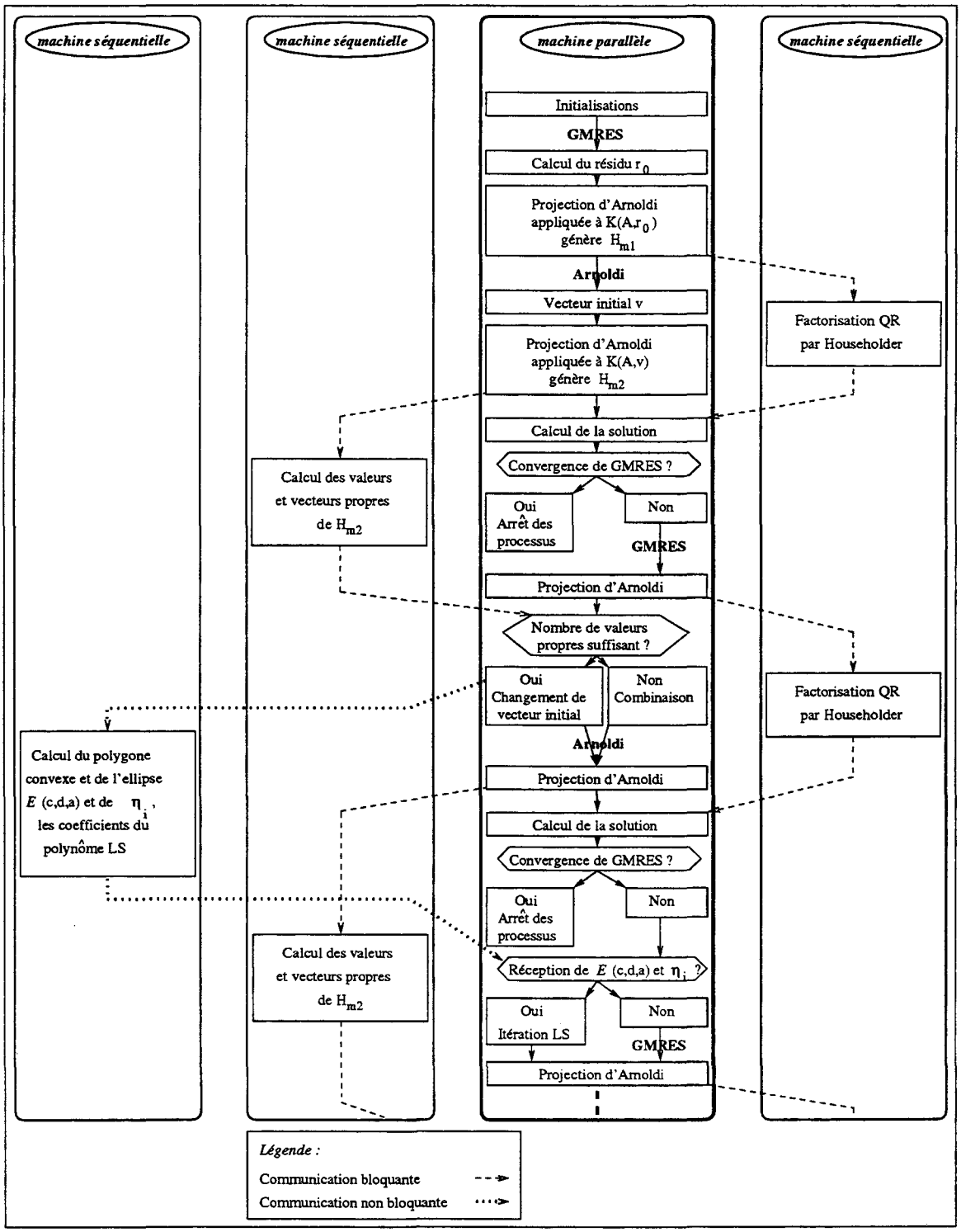

FIG. 4.12 - *Le schéma général de l'implantation entrelacée de la méthode GMRES(m_1)/LS(k,l)-Arnoldi(m_2).*

les algorithmes GMRES et Arnoldi sont synchrones, mais la réception des paramètres LS reste asynchrone. Cette synchronisation induit des délais d'attente de GMRES si la valeur de m_2 n'est pas bien ajustée, comme nous l'avons vu dans le paragraphe précédent.

4.4.2 Implantation de la méthode hybride entrelacée

Nous avons implanté cette version sur le site de l'ETCA[2], qui dispose d'une *Connection Machine* CM-5 avec 32 nœuds et de stations Sun. Ces différentes machines communiquent entre elles à l'aide du réseau Ethernet. La CM-5 exécute la partie parallèle de l'algorithme et les trois stations Sun exécutent les trois parties séquentielles de GMRES, Arnoldi et LS. Ces stations sont plus performantes dans les calculs séquentiels que le frontal de la CM-5, pour cette raison cette dernière ne sera utilisée dans cette implantation que pour accéder à la CM-5..

4.4.3 Résultats expérimentaux

Dans cette section nous testons les systèmes considérés dans la Section 4.3.4. Les résultats numériques (voir figures 4.13, 4.14 4.15 et 4.16) montrent que la méthode hybride parallèle entrelacée, avec des paramètres bien choisis, est meilleure que GMRES(m_1), comme c'est le cas de l'implantation asynchrone.

Des tests répétés sur un même exemple avec les mêmes paramètres donnent souvent les mêmes résultats. Cette version est donc pratiquement synchrone, même si la réception par la CM-5 des paramètres LS se fait d'une façon asynchrone.

Nous remarquons l'existence de pics dans les courbes représentant l'évolution de la norme du résidu au cours du temps, comme dans la version asynchrone. Mais ce qui diffère de la version asynchrone est que les pics ont la même largeur pour différentes valeurs de l (voir figure 4.15). Cela est dû au calcul des valeurs propres de H_{m_2} qui demande approximativement $10(m_2)^2$ opérations ce qui nécessite beaucoup de temps si la valeur de m_2 est grande. La partie parallèle de GMRES attend donc la partie séquentielle de Arnoldi puisque la valeur $m_2 = 50$ que nous avons pris pour la matrice "vp" est très grande.

Dans la courbe du temps total de la convergence en fonction de la dimension m_2

2. Établissement Technique Central de l'Armement, 16 bis avenue Prieur de la Cité d'Or, 94114 Arcueil Cedex, France.

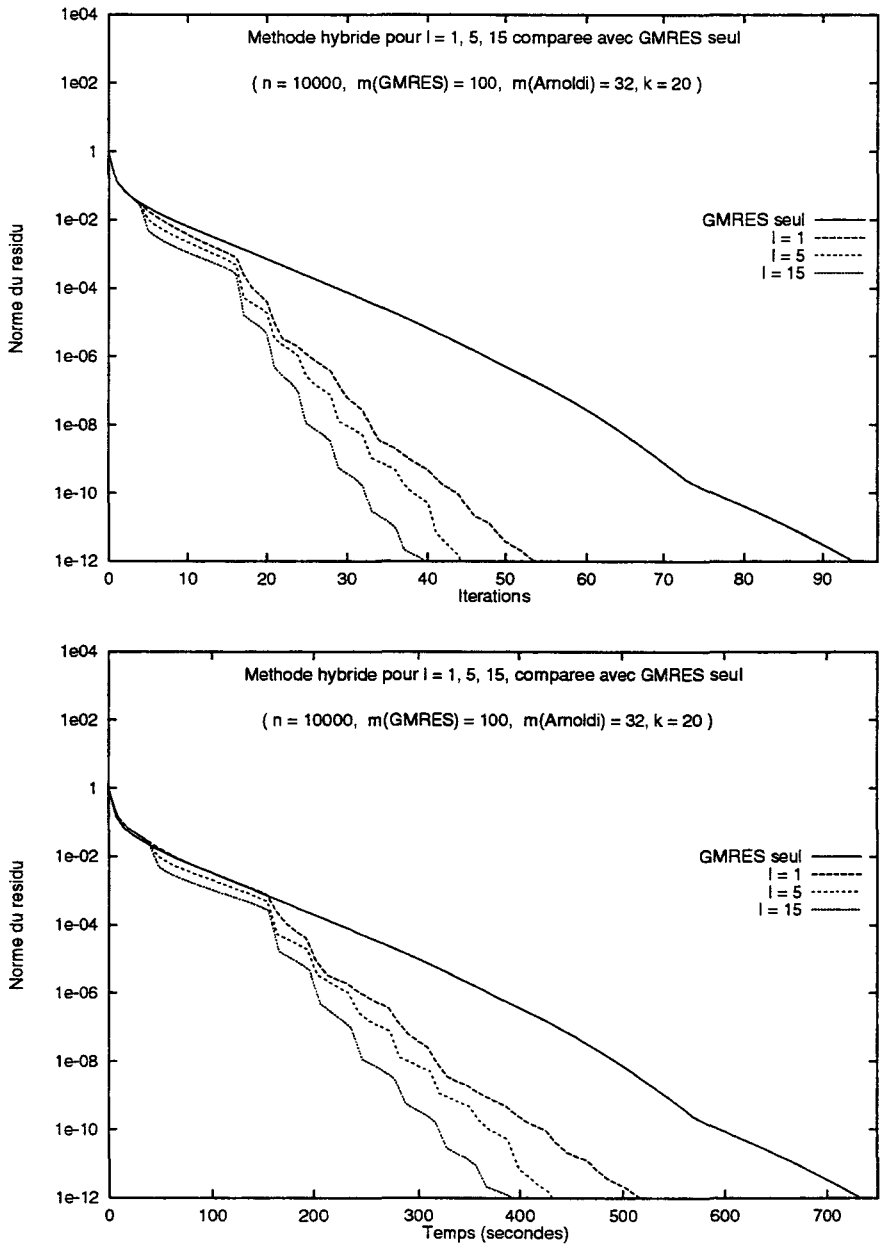

FIG. 4.13 - *L'évolution du temps de convergence de la méthode hybride parallèle entrelacée, comparée à GMRES(100), (la matrice "5-points").*

