Université Paris IX Dauphine

Thèse

présentée par **M. Vincent MARTIN**
pour obtenir le grade de docteur de l'Université Paris IX Dauphine
spécialité : mathématiques appliquées

Simulations multidomaines des écoulements en milieu poreux

Soutenue le 18 mars 2004 devant la Commission d'examen composée de :

Chavent	Guy	Président
Loth	Laurent	Examinateur
Pironneau	Olivier	Rapporteur
Roberts	Jean E.	Directrice de thèse
Russell	Thomas F.	Rapporteur
Weis	Pierre	Examinateur

Thèse préparée à l'INRIA (Projet Estime, Centre de Rocquencourt)

Caligula : C'était difficile à trouver.
Hélicon : Quoi donc ?
Caligula : Ce que je voulais.
Hélicon : Et que voulais-tu ?
Caligula, *toujours naturel* : La lune.
[...]
Caligula : Tu penses que je suis fou.
Hélicon : Tu sais bien que je ne pense jamais, je suis bien trop intelligent pour ça.
Caligula : Oui. Enfin ! Mais je ne suis pas fou et même je n'ai jamais été aussi raisonnable. Simplement, je me suis senti tout d'un coup un besoin d'impossible. *(Un temps)* Les choses telles qu'elles sont ne me semblent pas satisfaisantes.

[24, Acte I, Scène 4]

Pour Anne

Remerciements :
Je serai bref car la liste est longue et je ne voudrais oublier personne. Il ne faudrait pas en déduire pour autant que ma gratitude serait partielle ou limitée : toutes les personnes mentionnées ici m'ont véritablement aidé durant ma thèse et je leur en suis profondément reconnaissant. Elles mériteraient sans doute bien plus que ces quelques mots...

Pour commencer, je souhaiterais remercier Olivier Pironneau et Thomas Russell qui m'ont fait l'honneur d'accepter d'être les rapporteurs de ma thèse et de participer au jury.

Un grand merci également à Guy Chavent d'avoir présidé avec briole jury.

Je remercie bien évidemment Laurent Loth d'avoir suivi mes recherches depuis l'Andra. Cela m'amène tout naturellement à exprimer ma reconnaissance envers l'Andra pour le financement des trois années de thèse.

Merci à Pierre Weis qui par sa participation au jury a apporté une touche informatique qui contribuera peut-être à faire communiquer plus souvent les mondes numérique et informatique.

Ma plus profonde gratitude, accompagnée de l'expression de mon respect, va à Jean Roberts et à Jérôme Jaffré. Sans chercher à parler de leurs qualités scientifiques, je retiens qu'ils m'ont toujours soutenu, encouragé et guidé. Sans eux, je n'aurais sans doute pas surmonté cette épreuve que constitue la deuxième année de thèse... J'ai aussi particulièrement apprécié les moyens fournis aux doctorants du projet Estime dont j'ai très largement bénéficié.

Merci à François Clément, Arnaud Vodicka, Pierre Weis et Roberto Di Cosmo, avec qui j'ai travaillé au couplage de codes. Si notre collaboration interdisciplinaire s'est si bien passée, c'est sans doute grâce à l'humeur joyeuse qui régnait pendant nos réunions et à l'humour de tous les participants. Merci également à Amel Sboui pour le découpage de maillages, et bon courage pour la suite ! Un grand merci à Michel Kern pour ses blagues (mais aussi pour tout le temps qu'il a consacré à m'aider à clarifier ou résoudre mes problèmes mathématiques et informatiques). Une petite pensée amicale à Roland Morel, qui a survécu à ces quelques mois de stages passés avec moi... Dans la foulée, je remercie l'ensemble des personnes des projets Estime et Ondes pour leurs coups de main quand j'en avais besoin et leur gentillesse en général. Travailler dans cette ambiance est une chance certaine. Je voudrais dire un mot de remerciement en particulier aux secrétaires dont on oublie trop souvent les services rendus : Muriel De Bianchi, Héléna Kutniak, Sandrine Boute.

Merci à Jean-Frédéric Gerbeau et à Mohamed Belhadj pour la collaboration sur le code LifeV. C'était un réel plaisir de travailler ensemble. Merci à Marina Vidrascu pour ses moulinettes qui m'ont été bien utiles. Merci aux gens du bâtiment 16 que j'ai dérangés de temps à autres avec mes questions et qui m'ont toujours bien reçu.

Merci à Frédéric Nataf, Yves Achdou, Stéphane Del Pino et Sébastien Wagner avec qui j'ai travaillé lors du Cemracs 2001. Cette collaboration a marqué un tournant important dans ma thèse.

Un petit mot chaleureux et amical à Sandrine Fauqueux, Jacopo Grazzini, Alexis Paljic, Isabelle Champagne, Jeronimo Rodriguez, Julien Diaz, Xavier Jonsson, et tous les autres, qui ont égayé toutes ces années à l'Inria et qui me manquent déjà.

Pour finir, j'ai une pensée aux amis et à la famille pour leur soutien et leur patience : cette épreuve aura été autant la leur que la mienne.
Merci à mon père et à ma mère.
Merci à Anne.

Table des matières

Chapitre 1

Introduction

1.1 Contexte général

Ce mémoire est consacré aux simulations multidomaines de l'écoulement dans les milieux poreux. Les questions d'écoulement et de transport en milieu poreux, comme la plupart des problèmes environnementaux, ont acquis ces dernières années dans nos sociétés une importance qui paraît méritée. Les enjeux principaux liés à ces questions résident principalement dans la gestion des ressource en eau et la maîtrise des pollutions souterraines. Ce dernier thème préoccupe actuellement la plupart des pays industrialisés et motive de nombreuses recherches à travers le monde. La raison de cet engouement croissant est la question de la gestion des déchets nucléaires, résidus des activités civiles et militaires de ces vingt dernières années. Ces déchets s'accumulant et ayant une durée de nocivité dépassant largement l'échelle humaine, il convient de leur trouver une solution durable. Trois pistes sont généralement proposées : la transmutation des déchets radioactifs en isotopes moins dangereux ou dont la durée de vie est plus courte ; leur entreposage réversible et surveillé sous terre, mais à faible profondeur ; ou enfin leur stockage en couche géologique profonde. Ces trois voies sont actuellement explorées en France par l'Andra[1] —Agence Nationale pour la gestion des Déchets Radioactifs—, qui doit remettre au Parlement français les conclusions de ses études en 2006, afin d'éclairer le débat national sur le sujet.

Dans le cadre du stockage profond, la réalisation de simulations numériques fiables est indispensable, puisque c'est le seul moyen de prédire le comportement des déchets en sous-sol sur des durées de plusieurs milliers d'années. Il est notamment nécessaire d'effectuer des simulations du transport de radionucléides

[1] Voir l'URL `http ://www.andra.fr`.

en milieu souterrain afin de prévoir si des radioéléments dangereux sont susceptibles de ressortir dans la biosphère, et, si cela arrive, dans quelles proportions et dans quelles circonstances. Or, il s'avère que ces simulations sont entravées par de nombreuses difficultés d'ordre purement numérique dont nous dirons un mot à la Section 1.2. Dans ces conditions, il apparaît donc logique de faire appel aux mathématiques appliquées pour tenter de surmonter ces obstacles. L'Andra joue depuis quelques années un rôle moteur pour orienter la recherche numérique en France vers la problématique du stockage de déchets. Ma thèse, qui a été financée par l'Andra, s'inscrit modestement dans cette optique.

Nous donnons quelques idées du contexte scientifique dans la Section 1.2. Les sujets abordés dans cette étude sont présentés à la Section 1.3. La méthodologie générale est expliquée dans la Section 1.4. Enfin, nous décrivons brièvement le contenu de ce mémoire dans la Section 1.5.

Dans tout ce chapitre introductif, nous évoquons les équations de transport, alors que nous nous sommes limités dans notre travail à l'étude de l'écoulement. En fait, ce dernier sert de base pour les problèmes de transport. Il semble donc important de survoler certaines difficultés liées au transport afin de posséder une idée globale des problèmes de l'écoulement.

1.2 Contexte scientifique : problématiques liées au stockage de déchets radioactifs souterrains

La simulation numérique à grande échelle de l'écoulement et du transport de radionucléides dans le sous-sol autour d'un site de stockage de déchets pose de nombreux problèmes de différentes natures, rassemblant les énergies de différentes communautés scientifiques. On regroupe ces problèmes en trois grandes catégories.

La modélisation mathématique : au-delà des modèles les plus simples couramment utilisés, la prise en compte par exemple des phénomènes d'adsorption ou du comportement spécifique des argiles ou des argilites nécessite une interaction entre mathématiciens et physiciens pour construire des modèles représentant les interactions entre les échelles macroscopiques et microscopiques.

La collecte des données : la conception d'expériences dans un laboratoire souterrain, l'analyse de carottes issues de différents forages, les enregistrements piézométriques et les essais de puits mobilisent de nombreux expérimentateurs. De plus, le sous-sol étant difficile d'accès, ces données sont nécessairement ponctuelles et un gros travail d'interprétation est nécessaire pour en déduire les données distribuées nécessaires aux modèles. Ce travail implique la résolution de problèmes inverses.

Les méthodes numériques : les modèles étant choisis et les données rassemblées, il est nécessaire de repousser les limitations liées à la puissance des ordinateurs en utilisant les méthodes numériques les plus performantes. La modélisation à grande échelle, en trois dimensions, sur une longue période de temps, dans un milieu hétérogène, avec des modèles qui peuvent être très complexes, dépasse les capacités actuelles des ordinateurs et nécessite une réflexion sur le choix et la conception de méthodes numériques appropriées.

Cette présentation nécessairement simplifiée, devrait aussi faire mention des problèmes concernant les incertitudes sur les données et leur impact sur les résultats, ainsi que l'analyse de la sensibilité des résultats par rapport à des paramètres choisis.

Pour ce qui est des méthodes numériques, les problèmes de couplage sont le plus souvent à l'origine de beaucoup de difficultés. On peut en effet distinguer deux grandes familles de problèmes de couplage :

- le couplage de différents processus qui évoluent dans la même région de l'espace. Ainsi le couplage thermo-hydro-mécanique ou le couplage chimie-transport.
- le couplage de processus qui évoluent dans des régions différentes de l'espace. On peut donner comme exemples le couplage entre un aquifère où le transport est dominé par l'advection et une région où le processus de diffusion est dominant, ou bien le couplage entre un centre de stockage et le milieu géologique, ou encore le couplage entre des roches et les fractures qui les traversent.

Ces deux types de couplage présentent des difficultés numériques que les logiciels actuels prennent plus ou moins bien en compte. En général, la solution utilisée consiste à faire des simulations avec un seul modèle incluant tous les aspects physiques, ce qui rend les calculs coûteux et mène rapidement aux limites des possibilités actuelles des calculateurs, notamment pour les simulations en trois dimensions.

Nous nous intéressons principalement dans cette thèse au deuxième type de couplage. Nous nous sommes plus particulièrement occupés des problèmes liés à la diversité des échelles spatiales[2], et à l'hétérogénéité des milieux[3]. Nous avons également étudié les milieux poreux fracturés qui réunissent ces deux types de problèmes. Nous n'avons pas négligé les questions de performances et avons utilisé le parallélisme pour opérer certains couplages.

1.3 Sujets abordés

Nous avons mené des études principalement dans trois directions qui ont toutes en commun les simulations multidomaines et bien évidemment les écoulements en milieu poreux, comme le titre de la thèse l'indique.

Le raffinement local : considérons le calcul de l'écoulement et du transport des radionucléides en s'intéressant particulièrement au couplage entre le site de stockage et le milieu géologique environnant. Comme la source de radionucléides est très ponctuelle, il est nécessaire de raffiner en espace dans son voisinage. Si l'on utilise des maillages *structurés* —en rectangles en 2 dimensions—, le raffinement local classique en espace conduit à un raffinement même dans des zones qui sont éloignées de la zone d'intérêt. Si l'on utilise des maillages *non structurés*, le raffinement local est possible mais il ne permet plus d'utiliser les méthodes numériques particulièrement efficaces développées pour les maillages structurés.

Certaines méthodes de décomposition de domaines permettent de remédier à ces difficultés : il est possible de raffiner localement un sous-domaine du domaine de simulation sans avoir à raccorder le maillage local avec le maillage plus grossier utilisé dans l'ensemble du domaine. Nous avons donc étudié et utilisé une telle méthode.

Le couplage parallèle : pour pouvoir traiter des problèmes réalistes de grande taille en trois dimensions, nous avons cherché à coupler les sous-domaines en implémentant une version parallèle de la méthode de décomposition de domaines.

La modélisation des fractures : les fractures qui traversent fréquemment les milieux poreux engendrent dans les simulations de l'écoulement des difficultés numériques considérables, dues à une très forte hétérogénéité très localisée. Sur la faible épaisseur de la fracture, la perméabilité peut

[2]La géométrie d'un site de stockage a des distances caractéristiques de l'ordre du mètre, tandis que le domaine d'étude du milieu géologique doit s'étendre sur plusieurs dizaines de kilomètres

[3]Les coefficients de perméabilité peuvent varier sur 7 ordres de grandeurs entre deux couches voisines

en effet varier de plusieurs ordres de grandeurs, devenant un canal privilégié pour l'écoulement, ou au contraire une barrière géologique. Nous nous sommes donc attachés à modéliser les fractures ayant des perméabilités très fortes ou très faibles, quand elles sont d'une taille telle qu'il faut les considérer explicitement.

Le but est de les traiter comme des interfaces entre des sous-domaines qui représentent la roche avoisinante. Ici, la simulation multidomaine apparaît naturellement pour formaliser ce problème, et une technique de décomposition de domaines permet d'aider à sa résolution.

1.4 Méthodologies

1.4.1 Modèle mathématique

Le modèle mathématique que nous considérons est un modèle très simple : l'*équation de l'écoulement*, parfois appelée *équation du flux* :

$$\operatorname{div}\left(-\frac{k}{\mu}\operatorname{grad}(p+\rho g z)\right) = q \quad \text{dans } \Omega, \tag{1.1}$$

où p est la pression, k la perméabilité du milieu, ρ la masse volumique du fluide et μ sa viscosité, g l'accélération de la pesanteur, z la hauteur, et q un terme source L'équation de l'écoulement est une équation elliptique qui est obtenue à partir de la loi de la conservation de masse du fluide, avec l'hypothèse d'incompressibilité,

$$\operatorname{div}\mathbf{u} = q \quad \text{dans } \Omega, \tag{1.2}$$

où $\mathbf{u}$ est la vitesse de Darcy donnée par la *loi de Darcy* :

$$\mathbf{u} = -\frac{k}{\mu}\operatorname{grad}(p+\rho g z) \quad \text{dans } \Omega. \tag{1.3}$$

Nous mentionnons également pour que l'exposé soit tout à fait clair l'*équation du transport* que nous ne chercherons pas à résoudre ici. Cette équation gouverne le déplacement d'un contaminant et dépend de la précédente :

$$\phi R\frac{\partial c}{\partial t} + \operatorname{div}(\mathbf{u}c - D\operatorname{grad} c) + \lambda\phi R c = q\widehat{c} \quad \text{dans } \Omega, \tag{1.4}$$

où c est la concentration d'un élément radioactif dissous dans l'eau —par simplicité, nous supposons qu'il n'intervient qu'un seul élément radioactif dans le modèle—, ϕ est la porosité cinématique du milieu, R le coefficient de retard, $\mathbf{u}$ la vitesse de Darcy, t le temps, D le terme de diffusion, λ la constante de décroissance radioactive et $\widehat{c}$ la concentration du radionucléide dans le terme source.

Comme l'équation de l'écoulement est indépendante du temps et de la concentration, on peut la traiter d'abord et résoudre ensuite de façon séparée l'équation du transport. Pour certains contaminants, il est possible que la viscosité dépende de la concentration, mais on suppose généralement que les radionucléides sont présents en quantités suffisamment petites pour avoir une influence négligeable sur la viscosité μ.

L'équation de transport exprime la conservation de masse de l'élément radioactif. Il contient un terme d'accumulation, noté $\phi R\frac{\partial c}{\partial t}$, un terme de diffusion–dispersion, $\mathrm{div}\,(-D \text{ grad } c)$, un terme d'advection, $\mathrm{div}\,(\mathbf{u}c)$, et un terme de décroissance radioactive, $\lambda\phi Rc$. Précisons que le coefficient D dans le modèle le plus simple est une matrice diagonale et seule la diffusion moléculaire est prise en compte. Dans un modèle un peu plus sophistiqué, D est une matrice comportant aussi une partie dispersive non-diagonale dépendant de la vitesse de Darcy.

1.4.2 Discrétisation de l'équation du flux

Pour la discrétisation de l'équation du flux nous utilisons la méthode d'éléments finis mixtes hybrides. La méthode d'*éléments finis mixtes*, cf. [23], [29] et [83], est une méthode de Galerkin basée sur la formulation mixte faible de l'équation. L'équation du flux est donnée naturellement en formulation mixte : elle couple l'équation de la conservation de masse avec la loi de Darcy

$$\begin{aligned} \mathrm{div}\,\mathbf{u} &= q \quad \text{dans } \Omega, && (1.5)\\ \mathbf{u} &= -\frac{k}{\mu}\,\mathrm{grad}\,(p+\rho g z) \quad \text{dans } \Omega. && (1.6) \end{aligned}$$

Nous négligerons dans le reste de notre travail le terme de gravité, car sa présence ne joue pas un rôle important dans les méthodes étudiées.

La formulation mixte est particulièrement appropriée pour ce modèle, car elle donne un approximation directe de la vitesse de Darcy. En effet, notre intérêt n'est pas dans la pression p mais dans la concentration c, et dans l'équation du transport qui donne la concentration, c'est la vitesse de Darcy $\mathbf{u}$ qui intervient et non pas la pression. La méthode de différences finis et les méthodes classiques d'éléments finis donnent, en premier lieu, une approximation de la pression qui est dans un second temps différenciée pour donner l'approximation de la vitesse de Darcy qui est utilisée dans l'équation du transport. Cette différenciation est une seconde source d'erreur et la précision est réduite. De plus la pression obtenue est constante ou affine par morceaux et la vitesse qui en résulte en prenant la dérivée de la pression donne un flux à valeurs multivoques sur les interfaces du maillage, ce qui rend le schéma non-conservatif sur les mailles de discrétisation. Pour le schéma que nous utilisons, la pression est approchée dans un espace de fonctions constantes par morceau et la vitesse de Darcy $\mathbf{u}$ dans un espace de fonctions dont chaque composante est affine par morceaux et dont les composantes normales aux interfaces entre les éléments sont continues à travers les interfaces.

La méthode d'éléments finis mixtes-hybrides, conçue au départ comme une astuce pour résoudre le système linéaire associé aux méthodes mixtes, cf. [43] et [16], produit non seulement la même approximation de la pression et la même approximation de la vitesse mais aussi une approximation de la trace de pression sur les interfaces du maillage. Cette méthode se prête alors très bien aux méthodes de décomposition de domaines que nous utilisons, car nous avons à notre disposition la trace de la pression aussi bien que la trace normale de la vitesse sur les interfaces entre les sous-domaines.

Remarquons qu'avec l'emploi d'une règle de quadrature dans la construction de la matrice de masse, avec une discrétisation rectangulaire parallèle aux axes, on obtient un schéma équivalent au schéma de volumes finis centré sur les mailles standard, voir [28], [7]. Mentionnons également que les volumes finis ont connu ces dernières années de nombreux développements, voir par exemple [49].

1.4.3 Discrétisation de l'équation du transport

Disons un court mot sur l'équation du transport pour donner une idée de sa résolution. Elle peut s'écrire sous la forme

$$\begin{aligned} \phi R\frac{\partial c}{\partial t} + \text{div }(\mathbf{f}+\mathbf{r}) + \lambda\phi R c &= q\widehat{c} \quad \text{dans } \Omega, && (1.7)\\ \mathbf{r} &= -D \text{ grad } c \quad \text{dans } \Omega, && (1.8)\\ \mathbf{f} &= \mathbf{u}c \quad \text{dans } \Omega, && (1.9)\end{aligned}$$

où nous avons séparé le flux de concentration en sa partie diffusive $\mathbf{r}$ et sa partie convective $\mathbf{f}$. Cette séparation est utile pour les problèmes où la convection est importante.

Dans ce cas, cette séparation permet d'utiliser une méthode d'éléments finis mixtes-hybrides pour $\mathbf{r}$ alors que $\mathbf{f}$ sera obtenue par une méthode de type Godunov, voir [27], [90] et [32]. L'intérêt d'une telle méthode réside encore dans ses propriétés de conservation et dans sa facilité d'emploi dans une méthode de décomposition de domaines. En général, on utilise un schéma implicite pour la partie diffusive et le terme de décroissance radioactive, et un schéma explicite pour la partie convective, avec éventuellement des pas de temps différents pour ses deux parties (plus petits pour la partie convective).

Dans le cas où la convection est faible, alors le vecteur $\mathbf{f} + \mathbf{r}$ est approché dans son ensemble par la méthode d'éléments finis mixtes-hybrides pour donner une méthode de discrétisation centrée.

1.4.4 Méthode de décomposition de domaines

L'existence des ordinateurs modernes multi-processeurs ou des grappes de PC rend envisageable la modélisation numérique fine de problèmes physiques pratiques qui jusqu'à très récemment ont été traités par des méthodes assez grossières. Mais pour les problèmes où le domaine de calcul et/ou le temps de calcul est grand, spécialement pour les modèles en trois dimensions d'espace, les systèmes algébriques à résoudre sont d'une taille telle que les algorithmes numériques utilisés pour leur résolution doivent être conçus avec beaucoup de soin. Les méthodes les plus rapides pour la résolutions des systèmes linéaires creux —transformation de Fourier rapide, réduction cyclique par bloc, les méthodes multigrilles – sont efficaces pour les problèmes très réguliers. La décomposition de domaines est une méthode de calcul scientifique basée sur l'idée qu'on peut obtenir la solution d'un problème global en assemblant la solution de problèmes plus petits et plus régulier, cf. [91] ou [81].

Même si on place souvent l'origine des méthodes de décomposition de domaine au travail de H. A. Schwartz [89] en 1870, c'est un outil pour la résolution numérique des équations aux dérivées partielles relativement récent, les idées les plus importantes étant apparues dans les vingt dernières années. Aujourd'hui, c'est un domaine de recherche très fructueux ; depuis 1987 il y a une conférence annuelle internationale qui attire plusieurs centaines des meilleurs numériciens mondiaux.

Parmi les numériciens, on trouve au moins trois conceptions différentes de la décomposition de domaines : dans certains cercles, la décomposition de domaines veut dire la séparation du domaine physique en sous-domaines où les équations gouvernant les phénomènes physiques ne sont pas les mêmes et/ou en sous-domaines où il y a besoin de différents traitements numériques à cause, par exemple, de différence d'échelles. Pour les spécialistes dans la résolution de grands systèmes linéaires elle signifie une façon de construire un solveur en divisant le problème correspondant au système d'équations sur tout le domaine en des problèmes plus petits. Pour les spécialistes en calcul parallèle enfin la décomposition de domaine correspond à une façon de distribuer les données du problème sur les différents processeurs. Dans cette thèse, la technique de décomposition de domaines est exploitée sur les deux derniers plans, et un travail a été mis en place pour l'utiliser suivant le premier également.

La première raison pour laquelle nous avons décidé d'utiliser la décomposition de domaine tient à ses propriétés de flexibilité. Dans un ou plusieurs des sous-domaines autour du site de stockage, il est possible d'utiliser un maillage plus fin. Il y a, bien sûr, d'autres moyens de raffiner localement le maillage, en particulier l'utilisation de maillages non-structurés. Nous avons préféré choisir la décomposition de domaine pour son efficacité numérique. En particulier, on peut ainsi conserver des maillages rectangulaires, ce qui présente de nombreux avantages sur le plan numérique, spécialement pour les problèmes de grande taille obtenus avec la modélisation en trois dimensions.