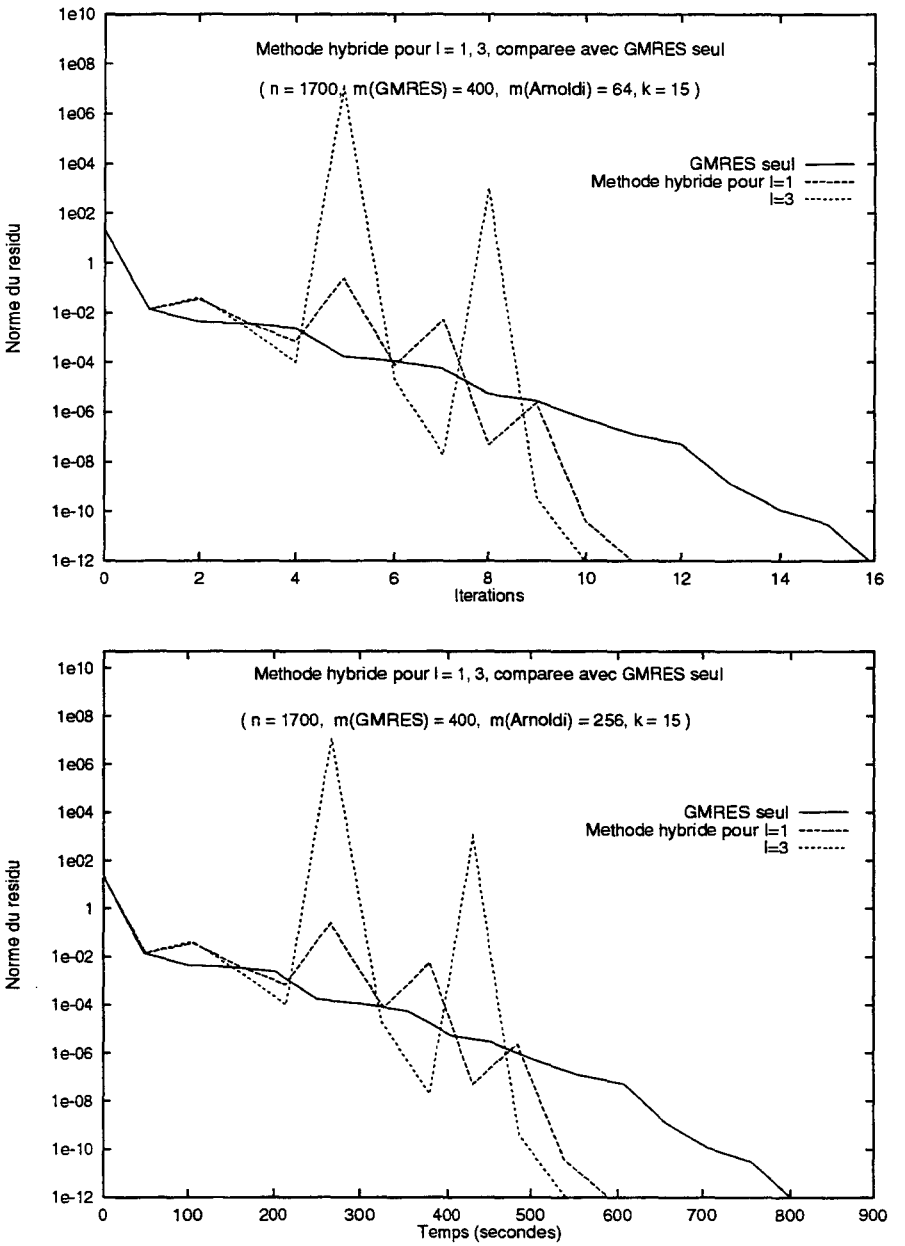

FIG. 4.14 - *L'évolution du temps de convergence de la méthode hybride parallèle entrelacée, comparée à GMRES(400), (la matrice "utm1700a").*

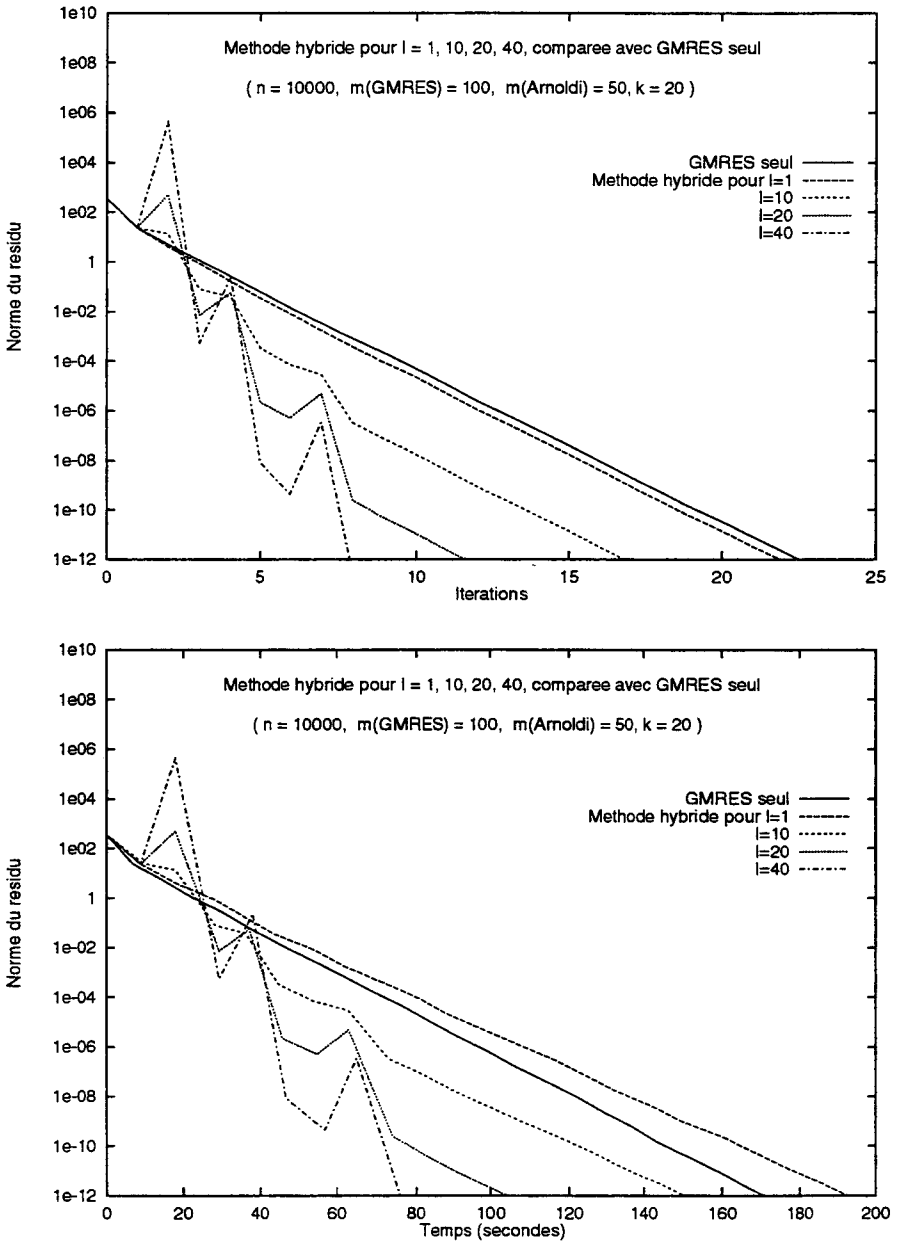

FIG. 4.15 - *L'évolution du résidu de la méthode hybride entrelacée comparée à GMRES(100), (la matrice "vp").*

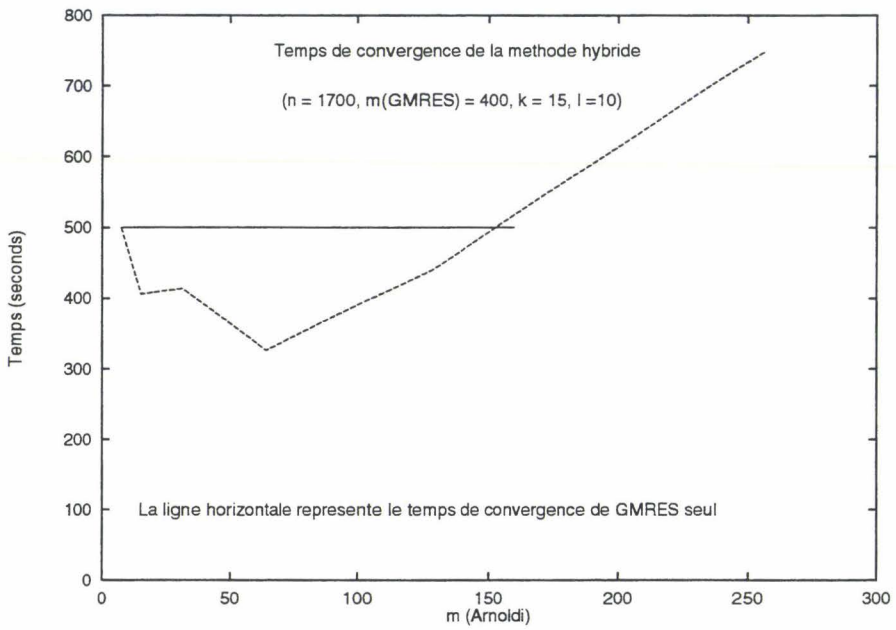

FIG. 4.16 - *Le temps de convergence en fonction de la taille m_2 du sous-espace de Krylov de la méthode d'Arnoldi, (la matrice "utm1700a").*

du sous-espace de projection d'Arnoldi (voir figure 4.16), nous constatons l'existence d'un optimum. En-dessous de cette valeur, m_2 est insuffisant pour calculer assez de valeurs propres. Au-dessus de cette valeur, le temps de calcul d'Arnoldi devient assez important et dans ce cas, GMRES attend que Arnoldi finisse ses calculs pour redémarrer. Cependant il y a une plage de valeurs de m_2 dans laquelle la méthode hybride est meilleure que GMRES tout seul, bien que des délais d'attente de GMRES existent.

4.5 Conclusion et travaux futurs

Les résultats numériques montrent l'intérêt de la méthode hybride parallèle hétérogène qui permet d'augmenter le degré de parallélisme de gros grains limité dans GMRES redémarré. La première implantation étant asynchrone nous permet d'exploiter des machines parallèles obsolètes, ayant une taille mémoire suffisante, pour accélérer la convergence de GMRES assigné à une machine parallèle puissante. La deuxième implantation étant entrelacée nous permet d'exploiter le temps perdu par une machine parallèle puissante pendant l'exécution de GMRES pour exécuter la méthode hybride parallèle et accélérer la convergence de GMRES.

Dans nos travaux futurs, nous envisagons d'implanter la méthode hybride sur d'autres réseaux hétérogènes de machines et d'approfondir l'étude et la modélisation de ce type d'applications.

Cette méthode hybride parallèle est particulièrement intéressante dans le cas de systèmes linéaires avec plusieurs seconds membres puisque le même polynôme de la méthode Moindres Carrés peut être appliqué à tous ces systèmes. Dans le futur, nous envisagons d'implanter la méthode hybride parallèle en remplaçant GMRES par Global-GMRES (Gl-GMRES) [36] ou par GMRES par bloc (BGMRES) [75, 68] pour la résolution des systèmes linéaires avec plusieurs seconds membres.

Chapitre 5

Les méthodes FOM et GMRES pondérées

5.1 Introduction

Certaines méthodes de Krylov de résolution de systèmes linéaires sont conçues afin de remédier à des inconvénients de la méthode GMRES (ou de FOM).
H. Sadok a proposé la méthode CMRH [67], [32] qui utilise le processus de Hessenberg qui est moins coûteux que le processus d'Arnoldi puisque la norme utilisée est la norme infinie ($\|v\|_\infty = \max_{i=1}^{n} |v_i|$) et les composantes de la matrice de Hessenberg obtenue sont faciles calculer.
C. Le Calvez et Y. Saad ont proposé la méthode ADOP [40], [41] dans laquelle le produit scalaire euclidien de deux vecteurs est remplacé par un produit scalaire discret des polynômes correspondants à ces vecteurs. Ce nouveau produit scalaire est moins coûteux et le processus de projection obtenu est plus parallèle que celui d'Arnoldi.
En vue d'accélérer la convergence nous considérons un autre produit scalaire plus coûteux que le produit scalaire euclidien, contrairement aux cas des méthodes CMRH et ADOP.

Dans ce chapitre, nous présentons deux nouvelles méthodes de résolution de systèmes linéaires, appelées WFOM (*weighted FOM*) et WGMRES (*weighted GMRES*) [23]. Pour accélérer la convergence, ces variantes de FOM (Full Orthogonalization Method) et GMRES redémarrés utilisent un produit scalaire noté $(.,.)_D$ où D est une matrice diagonale bien choisie. Ce produit scalaire est différent du produit scalaire euclidien, de plus, il change à chaque redémarrage de la méthode. Le but de

ce changement de produit scalaire est d'accélérer la convergence des composantes du résidu qui sont les plus éloignées de zéro par rapport à celles qui sont les plus proches de zéro. Pour ce faire, un poids approprié est associé à chaque terme du produit scalaire. Un choix naturel de ces poids est celui des composantes du résidu initial. Les méthodes WFOM et WGMRES sont appelées méthodes pondérées puisque nous avons introduit des poids dans le produit scalaire. L'implantation de ces méthodes pondérées est facilitée par l'introduction du processus d'Arnoldi relatif à ce nouveau produit scalaire, appelé Arnoldi pondéré.

Ce chapitre est organisé comme suit. Dans la Section 5.2, nous introduisons le processus d'Arnoldi pondéré et nous établissons quelques propriétés de ce processus. Après la description de ces méthodes dans la Section 5.3, quelques liens entre les nouvelles méthodes et celles d'origine sont établis dans la Section 5.4. Des résultats numériques en séquentiel sont présentés dans la Section 5.5 pour montrer les bonnes performances de WFOM(m) et WGMRES(m) comparés à FOM(m) et GMRES(m), ensuite nous concluons.

Nous avons choisi d'implanter ces quatres méthodes seulement en séquentiel afin de faciliter leur comparaison. Nous envisagons l'implantation des méthodes FOM et GMRES pondérées en parallèle dans nos travaux futurs.

5.2 Le processus d'Arnoldi pondéré

Avant de décrire le processus d'Arnoldi pondéré, nous allons introduire le D-produit scalaire et la norme associée.
Soient u et v deux vecteurs de $\mathbb{R}^n$, leur D-produit scalaire est défini par

$$(u, v)_D = v^T D u = \sum_{i=1}^{n} d_i u_i v_i,$$

où $D = diag(d_1, d_2, \ldots, d_n)$ est une matrice diagonale. Nous notons d le vecteur formé des éléments diagonaux de D, $d = (d_1, d_2, \ldots, d_n)^T$.

Ce produit scalaire est bien défini si et seulement si la matrice D est définie positive, c'est-à-dire si $d_i > 0, \quad \forall \ i \in \{1, \ldots, n\}$.

Dans ce cas, nous pouvons définir la D-norme associée à ce produit scalaire par

$$\|u\|_D = \sqrt{(u, u)_D} = \sqrt{u^T D u} = \sqrt{\sum_{i=1}^{n} d_i u_i^2}, \qquad \forall \ u \in \mathbb{R}^n.$$

Le vecteur d est choisi de telle sorte que $\|d\|_2 = \sqrt{n}$. Cette normalisation permet de retrouver la norme euclidienne dans le cas où tous les éléments de d sont égaux.

Le processus d'Arnoldi construit une base orthonormale du sous-espace de Krylov $K_m(A, v)$, tandis que le processus d'Arnoldi pondéré construit une base D-orthonormale de ce sous-espace vectoriel. L'algorithme ci-dessous décrit le nouveau processus utilisant le procédé de Gram-Schmidt modifié, en prenant comme vecteur de départ $\tilde{v}_1 = v/\|v\|_D$.

Algorithme 5.2.1 : *Arnoldi pondéré (Gram-Schmidt Modifié).*

Pour $j = 1, \ldots, m$
  $w = A\tilde{v}_j$,
  pour $i = 1, \ldots, j$
    $\tilde{h}_{i,j} = (w, \tilde{v}_i)_D$,
    $w = w - \tilde{h}_{i,j}\tilde{v}_i$,
  fin pour
  $\tilde{h}_{j+1,j} = \|w\|_D$, *si* $\tilde{h}_{j+1,j} = 0$ *Stop*,
  $\tilde{v}_{j+1} = w/\tilde{h}_{j+1,j}$,
fin pour.

La base construite par ce processus $\tilde{V}_m = [\tilde{v}_1, \ldots, \tilde{v}_m]$ est D-orthonormale, ce qui ce traduit par la relation

$$\tilde{V}_m^T D \tilde{V}_m = I_m. \tag{5.1}$$

Nous remarquons à partir de la normalisation des vecteurs de la base $\tilde{V}_m$ que le processus d'Arnoldi pondéré diffère du processus d'Arnoldi appliqué à la matrice A preconditionnée par la matrice diagonale D.

La matrice carrée de Hessenberg $\widetilde{H}_m$, dont les éléments non nuls $\tilde{h}_{i,j}$ sont obtenus par l'algorithme d'Arnoldi pondéré s'exprime sous la forme

$$\widetilde{H}_m = \tilde{V}_m^T D A \tilde{V}_m. \tag{5.2}$$

Nous définissons la matrice $\overline{\widetilde{H}}_m \in \mathbb{R}^{(m+1\times m)}$ comme suit

$$\overline{\widetilde{H}}_m = \begin{pmatrix} \widetilde{H}_m \\ \tilde{h}_{m+1,m} e_m^T \end{pmatrix}.$$

De même que pour Arnoldi, le nouveau processus vérifie la relation suivante

$$A\tilde{V}_m = \tilde{V}_{m+1}\overline{\overline{H}}_m. \tag{5.3}$$

Afin d'établir des relations entre les deux processus, nous rappellerons les relations vérifiées par les matrices construites par le processus d'Arnoldi, que nous notons V_m, H_m et $\overline{H}_m$

$$\begin{aligned} V_m^T V_m &= I_m \\ AV_m &= V_{m+1}\overline{H}_m \\ H_m &= V_m^T A V_m \end{aligned} \tag{5.4}$$

Proposition 5.2.1
Supposons que l'algorithme d'Arnoldi et celui d'Arnoldi pondéré 5.2 ne présentent pas de division par zéro avant l'étape m. Alors il existe une matrice triangulaire supérieure U_m, telle que

$$\tilde{V}_m = V_m U_m, \tag{5.5}$$
$$U_m = V_m^T \tilde{V}_m, \tag{5.6}$$
$$U_m^{-1} = \tilde{V}_m^T D V_m, \tag{5.7}$$

et la matrice $\overline{\overline{H}}_m$ s'exprime en fonction de $\overline{H}_m$ comme

$$\overline{\overline{H}}_m = U_{m+1}^{-1}\overline{H}_m U_m. \tag{5.8}$$

Preuve

Comme V_j et $\tilde{V}_j$ sont des bases du sous-espace de Krylov $K_j(A, v)$ pour tout entier $j \in \{1, \ldots, m\}$, $\tilde{V}_m$ s'exprime en fonction de V_m sous la forme

$$\tilde{V}_m = V_m U_m,$$

où U_m est une matrice triangulaire supérieure de dimensions $m \times m$.

En multipliant (5.5) à gauche par V_m^T, nous obtenons (5.6), et si nous multiplions (5.5) à gauche par $\tilde{V}_m^T D$, nous obtenons la formule (5.7), d'après la D-orthonormalité de $\tilde{V}_m$.

En remplaçant la relation (5.5) dans l'expression (5.3), nous obtenons

$$AV_mU_m = V_{m+1}U_{m+1}\overline{\overline{H}}_m,$$

et en utilisant (5.4) nous avons

$$V_{m+1}\overline{H}_mU_m = V_{m+1}U_{m+1}\overline{\overline{H}}_m.$$

Comme V_{m+1} est orthonormale, la dernière relation devient

$$\overline{H}_mU_m = U_{m+1}\overline{\overline{H}}_m.$$

Finalement, multiplions cette égalité à gauche par la matrice U_{m+1}^{-1}, nous obtenons ainsi le résultat (5.8). ∎

La relation (5.8) nous incite à nous demander si les matrices H_m et $\widetilde{H}_m$ sont semblables. Malheureusement, ceci est faux comme nous allons le montrer dans la proposition suivante.

Proposition 5.2.2
Sous les mêmes hypothèses que la proposition 5.2.1, la matrice $\widetilde{H}_m$ s'exprime en fonction de H_m comme

$$\widetilde{H}_m = U_m^{-1}H_mU_m + h_{m+1,m}u_{m,m}g_{m+1}e_m^T, \tag{5.9}$$

où le vecteur $g_{m+1} \in \mathbb{R}^m$ est obtenu à partir de la colonne $m+1$ de la matrice U_{m+1}^{-1} en éliminant sa dernière composante.

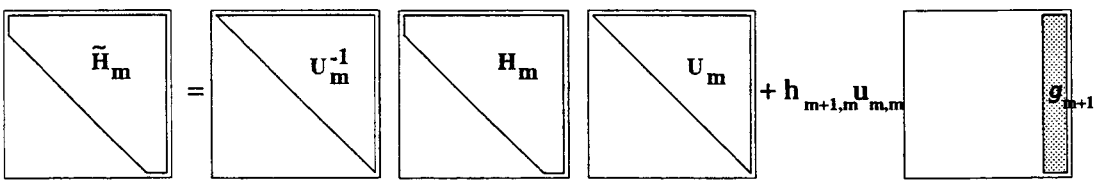

FIG. 5.1 - *La matrice $\widetilde{H}_m$ exprimée en fonction de H_m.*

Preuve

En vue d'éviter toute confusion avec la matrice U_m, nous notons $g_{i,j}$ $(1 \leq i,j \leq m)$ les composantes de la matrice U_m^{-1}.

Décomposons d'abord la matrice U_{m+1}^{-1} sous la forme

$$U_{m+1}^{-1} = \begin{bmatrix} U_m^{-1} & g_{m+1} \\ 0 \cdots 0 & g_{m+1,m+1} \end{bmatrix} = \begin{bmatrix} \widehat{U_{m+1}^{-1}} \\ 0 \cdots 0 \quad g_{m+1,m+1} \end{bmatrix}$$

où $g_{m+1} = (g_{1,m+1}, g_{2,m+1}, \ldots, g_{m,m+1})^T$. Ainsi, la matrice $\widehat{U_{m+1}^{-1}}$ représente les m premières lignes de la matrice U_{m+1}^{-1}.