Le principe de la décomposition de domaines est relativement simple. Le domaine de calcul est divisé en sous-domaines, en choisissant une initialisation qui assigne une condition sur les interfaces[4] ; les problèmes dans les sous-domaines sont alors résolus séparément. Aux itérations suivantes, les problèmes sont de nouveau résolus séparément dans les sous-domaines mais avec les conditions aux interfaces sur les sous-domaines déterminées à partir des résultats du calcul des itérations précédentes. Maintes variations sur ce principe existent : décomposition en sous-domaines se recouvrant ou ne se recouvrant pas, conditions de Dirichlet, de Neumann ou de Robin sur les bords intérieurs, sans ou avec relaxation[5], solutions dans tous les sous-domaines en parallèle ou solutions séquentielles, inclusion d'une solution sur une grille grossière ou non. Tous ces choix ont une influence sur la rapidité de la convergence du schéma et même sur la convergence ou non de la méthode.

Dans notre problème la décomposition en sous-domaines *ne se recouvrant pas* permet le plus de flexibilité. Dans ce cas, le problème algébrique est ramené, grâce aux opérateurs de type Poincaré-Steklov, à un problème sur les inconnues des interfaces entre sous-domaines, [57], [67], [9].

Des techniques comme la méthode des éléments joints permettent de traiter même des situations où les inconnues sur une interface provenant des deux sous-domaines adjacents ne coïncident pas nécessairement [21], et [13]. Nous avons étudié et utilisé une méthode un peu différente, décrite dans [2], [14], qui permet d'avoir des maillages qui ne se raccordent aux interface et qui est basée

[4]Les interfaces sont définies comme les parties du bord de chaque sous-domaine qui ne coïncident pas avec le bord du domaine original

[5]C'est-à-dire qu'on prend pour condition au bord celle donnée par l'itération précédente dans le sous-domaine voisin, ou on prend une moyenne pondérée de cette valeur avec une autre valeur telle que celle prise à l'itération précédente

sur des conditions d'interface de type Robin. De tels maillages non-raccordés peuvent provenir du fait qu'on a maillé séparément les différents sous-domaines. Ils peuvent aussi être recherchés pour effectuer un raffinement local dans l'un des sous-domaines. Mais pour que la méthode soit efficace, le défi est de choisir un préconditionneur pour le système qui rende rapide la convergence du système itératif [4], [36].

1.4.5 Calcul parallèle avec OcamlP3l

Les méthodes de décomposition de domaines sont en général intrinsèquement parallèles, nous avons donc cherché à profiter de cette qualité et à les implémenter de façon parallèle. Pour cela, nous avons travaillé en collaboration avec une équipe de chercheurs en informatique qui développent le système parallèle OcamlP3l qui, à notre avis, possède un avenir prometteur.

L'écriture de programmes parallèles n'est en effet pas chose aisée, et leur débogage est généralement un cauchemar. Afin de faire face à ces difficultés, une approche structurée de la programmation parallèle utilisant des squelettes*[6] —*skeletons* en anglais— et des techniques de compilation basée sur des modèles —*templates*— a été développée ces dernières années par plusieurs chercheurs dont le groupe P3L à Pise. Le système OcamlP3l, [39][7] marie le langage de programmation fonctionnelle Ocaml, [97][8] avec les squelettes p3l. Ceci a engendré un système de programmation parallèle performant et une méthodologie puissante : OcamlP3l permet à l'utilisateur d'écrire et de déboguer une version *séquentielle* de son programme —tâche qui, si elle n'est pas aisée en soi, peut être considérée comme routinière—, et ensuite la version parallèle est *automatiquement déduite* par une recompilation du programme source. Ainsi, l'avantage considérable de cette approche est-il que le programmeur ne doit se concentrer que sur la partie facile, la programmation séquentielle, et se repose sur le système OcamlP3l pour obtenir la partie complexe, la version parallèle. Un autre bénéfice accompagne le précédent : l'adéquation des sémantiques* des versions séquentielles et parallèles du programme n'est plus du ressort du programmeur ; c'est maintenant l'entière responsabilité du compilateur OcamlP3l.

[6] Les termes suivis d'un astérisque sont définis dans le lexique en Annexe C.1.

[7] Voir l'URL `http ://www.ocamlp3l.org/`.

[8] On prononce Ocaml avec le "O" de *O rage ! ô désespoir ! ô vieillesse ennemie !* [34, Acte I, scène IV], et "*avec le "ca" de* café *et le "mel" de* melba" [97], page xi. Voir les URL `http ://pauillac.inria.fr/ocaml/` ou `http ://www.ocaml.org/`.

1.5 Plan du mémoire

Le mémoire est divisé en trois chapitres principaux, au début desquels se trouve systématiquement leur contenu détaillé.

Dans le Chapitre 2, nous étudions les méthodes de décomposition de domaines non-conformes. Nous commençons par un bref survol bibliographique de ces méthodes, puis nous analysons en détail une méthode particulière, pour laquelle nous démontrons un résultat d'existence et d'unicité ainsi que des estimations d'erreur. Nous expliquons un moyen de transférer les variables à travers une interface quand les maillages ne se raccordent pas. Enfin, des tests numériques 2D illustrent le bon fonctionnement du raffinement local avec cette méthode.

Dans le Chapitre 3, nous parlons de l'implémentation de cette méthode en parallèle avec OcamlP3l. Nous commençons par donner quelques notions du langage Ocaml, qui nous servent ensuite à décrire complètement le système OcamlP3l ainsi que le code de couplage de décomposition de domaines. Deux simulations numériques suivent, la première montrant les performances du système, la deuxième traitant un écoulement réaliste issu de l'Andra.

Enfin, le Chapitre 4 est consacré au nouveau modèle de fractures. Nous décrivons comment il est obtenu, puis démontrons l'existence et l'unicité du problème continu. Une interprétation du problème discret nous fournit des estimations d'erreur, que viennent conforter des résultats numériques. Nous disons enfin un mot sur la résolution efficace du modèle.

Une conclusion propose quelques pistes pour poursuivre ce travail dans le Chapitre 5. Les Annexes A, B et C sont dédiées respectivement à des rappels sur les coordonnées barycentriques, sur le problème elliptique en une dimension, et à Ocaml.

Chapitre 2

Décompositions de domaines non-conformes

Dans ce chapitre, nous étudions une méthode de décomposition de domaines sans chevauchement, utilisant des maillages non-raccordés, et basée sur des conditions d'interface de type Robin, [14] ou [2].

Avant de décrire cette méthode, nous proposons dans la Section 2.1 quelques pistes bibliographiques concernant la décomposition de domaines.

Ensuite, dans la Section 2.2 consacrée aux problèmes multiblocs, nous posons le problème modèle, détaillons le découpage en sous-domaines et définissons les notations utilisées pour les maillages d'interface.

La Section 2.3 est entièrement dédiée à la présentation et à une analyse de la méthode de décomposition de domaines de Robin non-conforme. Résumons en quelques lignes son contenu. L'algorithme y est présenté sous deux formes différentes afin de faciliter sa mise en œuvre. En suivant la méthodologie de [14] et en se basant sur des estimations d'énergie, nous démontrons l'existence et l'unicité de la solution mixte discrète du problème de décomposition de domaines ; nous obtenons également des estimations d'erreur. L'originalité de ces résultats provient de la possibilité laissée à l'utilisateur de prendre des coefficients de Robin différents pour chaque interface, mais également différents sur les deux côtés d'une même interface. Quelques remarques sont faites également à propos du choix de "bons" coefficients de Robin.

La Section 2.4 concerne un sujet important qui, à notre connaissance, est rarement abordé dans la littérature sur les méthodes de décomposition de domaines non-conformes : la question de la transmission des variables d'un maillage d'interface à un autre. Dans le cadre des éléments finis mixtes de plus bas degré, cette communication s'interprète comme une simple projection L^2. Elle nécessite de calculer l'intersection des deux maillages qui vivent sur l'interface considérée, et en particulier de calculer les mesures des cellules de ce maillage d'intersection. Dans le cadre des maillages d'interface 2D plans, constitués de triangles, nous proposons un algorithme pour déterminer ces aires d'intersection. Cet algorithme a été mis au point lors de la thèse [69] et a été utilisé dans les simulations du Chapitre 3.

Des tests numériques simples en deux dimensions sont présentés dans la Section 2.5 ; ils illustrent le comportement de l'algorithme de décomposition de domaines dans différentes configurations, et indiquent notamment qu'il est préférable, quand c'est possible, de travailler avec des maillages d'interface emboîtés[1].

Enfin, la Section 2.6 sert de courte transition vers le chapitre suivant.

2.1 Pistes bibliographiques sur la décomposition de domaines

Les méthodes de décomposition de domaines ont maintenant une vingtaine d'années d'existence active et même intense, concrétisée par la tenue annuelle d'une conférence internationale qui leur est consacrée[2]. Bien évidemment, il serait illusoire de tenter de dresser une liste exhaustive de toutes les recherches pertinentes qu'un sujet aussi riche a pu susciter. Nous allons donc nous limiter à l'explication des méthodes de décomposition de domaines sans recouvrement, et notamment à trois méthodes : Neumann–Neumann et ses avatars, les méthodes de recollement par joints et la méthode de Robin non-conforme qui nous préoccupe plus particulièrement dans ce mémoire.

Tout d'abord, citons quelques ouvrages qui offrent une vision large des méthodes de décomposition de domaines : le livre de Smith, Bjørstad et Gropp [91] a une optique plutôt algébrique. Le livre très complet de Quarteroni et Valli [81] passe en revue de façon détaillée les différentes méthodes, notamment des algorithmes de Dirichlet–Neumann et de Neumann–Neumann. La monographie de Le Tallec [67] donne une analyse de la méthode de Scharz avec chevauchement et du préconditionneur de Neumann–Neumann ; les algorithmes y sont très clairement détaillés. Mentionnons enfin le livre de Wohlmuth [98] qui traite en particulier des méthodes de décomposition de domaines avec des maillages non-raccordés.

[1] Je qualifie deux maillages d'interface d'*emboîtés*, quand l'un des maillages est sous-maillage de l'autre

[2] Voir l'URL http ://www.ddm.org/.

2.1.1 Neumann–Neumann

Nous nous intéressons d'abord à la méthode de décomposition de domaines sans recouvrement dite de Neumann–Neumann. Un bon moyen d'expliquer son fonctionnement est de suivre les évolutions qu'elle a subies au cours du temps.

Dans le cadre des éléments finis mixtes, Wheeler avec Glowinski [57], puis avec Cowsar [37], ont commencé à mettre en place une méthode itérative de sous-structuration, basée sur la résolution d'un problème de type Steklov–Poincaré[3]. Le principe général est de faire apparaître un système linéaire symétrique défini positif dont les inconnues vivent sur l'interface. Notons pour fixer les idées ce système

$$S\lambda = f, \tag{2.1}$$

où λ est une inconnue correspondant à une trace de pression sur l'interface, f une donnée correspondant à une vitesse normale sur l'interface et S est l'opérateur discret de Dirichlet vers Neumann, encore appelé Steklov–Poincaré, et qui s'interprète comme un complément de Schur. On montre que S est symétrique défini positif pour un problème elliptique symétrique. La résolution de ce système se fait itérativement par une méthode de type gradient conjugué préconditionné en ne formant pas la matrice explicitement. A chaque itération, on doit résoudre un problème par sous-domaine avec des conditions sur l'interface de type Dirichlet. Dans ces conditions, la matrice du problème de Dirichlet est généralement factorisée une fois pour toutes dans une phase d'initialisation, de manière que les inversions qui ont lieu à chaque itération se fassent rapidement et précisément. Dans une décomposition en n sous-domaines, on montre, [67], que le conditionnement de S vérifie la majoration

$$\mathrm{cond} S \leq \frac{C}{H^2}\left(1 + \max \frac{H_i}{h_i}\right),$$

où h_i est le pas du maillage dans le sous-domaine d'indice $i, i = 1, 2, \ldots, n$ et H_i est le diamètre du sous-domaine i et H le diamètre maximal des sous-domaines et C est une constante par rapport à h_i et H_i. Ce conditionnement augmente fortement avec l'augmentation du nombre de sous-domaines, qui est proportionnel à $\frac{1}{H}$, et dépend du raffinement du maillage. Par ailleurs, la constante dépend également des sauts de perméabilités dans les sous-domaines.

L'étape suivante est de préconditionner ce système linéaire sur l'interface. L'idée est alors de prendre comme préconditionneur pour cet opérateur de Dirichlet vers Neumann global, un opérateur de Neumann vers Dirichlet dans

[3] Dans le premier article, les auteurs étudient en fait l'opérateur Neumann vers Dirichlet et non celui de Dirichlet vers Neumann.

chaque sous-domaine, qui tienne compte des sauts de perméabilité entre sous-domaines, voir par exemple l'article de Le Tallec, De Roeck et Vidrascu [93] et ses références. On obtient le préconditionneur de Neumann–Neumann proprement dit, qui nécessite à chaque itération du gradient conjugué préconditionné, en plus d'une résolution d'un problème de Dirichlet, une résolution d'un problème avec des conditions d'interface de type Neumann. Le coût en mémoire est donc de deux matrices factorisées, une pour le problème de Dirichlet, l'autre pour le problème de Neumann. En notant M^{-1} ce préconditionneur, on obtient un ordre de grandeur du conditionnement

$$\mathrm{cond}(M^{-1}S) \approx \frac{C}{H^2}\left(1 + \log\frac{H}{h}\right)^2,$$

où h est le pas maximal des maillages et la constante C ne dépend plus des sauts de perméabilités entre sous-domaines. Ce préconditionneur gomme donc la dépendance en h et les hétérogénéités, mais le système préconditionné supporte toujours mal l'augmentation du nombre de sous-domaines. Ceci s'explique par le fait qu'une information qui est émise à un bout du domaine doit traverser tous les sous-domaines un par un au cours des itérations pour arriver à l'autre bout du domaine[4].

La dernière étape provient d'une idée de Mandel [71], [72], appliquée aux éléments finis mixtes dans [36], qui consiste à faire transiter l'information globalement dans le domaine à l'aide d'un solveur grossier. Pour faire court, disons que ce solveur grossier équilibre —*balancing domain decomposition*— l'erreur à chaque itération en la répartissant entre tous les sous-domaines. Par la même occasion, il permet de traiter une difficulté technique du préconditionneur que je n'ai pas mentionnée par souci de simplification[5]. A chaque itération, ce solveur grossier nécessite une inversion d'un petit système matriciel, d'ordre le nombre de sous-domaines.

Pour résumer, la méthode de décomposition de domaines de Neumann–Neumann dans sa forme la plus évoluée, c'est-à-dire avec un solveur grossier pour l'équilibrage, constitue une méthode de résolution très efficace, car le condi-

[4]Cette analogie avec un problème d'évolution donne une bonne idée de la difficulté mais est assez inexacte, puisque les problèmes elliptiques sont statiques par définition. Il ne peut y avoir à proprement parler d'"émission" d'information.

[5]Les problèmes de Neumann peuvent être mal posés dans certains sous-domaines, quand il n'y a que des conditions de type Neumann sur le bord du sous-domaine. Le solveur grossier assure que l'intégrale des flux sur les bords de tels sous-domaines soient nulle, et garantit ainsi que tous les problèmes peuvent être résolus.

tionnement du système préconditionné devient, [67],

$$\mathrm{cond}(M^{-1}S) \leq \frac{C}{\max_i \alpha_i^2} \left(1 + \log \max_i \frac{H_i}{h_i}\right)^2,$$

où la constante C est indépendante du diamètre H_i, du pas de discrétisation h_i et du facteur de forme α_i[6] des sous-domaines d'indice i pour $i = 1, 2, \ldots, n$, n étant le nombre de sous-domaines, et des sauts de perméabilité entre sous-domaines. Le conditionnement est donc quasiment indépendant du nombre de mailles, du nombre de sous-domaine et des sauts de perméabilité entre les sous-domaines.

En revanche, le coût en ressources informatiques est relativement élevé, surtout en trois dimensions, car il nécessite le stockage de deux matrices factorisées par sous-domaine, une pour le problème de Dirichlet, et l'autre pour le problème de Neumann, plus une matrice pour le solveur grossier. Le parallélisme, intrinsèque à la méthode, devient très vite nécessaire sur des applications réalistes. Notons au passage l'existence de la méthode FETI [50] —*Finite Element Tearing and Interconnecting*— très similaire, qui est basée sur une formulation variationnelle duale et l'utilisation de multiplicateurs de Lagrange. Elle continue à avoir de nombreux développements à ce jour, [84].

La méthode de Neumann–Neumann a été étendue aux problèmes non-symétriques. Ainsi, dans [3], Achdou, Le Tallec, Nataf, et Vidrascu traitent-ils le problème de convection–diffusion. Le préconditionneur prend en compte le terme de convection en ajoutant à la vitesse normale sur l'interface un terme proportionnel à la trace de la variable scalaire. La condition d'interface devient alors une condition de type Robin ; le préconditionneur voit parfois son nom changé en Robin–Robin, voir aussi [5], [1]. Une autre application non-symétrique pour la mécanique se trouve dans [8].

Nous indiquons pour finir à ce sujet que nous avons programmé et testé en Matlab la méthode de Neumann–Neumann avec solveur grossier pour les problèmes d'écoulement, en collaboration avec Roland Morel pendant son stage de DESS, [74].

2.1.2 Méthode d'Aitken–Schwarz

Nous tenons à mentionner le travail de Garbey, Tromeur-Dervout et al., [17], [54] et [55] qui parviennent à faire de beaux calculs massivement parallèles sur des domaines réguliers en utilisant une décomposition de domaines à deux

[6] Le facteur de forme d'un sous-domaine est défini comme le rapport entre le diamètre de la plus grosse sphère incluse dans le sous-domaine et son diamètre.

niveaux. Elle vise à employer plusieurs grappes de PC *distantes*. Il s'agit dans cette méthodologie de découper le domaine en couches suivant une direction. Chaque couche est traitée séparément par un solveur parallèle sur l'une des grappes. Le couplage entre les couches s'opère grâce à la méthode d'Aitken–Schwarz, qui accélère dramatiquement la convergence.

2.1.3 Eléments joints pour maillages non-raccordés

La méthode dite des éléments joints —*mortar*— est née il y a une dizaine d'années au laboratoire de Paris VI. Elle a été initialement développée par Bernardi, Maday et Patera [21], [20], pour faire communiquer les éléments finis et les éléments finis spectraux : les éléments joints permettent de raccorder un domaine où vivent des éléments de Lagrange habituels et un autre où sont utilisés des éléments spectraux. Cette approche par éléments joints s'est révélée extrêmement féconde, et a vu depuis sa création de nombreux prolongements dans différents domaines, comme en atteste la richesse de la littérature à ce sujet, cf. les nombreuses références incluses dans [22]. Ainsi est-elle invoquée à présent pour recoller des sous-domaines qui ont le même type d'éléments mais sur des maillages non-raccordés.

Dans le cadre des éléments finis mixtes, Arbogast, Cowsar, Lawrence, Wheeler et Yotov ont analysé en détail la décomposition de domaines pour des maillages ne se raccordant pas aux interfaces, [13]. Une première étude peut être trouvée dans la thèse de Yotov [99]. Le principe est globalement le même que pour la résolution du problème d'interface (2.1). La différence essentielle réside dans l'espace des inconnues d'interface noté L_h. Quand les maillages sont raccordés, les inconnues d'interface vivent dans l'espace discret des multiplicateurs de Lagrange représentant les traces de pression, [16]. Ceci n'est évidemment plus possible pour des maillage non-raccordés, puisque les multiplicateurs de Lagrange des deux côtés d'une interface vivent sur deux maillages distincts. Conformément à la philosophie des éléments joints, l'espace L_h est donc défini sur un maillage d'interface distinct a priori des maillages d'interface provenant des sous-domaines. Les auteurs de [13] montrent l'existence, l'unicité de la solution et des estimations d'erreur, à la condition que

$$\text{si } \phi \in L_h \text{ et } \mathcal{Q}_{h,i}\phi = 0 \ , 1 \leq i \leq n, \text{ alors } \phi = 0,$$

où $\mathcal{Q}_{h,i}$ est la projection orthogonale sur l'espace des traces normales des vitesses du sous-domaine i, et n est le nombre de sous-domaines.

Cette condition impose de ne pas avoir d'espace de joints L_h trop riche. Plus l'espace L_h est riche, meilleure est la conservation de masse locale, mais on ne peut pas imposer trop de conservation locale sur un maillage non-concordant, faute de quoi, l'unicité de la solution est perdue. Plaçons-nous dans le cas des espaces de Raviart et Thomas de plus bas degré, [82], pour fixer les idées. Il est préconisé dans l'article de prendre comme espace celui des fonctions P_1 discontinues pour les triangles, ou Q_1 discontinues pour les quadrangles, sur un maillage qui ne soit pas trop fin. Un maillage qui peut toujours être utilisé, est l'un des deux maillages d'interface, déraffiné une fois, c'est-à-dire avec un pas deux fois plus grand.

Dans la pratique, la résolution du système d'interface s'effectue de la même manière que pour (2.1), avec deux phases de projection en plus à chaque itération, faisant intervenir $\mathcal{Q}_{h,i}$ et sa transposée.

Nous mentionnons enfin que Pencheva et Yotov montrent dans [56] que Neumann–Neumann avec un équilibrage par un solveur grossier s'étend de façon satisfaisante au cas des maillages non-raccordés grâce aux éléments joints.

2.1.4 Méthode de Robin non-conforme pour maillages non-raccordés

Nous étudions et utilisons dans ce mémoire une méthode de décomposition de domaines différentes de celles présentées jusqu'à présent. Cette méthode se base sur des conditions d'interface de type Robin et permet de traiter des sous-domaines non-raccordés. Dans [2], où Achdou, Japhet, Maday et Nataf l'étudient dans le cadre des volumes finis, les auteurs la dénomment "nouveau ciment" —*new cement*—, en référence au mortier des éléments joints, tandis que dans [14], où Arbogast et de Yotov l'analysent dans le cadre des éléments finis mixtes, elle est qualifiée de "non-mortier" —*non mortar*. Aucune de ces appellation ne me convenant, je la baptise pour ce mémoire la méthode de décomposition de domaines de *Robin non-conforme*, faute d'une meilleure idée.

Cette méthode tire ses sources de l'article [68] de Lions, qui étudie une méthode basée sur des conditions d'interface de type Robin dans le cas $\mathcal{H}^1(\cdot)$ pour des maillages raccordés. Elle a également été analysée par Després dans [44] et [45] pour les problèmes de Helmholtz, avec maillages raccordés. Une étude du problème d'écoulement a été menée par Douglas, Paes-Leme, Roberts et Wang dans [48] pour les éléments finis mixtes, et une autre par Després, Joly et Roberts analyse le problème de Maxwell, [46].

Dans [45], l'auteur propose des préconditionneurs astucieux pour permettre la convergence. Dans les problèmes elliptiques qui nous intéressent, à part sur un problème particulièrement raide, cf. Section 3.6.2, nous n'avons pas eu de problèmes de convergence nous incitant à programmer un préconditionneur.

L'optique consiste plutôt, avec cette méthode de Robin non-conforme, à jouer sur le coefficient de Robin pour accélérer la convergence.

Nous n'affirmons pas que cette méthode est meilleure ou plus performante qu'une de celles précédemment citées aux Sections 2.1.1, 2.1.2 ou 2.1.3.

Mentionnons qu'il serait sans doute possible et sûrement fructueux d'accélerer cette méthode avec Aitken, mais nous n'avons pas exploré cette voie plus avant.