À partir de la relation (5.8) et de la décomposition ci-dessus, nous pouvons écrire

$$\widetilde{H}_m = \widehat{U_{m+1}^{-1}} \overline{H}_m U_m,$$

nous avons aussi

$$\begin{aligned} \widetilde{H}_m &= [U_m^{-1}\ g_{m+1}] \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix} U_m, \\ &= U_m^{-1} H_m U_m + h_{m+1,m} g_{m+1} e_m^T U_m. \end{aligned}$$

Puisque la matrice U_m est triangulaire supérieure, le produit suivant $e_m^T U_m$ est égal à $u_{m,m} e_m^T$. Ainsi, nous obtenons la formule (5.9). ∎

Remarques :

1. Il est facile de voir à partir de la formule (5.9) que H_m et $\widetilde{H}_m$ sont semblables dans le cas où $u_{m,m} = 0$ ou $g_{m+1} = 0$. Le premier cas revient à dire que le degré du polynôme minimal de A pour v_1 est égal à m, le second cas est satisfait si v_{m+1} et $\tilde{v}_{m+1}$ sont linéairement dépendants.
2. La matrice $\widetilde{H}_m$ s'écrit en fonction de H_m et $h_{m+1,m}$, elle contient donc plus d'informations que la matrice H_m.

Corollaire 5.2.1
Sous les mêmes hypothèses que la proposition 5.2.1, la matrice H_m s'exprime en fonction de $\widetilde{H}_m$ comme

$$H_m = U_m \widetilde{H}_m U_m^{-1} + \frac{h_{m+1,m}}{u_{m+1,m+1}} u_{m+1} e_m^T, \tag{5.10}$$

où le vecteur $u_{m+1} \in \mathbb{R}^m$ est obtenu à partir de la colonne $m+1$ de la matrice U_{m+1} en éliminant la dernière composante.

Preuve

La démonstration de ce corollaire découle immédiatement de la relation (5.9) en multipliant cette relation à gauche par U_m et à droite par U_m^{-1}, et en utilisant la formule suivante

$$U_m g_{m+1} = -\frac{1}{u_{m+1,m+1}} u_{m+1}.$$

Cette relation se déduit de l'égalité $U_{m+1}U_{m+1}^{-1} = I_{m+1}$, en utilisant les décompositions de U_{m+1} et de U_{m+1}^{-1}. ∎

Complexité:

Notons N_{nz} le nombre d'éléments non nuls de la matrice A. Pour le processus d'Arnoldi comme pour celui d'Arnoldi pondéré, m étapes nécessitent le calcul de m produits matrice-vecteur avec un coût de $2mN_{nz}$ opérations.
Un produit scalaire euclidien coûte $2n$ opérations alors qu'un D-produit scalaire nécessite $3n$ opérations. Ainsi, l'étape j du processus d'Arnoldi a besoin de $2jn$ opérations, tandis que pour le processus d'Arnoldi pondéré utilisant le procédé de Gram-Schmidt modifié cette étape nécessite $3jn$ opérations. Il s'ajoute à ces opérations le coût du calcul du vecteur $j+1$ de la base pour chacun des processus qui s'élève à $2jn$ opérations. Le nombre total des opérations nécessitées par chacun des processus pour effectuer m étapes est résumé dans le tableau suivant

processus	nombre des opérations
Arnoldi	$2mN_{nz} + 2m^2n$
Arnoldi pondéré (GSM)	$2mN_{nz} + (5/2)m^2n$

Nous remarquons donc que le processus d'Arnoldi pondéré utilisant le procédé de Gram-Schmidt modifié est un peu plus coûteux que celui d'Arnoldi.

L'implantation en parallèle du processus d'Arnoldi pondéré (comme c'est le cas du processus d'Arnoldi vu dans le Chapitre 2) nécessite l'utilisation du procédé de Gram-Schmidt classique qui est plus parallèle que celui de Gram-Schmidt modifié. Le processus d'Arnoldi pondéré utilisant GSC peut être décrit par l'algorithme suivant

Algorithme 5.2.2 : *Arnoldi pondéré (Gram-Schmidt Classique).*

Pour $j = 1, \ldots, m$
  $w = A\tilde{v}_j$,
  $w' = DA\tilde{v}_j$,
  pour $i = 1, \ldots, j$
    $\tilde{h}_{i,j} = (w', \tilde{v}_i)$,
    $w = w - \tilde{h}_{i,j}\tilde{v}_i$,
  fin pour
  $\tilde{h}_{j+1,j} = \|w\|_D$, *si* $\tilde{h}_{j+1,j} = 0$ *Stop*,
  $\tilde{v}_{j+1} = w/\tilde{h}_{j+1,j}$,
fin pour.

Le nombre total d'opérations nécessité par cet algorithme pour effectuer m étapes est

$$2mN_{nz} + 2m^2n + 2mn.$$

Nous remarquons que cet algorithme est moins coûteux que l'algorithme 5.2 puisqu'il contient moins de produits de la matrice D par des vecteurs. De plus, cet algorithme est plus coûteux que le processus d'Arnoldi de seulement $2mn$ opérations, ce qui est relativement peu.

Cette version de l'algorithme d'Arnoldi pondéré est moins stable que le premier, cet incovénient peut être évité par l'utilisation de la technique de réorthogonalisation.

5.3 Description des méthodes FOM et GMRES pondérés

Comme dans toutes les méthodes de Krylov, le $m^{\text{ième}}$ $(m \geq 1)$ itéré de la méthode FOM pondérée ainsi que celui de la méthode GMRES pondérée appartiennent au sous-espace de Krylov affine $x_0 + K_m(A, r_0)$.

L'itéré x_m^{WF} de FOM pondéré est choisi de façon à ce que son résidu r_m^{WF} soit

D-orthogonal au sous-espace de Krylov $K_m(A, r_0)$

$$r_m^{\text{WF}} \perp_{\text{D}} K_m(A, r_0). \tag{5.11}$$

Cependant, l'itéré x_m^{WG} de GMRES pondéré est choisi de telle sorte qu'il minimise la D-norme de son résidu sur le sous-espace affine $x_0 + K_m(A, r_0)$.
Ainsi, cet itéré est solution du problème des moindres carrés suivant

$$\text{minimiser}_{x \in x_0 + K_m(A,r_0)} \|b - Ax\|_{\text{D}}. \tag{5.12}$$

Ces méthodes utilisent le D-produit scalaire et sa norme associée ; donc pour construire la solution sur le sous-espace affine $x_0 + K_m(A, r_0)$, nous construisons une base D-orthonormale du sous-espace de Krylov $K_m(A, r_0)$ par le processus d'Arnoldi pondéré décrit dans la Section précédente.

Un itéré de l'une des deux méthodes s'écrit sous la forme

$$x_m = x_0 + \tilde{V}_m y_m,$$

où $y_m \in \mathbb{R}^m$.

Ainsi, le résidu correspondant $r_m = b - Ax_m$ s'écrit

$$\begin{aligned} r_m &= b - A(x_0 + \tilde{V}_m y_m) \\ &= r_0 - A\tilde{V}_m y_m \\ &= \tilde{V}_{m+1}(\tilde{\beta} e_1 - \overline{\overline{H}}_m y_m), \end{aligned}$$

où $\tilde{\beta} = \|r_0\|_{\text{D}}$ et e_1 est le premier vecteur de la base canonique.

La méthode FOM pondérée consiste donc à trouver le vecteur y_m^{WF} solution du problème

$$\tilde{V}_m^T D \tilde{V}_{m+1}(\tilde{\beta} e_1 - \overline{\overline{H}}_m y_m^{\text{WF}}) = 0,$$

ce qui est équivalent au problème

$$\overline{\overline{H}}_m y_m^{\text{WF}} = \tilde{\beta} e_1. \tag{5.13}$$

Concernant la méthode GMRES pondérée, comme la matrice $\tilde{V}_{m+1}$ est D-orthonormale, nous avons

$$\begin{aligned} \|r_m\|_{\text{D}}^2 &= \|\tilde{V}_{m+1}(\tilde{\beta} e_1 - \overline{\overline{H}}_m y_m)\|_{\text{D}}^2 \\ &= \|\tilde{\beta} e_1 - \overline{\overline{H}}_m y_m\|_2^2, \end{aligned}$$

et donc le problème (5.12) se réduit à chercher le vecteur y_m^{WG} solution du problème de minimisation

$$\text{minimiser}_{y \in \mathbb{R}^m} \|\tilde{\beta} e_1 - \overline{\overline{H}}_m y\|_2. \tag{5.14}$$

Les solutions des problèmes (5.13) et (5.14) peuvent être obtenues par la factorisation QR des matrices $\widetilde{H}_m$ et $\overline{\overline{H}}_m$ respectivement, comme dans le cas des algorithmes FOM et GMRES [47], [59].

Dans le cas où m est égal au degré du polynôme minimal de A pour r_0, le sous-espace de Krylov est invariant et donc la solution obtenue par les deux méthodes est la solution exacte du système linéaire $Ax = b$.

Les algorithmes de FOM et GMRES pondérés nécessitent le stockage de la matrice $\widetilde{V}_m$, c'est-à-dire m vecteurs de dimension n à l'étape m. La valeur de m est donc limitée par les contraintes de stockage et par le risque de propagation des erreurs d'arrondi dans le processus d'Arnoldi pondéré. Le remède que nous pouvons adopter en vue d'éviter ces problèmes est celui utilisé pour FOM et GMRES, qui est le redémarrage de ces algorithmes après m itérations.

Les algorithmes ainsi obtenus sont appelés FOM pondéré redémarré et GMRES pondéré redémarré, ils sont notés WFOM(m) and WGMRES(m). Notons que les méthodes pondérées complètes c'est-à-dire sans redémarrage donnent des résultats semblables à ceux des méthodes non pondérées correspondantes. La pondération est utile dans le cas des méthodes redémarrées.

En prenant le choix $d = \sqrt{n}\dfrac{|r_0|}{\|r_0\|_2}$, ces algorithmes favorisent les composantes du résidu qui sont éloignées de zéro et donnent moins d'importance à celles qui sont voisines de zéro. Les algorithmes WFOM(m) et WGMRES(m) sont décrits ci-dessous.

Algorithme 5.3.1 : *WFOM(m) (WGMRES(m)).*

1. *Initialisation :*
   choisir x_0, m *et la tolérance* ε,
   calculer $r_0 = b - Ax_0$.

2. *Poser* $d = \sqrt{n}|r_0|/\|r_0\|_2$, *calculer* $\tilde{\beta} = \|r_0\|_D$ *et* $\tilde{v}_1 = r_0/\tilde{\beta}$.

3. *Construire la base D-orthonormale* $\widetilde{V}_m$ *par Arnoldi pondéré, en prenant* $\tilde{v}_1$ *comme vecteur initial.*

4. *Former l'approximation* x_m *:*
   WFOM : *Calculer la solution du système* $\widetilde{H}_m y_m = \tilde{\beta} e_1$
   par la factorisation QR de $\widetilde{H}_m$,
   calculer $x_m = x_0 + \widetilde{V}_m y_m$ *et* $r_m = b - Ax_m$.
   WGMRES : *Calculer la solution du problème*
   $$y_m = \arg\min_{y \in \mathbb{R}^m} \|\tilde{\beta} e_1 - \widetilde{\widetilde{H}}_m y\|_2$$
   par la factorisation QR de $\widetilde{\widetilde{H}}_m$,
   calculer $x_m = x_0 + \widetilde{V}_m y_m$ *et* $r_m = b - Ax_m$.