La comparaison entre la méthode de Robin non-conforme et le préconditionneur de Neumann–Neumann utilisant les éléments joints et un solveur grossier est également un point intéressant. Les deux méthodes sont optimales en précision, [14] et [13]. A priori, Neumann–Neumann devrait converger plus rapidement que Robin non-conforme pour un grand nombre de sous-domaines, quand le pas du maillage diminue ou quand les hétérogénéités deviennent importantes. Pour des problèmes symétriques elliptiques, Neumann–Neumann engendre un système linéaire symétrique défini positif, tandis que l'algorithme de Robin non-conforme est essentiellement non-symétrique. En revanche, Neumann–Neumann coûte environ deux fois plus cher en mémoire que Robin non-conforme. Si le coefficient de Robin est choisi de façon optimale, voir [78] et les références incluses, la convergence peut être fortement accélérée. Du point de vue de l'implémentation, la méthode de Robin non-conforme ne nécessite pas de maillage supplémentaire de l'interface ; or ce maillage est indispensable quand on utilise Neumann–Neumann, et peut être assez délicat à construire.

2.2 Description des problèmes multiblocs

2.2.1 Problème modèle

Soit Ω un domaine convexe de $\mathbb{R}^d$, $d = 2$ ou 3. Nous noterons $\Gamma = \partial\Omega$ la frontière supposée suffisamment régulière de Ω. Nous supposons que l'écoulement au sein de Ω est dirigé par l'équation de conservation associée à la loi de Darcy, voir [19], qui relie le gradient de la pression p à la vitesse de Darcy notée $\mathbf{u}$:

$$\begin{array}{rcll} \operatorname{div}\mathbf{u} & = & q & \text{dans } \Omega \\ \mathbf{u} & = & -\mathbf{K}\boldsymbol{\nabla} p & \text{dans } \Omega \\ p & = & \overline{p} & \text{sur } \Gamma, \end{array} \tag{2.2}$$

où p est la pression, $\mathbf{u}$ la vitesse de Darcy, $\mathbf{K}$ le tenseur de conductivité hydraulique (encore appelée perméabilité), q un terme source et $\overline{p}$ est la pression imposée sur la frontière Γ. Nous supposons que $\mathbf{K}$ est symétrique défini positif, et uniformément borné : il existe deux constantes positives $0 < K_{\min} < K_{\max}$ telles que pour tout vecteur ξ de $\mathbb{R}^d$, $0 < K_{\min}\, \xi^T\xi \leq \xi^T\mathbf{K}\xi \leq K_{\max}\, \xi^T\xi$ sur Ω.

2.2.2 Découpage du domaine

Afin de résoudre ce problème par une méthode de décomposition de domaines, nous découpons Ω en $n > 1$ sous-domaines sans chevauchement $\Omega_i, i \in I_n := \{1, 2, \ldots, n\}$. On notera ν_i la normale unitaire sortante de $\Omega_i, i \in I_n$ (et ν celle de Ω).

Ces sous-domaines sont parfois appelés blocs [14], en particulier quand les sous-domaines sont des rectangles (en 2D) ou des hexaèdres (en 3D) peu déformés auxquels un maillage structuré peut être appliqué.

Nous notons Γ_i la partie du bord de Ω_i, $i \in I_n$ qui est commune à la frontière de Ω. Ce bord Γ_i est donc le bord *extérieur* de Ω_i, $i \in I_n$. Ceci s'écrit :

$$\bar{\Omega} = \bigcup_{i \in I_n} \bar{\Omega}_i, \quad \text{et} \quad \Gamma_i = \partial\Omega_i \cap \Gamma, \quad i \in I_n = \{1, 2, \ldots, n\},$$

où $\bar{\Omega}_i$ est la fermeture de l'ouvert Ω_i. Soit Σ_i l'intérieur du complémentaire du bord extérieur de Ω_i. On appellera indifféremment Σ_i *frontière interne* ou *interface* de Ω_i. Nous introduisons également Σ la réunion de toutes les frontières internes.

$$\Sigma_i = \partial\Omega_i \backslash \partial\Omega, \quad i \in I_n, \qquad \Sigma = \bigcup_{i \in I_n} \Sigma_i.$$

Ensuite nous pouvons définir l'*interface entre les sous-domaines* Ω_i et Ω_j :

$$\Sigma_{ij} = \Sigma_{ji} = \Sigma_i \cap \Sigma_j, \quad i \neq j \in I_n, \qquad \Sigma_{ii} = \emptyset, \quad i \in I_n.$$

Si $\Sigma_{ij} \neq \emptyset$, les sous-domaines Ω_i et Ω_j, $i \neq j \in I_n$, sont dits *voisins.* On note qu'une interface Σ_{ij} ne peut être qu'une surface quand $d = 3$ (ou, en 2 dimensions, une courbe) ou l'ensemble vide : *deux sous-domaines ne sont voisins que quand leur interface est de dimension* $d - 1$.[7] Ainsi, en dimension 3, on ne considère pas que des sous-domaines ayant une courbe ou un point en commun sur leur frontière soient voisins. De même, en dimension 2, deux sous-domaines ne peuvent être voisins que s'ils se touchent par des lignes communes : sur la Figure 2.1, les sous-domaines Ω_1 et Ω_4 ne sont pas voisins car leurs fermetures ne s'intersectent que par un point.

[7] Ceci est possible dans la mesure où nous n'utilisons que des éléments finis mixtes ou des volumes finis pour la discrétisation et que les communications entre sous-domaines se font par conséquent au travers des côtés (2D) ou des facettes (3D). Les degrés de liberté *ne vivent pas* sur les sommets du maillage.

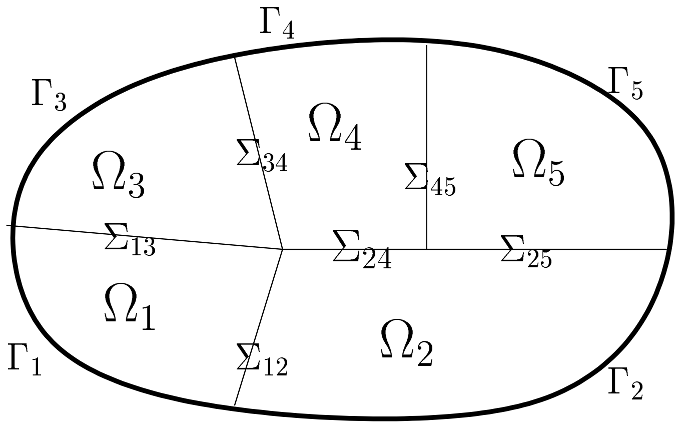


FIG. 2.1 – Domaine décomposé en 5 sous-domaines. Exemple en dimension 2.

Ceci amène des simplifications pour le traitement des communications puisqu'une interface de dimension $d-1$ ne peut être commune qu'à *deux* sous-domaines, alors qu'un sommet —et une arête en 3D— peut être commun à un nombre arbitraire de sous-domaines, comme on peut le voir sur la Figure 2.1.

Pour $i \in I_n$, nous appelons n_i le nombre de voisins que possède un sous-domaine Ω_i, et nous introduisons l'ensemble des indices de ces voisins que nous notons

$$\mathcal{N}_i := \{j_1, j_2, \dots, j_{n_i}\},$$

la lettre $\mathcal{N}$ se rapportant aux voisins —$\mathcal{N}$ comme "Noisins", ou plutôt comme *Neighbors* en américain. Nous ordonnons les ensembles I_n et $\mathcal{N}_i$ suivant l'ordre naturel de $\mathbb{N}$. Ainsi l'application $k \in \{1, 2, \dots, n_i\} \mapsto j_k \in \mathcal{N}_i$ est strictement croissante.

Nous avons aussi besoin de l'ensemble des couples ordonnés de voisins, noté

$$\mathcal{N} := \{(i,j), i \in I_n, j \in \mathcal{N}_i\}.$$

Cet ensemble est ordonné suivant l'ordre lexicographique sur $\mathbb{N}^2$. Nous introduisons s la permutation telle que

$$\forall (i,j) \in \mathcal{N}, \quad s(i,j) = (j,i) \in \mathcal{N}. \tag{2.3}$$

On remarque qu'une interface géométrique entre sous-domaines $\Sigma_{ij} = \Sigma_{ji}$, $(i,j) \in \mathcal{N}$ est comptée deux fois dans l'ensemble $\mathcal{N}$. Quand on aura besoin de faire intervenir une interface une seule fois, on utilisera la notation $i < j \in I_n$.

2.2.3 Problème de transmission

On note $p_i, \mathbf{u}_i, \mathbf{K}_i$, et q_i les restrictions de $p, \mathbf{u}, \mathbf{K}$, et q respectivement à Ω_i, $i \in I_n$, et $\overline{p}_i$ celle de $\overline{p}$ à Γ_i, $i \in I_n$. De façon générale, nous aurons naturellement tendance à utiliser l'indice i pour noter les variables se référant au sous-domaine Ω_i, $i \in I_n$. Nous pouvons réécrire le problème originel (2.2) comme le problème de transmission suivant :

$$\begin{aligned} \operatorname{div} \mathbf{u}_i &= q_i && \text{dans } \Omega_i, \quad i \in I_n, \\ \mathbf{u}_i &= -\mathbf{K}_i \boldsymbol{\nabla} p_i && \text{dans } \Omega_i, \quad i \in I_n, \\ p_i &= \overline{p}_i && \text{sur } \Gamma_i, \quad\ \ i \in I_n, \end{aligned} \tag{2.4}$$

avec les conditions de transmission sur l'interface

$$\begin{aligned} p_i &= p_j && \text{sur } \Sigma_{ij}, \quad i < j \in I_n, \\ \mathbf{u_i} \cdot \nu_\mathbf{i} &= -\mathbf{u_j} \cdot \nu_\mathbf{j} && \text{sur } \Sigma_{ij}, \quad i < j \in I_n. \end{aligned} \tag{2.5}$$

Il est clair que la solution du problème d'origine (2.2) satisfait (2.4) et (2.5). Réciproquement, d'après la régularité elliptique, la solution de (2.4) et (2.5) satisfait (2.2), ce qui établit l'équivalence des deux formulations.

Les deux équations de (2.5) assurent la continuité de la pression et de la composante normale de la vitesse (car $\nu_\mathbf{i} = -\nu_\mathbf{j}$) à travers chaque interface $\Sigma_{ij}(\neq \emptyset)$ entre deux voisins Ω_i et Ω_j, $(i,j) \in \mathcal{N}$. C'est la manière dont sont imposées ces deux conditions de transmission qui définissent la méthode de décomposition de domaines. En effet, un problème sur un sous-domaine n'est bien posé que si l'on impose une et une seule condition d'interface. Il faut donc mettre en œuvre un algorithme pour prendre en compte la seconde. C'est là toute la difficulté et l'art de la méthode de décomposition de domaines.

2.2.4 Maillages dans les sous-domaines

Soit $\mathcal{T}_i^h$ un maillage conforme éléments finis du sous-domaine Ω_i, $i \in I_n$, pour lequel les éléments ont un diamètre majoré par le réel $h > 0$. On suppose de plus que l'intérieur de chaque face d'élément se trouvant sur le bord est entièrement incluse soit dans Σ_i, soit dans $\Gamma_i = \partial\Omega_i \cap \Gamma$. Les maillages de deux sous-domaines voisins ne se raccordent pas nécessairement sur leur interface Σ_{ij}.

Précisons à présent les notations des maillages d'interface, dont nous rappelons qu'ils vivent en dimension $d-1$, si $\Omega \subset \mathbb{R}^d$. Nous venons juste de définir deux voisins comme des sous-domaines Ω_i et Ω_j, $i \neq j \in I_n$, tels que leur interface Σ_{ij} est non vide, et plus directement, tels que $(i,j) \in \mathcal{N}$.

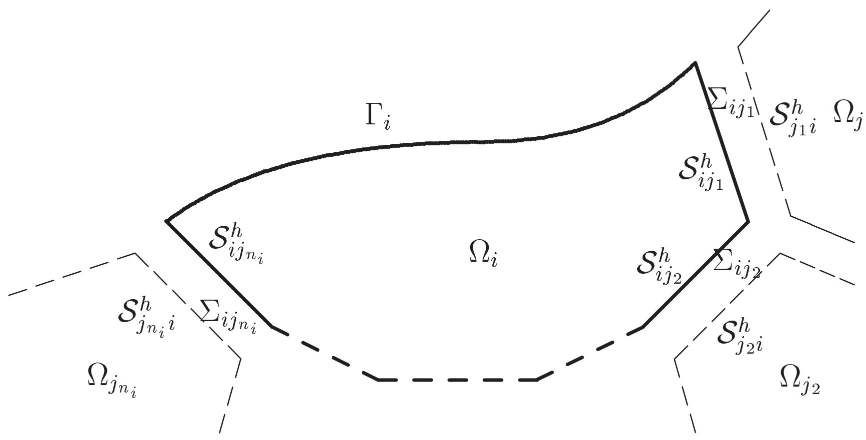


FIG. 2.2 – Le sous-domaine $\Omega_i, i \in I_n$, et ses n_i voisins Ω_{j_k}, $j_k \in \mathcal{N}_i$. Les maillages d'interface $\mathcal{S}^h_{ij_k}$ et $\mathcal{S}^h_{j_k i}$ de chaque côté de l'interface géométrique $\Sigma_{ij_k} = \Sigma_{j_k i}$ sont différents. Exemple en dimension 2.

Structure interne d'interface Pour un $i \in I_n$ et un $j \in \mathcal{N}_i$, la trace sur Σ_{ij} du maillage $\mathcal{T}^h_i$ du sous-domaine Ω_i est notée $\mathcal{S}^h_{ij}$. De la même manière et symétriquement, pour un $j \in I_n$ et un $i \in \mathcal{N}_j$, la trace sur $\Sigma_{ji} = \Sigma_{ij}$ du maillage $\mathcal{T}^h_j$ est appelée $\mathcal{S}^h_{ji}$. Comme les maillages $\mathcal{T}^h_i$ et $\mathcal{T}^h_j$ ne coïncident pas a priori, $\mathcal{S}^h_{ji} \neq \mathcal{S}^h_{ij}$. Voir un exemple en 2 dimensions de ces définitions dans la Figure 2.2.

Nous notons de plus $\mathcal{S}^h_i$ la trace sur Σ_i du maillage $\mathcal{T}^h_i$ de Ω_i. C'est également la réunion de tous les maillages d'interface issus du sous-domaine Ω_i. Ce maillage $\mathcal{S}^h_i$ sera appelé *maillage d'interface interne* de Ω_i.

$$\forall i \in I_n,\ \forall j \in \mathcal{N}_i \quad \mathcal{S}^h_{ij} = \mathcal{T}^h_i|_{\Sigma_{ij}}. \qquad \mathcal{S}^h_i = \biguplus_{j \in \mathcal{N}_i} \mathcal{S}^h_{ij},$$

où le signe $\biguplus$ symbolise l'union disjointe.

Nous appelons *structure interne d'interface* du domaine Ω, notée $\mathcal{S}^h$, la réunion de tous les maillages d'interface interne (chaque interface géométrique est donc comptée deux fois). La structure interne d'interface est ordonnée comme $\mathcal{N}$.

$$\mathcal{S}^h = \biguplus_{i \in I_n} \mathcal{S}^h_i = \biguplus_{i \in I_n} \biguplus_{j \in \mathcal{N}_i} \mathcal{S}^h_{ij} = \biguplus_{(i,j) \in \mathcal{N}} \mathcal{S}^h_{ij}.$$

Nous avons ainsi défini jusqu'à présent trois niveaux différents de maillages d'interface : l'interface entre deux sous-domaines, l'interface complète d'un sous-domaine et l'interface globale pour le domaine entier. Cependant, dans toute cette zoologie de notations, nous avons toujours fait intervenir les maillages d'interface d'un sous-domaine depuis le sous-domaine lui-même et non depuis ses voisins. C'est ce que nous faisons maintenant.

Structure externe d'interface Nous introduisons les maillages d'interface des voisins d'un sous-domaine Ω_i. Pour $(i,j) \in \mathcal{N}$, soit $\tilde{\mathcal{S}}_{ij}^h := \mathcal{S}_{ji}^h$ le maillage trace de $\mathcal{T}_j^h$ sur $\Sigma_{ij} = \Sigma_{ji}$. Nous appelons $\tilde{\mathcal{S}}_i^h$ le *maillage d'interface externe* du sous-domaine Ω_i, qui est la réunion de tous les maillages traces des voisins de Ω_i. Cet ensemble est aussi numéroté suivant $\mathcal{N}_i$.

$$\tilde{\mathcal{S}}_i^h = \biguplus_{j \in \mathcal{N}_i} \tilde{\mathcal{S}}_{ij}^h = \biguplus_{j \in \mathcal{N}_i} \mathcal{S}_{ji}^h.$$

De la même façon, nous introduisons $\tilde{\mathcal{S}}^h$ la *structure d'interface externe* du domaine Ω, qui est évidemment la réunion de tous les maillages d'interface externe. Cet ensemble est, comme $\mathcal{S}^h$, la réunion de tous les maillages d'interface, mais cette fois l'ensemble est ordonné suivant $s(\mathcal{N})$, où s est défini en (2.3),

$$\tilde{\mathcal{S}}^h = \biguplus_{i \in I_n} \tilde{\mathcal{S}}_i^h = \biguplus_{i \in I_n} \biguplus_{j \in \mathcal{N}_i} \tilde{\mathcal{S}}_{ij}^h = \biguplus_{i \in I_n} \biguplus_{j \in \mathcal{N}_i} \mathcal{S}_{ji}^h = \biguplus_{(i,j) \in s(\mathcal{N})} \mathcal{S}_{ij}^h.$$

Pour être parfaitement clair, nous explicitons sur un exemple les ordres sur $\mathcal{N}$ et sur $s(\mathcal{N})$. Si la décomposition en 5 sous-domaines est comme sur la Figure 2.1, alors l'ensemble des couples de voisins ordonnés suivant l'ordre lexicographique est

$$\begin{aligned} \mathcal{N} = \{&(1,2);(1,3);(2,1);(2,4);(2,5);\\ &(3,1);(3,4);(4,2);(4,3);(4,5);(5,2);(5,4)\}, \end{aligned}$$

et dans l'ordre $s(\mathcal{N})$, qui correspond d'ailleurs à l'ordre lexicographique tranposé, cet ensemble devient

$$\begin{aligned} s(\mathcal{N}) = \{&(2,1);(3,1);(1,2);(4,2);(5,2);\\ &(1,3);(4,3);(2,4);(3,4);(5,4);(2,5);(4,5)\}. \end{aligned}$$

Si le symbole $\prec$ désigne l'ordre lexicographique sur $\mathcal{N}$, alors l'ordre de $s(\mathcal{N})$ noté $\prec\prec$ est caractérisé par

$$s(a) \prec\prec s(b) \Longleftrightarrow a \prec b, \qquad a,b \in \mathcal{N}.$$

2.3 Algorithme de Robin non-conforme

2.3.1 Formulation continue

Soient $\alpha_{ij} > 0$ et $\alpha_{ji} > 0$ des coefficients de Robin réels pour l'interface $\Sigma_{ij} = \Sigma_{ji},\ (i,j) \in \mathcal{N}$. Les équations de continuité aux interfaces (2.5) peuvent se réécrire avec des conditions de Robin (2.7). Finalement, on obtient le nouveau problème de transmission avec des conditions de Robin

$$\begin{array}{rcll} \operatorname{div} \mathbf{u}_i & = & q_i & \text{dans } \Omega_i, \quad i \in I_n, \\ \mathbf{u}_i & = & -\mathbf{K}_i \boldsymbol{\nabla}\, p_i & \text{dans } \Omega_i, \quad i \in I_n, \\ p_i & = & \overline{p}_i & \text{sur } \Gamma_i, \quad\ \ i \in I_n, \end{array} \tag{2.6}$$

associé à

$$\begin{array}{rcll} -\mathbf{u_i} \cdot \nu_\mathbf{i} + \alpha_{ij} p_i & = & \mathbf{u_j} \cdot \nu_\mathbf{j} + \alpha_{ij} p_j & \text{sur } \Sigma_{ij}, \quad i < j \in I_n, \\ -\mathbf{u_j} \cdot \nu_\mathbf{j} + \alpha_{ji} p_j & = & \mathbf{u_i} \cdot \nu_\mathbf{i} + \alpha_{ji} p_i & \text{sur } \Sigma_{ij}, \quad i < j \in I_n. \end{array} \tag{2.7}$$

Il est facile de vérifier que les deux conditions de Dirichlet et Neumann (2.5) sont équivalentes aux deux conditions de Robin (2.7). D'abord, on multiplie la première équation de (2.5) par α_{ij} et on en soustrait la seconde équation, pour obtenir la première équation de (2.7). Ensuite, on multiplie la première équation de (2.5) par α_{ji} et on y ajoute la seconde équation, ce qui donne la deuxième équation de (2.7).

Donc finalement le problème (2.2) est équivalent au problème (2.6), (2.7).

La première équation de (2.7) s'interprète comme une condition de type Robin imposée sur le bord de Ω_i, tandis que la seconde correspond à une condition de Robin pour le sous-domaine Ω_j.

On présente une méthode itérative simple pour résoudre (2.6), (2.7) en (2.8). Etant donnés $\mathbf{u_j^0}$ et $p_j^0,\ j \in I_n$, on résout successivement pour $k = 1, 2, \ldots$

$$\begin{array}{rcll} \operatorname{div} \mathbf{u}_i^k & = & q_i & \text{dans } \Omega_i, \quad i \in I_n, \\ \mathbf{u}_i^k & = & -\mathbf{K}_i \boldsymbol{\nabla}\, p_i^k & \text{dans } \Omega_i, \quad i \in I_n, \\ p_i^k & = & \overline{p}_i & \text{sur } \Gamma_i, \quad\ \ i \in I_n, \\ -\mathbf{u_i^k} \cdot \nu_\mathbf{i} + \alpha_{ij} p_i^k & = & \mathbf{u_j^{k-1}} \cdot \nu_\mathbf{j} + \alpha_{ij} p_j^{k-1} & \text{sur } \Sigma_{ij}, \quad (i,j) \in \mathcal{N}. \end{array} \tag{2.8}$$

Remarque 1. Quelques remarques préliminaires peuvent être faites dès à présent.

- Cette méthode itérative a été initialement analysée par Lions dans le cadre $\mathcal{H}^1(\Omega)$ dans [68]. A partir d'arguments basés sur des estimations d'énergie, il y démontre la convergence de p_i^k vers p_i dans $\mathcal{H}^1_{\Gamma_i}(\Omega_i) = \{p \in \mathcal{H}^1(\Omega_i)/\ p = \overline{p}_i \text{ on } \Gamma_i\}$ quand k tend vers $+\infty$. Une autre démonstration dans un cas particulier peut être trouvée dans [81] pages 135–137. Pour les méthodes mixtes, une démonstration de la convergence existe également dans [48]. Dans ces références, le maillage est supposé raccordé aux interfaces.

- On vérifie aisément que les deux conditions de (2.7) sont assurées quand la convergence est atteinte : il suffit d'écrire la quatrième équation de (2.8) deux fois pour $\Sigma_{ij}, (i,j) \in \mathcal{N}$, une fois avec i et une seconde avec j.
- Cette méthode itérative n'est pas symétrique, bien que le Laplacien le soit. Dans la pratique, elle est accélérée avec une méthode de Krylov non-symétrique, comme Bi-CGStab [95], ou GMRes [86].
- Une version plus détaillée de la méthode itérative est donnée plus loin, cf. l'Algorithme 1.

2.3.2 Algorithme de point fixe

Nous précisons quelque peu la méthode itérative qui a été introduite en (2.8). Le but n'est pas ici de définir rigoureusement la méthode, mais juste de donner dès maintenant une idée des étapes nécessaires à la résolution pratique de (2.6)-(2.7). Une version complètement détaillée sera présentée plus loin, voir l'Algorithme 2, après que les différents opérateurs et espaces nécessaires auront été définis.

La solution du problème (2.6)-(2.7) peut être trouvée simplement en utilisant l'algorithme 1 de point fixe suivant.