5. *Redémarrage :*
   si $\|r_m\|_2 \leq \varepsilon$ *Stop*
   sinon calculer $x_0 = x_m$, $r_0 = r_m$, *et aller en 2.*

Remarquons que ces méthodes diffèrent des méthodes FOM(m) et GMRES(m) preconditionnées à gauche par la matrice diagonale D puisque les processus d'Arnoldi pondéré est celui d'Arnoldi preconditionné sont différents.

Quand une division par zéro (*breakdown*) survient à l'itération j, dans l'algorithme d'Arnoldi pondéré, ce qui veut dire que $\tilde{h}_{j+1,j} = 0$, j est alors le degré du polynôme minimal de A pour r_0. Dans ce cas le processus est arrêté et la solution du système est donnée par

$$x_j = x_0 + \widetilde{V}_j \widetilde{H}_j^{-1}(\tilde{\beta} e_1)$$

pour les deux méthodes.
Dans le cas où la quantité $\tilde{h}_{j+1,j}$ n'est pas nulle mais très proche de zéro, ce phénomène est dit *near-breakdown* et une approximation de la solution est donnée par la même formule ci-dessus.

Comme nous l'avons vu auparavant, l'algorithme de GMRES pondéré redémarré minimise la D-norme du résidu, cette norme est différente à chaque redémarrage

puisque la matrice D s'écrit en fonction du résidu r_0 initial utilisé pour le redémarrage. La norme euclidienne du résidu n'est donc pas monotone comme c'est le cas de l'algorithme de GMRES.

5.4 Liens avec FOM et GMRES

Commençons tout d'abord par établir un lien entre FOM et WFOM. Exprimons la correction $x_m^{\mathrm{WF}} - x_0 \in K_m(A, r_0)$ dans les bases $\widetilde{V}_m$ et V_m sous la forme

$$x_m^{\mathrm{WF}} - x_0 = \widetilde{V}_m y_m^{\mathrm{WF}} = V_m \check{y}_m^{\mathrm{WF}},$$

où $y_m^{\mathrm{WF}} \in \mathbb{R}^m$ et $\check{y}_m^{\mathrm{WF}} \in \mathbb{R}^m$ satisfont la relation $y_m^{\mathrm{WF}} = U_m^{-1}\check{y}_m^{\mathrm{WF}}$.
Rappelons que le vecteur y_m^{WF} est solution du système linéaire (5.13)

$$\widetilde{H}_m y_m^{\mathrm{WF}} = \tilde{\beta} e_1.$$

Dans ce système, remplaçons y_m^{WF} par $U_m^{-1}\check{y}_m^{\mathrm{WF}}$ et multiplions à gauche par U_m, nous obtenons

$$U_m \widetilde{H}_m U_m^{-1} \check{y}_m^{\mathrm{WF}} \;=\; U_m(\tilde{\beta} e_1).$$

Notons $\beta = \|r_0\|_2$, alors $U_m(\tilde{\beta}e_1) = \beta e_1$, puisque (βe_1) est les coordonnées de r_0 dans la base V_m et $(\tilde{\beta}e_1)$ ses coordonnées dans la base $\widetilde{V}_m$.
En utilisant la relation (5.10) du Corollaire 5.2.1, nous écrivons

$$\left(H_m - \frac{h_{m+1,m}}{u_{m+1,m+1}} u_{m+1} e_m^T\right) \check{y}_m^{\mathrm{WF}} = \beta e_1,$$

ainsi, $\check{y}_m^{\mathrm{WF}}$ est solution du système linéaire suivant

$$\widehat{H}_m \check{y}_m^{\mathrm{WF}} = \beta e_1, \tag{5.15}$$

où $\widehat{H}_m$ est la matrice dont les $m-1$ premières colonnes sont identiques à celles de la matrice H_m (voir figure 5.2) et $\widehat{H}_m$ est définie par

$$\widehat{H}_m = H_m - \frac{h_{m+1,m}}{u_{m+1,m+1}} u_{m+1} e_m^T.$$

Nous déduisons de ce résultat que la connaissance des composantes de $\tilde{v}_{m+1}$ dans la base orthonormale nous permet de calculer l'itéré x_m^{WF} à partir du processus d'Arnoldi.

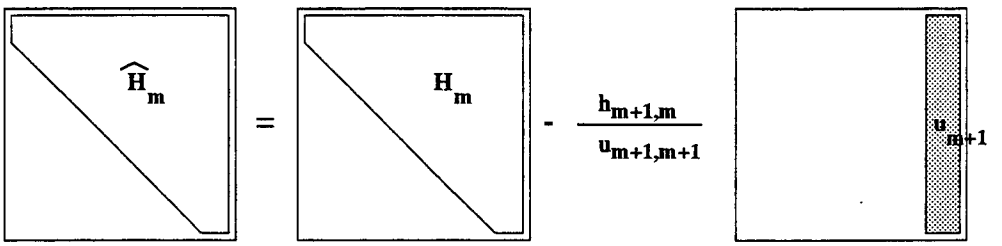

FIG. 5.2 - *La matrice $\widehat{H}_m$ en fonction de H_m.*

Ce résultat est très important puisqu'il nous permet de déterminer l'itéré x_m^{WF} de telle façon que son résidu soit colinéaire à une direction choisie au départ dans le sous-espace $K_{m+1}(A, r_0)$. Cette direction doit être choisie telle que la composante selon v_{m+1} soit non nulle, puisque dans la formule de $\widehat{H}_m$, nous divisons par cette composante.

Maintenant, nous allons établir le lien entre GMRES et WGMRES. Comme pour FOM et WFOM, nous écrivons la correction $x_m^{\mathrm{WG}} - x_0$ sous la forme

$$x_m^{\mathrm{WG}} - x_0 = \widetilde{V}_m y_m^{\mathrm{WG}} = V_m \check{y}_m^{\mathrm{WG}}.$$

Nous avons déjà vu que y_m^{WG} est solution du problème de minimisation (5.14)

$$y_m^{\mathrm{WG}} = \arg\min_{y \in \mathbb{R}^m} \quad \|\tilde{\beta} e_1 - \overline{\widetilde{H}}_m y\|_2.$$

Si nous cherchons $\check{y}_m^{\mathrm{WG}}$, les composantes de $x_m^{\mathrm{WG}} - x_0$ dans la base orthonormale, nous obtenons

$$\check{y}_m^{\mathrm{WG}} = \arg\min_{\check{y} \in \mathbb{R}^m} \quad \|\tilde{\beta} e_1 - \overline{\widetilde{H}}_m U_m^{-1} \check{y}\|_2,$$

ainsi, en utilisant la formule (5.8) nous avons

$$\check{y}_m^{\mathrm{WG}} = \arg\min_{\check{y} \in \mathbb{R}^m} \quad \|U_{m+1}^{-1}(\beta e_1 - \overline{H}_m \check{y})\|_2.$$

Finalement, dans le but d'obtenir une relation entre GMRES et WGMRES, nous pouvons voir le vecteur $\check{y}_m^{\mathrm{WG}}$ comme solution du même problème d'optimisation que celui de GMRES, mais, au lieu de la norme euclidienne, la norme correspondante à la matrice symétrique $U_{m+1}^{-T} U_{m+1}^{-1}$ est minimisée

$$\min_{\check{y} \in \mathbb{R}^m} \|\beta e_1 - \overline{H}_m \check{y}\|_{U_{m+1}^{-T} U_{m+1}^{-1}}.$$

Donc, contrairement au cas de FOM et WFOM, il est nécessaire de connaître les composantes de tous les vecteurs de $\widetilde{V}_{m+1}$ dans la base orthonormale V_{m+1} pour retrouver l'itéré x_m^{WG} à partir du processus d'Arnoldi.

5.5 Résultats expérimentaux

Dans cette Section nous allons présenter des exemples numériques pour comparer les méthodes FOM et GMRES avec leurs versions pondérées. Les essais numériques sont effectués sur un processeur Alpha, à l'aide du compilateur FORTRAN 77 en double précision. Les matrices testées sont non symétriques, creuses et proviennent du site MatrixMarket (contenant des collections très connues de matrices : Harwell-Boeing [18] et SPARSKIT [64]). Les matrices sont utilisées avec le format de stockage CSR et elles sont lu à partir de fichier ASCII sous le format Harwell-Boeing. Dans cette collection, une estimation du conditionnement est donnée pour certaines matrices test, nous reporterons cette estimation pour chaque exemple si elle est disponible.

Pour tous les systèmes, le second membre est un vecteur aléatoire dont les composantes sont uniformément prises dans l'intervalle $[0, 1]$. L'approximation initiale x_0 de la solution est le vecteur nul $x_0 = (0, \ldots, 0)$.
Le critère d'arrêt qui contrôle la convergence des algorithmes est le test

$$\|r\|_2/\|b\|_2 < epsilon,$$

la valeur de *epsilon* dépend de l'exemple. En général, cette valeur est la plus petite puissance de 10 qui nous permet d'arrêter l'algorithme avant la stagnation. L'unité de mesure du temps dans tous les exemples est la seconde.
Dans les tableaux de résultats, le symbol * indique que la convergence n'a pas été atteinte

Exemple 1 : la matrice **add20**

C'est un matrice de dimensions 2395 x 2395, qui provient du groupe HAMM parmi "*the independent sets and generators*" du site MatrixMarket, avec 17319 éléments non nuls. Le conditionnement est estimé à 1.76e+4.

Nous avons testé cet exemple avec plusieurs valeurs de m $(5, 10, 15, \ldots, 80)$, nous observons toujours que WFOM(m) est meilleur que FOM(m) et que WGMRES(m) est meilleur que GMRES(m). Afin d'illustrer cet exemple la figure 5.3 et la table suivante représentent les résultats obtenus avec $m = 10$ et nous avons pris $epsilon = 10^{-12}$.

méthode	FOM	WFOM	GMRES	WGMRES
itérations	865	400	772	136
temps(s)	79.44	37.34	75.25	13.17

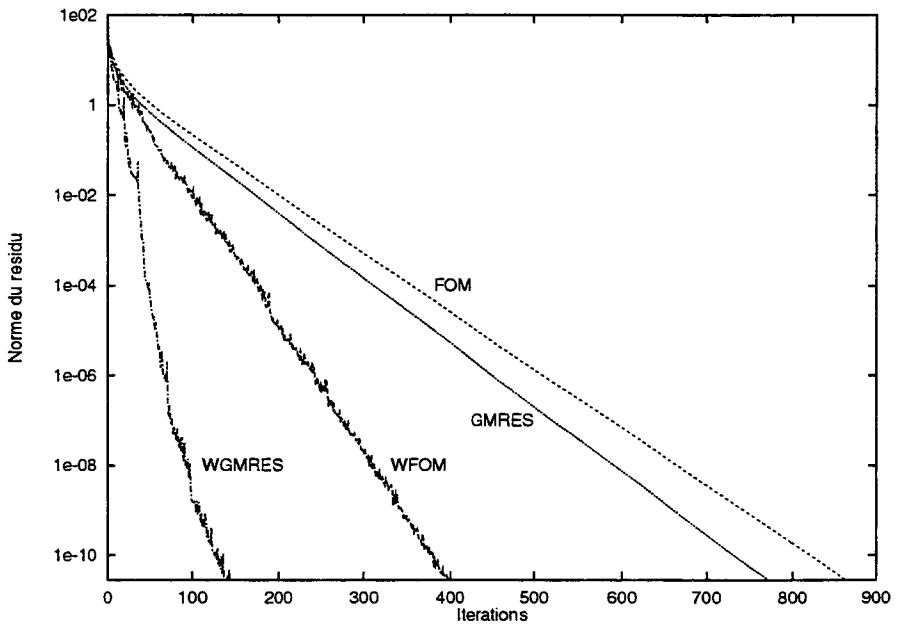

FIG. 5.3 - *Convergence des différentes méthodes pour la matrice add20, avec m=10.*

Exemple 2 : la matrice **orsirr_1**

La matrice test est obtenue à partir de la simulation 3D d'un réservoir d'huile. Cette matrice de dimensions 1030 x 1030 provient du groupe OILGEN de la collection Harwell-Boeing et contient 6858 éléments non nuls. Le conditionnement est estimé à 1e+2.
Nous prenons $epsilon = 10^{-11}$ et l'algorithme est arrêté si la convergence n'est pas atteinte après 2000 itérations.