Algorithme 1 (de point fixe, méthode de Robin non-conforme). *Soit une tolérance donnée tol > 0. Choisir λ_i^0, $i \in I_n$, résoudre itérativement pour $k = 1, 2, \dots$:*

1. *Pour chaque $i \in I_n$, étant donné λ_i^{k-1} trouver en parallèle $\mathbf{u}_i^k$ et p_i^k dans Ω_i tels que*

$$\begin{array}{rcll} div\, \mathbf{u}_i^k &=& q_i & \text{dans } \Omega_i, \quad i \in I_n, \\ \mathbf{u}_i^k &=& -\mathbf{K}_i \boldsymbol{\nabla} p_i^k & \text{dans } \Omega_i, \quad i \in I_n, \\ p_i^k &=& \overline{p}_i & \text{sur } \Gamma_i, \quad i \in I_n, \\ -\mathbf{u_i^k} \cdot \nu_\mathbf{i} + \alpha_{ij} p_i^k &=& \lambda_i^{k-1} & \text{sur } \Sigma_{ij}, \quad (i,j) \in \mathcal{N}. \end{array} \tag{2.9}$$

2. *En déduire : $\mu_{ij}^k = \mathbf{u_i^k} \cdot \nu_\mathbf{i} + \alpha_{ji} p_i^k$, sur Σ_{ij}, $(i,j) \in \mathcal{N}$.*
3. *Projeter μ_{ij}^k sur le sous-domaine voisin avec l'opérateur $P_{i \mapsto j}$ défini en (2.50) :*
$$\lambda_{ji}^k = P_{i \mapsto j} \mu_{ij}^k, \ (i,j) \in \mathcal{N}.$$
4. *Reconstituer λ_j^k, $j \in I_n$ à partir des λ_{ji}^k, $i \in \mathcal{N}_j$.*

5. *Si* $||\lambda_i^k - \lambda_i^{k-1}||_\infty < tol$, $\forall i \in I_n$, *sortir. Sinon retourner à 1. avec* λ_i^k.

□

L'opérateur $P_{i\mapsto j}$ permet de faire de façon conservative le transfert de μ_{ij}^k d'un maillage d'interface $\mathcal{S}_{ij}^h$ au maillage d'interface du voisin $\mathcal{S}_{ji}^h$: l'opérateur $P_{i\mapsto j}$ est la projection L^2 sur l'espace des inconnues vivant sur $\mathcal{S}_{ji}^h$, voir sa définition en (2.50) et son calcul dans la pratique dans la Section 2.4.

2.3.3 Problème multibloc faible

Pour un $i \in I_n$, nous notons $(\cdot,\cdot)_i$ le produit scalaire dans $L^2(\Omega_i)$ ou dans $(L^2(\Omega_i))^d$. De façon naturelle, l'indice i est omis pour le produit scalaire $(\cdot,\cdot)$ portant sur le domaine entier Ω. De la même manière, on introduit $||\cdot||_i$ la norme de $L^2(\Omega_i)$ ou $(L^2(\Omega_i))^d$, et $||\cdot||$ la norme prise sur Ω tout entier.

Soit enfin $\langle\cdot,\cdot\rangle_i$ le produit scalaire –qui peut, par abus de notation, devenir un crochet de dualité– sur $L^2(\Sigma_i)$, et $\langle\cdot,\cdot\rangle_{ij}$ le produit scalaire sur $L^2(\Sigma_{ij})$. Nous aurons besoin également des normes $|||\cdot|||_i$ et $|||\cdot|||_{ij}$ associées à ces produits scalaires.

On note que si $\Sigma_{ij} = \emptyset$, ce produit scalaire est identiquement nul. Cette remarque permettra d'opérer des sommations indifféremment soit sur toutes les interfaces (i.e. $\sum_{(i,j)\in I_n^2}$) en considérant un certain nombre d'interfaces vides, soit uniquement sur les interfaces réellement non vides (i.e. $\sum_{(i,j)\in\mathcal{N}}$).

Définissons à présent les espaces d'approximation utilisés dans cette partie.

$$\begin{aligned}
&\mathbf{Z_i} = \mathcal{H}(\mathrm{div},\Omega_i), \quad i \in I_n. \quad \mathbf{Z} = \bigoplus_{i\in I_n} \mathbf{Z_i} \not\subset \mathcal{H}(\mathrm{div},\Omega).\\
&N_i = L^2(\Omega_i), \qquad i \in I_n. \quad N = \bigoplus_{i\in I_n} N_i = L^2(\Omega).\\
&\Lambda_i = L^2(\Sigma_i), \qquad i \in I_n. \quad \Lambda = \bigoplus_{i\in I_n} \Lambda_i.
\end{aligned} \tag{2.10}$$

Tout au long du reste de ce chapitre nous allons supposer pour simplifier que les conditions imposées au bord extérieur du domaine sont de type Dirichlet homogène : $\overline{p} = 0$ on Γ.

Pour les démonstrations d'existence et d'estimations d'erreurs de cette section, nous introduisons le problème (2.11) qui est identique au problème original (2.2) à la modification suivante près : il est ajouté un terme cp à la divergence, qui peut représenter une discrétisation temporelle. Le coefficient $c > 0$ est alors la compressibilité du fluide.

Dans [48], une démonstration de la convergence de la méthode itérative *pour des maillages raccordés* est faite pour $c \geq 0$. On pourrait peut-être suivre le même schéma que dans [48] pour étendre les résultats qui suivent à $c = 0$.

En fait, les démonstrations sont basées sur des estimations d'énergie et requièrent la coercivité de la formulation. Celle-ci n'est jamais vérifiée dans une formulation mixte d'un problème elliptique. Il est donc indispensable d'ajouter ce terme $c > 0$ que l'on va supposer uniformément borné : il existe deux constantes positives $0 < c_{\min} < c_{\max}$ telles que $0 < c_{\min} \leq c \leq c_{\max}$ sur Ω.

$$\begin{array}{rcll} cp + \operatorname{div} \mathbf{u} & = & q & \text{dans } \Omega \\ \mathbf{u} & = & -\mathbf{K} \boldsymbol{\nabla} p & \text{dans } \Omega \\ p & = & \overline{p} & \text{sur } \Gamma. \end{array} \tag{2.11}$$

Avec les notations introduites jusqu'à présent, nous pouvons écrire la formulation mixte du problème. Pour cela, il faut multiplier la seconde équation de (2.11) par une fonction test $\mathbf{K^{-1}v}$, l'intégrer sur Ω_i puis intégrer par partie le terme en $\boldsymbol{\nabla} p$.

Une solution faible de (2.6)-(2.7) est un couple $(\mathbf{u}, p) \in \mathbf{Z} \times N$ tel que pour chaque $i \in I_n$, $p_i|_{\Sigma_i} \in \Lambda_i$, et $\mathbf{u_i} \cdot \nu_\mathbf{i}|_{\Sigma_i} \in \Lambda_i$, et

$$\begin{array}{rcll} (\mathbf{K^{-1}u},\, \mathbf{v})_i - (\operatorname{div} \mathbf{v},\, p)_i & = & -\left\langle \mathbf{v} \cdot \nu_\mathbf{i},\, p_i \right\rangle_i, & \mathbf{v} \in \mathbf{Z_i}, \\ (\operatorname{div} \mathbf{u},\, r)_i + (cp,\, r)_i & = & (q_i,\, r)_i, & r \in N_i, \\ \displaystyle\sum_{j \in I_n} \left\langle -\mathbf{u_i} \cdot \nu_\mathbf{i} + \alpha_{ij} p_i,\, \mu_i \right\rangle_{ij} & = & \displaystyle\sum_{j \in I_n} \left\langle \mathbf{u_j} \cdot \nu_\mathbf{j} + \alpha_{ij} p_j,\, \mu_i \right\rangle_{ij}, & \mu_i \in \Lambda_i. \end{array} \tag{2.12}$$

2.3.4 Espaces d'approximation discrets

Soit

$$\mathbf{Z_{h,i}} \times N_{h,i} \times \Lambda_{h,i} \subset \mathbf{Z_i} \times N_i \times \Lambda_i, \quad i \in I_n,$$

l'approximation habituelle par les éléments finis mixtes de plus bas degré. L'espace $\mathbf{Z_{h,i}}$ est l'espace de Raviart-Thomas (et Nédélec en 3D) d'ordre le plus bas ([82], [83], [79]), l'espace $N_{h,i}$ est celui des fonctions constantes par maille et $\Lambda_{h,i}$ est celui des multiplicateurs de Lagrange sur Σ_i, [23]. Nous rappelons que $Z_{h,i}$ est constitué de certaines fonctions vectorielles linéaires et que $\Lambda_{h,i}$ est fait de fonctions scalaires constantes par face en 3D (constantes par côté en 2D). Soient

$$\mathbf{Z_h} = \bigoplus_{i \in I_n} \mathbf{Z_{h,i}}, \qquad N_h = \bigoplus_{i \in I_n} N_{h,i}, \qquad \Lambda_h = \bigoplus_{i \in I_n} \Lambda_{h,i}.$$

Nous rappelons également que

$$\operatorname{div}\mathbf{Z_{h,i}} = N_{h,i} = P_0(\mathcal{T}_i^h) \qquad \Lambda_{h,i} = \mathbf{Z_{h,i}} \cdot \nu_\mathbf{i}|_{\Sigma_i} = P_0(\mathcal{S}_i^h), \quad i \in I_n,$$

où l'on a introduit $P_0(\Xi^h)$, l'espace des fonctions constantes par élément du maillage Ξ^h. En particulier, $P_0(\mathcal{S}_i^h)$ est l'espace des fonctions constantes par face ($d = 3$), ou par côté ($d = 2$), sur $\mathcal{S}_{ij}^h$.

On remarque que l'espace Λ_h contient deux inconnues par interface Σ_{ij}, $i \in I_n, j \in \mathcal{N}_i$: celle qui vit dans $P_0(\mathcal{S}_{ij}^h)$, et celle qui vit dans le maillage d'interface voisin $P_0(\mathcal{S}_{ji}^h) = P_0(\tilde{\mathcal{S}_{ij}^h})$. En procédant de la même façon que dans la section 2.2.4 pour les maillages d'interface, nous allons introduire deux espaces d'approximation différents sur l'interface géométrique entre deux sous-domaines Σ_{ij} : $\Lambda_{h,ij}$ et $\tilde{\Lambda}_{h,ij} = \Lambda_{h,ji}$. L'espace $\Lambda_{h,ij}$ s'appuie sur le maillage $\mathcal{S}_{ij}^h$ hérité du sous-domaine Ω_i, tandis que $\tilde{\Lambda}_{h,ij} = \Lambda_{h,ji}$ s'appuie sur le maillage $\tilde{\mathcal{S}_{ij}^h}$ de son voisin Ω_j.

$$
\begin{aligned}
\Lambda_{h,ij} &= \mathbf{Z_{h,i}} \cdot \nu_{\mathbf{i}}|_{\Sigma_{ij}} = P_0(\mathcal{S}_{ij}^h), \quad i \in I_n,\ j \in \mathcal{N}_i \quad (\text{i.e. } (i,j) \in \mathcal{N}),\\
\tilde{\Lambda}_{h,ij} = \Lambda_{h,ji} &= \mathbf{Z_{h,j}} \cdot \nu_{\mathbf{j}}|_{\Sigma_{ij}} = P_0(\mathcal{S}_{ji}^h), \quad j \in I_n,\ i \in \mathcal{N}_j.
\end{aligned}
\tag{2.13}
$$

Nous définissons à présent les espaces $\tilde{\Lambda}_{h,i}$ qui contiennent les fonctions vivant sur les maillages d'interface $\tilde{\mathcal{S}_i^h}$ des voisins de Ω_i, $i \in I_n$. Nous écrivons

$$
\begin{aligned}
\Lambda_{h,i} &= \bigoplus_{j \in \mathcal{N}_i} \Lambda_{h,ij} &&= \mathbf{Z_{h,i}} \cdot \nu_{\mathbf{i}}|_{\Sigma_i} = P_0(\mathcal{S}_i^h), \qquad i \in I_n,\\
\tilde{\Lambda}_{h,i} &= \bigoplus_{j \in \mathcal{N}_i} \tilde{\Lambda}_{h,ij} = \bigoplus_{j \in \mathcal{N}_i} \Lambda_{h,ji} &&= \mathbf{Z_{h,j}} \cdot \nu_{\mathbf{j}}|_{\Sigma_i} = P_0(\tilde{\mathcal{S}_i^h}), \qquad i \in I_n,
\end{aligned}
\tag{2.14}
$$

et sur les structures internes et externes

$$
\begin{aligned}
\Lambda_h &= \bigoplus_{i \in I_n} \Lambda_{h,i} = \bigoplus_{i \in I_n} \bigoplus_{j \in \mathcal{N}_i} \Lambda_{h,ij} = \bigoplus_{(i,j) \in \mathcal{N}} \Lambda_{h,ij},\\
\tilde{\Lambda}_h &= \bigoplus_{i \in I_n} \tilde{\Lambda}_{h,i} = \bigoplus_{i \in I_n} \bigoplus_{j \in \mathcal{N}_i} \tilde{\Lambda}_{h,ij} = \bigoplus_{i \in I_n} \bigoplus_{j \in \mathcal{N}_i} \Lambda_{h,ji} = \bigoplus_{(i,j) \in s(\mathcal{N})} \Lambda_{h,ji}.
\end{aligned}
\tag{2.15}
$$

2.3.5 Problème multibloc mixte discret

Ceci posé, nous pouvons écrire la formulation mixte faible du problème original (2.2) et, de manière équivalente, du problème de transmission (2.12) faisant intervenir les multiplicateurs entre éléments $\lambda_{h,i}$, voir [36], [13] et leurs références.

Nous cherchons $(\mathbf{u_h}, p_h, \lambda_h) \in \mathbf{Z_h} \times N_h \times \Lambda_h$ tels que

$$
\begin{aligned}
(\mathbf{K^{-1}u_h},\, \mathbf{v})_i - (\operatorname{div} \mathbf{v},\, p_h)_i &= -\langle \mathbf{v} \cdot \nu_{\mathbf{i}},\, \lambda_{h,i} \rangle_i\,, && \mathbf{v} \in \mathbf{Z_{h,i}},\\
(\operatorname{div} \mathbf{u_h},\, r)_i + (cp_h,\, r)_i &= (q_i,\, r)_i\,, && r \in N_{h,i},\\
\sum_{j \in \mathcal{N}_i} \langle -\mathbf{u_{h,i}} \cdot \nu_{\mathbf{i}} + \alpha_{ij}\lambda_{h,i},\, \mu_i \rangle_{ij} &= \sum_{j \in \mathcal{N}_i} \langle \mathbf{u_{h,j}} \cdot \nu_{\mathbf{j}} + \alpha_{ij}\lambda_{h,j},\, \mu_i \rangle_{ij}\,, && \mu_i \in \Lambda_{h,i}.
\end{aligned}
\tag{2.16}
$$

La dernière équation dans (2.16) peut être reformulée de façon équivalente en une condition qui exprime les sauts de pression en fonction des sauts de vitesse normale,

$$\sum_{j\in\mathcal{N}_i} \langle \alpha_{ij}(\lambda_{h,i}-\lambda_{h,j}),\, \mu_i\rangle_{ij} \;=\; \sum_{j\in\mathcal{N}_i} \langle \mathbf{u_{h,i}}\cdot\nu_\mathbf{i}+\mathbf{u_{h,j}}\cdot\nu_\mathbf{j},\, \mu_i\rangle_{ij}\,, \quad \mu_i\in\Lambda_{h,i}. \tag{2.17}$$

On introduit à présent une nouvelle notation pour les sauts de variable scalaire μ et les sauts de variable vectorielle normale $\mathbf{v_j}\cdot\nu_\mathbf{j}$ vivant sur une interface Σ_{ij}

$$[|\mu|]_{\Sigma_{ij}} \;=\; \mu_i-\mu_j, \quad [|\mathbf{v}\cdot\nu|]_{\Sigma_{ij}} \;=\; \mathbf{v_i}\cdot\nu_\mathbf{i}+\mathbf{v_j}\cdot\nu_\mathbf{j}, \quad (i,j)\in I_n. \tag{2.18}$$

Remarque 2. La formulation mixte (2.16) est analysée par Arbogast et Yotov dans [14]. Il y est démontré que le problème est bien posé et des estimations d'erreur sont données. Il y est également mis en avant que la conservation de masse locale est assurée partout sauf aux interfaces dont les maillages ne sont pas raccordés. La non-conservativité est le prix à payer si l'on veut travailler avec des maillages non-conformes. Cependant même si le flux peut ne pas être conservé à travers une portion quelconque de Σ, comme les fonctions-test $\mu_i\equiv 1$ appartiennent aux espaces $\Lambda_{h,i},\ i\in I_n$, la masse reste conservée globalement.

Une analyse de la discrétisation par volumes finis est également faite dans [2]. Des estimations d'erreurs sont démontrées avec une hypothèse assez forte de régularité du maillage.

2.3.6 Existence et estimation d'énergie

Nous suivons le schéma de la démonstration donnée par Arbogast et Yotov dans [14]. La seule différence est la généralisation à des coefficients de Robin variables selon les sous-domaines : dans [14], les calculs sont réalisés avec un seul coefficient de Robin (i.e. $\alpha_{ij}=\alpha,\ (i,j)\in\mathcal{N}$), tandis qu'ici tous les coefficients peuvent être différents : a priori $\alpha_{ij}\neq\alpha_{kl},\ (i,j),(k,l)\in\mathcal{N}$ et en particulier $\alpha_{ij}\neq\alpha_{ji},\ (i,j)\in\mathcal{N}$. Cette extension ne pose pas de difficulté importante : il suffit de trouver la fonction test adéquate pour les estimations d'énergie. Cette modification des fonctions tests doit cependant être réalisée avec précaution : au lieu d'utiliser la fonction test de [14], qui s'écrit $\mu_i|_{\Sigma_{ij}}=\frac{1}{2}(\lambda_{h,i}-1/\alpha\,(\mathbf{u_{h,i}}\cdot\nu_\mathbf{i}))\in\Lambda_{h,ij}$, nous devons prendre une fonction test pour chaque interface qui dépend des coefficients de Robin locaux. Nous avons pu mener les calculs avec

$$\mu_i|_{\Sigma_{ij}}=\frac{1}{\alpha_{ij}+\alpha_{ji}}\left[\alpha_{ji}\lambda_{h,i}-\mathbf{u_{h,i}}\cdot\nu_\mathbf{i}\right] \qquad \in\Lambda_{h,ij}. \tag{2.19}$$

Comme il a été remarqué dans [68], et confirmé dans [84], voir la Section 2.3.8, il peut être important de pouvoir choisir des coefficients différents selon les sous-domaines. Des analyses ont été réalisées qui montrent l'utilité de choix pertinents de ce coefficient de Robin, en particulier pour les problèmes de convection-diffusion, voir par exemple [3], [1], [5], ou plus récemment [87], [78].

On utilisera plusieurs fois le Lemme 1 qui se vérifie sans peine.

Lemme 1. *Soient trois suites réelles* $a_i,\ b_i,\ i \in I_n$*, et* $c_{ij},\ i,j \in I_n$ *alors*

$$\begin{aligned}
\sum_{i,j\in I_n} (a_i + a_j)\, c_{ij}\, b_i &= \frac{1}{2} \sum_{i,j\in I_n} (a_i + a_j)(c_{ij}\, b_i + c_{ji}\, b_j) \\
&= \frac{1}{2} \sum_{i,j\in I_n} (a_i + a_j)\, c_{ij}\, (b_i + b_j) \qquad si\ c_{ij} = c_{ji} \quad (2.20) \\
\sum_{i,j\in I_n} (a_i - a_j)\, c_{ij}\, b_i &= \frac{1}{2} \sum_{i,j\in I_n} (a_i - a_j)(c_{ij}\, b_i - c_{ji}\, b_j) \\
&= \frac{1}{2} \sum_{i,j\in I_n} (a_i - a_j)\, c_{ij}\, (b_i - b_j) \qquad si\ c_{ij} = c_{ji} \quad (2.21) \\
\sum_{i,j\in I_n} (a_i + a_j)\, c_{ij}\, b_i &= \sum_{i,j\in I_n} a_i\, (c_{ij}\, b_i + c_{ji}\, b_j) \qquad (2.22) \\
\sum_{i,j\in I_n} (a_i - a_j)\, c_{ij}\, b_i &= \sum_{i,j\in I_n} a_i\, (c_{ij}\, b_i - c_{ji}\, b_j) \qquad (2.23)
\end{aligned}$$

Dans les première et seconde équations de (2.16), on prend comme fonctions tests $\mathbf{v} = \mathbf{u_h} \in \mathbf{Z_{h,i}}$ et $r = p_h \in N_{h,i}$, puis on les additionne entre elles. Finalement après élimination des termes en divergence et somme pour $i \in I_n$, on obtient :

$$\left(\mathbf{K^{-1}u_h},\, \mathbf{u_h}\right) + (cp_h,\, p_h) = (q,\, p_h) - \sum_{i\in I_n} \langle \mathbf{u_{h,i}} \cdot \nu_\mathbf{i},\, \lambda_{h,i} \rangle_i \,.$$

Ensuite, on applique la fonction test 2.19 dans les équations sur l'interface (2.17). Dans un souci de clarté, nous découpons la fonction test en deux. Tout d'abord, prenons $\mu_i|_{\Sigma_{ij}} = \alpha_{ji}/(\alpha_{ij}+\alpha_{ji})\ \lambda_{h,i} \in \Lambda_{h,i}$, et sommons sur $i \in I_n$. On rappelle que le produit scalaire est identiquement nul sur une interface vide, c'est-à-dire si $\Sigma_{ij} = \emptyset$. On utilise le Lemme 1 —l'équation (2.21) s'applique car l'expression $(\alpha_{ij}\alpha_{ji})/(\alpha_{ij} + \alpha_{ji})$ est symétrique en (i,j)— pour écrire

$$\frac{1}{2} \sum_{i,j\in I_n} \left\langle \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij} + \alpha_{ji}} [|\lambda_h|]_{\Sigma_{ij}},\, [|\lambda_h|]_{\Sigma_{ij}} \right\rangle_{ij} = \sum_{i,j\in I_n} \left\langle [|\mathbf{u_h} \cdot \nu|]_{\Sigma_{ij}},\, \frac{\alpha_{ji}}{\alpha_{ij} + \alpha_{ji}} \lambda_{h,i} \right\rangle_{ij} .$$

Dans un deuxième temps, prenons l'autre moitié de la fonction test de 2.19 : la fonction μ_i telle que $\mu_i|_{\Sigma_{ij}} = -1/(\alpha_{ij}+\alpha_{ji})\ (\mathbf{u_{h,i}} \cdot \nu_\mathbf{i}) \in \Lambda_{h,ij}$ et sommons encore pour $i \in I_n$. On invoque deux fois le Lemme 1 —une fois l'équation (2.21) pour le membre de droite avec l'expression $1/(\alpha_{ij}+\alpha_{ji})$ qui est symétrique en (i,j), et une fois l'équation (2.23) pour le membre de gauche—, pour obtenir finalement

$$-\sum_{i,\,j\in I_n}\left\langle \lambda_{h,i},\, \frac{\alpha_{ij}\mathbf{u_{h,i}}\cdot\nu_\mathbf{i} - \alpha_{ji}\mathbf{u_{h,j}}\cdot\nu_\mathbf{j}}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij} = -\frac{1}{2}\sum_{i,\,j\in I_n}\left\langle [|\mathbf{u_h}\cdot\nu|]_{\Sigma_{ij}},\, \frac{[|\mathbf{u_h}\cdot\nu|]_{\Sigma_{ij}}}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}.$$

Il ne reste plus qu'à sommer ces trois équations, éliminer certains termes afin d'écrire

$$\begin{aligned}\left(\mathbf{K^{-1}u_h},\, \mathbf{u_h}\right) + (cp_h,\, p_h) + \frac{1}{2}\sum_{i,\,j\in I_n}\left\langle \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}}[|\lambda_h|]_{\Sigma_{ij}},\, [|\lambda_h|]_{\Sigma_{ij}}\right\rangle_{ij}\\ +\frac{1}{2}\sum_{i,\,j\in I_n}\left\langle [|\mathbf{u_h}\cdot\nu|]_{\Sigma_{ij}},\, \frac{[|\mathbf{u_h}\cdot\nu|]_{\Sigma_{ij}}}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij} = (q,\, p_h)\,.\end{aligned}$$

Tous les termes du membre de gauche sont positifs. Les deux premiers sont minorés par la norme L^2 de la pression et de la vitesse, grâce aux hypothèses de positivité sur les coefficients. Les deux autres termes représentent des normes pondérées par les α_{ij} de l'erreur faite sur les pressions et les vitesses normales aux interfaces. Cette erreur est due au fait que l'on n'impose que faiblement la continuité de la condition de Robin sur les interfaces. On peut remarquer que le facteur $\frac{1}{2}$ devant les sommations provient du fait que ces sommations font intervenir *deux fois* chaque interface Σ_{ij}. D'autre part, quand les coefficients de Robin sont tous égaux, $\alpha_{ij} = \alpha,\ (i,j) \in \mathcal{N}$, nous retombons sur l'équation (16) de [14].

Il ne reste qu'à maîtriser le terme $(q,\, p_h)$ en faisant la remarque générale que pour tout $\varepsilon > 0$, si a et b vivent dans $L^2(\Omega)$, alors

$$(a,\, b) \leq \frac{1}{2}\left[\varepsilon||a||^2 + \frac{1}{\varepsilon}||b||^2\right]. \tag{2.24}$$

Finalement en utilisant les notations introduites dans la Section 2.3.3, nous obtenons le

Théorème 1. *Soit une famille de coefficients de Robin $\alpha_{ij} > 0,\ (i,j) \in \mathcal{N}$,* pouvant être tous différents *: a priori $\alpha_{ij} \neq \alpha_{kl},\ (i,j),(k,l) \in \mathcal{N}$ et en particulier $\alpha_{ij} \neq \alpha_{ji},\ (i,j) \in \mathcal{N}$. Si les perméabilités $\mathbf{K}$ et la compressibilité c sont bornées uniformément positivement, alors il existe une constante C indépendante de h*

et des α_{ij}, $(i,j) \in \mathcal{N}$ telle que la solution du problème 2.16 vérifie l'estimation d'énergie

$$||p_h|| + ||\mathbf{u_h}|| + \left[\sum_{(i,j)\in\mathcal{N}} \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}} |||\lambda_{h,i} - \lambda_{h,j}|||_{ij}^2 \right]^{1/2} + \left[\sum_{(i,j)\in\mathcal{N}} \frac{1}{\alpha_{ij}+\alpha_{ji}} |||\mathbf{u_{h,i}}\cdot\nu_\mathbf{i} + \mathbf{u_{h,j}}\cdot\nu_\mathbf{j}|||_{ij}^2 \right]^{1/2} \leq C\ ||q||.$$

Remarque 3. Ce résultat est analogue à celui de [14], mais étendu au cas des coefficients de Robin variables par interfaces. D'ailleurs, sans difficulté supplémentaire, nous obtiendrions comme dans [14] les mêmes résultats dans le cas d'éléments finis d'ordres plus élevés, [23]. Cette remarque reste valable pour la démonstration des estimations d'erreur de la Section 2.3.7 qui suit. Les estimations obtenues seraient alors identiques, mais avec un ordre plus élevé.

Par ailleurs, nous précisons que, par manque de temps, nous n'avons pas cherché à faire d'expérimentations numériques avec des coefficients de Robin différents sur une même interface, c'est-à-dire tels que $\alpha_{ij} \neq \alpha_{ji}, (i,j) \in \mathcal{N}$. En revanche, nous avons utilisé abondamment la possibilité de faire varier les coefficients de Robin interface par interface.

En dimension finie, l'injectivité d'une matrice carrée suffit à établir son inversibilité. Le système linéaire (2.16) est carré, donc on étudie le problème homogène, c'est-à-dire avec un terme source nul $q = 0$. Le théorème précédent nous indique immédiatement que p_h et $\mathbf{u_h}$ sont nuls. La nullité des $\lambda_i,\ i \in I_n$ provient de la première équation de (2.16), car $\mathbf{Z_{h,i}} \cdot \nu_\mathbf{i}|_{\Sigma_i} = \Lambda_{h,i}$. Ceci prouve le

Corollaire 1. *Sous les mêmes hypothèses de coercivité que le Théorème 1, et avec la même généralité concernant les coefficients de Robin, le problème (2.16) admet une unique solution.*

2.3.7 Estimations d'erreur

Suivant à nouveau les pas de [14], nous donnons dans ce paragraphe la démonstration des estimations d'erreur pour le problème (2.16), dans le cas des espaces de Raviart-Thomas de plus bas degré[8], et avec la généralisation à des coefficients de Robin α_{ij} variables par interface de sous-domaine.

[8]Cette restriction n'est pas réellement nécessaire. Voir la Remarque 3.

Il existe $\Pi_{h,i}$ un opérateur de projection sur $\mathbf{Z_{h,i}}$ vérifiant pour un $\mathbf{v} \in (\mathcal{H}^{1/2+\varepsilon}(\Omega))^d \cap \mathbf{Z_i}$

$$\begin{aligned} (\mathrm{div}\,(\mathbf{v} - \Pi_{h,i}\mathbf{v}),\ r)_i &= 0, \qquad r \in N_{h,i}, & (2.25)\\ \langle(\mathbf{v} - \Pi_{h,i}\mathbf{v}) \cdot \nu_i,\ \mathbf{w} \cdot \nu_i\rangle_i &= 0, \qquad \mathbf{w} \in \mathbf{Z_{h,i}}, & (2.26)\\ ||\mathbf{v} - \Pi_{h,i}\mathbf{v}||_i &\leq Ch||\mathbf{v}||_i, & (2.27)\\ ||\mathrm{div}\,(\mathbf{v} - \Pi_{h,i}\mathbf{v})||_i &\leq Ch||\mathrm{div}\,\mathbf{v}||_i, & (2.28) \end{aligned}$$

où C représente différentes constantes indépendantes de h. Pour un $i \in I_n$, nous avons besoin de la projection L^2 d'une fonction $f \in L^2(\Omega)$ notée $\hat{f} \in N_h$ définie par

$$\left(f - \hat{f},\ r\right) = 0, \qquad r \in N_h, \tag{2.29}$$

vérifiant l'estimation

$$||f - \hat{f}|| \leq Ch. \tag{2.30}$$

De la même manière, la projection L^2 d'une fonction $f_i \in L^2(\Sigma_i)$ est notée $\bar{f}_i \in \Lambda_{h,i}$. [9] Elle est définie par

$$\left\langle f_i - \bar{f}_i,\ \mu_i\right\rangle_i = 0, \qquad \mu_i \in \Lambda_{h,i}, \tag{2.31}$$

et vérifie la majoration

$$||f - \bar{f}_i||_i \leq Ch, \tag{2.32}$$

dont on déduit l'estimation (2.33), si $\mathbf{v} \cdot \nu_i \in L^2(\Sigma_i)$, car alors $\Pi_{h,i}\mathbf{v} \cdot \nu_i = \overline{\mathbf{v} \cdot \nu_i}$,

$$||(\mathbf{v} - \Pi_{h,i}\mathbf{v}) \cdot \nu_i||_i \leq Ch. \tag{2.33}$$

L'opérateur de projection L^2 sur les interfaces sera précisé plus loin (voir l'équation (2.54)) à partir des projections sur les interfaces Σ_{ij} entre deux sous-domaines. De plus un algorithme menant à une implémentation de ces projections est précisé dans la Section 2.4.

Afin de raccourcir les écritures, nous notons les erreurs

$$\varphi = p - p_h, \qquad \psi = \mathbf{u} - \mathbf{u_h}, \quad \text{et} \quad \phi_i = p_i - \lambda_{h,i}.$$

Nous omettons ici et dans la suite les indices h qui alourdiraient les notations sans gain de clarté.

[9] Aucune confusion avec les conditions aux limites de Dirichlet, notées $\overline{p}$, n'est possible puisque, dans cette section, ces conditions aux limites sont supposées nulles.

La soustraction du système discret (2.16) à (2.12), puis l'utilisation de (2.29) et (2.25) pour projeter φ et ψ, donnent

$$\begin{array}{rcll}(\mathbf{K^{-1}}\psi,\, \mathbf{v})_i - (\operatorname{div} \mathbf{v},\, \hat{\varphi})_i & = & -\left\langle \mathbf{v}\cdot\nu_{\mathbf{i}},\, \phi_i\right\rangle_i, & \mathbf{v}\in \mathbf{Z_{h,i}},\\ (\operatorname{div} \Pi_{h,i}\psi,\, r)_i + (c\varphi,\, r)_i & = & 0, & r\in N_{h,i},\\ \displaystyle\sum_{j\in\mathcal{N}_i} \left\langle \alpha_{ij}(\phi_i-\phi_j),\, \mu_i\right\rangle_{ij} & = & \displaystyle\sum_{j\in\mathcal{N}_i} \left\langle \psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}} + \psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\, \mu_i\right\rangle_{ij}, & \mu_i\in\Lambda_{h,i}.\end{array} \tag{2.34}$$

Nous ferons appel fréquemment dans la suite aux identités (2.35)

$$\begin{array}{rclcl}\Pi_{h,i}\psi & = & \psi + (\Pi_{h,i}\psi - \psi) & = & \psi + (\Pi_{h,i}\mathbf{u} - \mathbf{u}),\\ \hat{\varphi} & = & \varphi + (\hat{\varphi}-\varphi) & = & \varphi + (\hat{p}-p),\\ \bar{\phi}_i & = & \phi_i + (\bar{\phi}_i - \phi_i) & = & \phi_i + (\bar{p}_i - p_i).\end{array} \tag{2.35}$$

On prend comme fonctions test $\mathbf{v} = \Pi_{h,i}\psi \in \mathbf{Z_{h,i}}$ et $r = \hat{\varphi} \in N_{h,i}$ pour les deux premières équations de (2.34), puis on les additionne entre elles avant de sommer pour $i \in I_n$, et on utilise deux des identités (2.35), afin d'écrire

$$\begin{array}{rcl}(\mathbf{K^{-1}}\psi,\, \psi) + (c\varphi,\, \varphi) & = & \displaystyle\sum_{i\in I_n} \left(\mathbf{K^{-1}}\psi,\, \mathbf{u} - \Pi_{h,i}\mathbf{u}\right)_i + (c\varphi,\, p-\hat{p})\\ & & \displaystyle -\sum_{i\in I_n} \left\langle \Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}},\, \phi_i\right\rangle_i.\end{array} \tag{2.36}$$

Les termes du membre de gauche dans (2.36) sont—aux coefficients près—les normes des erreurs. Les deux premiers termes du membre de droite sont contrôlés en utilisant l'inéquation (2.24) grâce à (2.30) et (2.27). Il ne reste qu'à maîtriser le dernier terme en faisant intervenir les conditions d'interface. Pour cela, prenons une fonction test de type Robin μ_i telle que

$$\mu_i|_{\Sigma_{ij}} = \frac{1}{\alpha_{ij}+\alpha_{ji}} \left[\alpha_{ji}\bar{\phi}_i - \Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i\right] \qquad \in \Lambda_{h,ij}$$

dans la dernière équation de (2.34). A nouveau, pour la lisibilité, nous séparons la fonction test en deux parties. Premièrement, pour un $\mu_i = \alpha_{ji}/(\alpha_{ij}+\alpha_{ji})\, \bar{\phi}_i$, on fait la sommation sur $i \in I_n$ et on applique deux fois le Lemme 1 : une fois l'équation (2.21) à gauche, et une fois l'équation (2.22) à droite. Ceci donne

$$\frac{1}{2}\sum_{i,j\in I_n} \left\langle \phi_i - \phi_j,\, \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}}(\bar{\phi}_i - \bar{\phi}_j)\right\rangle_{ij} = \sum_{i,j\in I_n}\left\langle \psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}},\, \frac{\alpha_{ji}\bar{\phi}_i + \alpha_{ij}\bar{\phi}_j}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}. \tag{2.37}$$

On a l'égalité suivante d'après les propriétés sur les projections (2.26) et (2.31) :

$$\left\langle \psi\cdot\nu_i,\, \bar{\phi}_i\right\rangle_{ij} = \left\langle \Pi_{h,i}\psi\cdot\nu_i,\, \phi_i\right\rangle_{ij},$$

qui, en invoquant la troisième identité de (2.35), nous permet d'écrire (2.37) sous la forme

$$\begin{aligned}\frac{1}{2}\sum_{i,j\in I_n}&\left\langle \phi_i-\phi_j,\,\frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}}(\phi_i-\phi_j)\right\rangle_{ij}\\ &=\sum_{i,j\in I_n}\left\langle \Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}},\,\frac{\alpha_{ji}\phi_i}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}+\sum_{i,j\in I_n}\left\langle \psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}},\,\frac{\alpha_{ij}\bar{\phi}_j}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}\\ &\quad+\frac{1}{2}\sum_{i,j\in I_n}\left\langle \phi_i-\phi_j,\,\frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}}[(p_i-\bar{p}_i)-(p_j-\bar{p}_j)]\right\rangle_{ij}.\end{aligned}\tag{2.38}$$

Deuxièmement, nous prenons $\mu_i|_{\Sigma_{ij}}=-1/(\alpha_{ij}+\alpha_{ji})\ [\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i]$ dans la dernière équation de (2.34). Encore une fois, nous sommons sur $i\in I_n$ et utilisons le Lemme 1 —plus précisément l'équation (2.20)— une fois à droite pour obtenir

$$\begin{aligned}-\sum_{i,j\in I_n}&\left\langle \phi_i-\phi_j,\,\frac{\alpha_{ij}\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}\\ &=-\frac{1}{2}\sum_{i,j\in I_n}\left\langle \psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\,\frac{\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i+\Pi_{h,j}\psi_{\mathbf{j}}\cdot\nu_j}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}.\end{aligned}\tag{2.39}$$

De plus, les définitions des projections (2.26) et (2.31) permettent d'établir

$$\left\langle(\mathbf{u}-\Pi_{h,j}\mathbf{u})\cdot\nu_{\mathbf{j}},\,\bar{\phi}_j\right\rangle_{ij}=0,\qquad \left\langle\Pi_{h,j}\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\,p_j-\bar{p}_j\right\rangle_{ij}=0.$$

Ceci, ajouté aux identités (2.35) implique les égalités

$$\begin{aligned}\left\langle\phi_j,\,\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i\right\rangle_{ij} &= \left\langle\psi_{\mathbf{i}}\cdot\nu_i,\,\bar{\phi}_j\right\rangle_{ij}-\left\langle(\mathbf{u}-\Pi_{h,i}\mathbf{u})\cdot\nu_{\mathbf{i}},\,\bar{\phi}_j\right\rangle_{ij}+\left\langle\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}},\,p_j-\bar{p}_j\right\rangle_{ij}\\ &= \left\langle\psi_{\mathbf{i}}\cdot\nu_i,\,\bar{\phi}_j\right\rangle_{ij}+\left\langle(\mathbf{u}-\Pi_{h,i}\mathbf{u})\cdot\nu_{\mathbf{i}},\,\bar{\phi}_i-\bar{\phi}_j\right\rangle_{ij}\\ &\quad+\left\langle\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\Pi_{h,j}\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\,p_j-\bar{p}_j\right\rangle_{ij},\end{aligned}$$

qui à leur tour, multipliées par le facteur $\alpha_{ij}/(\alpha_{ij}+\alpha_{ji})$ et en utilisant encore (2.35), permettent de transformer (2.39) en

$$\begin{aligned}\frac{1}{2}\sum_{i,j\in I_n}&\left\langle\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\,\frac{\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}}}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}\\ &=\frac{1}{2}\sum_{i,j\in I_n}\left\langle[|\psi\cdot\nu|],\,\frac{[|(\mathbf{u}-\Pi_h\mathbf{u})\cdot\nu|]}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}+\sum_{i,j\in I_n}\left\langle\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_i,\,\frac{\alpha_{ij}\phi_i}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}\\ &\quad-\sum_{i,j\in I_n}\left\langle\psi_{\mathbf{i}}\cdot\nu_i,\,\frac{\alpha_{ij}\bar{\phi}_j}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}-\sum_{i,j\in I_n}\left\langle(\mathbf{u}-\Pi_{h,i}\mathbf{u})\cdot\nu_{\mathbf{i}},\,\frac{\alpha_{ij}(\bar{\phi}_i-\bar{\phi}_j)}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}\\ &\quad-\sum_{i,j\in I_n}\left\langle\Pi_{h,i}\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\Pi_{h,j}\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}},\,\frac{\alpha_{ij}(p_j-\bar{p}_j)}{\alpha_{ij}+\alpha_{ji}}\right\rangle_{ij}.\end{aligned}\tag{2.40}$$

Nous regroupons les équations (2.36), (2.38) et (2.40). Cinq termes s'éliminent : trois termes en $\langle \Pi_{h,i}\psi_{\mathbf{i}} \cdot \nu_i,\, \phi_i \rangle_{ij}$ et deux en $\left\langle \psi_{\mathbf{i}} \cdot \nu_i,\, \frac{\alpha_{ij}\bar{\phi}_j}{\alpha_{ij}+\alpha_{ji}} \right\rangle_{ij}$. Tous les termes restants sont facilement contrôlés grâce à la coercivité et à la majoration (2.24), sauf les deux derniers termes de (2.40). Préoccupons-nous donc de l'avant dernier terme de (2.40) que nous appellerons A et du dernier terme de (2.40) que nous nommerons B. Pour A et B, il faut utiliser encore une fois les équations (2.35) afin de faire apparaître respectivement $(\phi_i - \phi_j)$ et $\psi_{\mathbf{i}} \cdot \nu_{\mathbf{i}} + \psi_{\mathbf{j}} \cdot \nu_{\mathbf{j}}$. Commençons par A :

$$
\begin{aligned}
A &:= -\sum_{i,j \in I_n} \left\langle (\mathbf{u} - \Pi_{h,i}\mathbf{u}) \cdot \nu_{\mathbf{i}},\, \frac{\alpha_{ij}(\bar{\phi}_i - \bar{\phi}_j)}{\alpha_{ij} + \alpha_{ji}} \right\rangle_{ij} \\
&= -\sum_{i,j \in I_n} \left\langle \frac{(\mathbf{u} - \Pi_{h,i}\mathbf{u}) \cdot \nu_{\mathbf{i}}}{\sqrt{\alpha_{ij} + \alpha_{ji}}},\, \frac{\alpha_{ij}(\phi_i - \phi_j)}{\sqrt{\alpha_{ij} + \alpha_{ji}}} \right\rangle_{ij} \\
&\quad - \sum_{i,j \in I_n} \left\langle \frac{(\mathbf{u} - \Pi_{h,i}\mathbf{u}) \cdot \nu_{\mathbf{i}}}{\sqrt{\alpha_{ij} + \alpha_{ji}}},\, \frac{\alpha_{ij}[(\bar{p}_i - p_i) - (\bar{p}_j - p_j)]}{\sqrt{\alpha_{ij} + \alpha_{ji}}} \right\rangle_{ij} .
\end{aligned}
\tag{2.41}
$$

Dans (2.41), la valeur absolue de la deuxième somme se majore facilement, en utilisant la majoration (2.24) avec $\varepsilon = 1$, par

$$
\frac{1}{2} \sum_{i,j \in I_n} \left\{ \frac{|||(\mathbf{u} - \Pi_{h,i}\mathbf{u}) \cdot \nu_i|||_{ij}^2}{\alpha_{ij} + \alpha_{ji}} + 2\frac{\alpha_{ij}^2 + \alpha_{ji}^2}{\alpha_{ij} + \alpha_{ji}} |||p_i - \bar{p}_i|||_{ij}^2 \right\} .
$$

La première somme dans (2.41) doit être traitée avec plus de précautions, de façon à maîtriser le terme contenant une norme de l'erreur $(\phi_i - \phi_j)$. Nous invoquons à nouveau la majoration (2.24), en prenant cette fois $\varepsilon = \dfrac{\eta\alpha_{ji}}{\alpha_{ij}}$, avec $\eta > 0$, pour majorer la valeur absolue de la première somme dans (2.41) par

$$
\frac{1}{2} \sum_{i,j \in I_n} \left\{ \frac{1}{\eta} \frac{\alpha_{ij}\, |||(\mathbf{u} - \Pi_{h,i}\mathbf{u}) \cdot \nu_i|||_{ij}^2}{\alpha_{ji}(\alpha_{ij} + \alpha_{ji})} + \eta \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij} + \alpha_{ji}} |||\phi_i - \phi_j|||_{ij}^2 \right\} .
$$

Passons maintenant au terme B qui s'écrit, en utilisant (2.35),

$$
\begin{aligned}
B &:= -\sum_{i,j \in I_n} \left\langle \Pi_{h,i}\psi_{\mathbf{i}} \cdot \nu_{\mathbf{i}} + \Pi_{h,j}\psi_{\mathbf{j}} \cdot \nu_{\mathbf{j}},\, \frac{\alpha_{ij}(p_j - \bar{p}_j)}{\alpha_{ij} + \alpha_{ji}} \right\rangle_{ij} \\
&= -\sum_{i,j \in I_n} \left\langle \frac{\psi_{\mathbf{i}} \cdot \nu_{\mathbf{i}} + \psi_{\mathbf{j}} \cdot \nu_{\mathbf{j}}}{\sqrt{\alpha_{ij} + \alpha_{ji}}},\, \frac{\alpha_{ij}(p_j - \bar{p}_j)}{\sqrt{\alpha_{ij} + \alpha_{ji}}} \right\rangle_{ij} \\
&\quad - \sum_{i,j \in I_n} \left\langle \frac{[|(\Pi_h\mathbf{u} - \mathbf{u}) \cdot \nu|]}{\sqrt{\alpha_{ij} + \alpha_{ji}}},\, \frac{\alpha_{ij}(p_j - \bar{p}_j)}{\sqrt{\alpha_{ij} + \alpha_{ji}}} \right\rangle_{ij} .
\end{aligned}
\tag{2.42}
$$

Nous majorons la valeur absolue de la deuxième somme dans (2.42) à l'aide de la majoration (2.24) avec $\varepsilon = 1$, par

$$\frac{1}{2}\sum_{i,j\in I_n}\left\{4\frac{|||(\mathbf{u}-\Pi_{h,i}\mathbf{u})\cdot\nu_i|||_{ij}^2}{\alpha_{ij}+\alpha_{ji}}+\frac{\alpha_{ji}^2|||p_i-\bar{p}_i|||_{ij}^2}{\alpha_{ij}+\alpha_{ji}}\right\}.$$

De même, la valeur absolue de la première somme dans (2.42) se majore, grâce à la majoration (2.24) avec $\varepsilon = \eta > 0$, par

$$\frac{1}{2}\sum_{i,j\in I_n}\left\{\eta\frac{|||\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}}|||_{ij}^2}{\alpha_{ij}+\alpha_{ji}}+\frac{1}{\eta}\frac{\alpha_{ji}^2|||p_i-\bar{p}_i|||_{ij}^2}{\alpha_{ij}+\alpha_{ji}}\right\}.$$

Finalement, nous regroupons ces inéquations, et utilisons les arguments de coercivité, la majoration (2.24) et la remarque qu'il existe $C>0$ tel que $a\alpha_{ij}^2 + b\alpha_{ij}\alpha_{ji} + c\alpha_{ji}^2 \leq C(\alpha_{ij}+\alpha_{ji})^2$, pour a, b et $c > 0$, car α_{ij} et α_{ji} sont positifs, pour obtenir l'existence d'une constante $C > 0$ indépendante des $\alpha_{ij}, (i,j) \in \mathcal{N}$ telle que

$$\begin{aligned}
||\varphi||^2 + ||\psi||^2 &+ \sum_{(i,j)\in\mathcal{N}}\left[\frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij}+\alpha_{ji}}|||\phi_i-\phi_j|||_{ij}^2+\frac{|||\psi_{\mathbf{i}}\cdot\nu_{\mathbf{i}}+\psi_{\mathbf{j}}\cdot\nu_{\mathbf{j}}|||_{ij}^2}{\alpha_{ij}+\alpha_{ji}}\right]\\
&\leq C\Bigg(||p-\hat{p}||^2+\sum_{i\in I_n}||\mathbf{u}-\Pi_{h,i}\mathbf{u}||_i^2\\
&\quad+\sum_{(i,j)\in\mathcal{N}}\left[(\alpha_{ij}+\alpha_{ji})|||p_i-\bar{p}_i|||_{ij}^2+\frac{1}{\alpha_{ji}}|||(\mathbf{u}-\Pi_{h,i}\mathbf{u})\cdot\nu_i|||_{ij}^2\right]\Bigg).
\end{aligned} \tag{2.43}$$

Cette estimation permet, en invoquant les majorations (2.27), (2.30), (2.32) et (2.33), de prouver le Théorème 2. La seconde inéquation est obtenue en prenant $r = \operatorname{div}\Pi_{h,i}\psi$ dans la seconde équation de (2.34) et en faisant intervenir encore une fois (2.35).