Dans la table 5.1, nous reportons les résultats correspondants à différentes valeurs de m. La figure 5.4 représente les courbes des quatre méthodes avec $m = 20$. Nous remarquons que, pour toutes les dimensions de sous-espace de Krylov, chaque méthode pondérée converge en moins d'itérations et en moins de temps que la méthode non pondérée correspondante.
Dans le cas particulier où $m = 10$, la méhode WGMRES converge, cependant la norme du résidu de GMRES stagne à 1.65e+01. La convergence de FOM et WFOM est lente, mais, à l'itération maximale 2000, la norme du résidu est réduite à 5.24E-04 pour FOM et 2.26E-06 pour WFOM.

Dans la table 5.1, nous notons en gras le temps minimal relatif de chaque méthode. Nous remarquons que la dimension du sous-espace de Krylov correspondant à l'optimum pour WGMRES (respectivement WFOM) est plus petite que celle relative à GMRES (respectivement FOM), donc le stockage nécessité par les méthodes

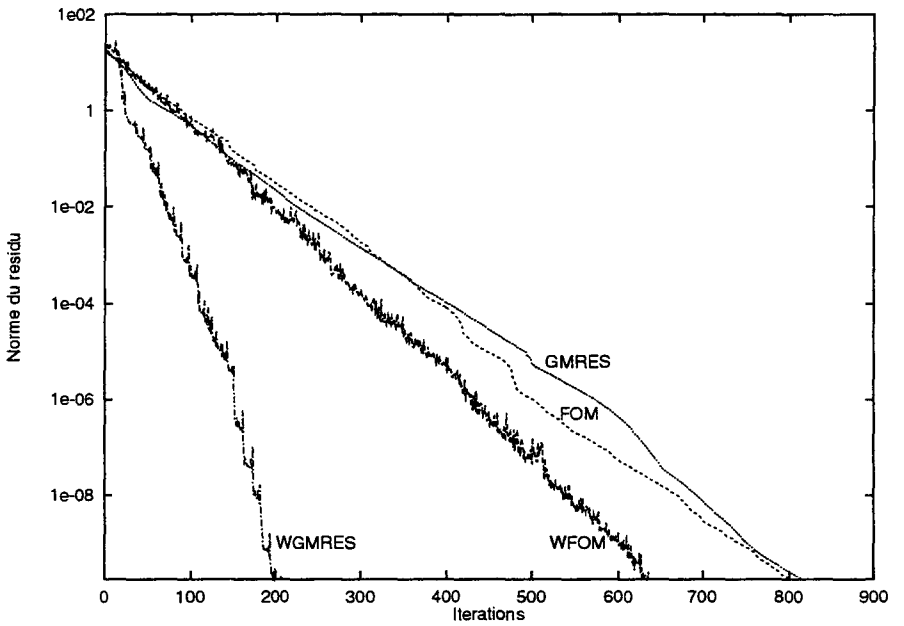

FIG. 5.4 - *Convergence des différentes méthodes pour la matrice orsirr_1, avec m=20.*

m	FOM		WFOM		GMRES		WGMRES	
	itér	temps	itér	temps	itér	temps	itér	temps
10	2000	75.35	2000	75.96	2000	73.66	1734	68.46
	$\|r\|_2$=5.24E-04		$\|r\|_2$=2.26E-06		$\|r\|_2$=1.65E+01			
20	800	77.62	630	66.89	815	82.75	198	**21.08**
30	325	63.40	167	35.67	300	57.56	120	25.59
40	142	44.35	82	**27.09**	156	47.88	64	21.93
50	64	**30.12**	53	27.90	77	35.85	49	24.48
60	51	31.78	39	27.31	54	**34.09**	39	27.32
70	42	36.37	32	31.58	42	35.94	28	27.05
80	36	39.01	26	31.21	34	37.13	27	33.01

TAB. 5.1 - *Résultats obtenus avec la matrice orsirr_1.*

pondérées pour obtenir la convergence optimale n'est pas plus élevé que celui des méthodes classiques.

Exemple 3 : la matrice **fs_541_2**

Cette matrice de dimensions 541 x 541 provient du groupe FACSIMILE des problèmes de la pollution atmosphérique de la collection Harwell-Boeing. Elle possède 4285 éléments non nuls. Le conditionnement est estimé à 7.7e+11.

Pour cet exemple nous prenons $m = 40$ et $epsilon = 10^{-10}$.

Les résultats donnés dans la figure 5.5 et dans la table suivante montrent qu'exceptionnellement FOM converge plus vite que GMRES qui stagne à la valeur 1.16e+01. La méthode WGMRES converge, bien que jusqu'à l'itération 107 la courbe de la norme du résidu oscille au-dessus de celle de GMRES.

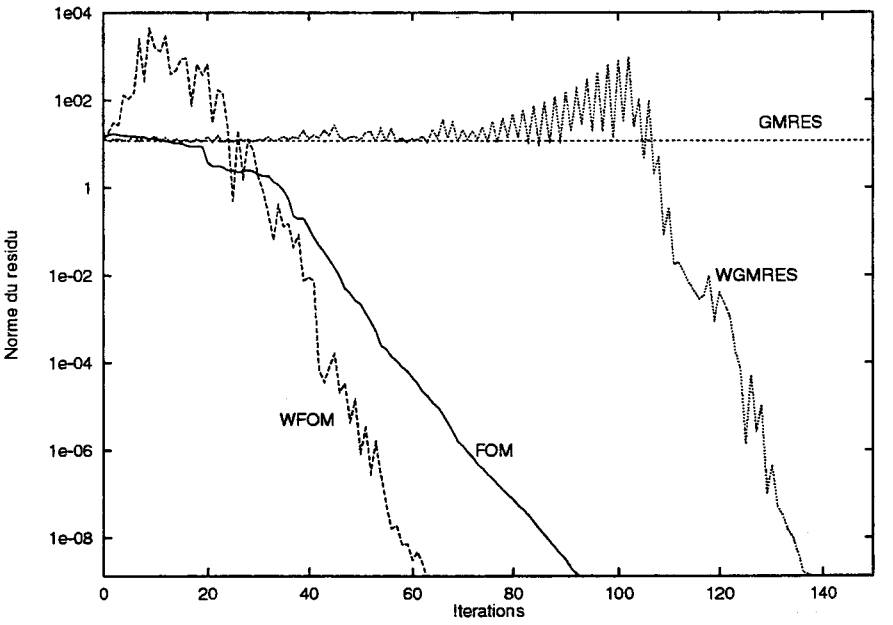

FIG. 5.5 - *Convergence des différentes méthodes pour la matrice fs_541_2, avec m=40.*

méthode	FOM	WFOM	GMRES	WGMRES
itérations	93	63	*	138
temps(s)	14.67	12.75	*	27.50

Exemple 4 : la matrice **bfw782a**

Cette matrice test de dimensions 782 x 782 provient du groupe BFWAVE de la collection NEP. Cette matrice admet 7514 éléments non nuls et son conditionnement est estimé à 4.6e+03.
Les résultats sont présentés dans la figure 5.6 et la table ci-dessous, avec $m = 20$ et $epsilon = 10^{-12}$.

Cet exemple est étudié afin de montrer que l'amélioration apportée par les méthodes pondérées ne l'est pas seulement du point de vue du nombre d'itérations mais aussi du point de vue du temps d'exécution.

Ainsi, nous remarquons que les courbes dans les deux tracés de la figure 5.6 ont la même allure.