Théorème 2. *Sous les hypothèses du Théorème 1, et avec la même généralité concernant les coefficients de Robin variables, c'est-à-dire tels que a priori $\alpha_{ij} \neq \alpha_{kl}$, $(i,j),(k,l) \in \mathcal{N}$, il existe une constante C dépendant des α_{ij} mais* *indépendante de h telle que*

$$||p - p_h|| + ||\mathbf{u} - \mathbf{u_h}|| + \left[\sum_{(i,j)\in\mathcal{N}} \frac{\alpha_{ij}\alpha_{ji}}{\alpha_{ij} + \alpha_{ji}} |||\lambda_{h,i} - \lambda_{h,j}|||_{ij}^2 \right]^{1/2} + \left[\sum_{(i,j)\in\mathcal{N}} \frac{|||\mathbf{u_{h,i}} \cdot \nu_\mathbf{i} + \mathbf{u_{h,j}} \cdot \nu_\mathbf{j}|||_{ij}^2}{\alpha_{ij} + \alpha_{ji}} \right]^{1/2} \leq \ Ch,$$

$$\left[\sum_{i\in I_n} ||\mathrm{div}\ (\mathbf{u} - \mathbf{u_{h,i}})||_i^2 \right]^{1/2} \leq \ Ch.$$

Il est possible de généraliser simplement ces estimations avec des éléments d'ordre supérieur, voir [14].

Nous refermons maintenant la parenthèse qui avait été ouverte pour des raisons théoriques à la Section 2.3.3 : nous supprimons dorénavant le terme cp de (2.11) et revenons au problème originel (2.6)-(2.7).

2.3.8 Comment choisir les coefficients de Robin ?

Comme toutes les méthodes laissant un paramètre libre, la méthode de décomposition de domaines de Robin–Robin non-conforme possède une grande qualité et un défaut majeur : elle a un paramètre libre. C'est un atout puisqu'il est théoriquement possible de prendre *le* meilleur paramètre, en l'occurence le coefficient de Robin *optimal* qui permet de converger très rapidement, avec une grande précision ; mais c'est aussi un défaut important, car la plupart du temps, l'utilisateur ne sait jamais quel paramètre choisir pour son application et doit tâtonner pour arriver à un résultat.

Nous tentons donc dans cette section de donner quelques éléments de réponse à la question du choix des coefficients de Robin. Nos expériences numériques ont prouvé qu'il s'avère crucial de choisir correctement les coefficients de Robin dans une simulation où les perméabilités et les dimensions du domaine sont réalistes, voir la Section 3.6.2. Toute tentative plus ou moins naïve de faire converger la méthode sans le “bon” α s'est soldée par un échec cuisant : Bi-CGStab stagne ou diverge. Avant d'essayer de faire sentir le pourquoi de ces échecs, ainsi que le remède que nous avons trouvé, présentons deux idées qui existent dans la littérature.

Tout d'abord, Roux, Magoulès et Series, [84], remarquent très justement que si l'on ajoute un complément de Schur judicieux aux conditions d'interface, leur méthode de FETI à deux multiplicateurs de Lagrange converge en 1 itération quand il y a deux sous-domaines. Ceci est particulièrement agréable, mais impraticable dans la mesure où les matrices de Schur sont pleines et coûteuses à calculer, surtout quand le nombre de sous-domaines augmente. L'idée suivante est alors d'essayer d'approcher le complément de Schur à moindres frais, mais en conservant autant que possible l'accélération. Divers moyens sont testés dans [85] : une première approximation du complément de Schur est une matrice diagonale. Dans ce cas, l'opérateur d'interface peut s'interpréter comme une condition de Robin avec des coefficients de Robin particuliers[10]. Cette approximation n'utilise que des données d'interface et se révèle limitée, car le complément de Schur est essentiellement non-local. En revanche, une approximation utilisant un petit morceau de maillage autour de l'interface —un *patch*— semble donner des résultats satisfaisants.

Dans [78], Nataf part d'un constat similaire : si la condition d'interface dans la méthode de décomposition de domaines contient un opérateur symétrique défini positif plein bien particulier, la convergence est achevée en au plus 2 itérations pour deux sous-domaines. L'auteur essaie donc d'approcher cet opérateur par une matrice diagonale, auquel cas la condition d'interface devient une condition de Robin. L'analyse pour 2 sous-domaines lui permet de trouver une matrice diagonale quasi-optimale qui dépend d'une valeur calculée à partir des valeurs propres maximale et minimale d'une certaine matrice. Un autre opérateur approché est également étudié. Les tests numériques semblent montrer que la matrice diagonale quasi-optimale donne des résultats un peu moins bons que le préconditionneur de Neumann–Neumann quand existent de forts contrastes. Cette méthodologie permet donc de déterminer des coefficients de Robin quasi-optimaux.

Nous n'avons expérimenté aucune des deux techniques que nous venons de présenter, faute de temps, et nous ignorons le coût en durée de développement qu'elles représentent. Nous donnons à présent les résultats de l'étude, essentiellement expérimentale et basée sur des arguments physiques, que nous avons menée pour la simulation de la Section 3.6.2. Nous précisons à nouveau que, par manque de temps également, nous n'avons pas fait non plus de simulations dans lesquelles les coefficients de Robin α_{ij} et α_{ji} sont différents.

[10] Plaçons nous dans le cas où nous avons deux sous-domaines Ω_1 et Ω_2. Dans [84], il est montré que le complément de Schur adéquat pour un sous-domaine est alors le complément de Schur de son voisin. Les deux opérateurs d'interfaces sont différents. Dans ce cadre, il y a un intérêt à avoir deux coefficients α_{12} et α_{21} *distincts*.

Synthèse de l'étude expérimentale concernant les α_{ij}

L'une des raisons qui peuvent mettre en défaut une méthode de décomposition de domaines basée sur des coefficients de Robin, est la *valeur* des perméabilités, indépendamment de leurs sauts. En effet, pour aboutir à la convergence, il faut obtenir simultanément la continuité des pressions et des vitesses normales sur les interfaces. Or, les vitesses normales et les pressions peuvent être d'ordres radicalement différents. En particulier, les vitesses sont proportionnelles aux perméabilités : quand celles-ci sont de l'ordre de 10^{-13}, tandis que les pressions sont plutôt de l'ordre de 100, comme c'est le cas dans la simulation de la Section 3.6.2, la vitesse normale est complètement négligeable devant le terme en pression dans la quantité $\mathbf{K}\boldsymbol{\nabla} p \cdot \mathbf{n} + \alpha p$, à moins que le coefficient de Robin α ne soit lui aussi très petit. On comprend dans ces conditions que si l'on ne fait pas dépendre les coefficient de Robin des perméabilités, il est illusoire d'espérer obtenir à la fois la continuité de la pression et la continuité de la vitesse normale en itérant sur les quantités $\mathbf{K}\boldsymbol{\nabla} p \cdot \mathbf{n} + \alpha p$ et $-\mathbf{K}\boldsymbol{\nabla} p \cdot \mathbf{n} + \alpha p$, qui sont presque égales.

D'autre part, des considérations simples mais indispensables d'analyse dimensionnelle nous poussent à prendre les coefficients de Robin homogènes au rapport d'une perméabilité par une longueur, ce que nous écrivons

$$[\alpha] = \frac{[\mathbf{K}]}{[L]}. \tag{2.44}$$

Nous devons donc trouver et utiliser une distance caratéristique du domaine.

De plus, il est nécessaire de tenir compte des *sauts* de perméabilité qui constituent en général la plus grosse difficulté des écoulements en milieu poreux. Là encore, nous nous laissons guider par une analyse très simple du problème elliptique 1D avec des coefficients constants par morceaux. Nous montrons en Annexe B, que la vitesse est dépendant du rapport de la moyenne harmonique des perméabilités par la longueur totale du domaine, cf. l'équation (B.6). Il devient par conséquent logique de prendre des coefficients de Robin, que nous avons notés α_{ij}^0 dans la Section 3.6.2, tels que

$$\alpha_{ij}^0 \quad = \quad \frac{2\mathbf{K_i}\mathbf{K_j}}{(\mathbf{K_i}+\mathbf{K_j})L_\Omega^{ij}} = \frac{\mathrm{MoyH}\left(\mathbf{K_i},\mathbf{K_j}\right)}{L_\Omega^{ij}} \quad (i,j) \in \mathcal{N}, \tag{2.45}$$

où L_Ω^{ij} est la longueur caractéristique du domaine global Ω selon la direction normale à l'interface Σ_{ij}. La définition de la moyenne harmonique, notée $\mathrm{MoyH}\,(\cdot)$, est rappelée en Annexe B à l'équation (B.4). Ces coefficients $\alpha_{ij}^0, (i,j) \in \mathcal{N}$, ont fourni des résultats tout à fait acceptables, voir la Section 3.6.2, et nous pensons qu'ils peuvent être utilisés avec succès dans de nombreuses applications.

Une analyse théorique des propriétés de ce choix particulier pour les coefficients de Robin serait bien entendu nécessaire.

Enfin, nous mentionnons qu'un degré de raffinement supplémentaire a été expérimenté avec un succès mitigé : il est possible de tenir compte du facteur de forme du domaine. On prend alors des coefficients de Robin, notés α_{ij}^3 dans la Section 3.6.2, tels que

$$\alpha_{ij}^3 \quad = \quad \frac{2\mathbf{K_i}\mathbf{K_j}}{(\mathbf{K_i}+\mathbf{K_j})L_\Omega^{ij}}\,\frac{L_\Omega^{\max}}{L_\Omega^{\min}} = \frac{\mathrm{MoyH}\,(\mathbf{K_i},\mathbf{K_j})}{L_\Omega^{ij}}\,\frac{L_\Omega^{\max}}{L_\Omega^{\min}} \quad (i,j)\in\mathcal{N}, \tag{2.46}$$

où $L_\Omega^{\max}$ et $L_\Omega^{\min}$ sont respectivement les longueurs maximale et minimale du domaine Ω. La pertinence d'un tel choix est en revanche moins claire.

Pour des détails supplémentaires, nous engageons le lecteur à se reporter à l'étude expérimentale de la Section 3.6.2.

2.3.9 Opérateurs d'interface

Nous revenons à présent sur l'algorithme itératif proprement dit. Le but est de le préciser autant que possible dans un formalisme mathématique *fonctionnel*, qui sera très proche de l'implémentation dans l'environnement OcamlP3l.

Nous commençons par définir l'opérateur discret S_i de Robin–Robin[11] en (2.47). Cet opérateur est défini séparément pour chaque sous-domaine. Il prend en entrée des conditions d'interface de type Robin sur sa frontière interne. Il résout le problème de Robin dans le sous-domaine, puis renvoie une valeur sur sa frontière interne, qui s'interprète comme une condition de Robin mais vue cette fois-ci depuis ses voisins.

$$\begin{array}{rcll} S_i : \Lambda_{h,i}\times N_{h,i} & \longrightarrow & \Lambda_{h,i}, & i\in I_n \\ S_i(x_{h,i},q_i)|_{\Sigma_{ij}} & = & \mathbf{u_{h,i}}\cdot\nu_\mathbf{i}+\alpha_{ji}p_{h,i}, & i\in I_n, \quad \forall j\in\mathcal{N}_i, \end{array} \tag{2.47}$$

tel que $(\mathbf{u_{h,i}},p_{h,i})(x_{h,i})$ est la solution du problème de Robin (2.48) posé sur $\Omega_i,\ i\in I_n$.

Chercher $(\mathbf{u_{h,i}},p_{h,i},\lambda_{h,i})\in\mathbf{Z_{h,i}}\times N_{h,i}\times\Lambda_{h,i}$ tels que

$$\begin{array}{rcll} (\mathbf{K^{-1}u_{h,i}},\,\mathbf{v})_i-(\mathrm{div}\,\mathbf{v},\,p_{h,i})_i & = & -\langle\mathbf{v}\cdot\nu_\mathbf{i},\,\lambda_{h,i}\rangle_i\,, & \mathbf{v}\in\mathbf{Z_{h,i}}, \\ (\mathrm{div}\,\mathbf{u_{h,i}},\,r)_i & = & (q_i,\,r)_i\,, & r\in N_{h,i}, \\ \displaystyle\sum_{j\in\mathcal{N}_i}\langle-\mathbf{u_{h,i}}\cdot\nu_\mathbf{i}+\alpha_{ij}\lambda_{h,i},\,\mu_i\rangle_{ij} & = & \displaystyle\sum_{j\in\mathcal{N}_i}\langle x_{h,ij},\,\mu_i\rangle_{ij}\,, & \mu_i\in\Lambda_{h,i}. \end{array} \tag{2.48}$$

[11] A ne pas confondre avec le préconditionneur dit de Robin–Robin introduit dans [3], nommé ainsi par analogie avec le préconditionneur de Neumann–Neumann, [67].

Le problème de Robin (2.48) est bien posé, [83]. Il est en général écrit sous une forme équivalente qui ne fait plus intervenir les multiplicateurs de Lagrange : une simple élimination peut nous en débarrasser pour obtenir (2.49). Cette remarque a pour but de faire le lien avec les problèmes de Robin que nous introduisons lors de l'étude des fractures, cf. la Section 4.4.2 et en particulier l'équation (4.22).

Chercher $(\mathbf{u_{h,i}}, p_{h,i}) \in \mathbf{Z_{h,i}} \times N_{h,i}$ tels que

$$\begin{aligned} \left(\mathbf{K^{-1}u_{h,i}},\, \mathbf{v}\right)_i - \left(\operatorname{div} \mathbf{v},\, p_{h,i}\right)_i &= -\sum_{j\in\mathcal{N}_i} \left\langle \frac{\mathbf{u_{h,i}}\cdot\nu_{\mathbf{i}} + x_{h,ij}}{\alpha_{ij}},\, \mathbf{v}\cdot\nu_{\mathbf{i}} \right\rangle_{ij}, \quad && \mathbf{v} \in \mathbf{Z_{h,i}}, \\ \left(\operatorname{div} \mathbf{u_{h,i}},\, r\right)_i &= \left(q_i,\, r\right)_i, && r \in N_{h,i}. \end{aligned} \tag{2.49}$$

Pour un couple de voisins (Ω_i, Ω_j), $(i, j) \in \mathcal{N}$, nous définissons sur leur interface un opérateur de projection en (2.50) et un opérateur de restriction en (2.51). En outre, pour un sous-domaine Ω_i $i \in I_n$, nous introduisons sur sa frontière interne un opérateur de restriction en (2.52) et un opérateur de projection en (2.54). Enfin nous ajoutons deux dernières notations pour les opérateurs globaux vivant sur la structure complète d'interface en (2.55) et (2.56).

Soit $P_{i\to j}$ l'opérateur de projection L^2 vers $\tilde{\Lambda}_{h,ij} = \Lambda_{h,ji}$. C'est cet opérateur qui réalise les projections, transférant un $x_{ij} \in \Lambda_{h,ij}$ dans $\tilde{\Lambda}_{h,ij}$. Ce passage d'information d'un maillage à un autre est une phase délicate à réaliser dans la pratique ; la Section 2.4 est dédiée à l'explication de la méthode retenue pour ce faire.

$$\begin{aligned} P_{i\to j} : L^2(\Sigma_{ij}) &\longrightarrow \tilde{\Lambda}_{h,ij}, \\ x_{ij} &\longmapsto P_{i\to j}x_{ij} \quad \text{tel que} \\ \langle x_{ij} - P_{i\to j}x_{ij},\, \tilde{\mu}_{ij}\rangle_{ij} &= 0, \qquad \tilde{\mu}_{ij} \in \tilde{\Lambda}_{h,ij}. \end{aligned} \tag{2.50}$$

Soit également l'opérateur de restriction de $\Lambda_{h,i}$ vers $\Lambda_{h,ij}$

$$\begin{aligned} R_{ij} : \Lambda_{h,i} &\longrightarrow \Lambda_{h,ij}, \\ x_i = \left(x_{ij_1}, x_{ij_2}, \ldots, x_{ij_{n_i}}\right)^\top \text{ sur } \mathcal{S}_i^h &\longmapsto x_{ij} \text{ sur } \mathcal{S}_{ij}^h. \end{aligned} \tag{2.51}$$

Maintenant, pour un $i \in I_n$ nous définissons deux opérateurs de restriction : un qui va de l'espace Λ_h défini en (2.15) vers $\Lambda_{h,i}$, et l'autre de l'espace $\tilde{\Lambda}_h$ vers $\tilde{\Lambda}_{h,i}$. Les transposées de ces opérateurs de restriction sont des opérateurs de prolongement ou de relèvement correspondants.

$$\begin{aligned} R_i : \Lambda_h &\longrightarrow \Lambda_{h,i}, \\ x = (x_1, x_2, \ldots, x_n)^\top \text{ sur } \mathcal{S}^h &\longmapsto x_i \text{ sur } \mathcal{S}_i^h. \\ \tilde{R}_i : \tilde{\Lambda}_h &\longrightarrow \tilde{\Lambda}_{h,i}, \\ \tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)^\top \text{ sur } \tilde{\mathcal{S}}^h &\longmapsto \tilde{x}_i \text{ sur } \tilde{\mathcal{S}}_i^h. \end{aligned} \tag{2.52}$$

Par un abus de notation, nous utilisons la permutation s définie en (2.3) comme une permutation de Λ_h vers $\tilde{\Lambda}_h$:

$$\begin{array}{rcl} s : \Lambda_h & \longrightarrow & \tilde{\Lambda}_h, \\ s(x_h) & = & \tilde{x}_h. \end{array} \tag{2.53}$$

Soit P_i l'opérateur de projection de $L^2(\Sigma_i),\ i \in I_n$ vers $\tilde{\Lambda}_{h,i}$. Cet opérateur réalise la projection depuis $\Lambda_{h,i}$ qui vit sur le maillage d'interface *interne*, vers $\tilde{\Lambda}_{h,i}$ qui s'appuie sur le maillage d'interface *externe*.

$$\begin{array}{rcl} P_i : L^2(\Sigma_i) & \longrightarrow & \tilde{\Lambda}_{h,i}, \\ x_i = \left(x_{ij_1}, x_{ij_2}, \ldots, x_{ij_{n_i}}\right)^\top & \longmapsto & \left(P_{i\to j_1} x_{ij_1}, P_{i\to j_2} x_{ij_2}, \ldots, P_{i\to j_n} x_{ij_n}\right)^\top . \end{array} \tag{2.54}$$

A partir de ces opérateurs d'interface, nous pouvons mettre en place les opérateurs globaux

$$\begin{array}{rcl} P : L^2(\Sigma) = \bigoplus_{i\in I_n} L^2(\Sigma_i) & \longrightarrow & \tilde{\Lambda}_h, \\ x = (x_1, x_2, \ldots, x_n)^\top & \longmapsto & (P_1 x_1, P_2 x_2, \ldots, P_n x_n)^\top , \\ Px & = & \displaystyle\sum_{i\in I_n} \tilde{R}_i^\top P_i R_i x, \end{array} \tag{2.55}$$

et

$$\begin{array}{rcl} S : \Lambda_h \times N_h & \longrightarrow & \tilde{\Lambda}_h, \\ (x_h, q) & \longmapsto & (S_1(x_{h,1}, q_1), S_2(x_{h,2}, q_2), \ldots, S_n(x_{h,n}, q_n))^\top , \\ S(x_h, q) & = & \displaystyle\sum_{i\in I_n} R_i^\top S_i(R_i x_h, q_i). \end{array} \tag{2.56}$$

Avec toutes ces notations, le problème (2.16) (avec $c = 0$), s'écrit de manière équivalente comme un problème de point fixe posé sur l'interface, défini en (2.57).

Chercher $x_h \in \Lambda_h$ tel que

$$s\left(PS(x_h, q)\right) = x_h \quad \text{dans } \Lambda_h. \tag{2.57}$$

Dans la pratique, la bilinéarité de S permet de déplacer $S(0, q)$ dans le membre de droite. De plus il n'est pas nécessairement utile de former explicitement P : la projection peut être faite juste après le calcul des $S_i R_i(x_h, 0)$ sur chaque interface. Ainsi les équations (2.58) et(2.59) sont deux expressions équivalentes du même problème (2.57).

$$\mathrm{Id}_{\Lambda_h} x_h - sPS(x_h, 0) = sPS(0, q) \quad \text{dans } \Lambda_h. \tag{2.58}$$

$$\mathrm{Id}_{\Lambda_h} x_h - s \left(\sum_{i \in I_n} \tilde{R}_i^\top P_i S_i(R_i x_h, 0) \right) = s \left(\sum_{i \in I_n} \tilde{R}_i^\top P_i S_i(0, q_i) \right) \quad \text{dans } \Lambda_h. \quad (2.59)$$

Une fois écrit sur une base de Λ_h, le problème de point fixe (2.58) de la méthode de Robin–Robin devient le système linéaire noté

$$\mathcal{A}^{RR} x = b, \qquad (2.60)$$

où $\mathcal{A}^{RR}$ est une matrice carrée non symétrique réelle d'ordre la dimension de Λ_h, où x, l'inconnue, et b, le second membre, sont des vecteurs réels de taille l'ordre de $\mathcal{A}^{RR}$. La matrice $\mathcal{A}^{RR}$ est creuse par blocs ; en dehors du bloc diagonal qui est en fait une diagonale de 1 correspondant à l'identité Id_{Λ_h}, les seuls blocs non nuls sont les blocs $(i, j) \in \mathcal{N}$, qui tiennent compte de la contribution de $S_i|_{\Sigma_{ij}}$. Chaque bloc extra diagonal est donc plein a priori, car les S_i font intervenir la résolution d'un problème de Robin (2.48). Il est donc très coûteux en espace mémoire de calculer explicitement la matrice $\mathcal{A}^{RR}$. On ne le fait d'ailleurs en général jamais.

En effet, dans la pratique, le système (2.60) est résolu itérativement par une méthode de Krylov non symétrique, comme Bi-CGStab [95], ou GMRes [86], qui ne nécessite que la donnée du produit matrice vecteur ($x \mapsto \mathcal{A}^{RR} x$) et non la matrice $\mathcal{A}^{RR}$ elle-même.

Nous explicitons ce produit matrice vecteur dans l'Algorithme 2 suivant.

2.3.10 Algorithme de produit matrice vecteur

Le produit matrice vecteur correspondant au système (2.60) —sans calcul explicite de la matrice— peut être facilement implémenté à partir des équations (2.58) ou (2.59). C'est ce que nous faisons ici.

Algorithme 2 (de produit matrice vecteur, méthode de Robin non-conforme). *Soit $x \in \Lambda_h$, on calcule le produit matrice vecteur $\xi = \mathcal{A}^{RR} x \in \Lambda_h$ comme suit.*

Pour $i \in I_n$
  $x_i = R_i x \qquad (\Lambda_h \to \Lambda_{h,i})$
  Pour $j \in \mathcal{N}_i$
    $x_{ij} = R_{ij} x_i \qquad (\Lambda_{h,i} \to \Lambda_{h,ij})$

Pour $j \in I_n$
  $\mu_j = S_j(x_j, 0) \qquad (\Lambda_{h,j} \to \Lambda_{h,j})$
  Pour $i \in \mathcal{N}_j$
    $\mu_{ji} = R_{ji} \mu_j \qquad (\Lambda_{h,j} \to \Lambda_{h,ji})$

$$\begin{aligned}
&\qquad\qquad \tilde{\mu}_{ji} = P_{j\to i}\mu_{ji} \qquad (\Lambda_{h,ji} \to \tilde{\Lambda}_{h,ji} = \Lambda_{h,ij})\\
&\qquad\qquad z_{ij} = x_{ij} - \tilde{\mu}_{ji} \qquad (\Lambda_{h,ij} \to \Lambda_{h,ij})\\
&\xi = 0\\
&\textit{Pour } i \in I_n\\
&\qquad y_i = 0\\
&\qquad \textit{Pour } j \in \mathcal{N}_i\\
&\qquad\qquad y_i = y_i + R_{ij}^T z_{ij} \qquad (\Lambda_{h,ij} \to \Lambda_{h,i})\\
&\qquad \xi = \xi + R_i^T y_i \qquad (\Lambda_{h,i} \to \Lambda_h).
\end{aligned}$$

□

Remarque 4. Quelques commentaires sur cet Algorithme 2 seront peut-être appréciés. Tout d'abord, ce n'est qu'une expression sous forme d'opérateurs du produit matrice vecteur associé à l'Algorithme de point fixe 1. Ensuite l'algorithme tel qu'il est écrit n'est pas optimal en termes de mémoire requise : un certain nombre de variables intermédiaires sont utilisées ici à seules fins de lisibilité et doivent dans la pratique être omises. On peut ainsi programmer l'algorithme avec seulement 2 séries de vecteurs d'interfaces $\lambda_{ij},\ \mu_{ij},\ (i,j) \in \mathcal{N}$, en plus de x et de ξ.

Enfin, nous décrivons partie par partie ce qui est réalisé.

- Le premier bloc est en fait la restriction du vecteur défini sur la structure interne globale aux interfaces entre sous-domaines.
- Le deuxième bloc constitue le cœur de l'algorithme. Il est fait appel au solveur de sous-domaine qui résout le problème de Robin. La résolution dans chaque sous-domaine est en général effectuée grâce à un solveur direct (de type LU ou Choleski). La matrice du problème dans chaque sous-domaine est factorisée une seule fois lors d'une phase d'initialisation. La résolution du système linéaire à chaque appel du produit matrice vecteur est alors à la fois précise et rapide.
  Ensuite le résultat est projeté, interface par interface, au bon voisin. Pour cela, il est évidemment nécessaire de restreindre au préalable ce résultat à chacune des interfaces entre sous-domaine. Enfin on prend la différence entre la valeur projetée et la valeur initiale sur l'interface.
  C'est au moment de faire cette soustraction que la permutation est réalisée implicitement.
- Le dernier bloc décrit la reconstitution du résultat final à partir des petits bouts disséminés sur chacune des interfaces.

2.4 Projection sur les interfaces planes

Nous expliquons ici la méthode numérique utilisée pour mettre en œuvre l'opérateur de projection L^2 noté $P_{i\mapsto j}$ et défini en (2.50). Le transfert d'information d'un maillage à un autre est une opération délicate qui doit être effectuée de façon propre : deux questions en particulier doivent retenir l'attention. D'une part, il est impératif d'éviter les algorithmes de complexité quadratique, ou pire, faute de quoi la projection sera extrêmement lente. D'autre part, il faut que les approximations qui sont faites au cours des calculs n'engendrent pas d'erreurs catastrophiques. Nous verrons des illustrations de tels comportements non souhaitables obtenus au départ avec notre code ainsi que les remèdes qui ont été apportés.

Dorénavant, et ce jusqu'à la fin de cette section, nous travaillerons dans le cas le plus complexe, c'est-à-dire dans le cas 3D, $d = 3$, où les interfaces sont donc de dimension 2. Le problème de la projection quand $d = 2$ est en effet beaucoup plus simple puisque les interfaces sont linéaires, et ne mérite sans doute pas autant d'attention que le cas des interfaces 2D. En revanche, un problème beaucoup plus ardu, que nous n'aborderons pas non plus, est celui de la projection d'un maillage 3D sur un autre maillage 3D. Cette question qui est déplacée ici puisque nous ne faisons transférer nos informations qu'à travers des interfaces de dimension $d-1$, pourrait sans doute être résolue par des techniques similaires à celles qui sont introduites ici, [70].

De plus, nous nous limitons au cas où l'interface géométrique entre deux sous-domaines Σ_{ij} est *de dimension 2 et plane*. Des difficultés supplémentaires considérables peuvent en effet survenir quand la surface Σ_{ij} n'est pas plane. On peut voir sur la Figure 2.3 que les projections $P_{i\to j}$ peuvent être mal définies dans ce cas à cause du chevauchement des maillages.

Dans la pratique, les maillages d'interface $\mathcal{S}_{ij}^h$ et $\mathcal{S}_{ji}^h$ sont généralement obtenus à partir de deux maillages 3D $\mathcal{T}_i^h$ et $\mathcal{T}_j^h$, $(i, j) \in \mathcal{N}$, et sont par conséquents des maillages *réputés* plans[12] et en fait plongés dans $\mathbb{R}^3$. Or, nous avons besoin pour la méthode exposée dans cette section de deux maillages *réellement plans et ne dépendant que de 2 variables*. Nous devons donc les extraire ces deux maillages à partir des maillages d'interface issus des maillages 3D $\mathcal{T}_i^h$ et $\mathcal{T}_j^h$, $(i, j) \in \mathcal{N}$. Pour cela, pour chaque maillage d'interface 3D $\mathcal{S}_{ij}^h$ et $\mathcal{S}_{ji}^h$, nous commençons par effectuer une simple régression linéaire afin de trouver les deux plans les plus proches des nuages de points 3D de $\mathcal{S}_{ij}^h$ et $\mathcal{S}_{ji}^h$. Nous déterminons ensuite un plan intermédiaire entre les deux plans ainsi obtenus[13], et le

[12] En réalité, ils sont plans aux erreurs d'arrondis du maillage près.

[13] Par exemple, on peut prendre le plan moyen. En géométrie exacte les deux plans des deux maillages sont censés être identiques ; ils doivent donc être "proches" numériquement

considérons comme le plan "réel" porté par Σ_{ij}. Puis chaque point 3D des deux maillages d'intersection est projeté sur ce plan. Finalement, nous avons créé deux maillages d'interface *à deux dimensions*, vivant sur un même plan Σ_{ij}, et issus des maillages d'interface $\mathcal{S}_{ij}^h$ et $\mathcal{S}_{ji}^h$. Ce sont ces maillages d'interface 2D qui servent dans les calculs de la projection[14]. Nous utiliserons dans la suite les mêmes notations, $\mathcal{S}_{ij}^h$ et $\mathcal{S}_{ji}^h$, pour les maillages d'interface plongés dans $\mathbb{R}^3$ et pour les maillages vivant dans le plan commun Σ_{ij}. Ceci n'induira pas de confusion puisque, jusqu'à la fin de ce Chapitre, nous ne travaillons que sur des maillages plans 2D.

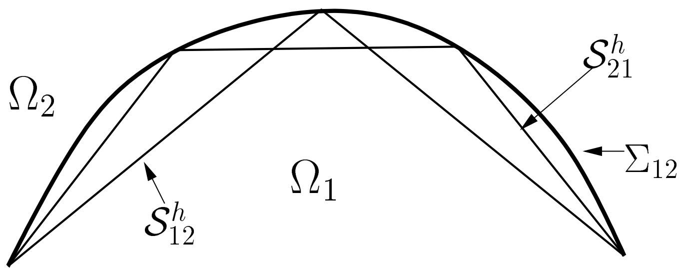


FIG. 2.3 – Exemple en 2 dimensions de maillages non raccordés sur une interface courbe. Les maillages s'interpénètrent.

2.4.1 Projection L^2 de Λ_{ij}^h sur Λ_{ji}^h

Soient deux voisins Ω_i et Ω_j, $(i,j) \in \mathcal{N}$. Nous supposons donc l'interface Σ_{ij} plane. Soit un $x_{ij} \in \Lambda_{ij}^h$, nous voulons le projeter sur $\tilde{\Lambda}_{ij}^h = \Lambda_{ji}^h$ avec la projection L^2 notée $P_{i\mapsto j}$ et définie en (2.50). Nous notons $\tilde{x}_{ij} = P_{i\mapsto j} x_{ij} \in \tilde{\Lambda}_{ij}^h$ le projeté de x_{ij}.

Par définition, le projeté satisfait

$$\langle \tilde{x}_{ij},\, \tilde{\mu}_{ij} \rangle_{ij} = \langle x_{ij},\, \tilde{\mu}_{ij} \rangle_{ij}, \qquad \tilde{\mu}_{ij} \in \tilde{\Lambda}_{h,ij}. \tag{2.61}$$

Nous rappelons que les Λ_{ij}^h sont constitués de fonctions constantes par faces. On peut donc écrire x_{ij} dans la base canonique de Λ_{ij}^h, qui est l'ensemble des fonctions caractéristiques des faces f de $\mathcal{S}_{ij}^h$, notées 1_f. De la même manière, le terme $\tilde{x}_{ij}$ peut être décomposé dans la base canonique de $\tilde{\Lambda}_{ij}^h$ qui est constitué

et le choix du plan intermédiaire ne devrait pas trop influer sur le résultat : ceci induit une erreur systématique que l'on peut espérer petite.

[14]Bien évidemment, si les maillages 2D d'interface existent par ailleurs, il vaut mieux ne pas s'en priver et éviter de faire ce qui précède.

des fonctions caractéristiques des faces $\tilde{f}$ de $\tilde{\mathcal{S}_{ij}^h} = \mathcal{S}_{ji}^h$, notées $1_{\tilde{f}}$. Ainsi

$$x_{ij} = \sum_{f \in \mathcal{S}_{ij}^h} x_{ij}^f 1_f, \qquad \tilde{x}_{ij} = \sum_{\tilde{f} \in \tilde{\mathcal{S}_{ij}^h}} \tilde{x}_{ij}^{\tilde{f}} 1_{\tilde{f}}.$$

En prenant successivement $\tilde{\mu}_{ij} = 1_{\tilde{f}}$ pour chacune des faces $\tilde{f}$ de $\tilde{\mathcal{S}_{ij}^h}$, on obtient de façon immédiate à partir de (2.61)

$$P_{i \mapsto j} x_{ij}|_{\tilde{f}} = \tilde{x}_{ij}^{\tilde{f}} = \sum_{f \in \mathcal{S}_{ij}^h} x_{ij}^f \frac{\text{mes}(f \cap \tilde{f})}{\text{mes}(\tilde{f})}, \qquad \tilde{f} \in \tilde{\mathcal{S}_{ij}^h}, \quad (i,j) \in \mathcal{N}, \tag{2.62}$$

où $\text{mes}(\mathcal{O})$ est la mesure de l'ensemble $\mathcal{O}$. Sous cette forme (2.62), la projection apparaît comme une moyenne pondérée par les mesures des aires des mailles intersectées. La conservation *globale* des flux à travers les interfaces est bien assurée, voir aussi la Remarque 2 de la Section 2.3.5.

2.4.2 Maillage intersecté

Le calcul de la projection $P_{i \mapsto j}$ revient donc à déterminer les aires du maillage intersecté, qui est obtenu en prenant l'intersection des faces de deux maillages $2D$ d'interface :

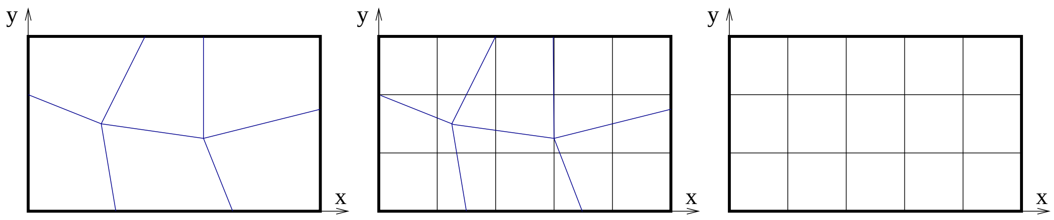
$$\mathcal{S}_{ij}^h \cap \mathcal{S}_{ji}^h = \left\{ f \cap \tilde{f} \;/\; f \in \mathcal{S}_{ij}^h \text{ et } \tilde{f} \in \tilde{\mathcal{S}_{ij}^h} \right\}.$$

On peut voir un exemple de ce maillage intersecté dans la Figure 2.4. Nous ne supposons pas que les maillages sont structurés ni réguliers a priori. Sans perte de généralité, nous supposons donc que les maillages $\mathcal{S}_{ij}^h$ et $\tilde{\mathcal{S}_{ij}^h}$ sont constitués *exclusivement de triangles*. Si un maillage contient d'autres polygones, il suffit de se ramener à ce cas en coupant de façon ad hoc les polygones en triangles[15].

Nous présentons maintenant la procédure que nous avons utilisée pour calculer ces maillages intersectés. Cette méthode a été mise au point par Martial Mancip pendant sa thèse, [69], et programmée lors de son Post-Doc dans le projet Estime à l'Inria Rocquencourt, [70]. Elle a été utilisée et améliorée au cours de tests que nous avons réalisés avec la méthode de décomposition de domaines non conforme présentée auparavant (voir Section 2.3).

Il faut admettre que le calcul d'intersection de maillage n'est sans doute pas parfait : sa robustesse peut encore être améliorée. Ainsi, des difficultés liées à la

[15] Ainsi, comme nous travaillons avec des hexaèdres dans notre code, nous obtenons des faces quadrangulaires que nous divisons en deux triangles.



$\mathcal{S}_{ij}^h$ $\mathcal{S}_{ij}^h \cap \mathcal{S}_{ji}^h$ $\mathcal{S}_{ji}^h = \tilde{\mathcal{S}}_{ij}^h$

FIG. 2.4 – Exemple en 3 dimensions de deux maillages non raccordés sur une interface plane. Le maillage 2D de gauche $\mathcal{S}_{ij}^h$ est la trace sur l'interface Σ_{ij}, $(i,j) \in \mathcal{N}$, d'un maillage 3D $\mathcal{T}_i^h$ de Ω_i. Idem pour le maillage de droite $\mathcal{S}_{ji}^h$. Le maillage du milieu est le "maillage intersecté". Les maillages sont constitués de quadrangles mais peuvent également être faits de triangles. Ils ne se raccordent pas et ne sont pas sous-maillages l'un de l'autre.

très forte anisotropie de certains maillages pour les calculs de déchets ont-elles été résolues tout récemment. Cependant, il faut comprendre que ce problème est complexe. Ceci est en grande partie dû à la précision finie des calculs : deux arêtes réputées confondues peuvent en effet ne pas avoir de point commun à la précision de la machine près. Ou bien encore, deux arêtes disjointes en géométrie exacte peuvent se révéler avoir un ou plusieurs points d'intersection dans les calculs. Ces erreurs numériques peuvent s'avérer catastrophiques si elles ne sont pas prises en compte proprement. Un cataclysme[16] qui nous est ainsi arrivé est la disparition pure et simple d'une maille : deux triangles venant du "même" maillage, donc censés être égaux, avaient, d'après nos calculs, leur intersection d'aire nulle... Paradoxalement, le cas où nous cherchons l'intersection de deux maillages "identiques", qui semble le plus simple a priori, se révèle très compliqué numériquement.

Pour remédier à cela, notre stratégie a été l'adaptation des seuils de tolérance, ou epsilons, qui interviennent dans tous les tests de comparaison entre flottants (proximité de deux triangles, nullité d'une aire d'intersection, intersection de deux droites parallèles, appartenance d'un point à un triangle, etc. , voir les Algorithmes 3 et 4, et l'Annexe A, notamment la Section A.2.3). Nous croyons qu'il est important de faire dépendre ces seuils des maillages, en particulier de l'anisotropie et des tailles caractéristiques des mailles.

Afin de clarifier l'exposé, nous avons découpé la procédure en deux parties : l'Algorithme 3 décrit comment obtenir le maillage intersecté globalement et fait appel de nombreuses fois à l'Algorithme 4 qui permet de calculer l'aire de l'in-

[16]N'ayons pas peur des mots !

tersection de 2 triangles. Dans un souci de simplification, nous ne rentrons pas ici dans tous les détails, et en particulier nous travaillons en “géométrie exacte”, ce qui signifie que nous ignorons délibérément les problèmes liés aux approximations et autres erreurs d'arrondis. Cependant, ces problèmes sont abordés dans l'Annexe A.

Algorithme 3 (d'intersection de 2 maillages en triangles, Martial Mancip). *Soient deux maillages plans* $\mathcal{S}_1^h$ *et* $\mathcal{S}_2^h$ *du même domaine* $\Sigma \in \mathbb{R}^2$ *constitués de triangles exclusivement. Pour* $i = 1, 2$*, on note* $\mathcal{S}_i^h = \bigcup_{k=1}^{M_i} T_k^i$*, où* T_k^i *est un triangle de* $\mathcal{S}_i^h$ *et* M_i *le nombre de ces triangles.*

1. *Pour les triangles* T^1 *de* $\mathcal{S}_1^h$*, calculer et stocker la matrice permettant de calculer les coordonnées barycentriques d'un point du plan par rapport au triangle* T^1*. Cette matrice qui s'obtient par l'inversion d'une matrice* 3×3 *est notée* $\mathcal{M}^{-1}$ *dans l'Annexe A.*
2. *De même, pour tous les triangles* T^2 *de* $\mathcal{S}_2^h$*, calculer et stocker la matrice permettant de calculer les coordonnées barycentriques d'un point du plan par rapport au triangle* T^2*.*
3. *Pour chaque triangle* T^1 *de* $\mathcal{S}_1^h$*, déterminer les* $M_2(T^1)$ *triangles de* $\mathcal{S}_2^h$ *notés* $T_{l_\alpha}^2, \alpha \in \{1, \dots, M_2(T^1)\}$*, avec les* α *tels que* $l_\alpha \in \{1, 2, \dots, M_2\}$*, qui sont “proches[17]” de* T^1 *à* $2h$ *près. Puis, faire :*
   (a) *Pour chaque triangle proche* $T_{l_\alpha}^2, \alpha = 1, \dots M_2(T^1)$*, calculer l'aire intersectée* $mes(T^1 \cap T_{l_\alpha}^2)$ *(cf. Algorithme 4 ci-dessous). Si cette aire est positive, stocker le triplet* $(T^1, T_{l_\alpha}^2, mes(T^1 \cap T_{l_\alpha}^2))$*.*
   (b) *Vérifier que la somme*
   $$\sum_{\alpha=1}^{M_2(T^1)} mes(T^1 \cap T_{l_\alpha}^2)$$
   est égale à $mes(T^1)$*. Si ce n'est pas le cas, la méthode est mise en échec : soit des triangles proches de* T^1 *n'ont pas été trouvés, soit le calcul des aires de triangles intersectés a été mis en défaut.*

□

Déterminer la proximité de deux triangles est un problème à traiter prudemment : si le test de proximité est fait naïvement en vérifiant pour chaque triangle de $\mathcal{S}_1^h$ la distance avec tous les triangles de $\mathcal{S}_2^h$, l'algorithme est de complexité

[17] Le problème de la proximité de deux triangle est abordé ci-dessous.

quadratique et il devient très vite insupportablement lent. Une méthode possible bien connue (par exemple [53], chap. 17.2, pp 577–579) consiste à prédécouper la boîte rectangulaire englobant le maillage en boîtes rectangulaires de diamètre $\Delta \geq\approx 2h$, constituant une grille structurée grossière du domaine plan Σ, et affecter à chaque triangle des deux maillages le numéro de la boîte qui le contient. Ce pré-tri permet d'accélérer notablement la détermination de la proximité entre deux triangles car le tri est maintenant à deux niveaux. L'algorithme peut se décrire de la façon suivante.

Pour un triangle T^1 du maillage $\mathcal{S}_1^h$, nous commençons par déterminer la boîte B qui le contient. Nous stockons alors deux informations : le fait que T^1 est dans B et inversement que B contient T^1. Nous conservons ainsi pour chaque boîte la liste des triangles de $\mathcal{S}_1^h$ qu'elle contient. Nous procédons de la même manière pour les triangles de $\mathcal{S}_2^h$. Une fois ce pré-tri terminé, pour un triangle T^1 de $\mathcal{S}_1^h$, nous lisons la boîte B qui le contient, et déterminons les boîtes voisines de B : il en existe au plus 4 en deux dimensions. Ces cinq boîtes, au plus, fournissent la liste des triangles de $\mathcal{S}_2^h$ qui sont susceptibles d'être proches de T^1 à $2h$ près. Pour chacun de ces triangles de $\mathcal{S}_2^h$ sélectionnés, qui sont notés T^2 et dont le nombre est grandement restreint par rapport au nombre total de triangles de $\mathcal{S}_2^h$, nous calculons effectivement la distance entre T^1 et T^2 et ne gardons que les triangles "proches".

2.4.3 Intersection de deux triangles

Le calcul de la projection dans le cas de deux maillages plans revient donc en dernière analyse à calculer l'intersection de deux triangles T^1 et T^2 quelconques, non réduits à un point ni à un segment. On utilisera les notations suivantes : les sommets du triangle $T^i,\ i = 1, 2$, sont notés $A_k^i,\ k = 1, 2, 3$. L'arête opposée au sommet A_k^i est appelée $\mathcal{E}_k^i$, voir Figure 2.5.

Ce qu'on sait a priori sur cette intersection $T^1 \cap T^2$, que l'on appellera désormais *polygone d'intersection*, est assez limité :

Lemme 2. *L'intersection de deux triangles quelconques non dégénérés est un polygone strictement convexe, possédant de 0 à 6 sommets.*

On peut répertorier 7 classes de cas différents : voir la Figure 2.6. Nous n'avons mentionné ces différentes configurations d'intersection que pour l'illustration de l'exposé. L'Algorithme 4 suivant n'est en aucune manière basé sur le listage de ces différents cas de figure : ceci serait à notre avis lent et laborieux du fait de la multiplicité de ces cas et des nombreuses dégénérescences possibles.

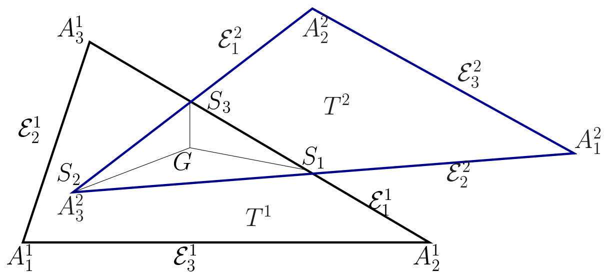


FIG. 2.5 – Deux triangles s'intersectant : notations des sommets et des arêtes. Les sommets du triangle $T^i, i = 1, 2$, sont notés $A_k^i, k = 1, 2, 3$. L'arête opposée au sommet A_k^i est nommée $\mathcal{E}_k^i$.

En effet, diverses dégénérescences peuvent surgir : quand le sommet d'un triangle appartient à un côté de l'autre, ou encore se superpose sur un sommet de l'autre. Enfin, un côté d'un triangle peut se confondre partiellement ou en totalité avec un côté de l'autre triangle. On considère que ces cas limite sont contenus dans les 7 précédents et ne sont pas montrés dans la Figure 2.6. Evidemment ces dégénérescences peuvent engendrer des difficultés numériques plus ou moins graves.