méthode	FOM	WFOM	GMRES	WGMRES
itérations	*	314	506	178
temps	*	24.52	36.15	14.37

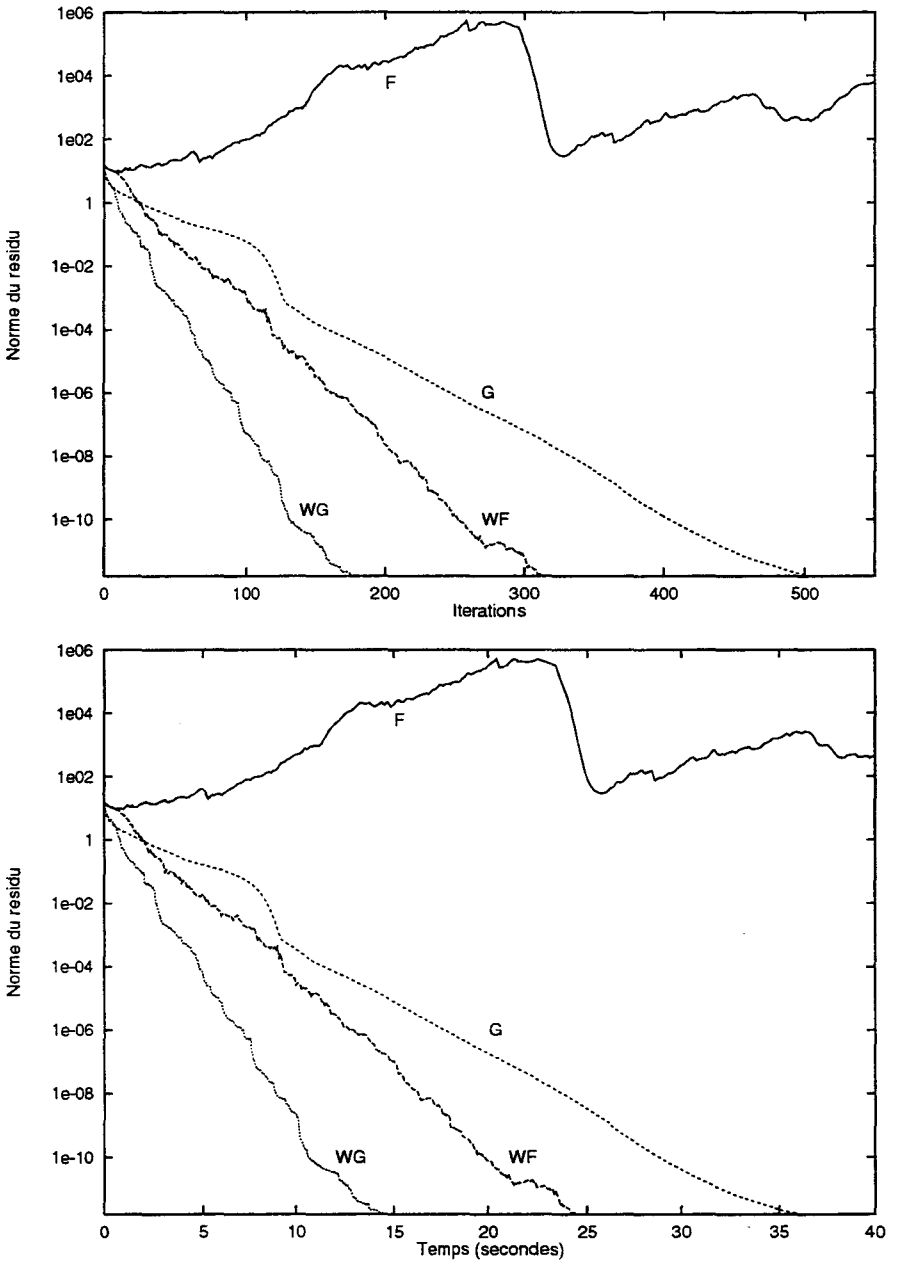

FIG. 5.6 - *La matrice bfw782a, avec m=20.*

Exemple 5 : la matrice **memplus**

Cette matrice de dimensions 17758 x 17758 provient du groupe HAMM, l'un des groupes de "*the independent sets and generators*" du site MatrixMarket. Cette matrice possède 126150 éléments non nuls. Aucune estimation du conditionnement n'est donnée.
Les résultats sont présentés dans la figure 5.7 et dans la table suivante, avec $m = 30$ et $epsilon = 10^{-12}$.

Nous étudions cet exemple pour montrer les performance des méthodes WFOM et WGMRES sur les systèmes creux de grande taille. Le facteur d'amélioration relativement au temps est de 1.63 pour WFOM et 1.82 pour WGMRES.

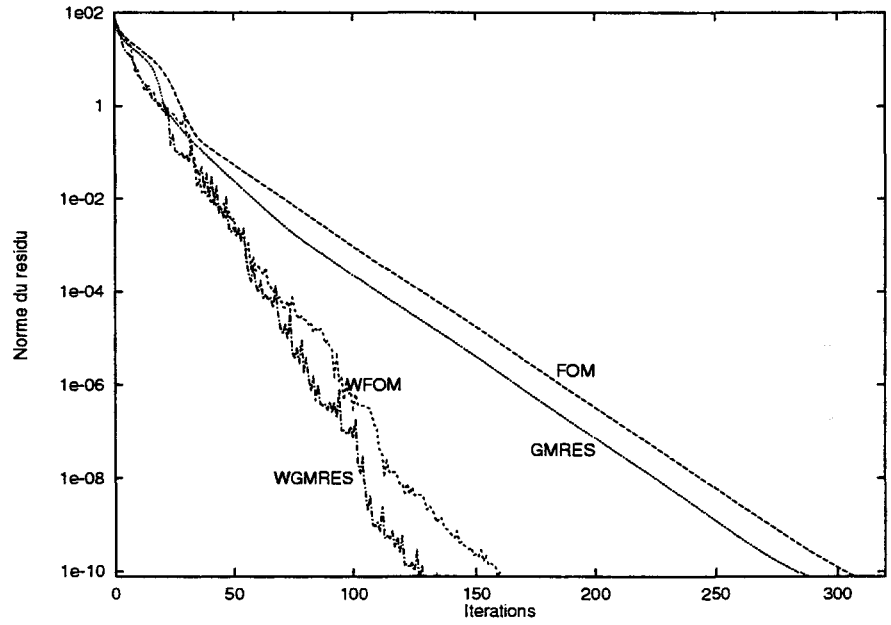

FIG. 5.7 - *Convergence des différentes méthodes pour la matrice memplus, avec m=30.*

méthode	FOM	WFOM	GMRES	WGMRES
itérations	308	161	289	127
temps(s)	1341.01	819.76	1245.49	681.79

5.6 Conclusion et travaux futurs

Dans ce chapitre, nous avons présenté deux nouvelles méthodes itératives WFOM et WGMRES pour la résolution des systèmes linéaires. Les différents exemples traités

dans la Section précédente illustrent l'intérêt de ces méthodes, même si la convergence est irrégulière puisque la norme du résidu présente des oscillations.

Toutes les relations déjà établies entre FOM et GMRES [10] [15] sont aussi valables pour WFOM et WGMRES. En particulier, les itérés de la méthode WGMRES peuvent être obtenus à partir de ceux de la méthode WFOM par une technique similaire à celle du lissage MRS (Minimal Residual Smoothing technique) [77] et dans laquelle la norme change à chaque redémarrage [33]. La technique de lissage MRS peut servir à rendre monotone la convergence de WFOM et WGMRES.

Comme le nombre d'opérations du processus Weighted-Arnoldi utilisant le procédé de Gram-Schmidt classique n'est pas très grand par rapport à celui du processus d'Arnoldi et puisque les méthodes pondérées convergent en moins d'itérations, les versions parallèles de ces méthodes semblent meilleures que celles des méthodes d'origine.

Dans la Section 5.4, nous avons vu que l'itéré x_m^{WF} de WFOM peut être obtenu facilement à partir du processus d'Arnoldi si les composantes du vecteur $\tilde{v}_{m+1}$ (dans la base construite par Arnoldi) sont connues. Le vecteur $\tilde{v}_{m+1}$ constitue une direction du vecteur résidu de WFOM.
La direction du résidu dans la méthode FOM peut donc être choisie dans le sous-espace de Krylov $K_{m+1}(A, r_0)$ en changeant la matrice H_m, construite par le processus d'Arnoldi, par la matrice $\widehat{H}_m$ (voir figure 5.2) dépendant de la nouvelle direction du résidu. Cette idée peut nous mener à construire une méthode puissante si le problème du choix des directions des résidus est résolu.

Dans les travaux futurs, nous allons essayer de résoudre le problème ouvert que nous venons de soulever, trouver d le vecteur optimal, appliquer la technique de pondération aux méthodes QMR [28] et TFQMR [27] basées sur le processus de Lanczos [39] ainsi que préconditionner les méthodes pondérées par des préconditionnements comme SSOR, ceux basés sur la factorisation LU incomplète (ILU) [59] et un préconditionneur polynomial [59]. L'implantation de toutes ces méthodes en parallèle est aussi envisagée.

Conclusion générale et perspectives

Conclusion générale

Cette thèse débute par une comparaison des processus d'orthonormalisation de famille de vecteurs ; Gram-Schmidt Classique et Gram-Schmidt modifié. Le processus Gram-Schmidt modifié est celui qui est le plus stable numériquement et est le plus parallélisable dans le cas où la famille de vecteurs est connue au début. Par contre, pour une famille de vecteurs générant un sous-espace de Krylov, qui se calcule au fur et à mesure, le processus de Gram-Schmidt Classique réorthogonalisé est stable numériquement et est le mieux adapté au parallélisme. Nous avons donc utilisé ce dernier pour l'implantation des méthodes GMRES et Arnoldi.

De nos jours, les problèmes à résoudre sont de plus en plus difficiles et ont une complexité croissante. Les méthodes hybrides sont de nouvelles méthodes composées de plusieurs algorithmes et chaque algorithme est adapté à un type d'architecture de machines et à un modèle de programmation (data-parallèle, parallélisme de tâches,...). Les environnements hétérogènes sont donc le milieu pour le développement de ces méthodes hybrides hétérogènes qui sont très prometteuses pour les problèmes de très grande taille. La méthode hybride parallèle GMRES(m_1)/LS(k,l)-Arnoldi(m_2) que nous avons introduite est un exemple de ces méthodes, elles nous permet d'augmenter le degré de parallélisme de gros grains qui est limité dans la méthode GMRES et d'accélérer ainsi la convergence. Les résultats numériques que nous avons présentés montrent l'intérêt de cette méthode que nous avons implanté de deux façon différentes, l'une asynchrone et l'autre entrelacée sur deux résaux hétérogènes.

Nous avons également introduit les nouvelles méthodes itératives WFOM et WGMRES pour la résolution des systèmes linéaires et nous avons établi les liens les reliant aux méthodes FOM et GMRES. Ces méthodes convergent plus vite que les méthodes d'origine. Nous pensons que même si les méthodes pondérées sont un

peu plus coûteuses, elles sont plus attractives surtout leur implantation dans un environnement parallèle.

Perspectives

Nos futurs travaux sont naturellement dans la continuation des travaux présentés dans cette thèse. Nous avons déjà commencé à réaliser une partie de ces travaux et nous envisagons de réaliser le reste dans le futur proche.

En plus d'implanter les méthodes pondérées en parallèle, nous envisagons de remplacer dans la méthode hybride la méthode GMRES par la méthode WGMRES.

La méthode hybride parallèle est très intéressante dans le cas des systèmes linéaires avec plusieurs seconds membres. Pour cela, nous sommes en train de remplacer dans la méthode hybride parallèle la méthode GMRES par la méthode Global-GMRES (Gl-GMRES) [36] et nous envisagons aussi de remplacer GMRES par un GMRES par bloc (BGMRES) [75, 68]. Nous projetons également de faire la même chose avec la méthode GMRES pondéré.

Nous avons commencé à effectuer des tests sur le choix de la direction des résidus de la méthode FOM (voir Section 5.6). Nous avons pris comme direction du résidu de la méthode FOM avant le redémarrage le vecteur de Ritz dominant dans l'expression du résidu initial. Des tests effectués en Matlab avec des matrices issues du site MatrixMarket ont donné des résultats intéressants. Nous envisagons également d'adapter ce choix à la méthode GMRES.
Les méthodes que nous pouvons obtenir de cette façon sont elles aussi hétérogènes, elles peuvent donc être implantées sur des réseaux hétérogènes. Ces points la meritent d'être développés plus profondément.

Bibliographie

[1] W. E. ARNOLDI, *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., 9 (1951) 17–29.

[2] Z. BAI, D. HU, L. REICHEL, *An implementation of the GMRES method using QR factorization*, In J. Dongarra, K. Kenedy, P. Messina, D. C. Sorensen, and R. G. Voigt, editors, *Fifth SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, 1992.

[3] Z. BAI, D. DAY, J. DEMMEL, J. DONGARRA, *A test matrix collection for non-Hermitian eigenvalue problems*, Release 1.0, 1996.

[4] A. BEGUELIN, J. DONGARRA, A. GEIST, R. MANCHEK, V. SUNDERAM, *A user's guide to PVM-Parallel Virtual Machine*, Research Report ORNL/TM-11826, Oak Ridge National Laboratory, 1992.

[5] M. BENNANI, *A propos de la stabilité de la résolution d'équations sur ordinateurs*, Thèse, Institut National Polytechnique de Toulouse, Décembre 1991.

[6] G. BERGÈRE, *Contribution à une programmation parallèle hétérogène de méthodes numériques hybrides*, Thèse, Université des Sciences et Technologies de Lille, à paraître.

[7] G. BERGÈRE, A. ESSAI, S. PETITON, *Utilisation asynchrone de machines parallèles hétérogènes pour l'accélération de la résolution de systèmes linéaires*, *Actes de RenPar'9*, Lausanne, 1997.

[8] T. BLANK, *The MasPar MP-1*, IEEE Transactions on Computers, (1991) 20–24.

[9] R. F. BOISVERT, R. POZO, K. REMINGTON, R. BARRETT, J. J. DONGARRA, *The Matrix Market: A web resource for test matrix collections*, In the Quality of Numerical Software: Assessment and Enhancement, (R. F. Boisvert, ed.), Chapman & Hall, London, (1997) 125-137.

[10] P. N. BROWN, *A theoretical comparison of the Arnoldi and GMRES algorithms*, SIAM J. Sci. Statist. Comput., 12 (1991) 58–78.

[11] G. BRUSSINO, V. SONNAD, *A comparison of direct and preconditioned iterative techniques for sparse, unsymmetric systems of linear equations*, Int. J. Numer. Meth. Engrg., 28 (1989) 801–815.

[12] D. CALVETTI, J. PETERSON, L. REICHEL, *A parallel implementation of the GMRES method*, Numer. Lin. Alg, To appear.

[13] L. COLOMBET, *Parallélisation d'applications pour des réseaux de processeurs homogènes ou hétérogènes*, Thèse, Institut National Polytechnique de Gronoble, Octobre 1994.

[14] M. COSNARD, D. TRYSTRAM, *Algorithmes et architectures parllèles*, InterEditions, Paris, 1993.

[15] J. CULLUM, A. GREENBAUM, *Relations between Galerkin and norm-minimizing iterative methods for solving linear systems*, SIAM J. Matrix Anal. Appl., 17 (1996) 223–247.

[16] J. DE RUMEUR, *Communications dans les réseaux de processeurs*, Masson, Paris, 1994.

[17] E. DE STURLER, H. A. VAN DER VORST, *Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers*, Appl. Numer. Math., 18 (1995) 441–459.

[18] I. S. DUFF, R. G. GRIMES, J. G. LEWIS, *User's guide for the Harwell-Boeing sparse matrix collection (Release I)*, Tech. Rep. TR/PA/92/86, CERFACS, Toulouse, France, 1992.

[19] G. EDJLALI, *Contribution à la parallélisation de méthodes itératives hybrides pour matrices creuses sur architectures hétérogènes*, Thèse, Université de Paris VI, 1994.

[20] G. EDJLALI, S. PETITON, N. EMAD, *Interleaved Parallel Hybrid Arnoldi Method for a Parallel Machine and a Network of Workstations*, (1996).

[21] A. EDREI, *Sur les déterminants récurrents et les singularités d'une fonction donnés par son developpement de Taylor*, Composito. Math., 7 (1939) 20–88.

[22] H. C. ELMAN, Y. SAAD, P. SAYLOR, *A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems for linear equations*, SIAM J. Sci. Statist. Comput., 7 (1986) 840–855.

[23] A. ESSAI, *Weighted FOM and GMRES for solving nonsymmetric linear systems*, à paraître dans Numer. Algo.

[24] A. ESSAI, G. BERGÈRE, S. PETITON, *Heterogeneous parallel hybrid GMRES/LS-Arnoldi method*, In J. Dongarra, P. Messina, D. Sorensen, and R. Voigt, editors, *Parallel Processing for Scientific Computing*, SIAM, accepté pour paraître en 1999.

[25] M. J. FLYNN, *Very high-speed computing systems*, Proceedings of the IEEE, (1966) 1901–1909.

[26] I. FOSTER, *Task Parallelism and High-Performance Languages*, Mathematics and Computer Science Division, Argonne National Laboratory, 1995.

[27] R. W. FREUND, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear system*, SIAM J. Sci. Comput., 14 (1993) 470–482.

[28] R. W. FREUND, N. M. NACHTIGAL, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991) 315–339.

[29] G. H. GOLUB, C. F. VAN LOAN, *Matrix Computations*, The Johns University Press, Baltimore, third edition, 1994.

[30] L. HAGEMAN, D. M. YOUNG, *Applied iterative methods*, Academic Press, 1981.

[31] M. R. HESTENES, E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. stand., 49 (1952) 409–435.

[32] M. HEYOUNI, *Méthode de Hessenberg généralisée et applications*, Thèse, Université des Sciences et Technologies de Lille, Décembre 1996.

[33] M. HEYOUNI, H. SADOK, *On variable smoothing procedure for Krylov subspaces methods*, Lin. Alg. Appl., 286 (1998) 131–149.

[34] C. HIRSCH, *Numerical Computation of Internal and External Flows*, John Wiley and Sons, NY, 1988.

[35] W. JALBY, B. PHILIPPE, *Stability analysis and improvement of the block Gram–Schmidt algorithm*, SIAM J. Sci. Statist. Comput., 12 (1991) 1058–1073.

[36] K. JBILOU, A. MESSAOUDI, H. SADOK, *Global FOM and GMRES algorithms for solving linear systems of equations with multiple right-hand sides*, submitted.

[37] C. JOHNSON, *Numerical Solutions of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, Cambridge, UK, 1987.

[38] G. T. KELLEY, *Iterative methods for linear and nonlinear equations*, SIAM, Philadelphia, 1995.

[39] C. LANCZOS, *Solution of systems of linear equations by minimized iteration*, J. Res. Nat. Bur. stand., 49 (1952) 33–53.

[40] C. LE CALVEZ, Y. SAAD, *Modified Krylov acceleration for parrallel environments*, à paraître dans Numer. Appl. Math.

[41] C. LE CALVEZ, *Accélération de méthodes de Krylov pour la résolution de systèmes linéaires creux sur machines parallèles*, Thèse, Université des Sciences et Technologies de Lille, Janvier 1999.

[42] C. LE CALVEZ, B. MOLINA, *Implicitly restarted and deflated FOM and GMRES*, esoumis à Numer. Algo.

[43] C. E. LEISERSON, *Fat-trees: universal networks for hardware-efficient supercomputing*, IEEE Transactions on Computers, (1985) 892–901.

[44] F. LEJA, *Sur certaines suites liées aux ensembles plans et leur application à la représentation conforme*, Ann. Polon. Math., 4 (1957) 8–13.

[45] T. A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977) 307–327.

[46] T. A. MANTEUFFEL, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Numer. Math., 31 (1978) 183–208.

[47] R. S. MARTIN, G. PETERS, J. H. WILKINSON, *The QR Algorithm for Real Hessenberg Matrices*, Numer. Math., 14 (1970) 219–231.

[48] *MasPar Fortran Reference Manual, PN 9303-0300, (1993).*

[49] *MasPar Fortran User Guide, PN 9303-0100, (1993).*

[50] MESSAGE PASSING INTERFACE FORUM, *Document for a standard Message-Passing Interface* , 1993.

[51] R. B. MORGAN, *A restarted GMRES method augmented with eigenvectors* , SIAM J. Matrix Anal. Appl., 16 (1995) 1154–1171.

[52] N. M. NACHTIGAL, L. REICHEL, L. N. TREFETHEN, *A Hybrid GMRES Algorithm for nonsymetric linear systems* , SIAM J. Matrix Anal. Appl., 13 (1992) 796–825.

[53] S. PETITON, *Parallel QR Algorithm for Iterative Subspace Methods on the Connection Machine (CM2)*, In J. Dongarra, P. Messina, D. Sorensen, and R. Voigt, editors, *Parallel Processing for Scientific Computing*, SIAM, (1990).

[54] S. PETITON, *Data parallel sparse matrix computation on CM-2 and CM-5 for iterative methods*, University of Minnesota, AHPCRC (1993).

[55] S. PETITON, *Parallel subspace method for non-Hermitian eigenproblems on the Connection Machine (CM2)*, Appl. Numer. Math., 10 (1992) 19–35.

[56] B. PHILIPPE, R. B. SIDJE, *Parallel algorithm for Arnoldi procedure, in Proc. of the Second International Symposium on Iterative Methods in Linear Algebra*, IMACS, Sofia, 1995.

[57] L. REICHEL, *Newton interpolation at Leja points*, BIT, 30 (1990) 332–346.

[58] Y. SAAD, *Numerical methods for large eigenvalue problems*, Manchester University Press, Manchester (1992).

[59] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, New York, 1996.

[60] Y. SAAD, *Variation on Arnoldi's method for computing eigenelements of large unsymmetric matrices*, Linear Algebra Appl., 34 (1980) 269–295.

[61] Y. SAAD, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comput., 37 (1981) 105–126.

[62] Y. SAAD, *On the condition number of modified moment matrices arising in least squares approximation in the complex plane*, Numer. Math., 48 (1986) 337–347.

[63] Y. SAAD, *Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1987) 155–169.

[64] Y. SAAD, *SPARSKIT: a basic tool kit for sparse matrix computations*, Technical Report 90-20, RIACS, NASA Ames Research Center (1990).

[65] Y. SAAD, *A flexible inner-outer preconditionned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993) 461–469.

[66] Y. SAAD, M. H. SCHULTZ, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986) 856–869.

[67] H. SADOK, *Analysis of the convergence of the minimal and the orthogonal residual methods*, submitted.

[68] V. SIMONCINI, E. GALLOPOULOS *Convergence Properties of Block GMRES and Matrix Polynomials*, Lin. Alg. Appl., 247 (1996) 97-119.

[69] D. C. SMOLARSKI, P. E. SAYLOR, *Optimum parameters for the solution of linear equations by the Richardson's iteration*, (1982), non publié.

[70] D. C. SMOLARSKI, P. E. SAYLOR, *An optimum iterative method for solving any linear system with a square matrix*, BIT, 28 (1988) 163–178.

[71] R. B. SIDJE, B. PHILIPPE , *Parallel Krylov subspace basis computation*, Actes du 2ème Colloque Africain sur la Recherche en Informatique CARI'94, (1994) 421–440.

[72] THINKING MACHINES CORPORATION, *CM-5*, 1992.

[73] H. A. VAN DER VORST, C. VWIK, *The superlinear convergence of GMRES*, J. Comput. Appl. Math., 48 (1993) 327–341.

[74] R. S. VARGA, *Matrix iterative analysis*, Prentice Hall, 1962.

[75] B. VITAL, *Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur*, Thèse, Université de Rennes, 1990.

[76] D. M. YOUNG, *Iterative solution of large linear systems*, Academic Press, 1971.

[77] L. ZHOU, H. F. WALKER, *Residual smoothing techniques for iterative methods*, SIAM J. Sci. Statist. Comput., 15 (1994) 297–312.