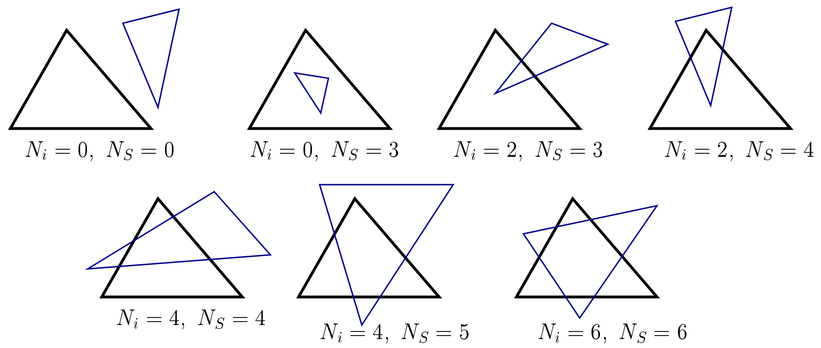


FIG. 2.6 – Liste de 7 polygones obtenus par l'intersection de 2 triangles.

On appelle $S_r, r = 1, 2, \ldots N_S$ les N_S sommets du polygone d'intersection. Le nombre de points d'intersections entre les côtés des deux triangles est noté N_i et vaut obligatoirement $N_i = 0, 2, 4$ ou 6, sauf dans les cas limites cités ci-dessus.

Algorithme 4 (d'intersection de 2 triangles, Martial Mancip). *Soient deux triangles non dégénérés T^1 et T^2. On cherche à calculer l'aire du polygone d'intersection $T^1 \cap T^2$, voir la Figure 2.5. On procède comme suit.*

1. *Détermination de la suite des points qui sont les sommets du polygone*

d'intersection notée

$$\mathcal{P}_S := (S_r)_{r=1,2,\ldots N_S}\,.$$

Ce sont soit des points d'intersection entre une arête de T^1 et une arête de T^2, soit des sommets de T^1 ou de T^2.

(a) *On calcule les points d'intersection entre arêtes des deux triangles. Pour ce faire, pour chaque arête $\mathcal{E}_k^1, k = 1,2,3$, et chaque arête $\mathcal{E}_l^2, l = 1,2,3$, on calcule l'éventuel point d'intersection de la droite portée par l'arête $\mathcal{E}_k^1$ avec la droite portée par le segment $\mathcal{E}_l^2$.*

$$P_{kl} = (\mathcal{E}_k^1) \cap (\mathcal{E}_l^2), \qquad k,l \in \{1,2,3\}.$$

Si les droites sont parallèles ou confondues, on rejette les éventuels P_{kl}. En effet, si les droites sont parallèles et disjointes, il n'y a pas d'intersection. Si elles sont confondues, alors les arêtes peuvent éventuellement s'intersecter, mais dans ce cas, un point de leur intersection qui est aussi un sommet du polygone d'intersection sera aussi un sommet d'au moins un triangle, et cela sera détecté à l'étape suivante.

Les points P_{kl} ne sont pas tous des sommets S_r du polygone d'intersection, car les droites peuvent s'intersecter à l'extérieur des triangles. Aussi, ne garde-t-on que les points d'intersection P_{kl} qui sont effectivement contenus dans les deux triangles T^1 et T^2, définissant la suite des points

$$\mathcal{P}_{S,\ inter} = \left(P_{kl} \subset T^1 \cap T^2\right)_{k,l=1,2,3}.$$

(b) *On ajoute les éventuels sommets $A_l^2,\ l = 1,2,3$, de T^2 qui sont aussi contenus dans T^1. On ajoute également les sommets $A_k^1,\ k = 1,2,3$, de T^1 qui sont inclus dans T^2. Finalement la suite des sommets du polygone d'intersection s'écrit*

$$\mathcal{P}_S = \mathcal{P}_{S,\ inter} \oplus \left(A_k^1 \subset T^2\right)_{k=1,2,3} \oplus \left(A_l^2 \subset T^1\right)_{l=1,2,3},$$

où nous avons noté $\oplus$ la concaténation de suites finies.

- *L'outil de base pour calculer l'intersection de droites ou l'appartenance d'un point à un triangle est le même : les coordonnées barycentriques, voir l'Annexe A. En particulier, la manière de procéder pour déterminer si un point du plan appartient à un triangle ou non est expliquée dans la Section A.2.3.*

- *On doit noter que des points peuvent être trouvés plusieurs fois : un sommet peut être également l'intersection de deux arêtes. Nous n'enlevons pas les doublons. Cela n'est pas gênant car les points ainsi obtenus plusieurs fois auront une contribution nulle à l'aire finale.*

2. *On a obtenu la liste $\mathcal{P}_S$ des N_S sommets du polygone strictement convexe dont on cherche à calculer l'aire.*

   On calcule l'isobarycentre G de la suite $\mathcal{P}_S$. Certains points étant potentiellement comptés plusieurs fois, ce barycentre n'est pas forcément au centre du polygone, mais il est nécessairement dans son intérieur *puisque le polygone est* convexe, *ce qui nous permet d'ordonner les points de $\mathcal{P}_S$.*

   Pour cela, on calcule $(N_S - 1)$ angles θ_r, $r = 2, 3, \ldots, N_S$ de l'intervalle $[0, 2\Pi]$. Chaque θ_r est défini comme l'angle entre les deux vecteurs unitaires
$$\frac{\overrightarrow{GS_1}}{||\overrightarrow{GS_1}||} \quad et \quad \frac{\overrightarrow{GS_r}}{||\overrightarrow{GS_r}||}.$$
   Ce calcul est opéré avec un simple arccosinus *et un test de positivité.*

   On trie ensuite suivant l'ordre croissant la liste des angles $(\theta_r)_{r=2,3,\ldots,N_S}$. Celle-ci devient la liste $(\theta_{r_\alpha})_{\alpha=2,3,\ldots,N_S}$ avec une indexation bijective α sur $\{1, 2, \ldots, N_S\}$ telle que $r_1 = 1$ et
$$\theta_{r_{\alpha-1}} \leq \theta_{r_\alpha}, \quad 2 \leq \alpha \leq N_S.$$
   La liste des points $\mathcal{P}_S$ devient alors la liste des sommets du polygone d'intersection orientés dans le sens direct
$$\tilde{\mathcal{P}}_S := (S_{r_\alpha})_{\alpha \in \{1,2,\ldots,N_S\}},$$
   Finalement l'aire du polygone d'intersection que nous recherchons peut être calculée comme la somme des aires des triangles $GS_{r_\alpha}S_{r_{\alpha+1}}$
$$mes(T^1 \cap T^2) = \sum_{\alpha=1}^{N_S-1} mes(GS_{r_\alpha}S_{r_{\alpha+1}}). \tag{2.63}$$
   Dans la somme de (2.63), on remarque que la contribution à l'aire finale de deux points S_{r_α} et $S_{r_{\alpha+1}}$ confondus *est nulle. La suppression des doublons dans la liste des sommets $\mathcal{P}_S$ n'est donc pas indispensable.*

□

2.4.4 Illustration d'un calcul d'intersection de maillages

Nous montrons une illustration d'un calcul d'intersection de maillage sur la Figure 2.7. Le premier maillage est dans ce cas structuré et régulier, et l'autre est constitué de quadrangles déformés. Le maillage d'intersection montré sur la partie droite de la Figure 2.7 a été obtenu avec l'Algorithme 3 décrit ci-dessus. Nous remarquons que chaque quadrangle a bien été découpé en deux triangles pour calculer le maillage d'intersection.

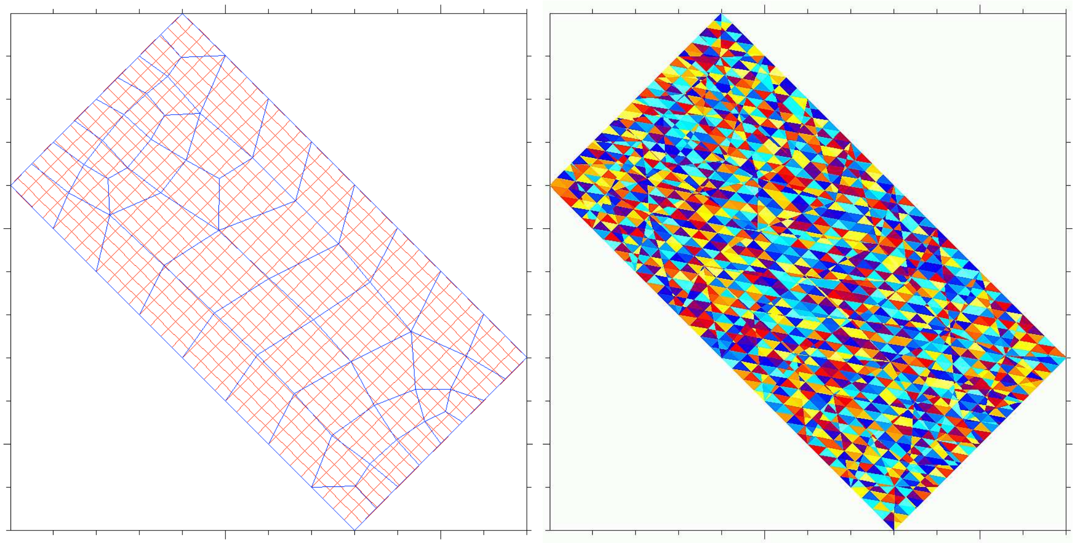


FIG. 2.7 – Gauche : deux maillages 2D superposés. Droite : l'intersection des deux maillages. Images fournies par Martial Mancip.

2.5 Tests numériques 2D de la méthode de décomposition de domaines

Ces premiers résultats numériques constituent la concrétisation d'un travail réalisé pendant et immédiatement après le Cemracs 2001, en collaboration avec Sébastien Wagner[18], sous la direction de Frédéric Nataf[19] et d'Yves Achdou[20]. Le but était alors de tester la méthode de décomposition de domaines

[18]Laboratoire Modélisation Numérique et Couplages, Avenue George Pompidou, BP 56, 83162 La Valette du Var Cédex, France et Joint Research Centre - Environment Institute, 1 Via Enrico Fermi, 21020 Ispra (Va), Italy, sebastien.wagner@jrc.it

[19]CMAP, Ecole Polytechnique, Palaiseau, France.

[20]Université Paris VII, France.

non conformes définie à l'équation (2.58) dans le cadre de la problématique du stockage de déchets.

Ces expériences numériques sont donc relativement simples et reprennent des idées de tests fournies dans [2]. L'originalité principale de la présente étude réside dans la décomposition du domaine d'origine en deux sous-domaines *emboîtés* l'un dans l'autre : ceci permet de faire à moindres frais un raffinement local autour d'une zone jugée critique. Dans le cas des problèmes de déchets souterrains, cela peut servir pour un raffinement autour du site de stockage.

Dans toute cette partie, nous nous limitons à la résolution du problème (2.6) avec un découpage en deux sous-domaines de dimension 2 et une perméabilité constante $\mathbf{K} = 1$. De plus, tous les sous-domaines sont maillés avec des grilles rectangulaires et nous utilisons une formule de quadrature dans la méthode des éléments finis mixtes, afin d'éliminer les inconnues de vitesse. Ceci permet de se ramener à la méthode des volumes finis (voir [29] par exemple).

La méthode de Krylov employée est Bi-CGStab, avec un critère d'arrêt pour tester la convergence tel que

$$\sup_{(i,j)\in\mathcal{N}} || \left((\mathbf{u_{h,i}} + \mathbf{u_{h,j}}) \cdot \nu_\mathbf{i} + \alpha_{ij}(p_{h,i} - p_{h,j}) \right) |_{\mathcal{S}_{ij}^h} ||_\infty \leq 10^{-8}.$$

2.5.1 Cas test académique 2D

Le domaine est le carré unité $\Omega = [0,1]^2$. Le premier sous-domaine $\Omega_1 = [0.6, 0.8]^2$ est totalement entouré par le second $\Omega_2 = \Omega \backslash \Omega_1$, qui est donc non convexe puisqu'il possède un trou. Le terme source q et les conditions aux limites de type Dirichlet $\overline{p}$ sont choisies de telle sorte que la solution exacte soit la fonction régulière $p(x, y) = x^3 y^2 + sin(xy)$.

Nous donnons les normes L^∞ et H^1 de l'erreur pour des sous-domaines ayant des maillages non conformes raffinés successivement.

Les expériences notées 1 et 2 sont faites avec des maillages réguliers, et un raffinement local dans le sous-domaine interne. La grille fine est un sous-maillage de la grille grossière : tous les nœuds de l'interface entre Ω_1 et Ω_2 sont des nœuds de la frontière de Ω_1. Je qualifie les maillages dans ce cas d'*emboîtés*. Dans le cas 1, le maillage interne est deux fois plus fin que le maillage externe. Dans le cas 2, il y a un facteur de raffinement de 10. Pour l'expérience 3, les maillages ne sont pas emboîtés, mais sont totalement non coïncidents : à part les sommets de Ω_1, aucun point des deux interfaces ne sont les mêmes. Nous donnons dans chaque cas le nombre de mailles selon les axes Ox et Oy. Une grille initiale est choisie, puis est raffinée successivement en divisant un rectangle en quatre rectangles égaux, c'est-à-dire en divisant le pas du maillage h par un facteur 2.

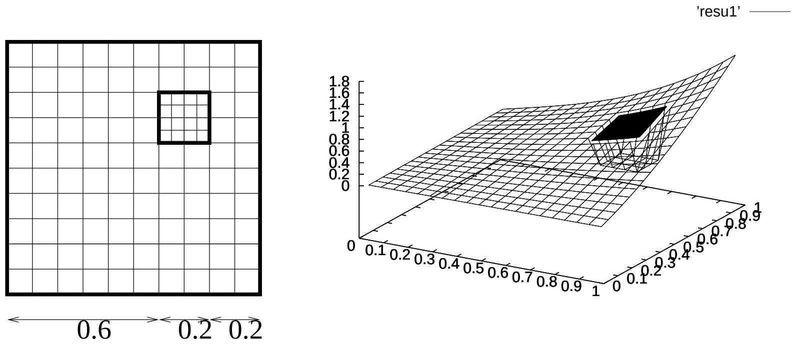


Fig. 2.8 – Gauche : exemple de maillages **emboîtés** pour le problème 2D académique. Le maillage du domaine intérieur est un sous-maillage de celui du domaine extérieur. Grilles (10×10) dans Ω_2 et (4×4) dans Ω_1. Droite : pression au centre des mailles dans l'expérience 2.

Expérience 1 Grilles : $(5 * 2^n \times 5 * 2^n)$ dans $\Omega_2; (2 * 2^n \times 2 * 2^n)$ dans $\Omega_1, n = 0, 1, 2, 3, 4, 5$. Un exemple est présenté pour $n = 1$ dans la Figure 2.8. Les maillages sont emboîtés et les maillages réguliers.

Expérience 2 Grilles : $(5 * 2^n \times 5 * 2^n)$ dans $\Omega_2; (10 * 2^n \times 10 * 2^n)$ dans $\Omega_1, n = 0, 1, 2, 3, 4$. Les maillages sont emboîtés et les maillages réguliers.

Expérience 3 Grids : $(5 * 2^n \times 5 * 2^n)$ dans $\Omega_2; (2 * (2^n + 1) \times 2 * (2^n + 1))$ dans $\Omega_1, n = 0, 1, 2, 3, 4, 5$. Les maillages ne sont *pas* emboîtés et les maillages réguliers.

Résultats

Nous présentons sur la Figure 2.8 la solution du problème 2 avec $2^n = 4$. Les maillages sont réguliers : 20×20 pour le sous-domaine externe et 40×40 pour le sous-domaine interne. Pour des raisons de visualisation, nous avons ajouté des 0 à la solution du domaine Ω_2 dans la zone du sous-domaine Ω_1 afin de tracer cette solution sur une grille carrée régulière. Ceci explique les lignes sans sens physiques qui passent sous le carré noir. D'autre part, la solution du sous-domaine interne apparaît noire car le maillage est trop fin pour la résolution de l'image.

Dans la Figure 2.9, nous avons dessiné la valeur absolue de l'erreur en chaque point pour chaque sous-domaine.

Nous notons η le facteur de réduction de l'erreur en norme H^1 d'une grille de pas de maillage h_1 à la grille suivante de pas h_2. Il est calculé par la formule $\frac{||\text{erreur}_{h_1}||_{H^1}}{||\text{erreur}_{h_2}||_{H^1}} = \left(\frac{h_1}{h_2}\right)^\eta$.

Lors des expériences 1 et 2, l'ordre asymptotique de convergence dans la

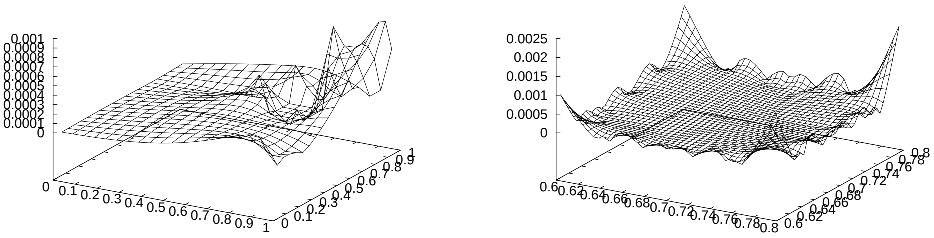


FIG. 2.9 – Erreur en norme infinie : Expérience 2. Gauche : dans le sd externe Ω_2. Echelle : de 0 à $1E-3$. Droite : dans le sous-domaine interne Ω_1. Echelle : entre 0 et $2.5E-3$.

2^n	$1/h$	$\|\|\cdot\|\|_\infty$	$\|\|\cdot\|\|_{H^1}$	réduction d'erreurH^1	η
1	5	$1.021E-2$	$2.397E-2$		
2	10	$3.610E-3$	$9.932E-3$	2.41	1.27
4	20	$1.071E-3$	$4.527E-3$	2.19	1.13
8	40	$3.053E-4$	$1.638E-3$	2.76	1.47
16	80	$8.519E-5$	$5.897E-4$	2.77	1.47
32	160	$2.357E-5$	$2.108E-4$	2.80	1.48

TAB. 2.1 – Expérience 1 : erreurs comme fonctions de h.

2^n	$1/h$	$\|\|\cdot\|\|_\infty$	$\|\|\cdot\|\|_{H^1}$	réduction d'erreurH^1	η
1	5	$2.784E-2$	$4.744E-2$		
2	10	$7.379E-3$	$1.464E-2$	3.24	1.70
4	20	$2.054E-3$	$5.054E-3$	2.90	1.53
8	40	$5.617E-4$	$1.777E-3$	2.84	1.51
16	80	$1.520E-4$	$6.281E-4$	2.83	1.50

TAB. 2.2 – Expérience 2 : erreurs comme fonctions de h.

2^n	$1/h$	$\|\|\cdot\|\|_\infty$	$\|\|\cdot\|\|_{H^1}$	réduction d'erreurH^1	η
1	5	$1.610E-2$	$3.99E-2$		
2	10	$1.636E-2$	$3.22E-2$	1.24	0.31
4	20	$1.164E-2$	$2.10E-2$	1.53	0.61
8	40	$5.250E-3$	$8.78E-3$	2.39	1.26
16	80	$1.730E-3$	$2.70E-3$	3.25	1.70
32	160	$4.926E-4$	$7.43E-4$	3.64	1.86
48	240	$2.287E-4$	$3.43E-4$	2.17	1.91

TAB. 2.3 – Expérience 2 : erreurs comme fonctions de h.

norme H^1 est en $\mathcal{O}(h^{3/2})$, comme le montrent les Tables 2.1 et 2.2. La super convergence qui existe au centre des mailles pour la méthode des volumes finis est conservée partiellement avec cette méthode de décomposition de domaines dans le cas où les maillages sont *réguliers et non concordants mais emboîtés*. L'ordre $\mathcal{O}(h^{1/2})$ qui manque par rapport à l'ordre $\mathcal{O}(h^2)$ qui est attendu théoriquement avec les volumes finis sur des maillages réguliers est probablement le coût à payer pour avoir des maillages non concordants. On note de plus que les résultats restent sensiblement les mêmes pour les deux tests, bien que le maillage du sous-domaine interne soit 5 fois plus fin dans le cas 2 que dans le cas 1. L'algorithme semble donc peu dépendant du rapport entre les maillages des deux sous-domaines dans cette expérience où la solution est très régulière.

Les résultats sont moins nets pour l'expérience 3, qui correspond à des maillages qui ne sont pas emboîtés, voir la Table 2.3. L'ordre de convergence augmente et semble vouloir atteindre un $\mathcal{O}(h^2)$. Cependant, le taux de convergence part d'un niveau bien inférieur aux cas précédents. Ainsi, si on compare avec les erreurs du cas 1 qui est analogue au cas 3, mais avec des maillages emboîtés, pour un raffinement égal de $2^n = 32$, on obtient dans un cas 2.10^{-4} contre 7.10^{-4} dans l'autre. L'erreur H^1_h est donc ici 3.5 fois plus forte dans le cas non emboîté que dans le cas emboîté.

2.5.2 Cas test de stockage simplifié

Nous montrons maintenant une expérience numérique visant à modéliser une couche géologique contenant cinquante alvéoles qui libèrent un contaminant. Le problème est statique, 2D et avec des coefficients normalisés à 1. Aucune interprétation physique n'est donc réalisable.

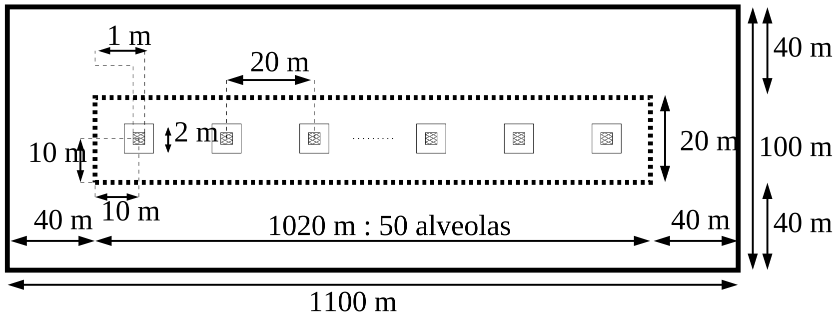


FIG. 2.10 – Géométrie de la couche géologique simplifiée.

La géométrie est présentée dans la Figure 2.10. Le domaine $\Omega = [0, 1100] \times [0, 100]$, en mètres, est décomposé en deux sous-domaines $\Omega_1 = [40, 1060] \times [40, 60]$, qui contient tous les termes source, et $\Omega_2 = \Omega \backslash \Omega_1$. L'interface est tracée en ligne discontinue. Les sources sont définies par $q = 1$ dans les alvéoles représentées comme des carrés gris sur la Figure 2.10, et 0 partout ailleurs.

Résultats

On peut visualiser les résultats de calcul sur les Figures 2.11 et 2.12.

Les calculs ont été réalisés sur deux maillages différents. Lors du premier, la grille du sous-domaine externe Ω_2 était constituée de 110×10 mailles carrées de longueur $10m \times 10m$. Le sous-domaine interne Ω_1 possédait 2040×40 mailles carrées de taille $0.5m \times 0.5m$. Pour le second calcul, le maillage externe était inchangé à 110×10 mailles. Le maillage interne était raffiné par un facteur deux : il contenait 4080×80 mailles carrées de longueur $0.25m \times 0.25m$.

Dans les deux cas, Bi-CGStab converge en 34 itérations et moins d'une dizaine de minutes avec un PC commun.

On note que le maillage externe est beaucoup moins fin que le maillage autour du site de stockage : *le maillage dans la couche géologique est de 20 à 40 fois plus grossier que dans le voisinage des déchets.* Ceci n'a pas d'incidence sur la rapidité de convergence de la méthode dans ce cas.

2.6 Conclusion

Après cette analyse de la méthode de décomposition de domaines de Robin non-conforme et ces premiers tests simples en deux dimensions qui montrent son utilité pour le raffinement local, nous allons aborder maintenant des simulations plus complexes en trois dimensions. Pour cela, nous avons besoin de

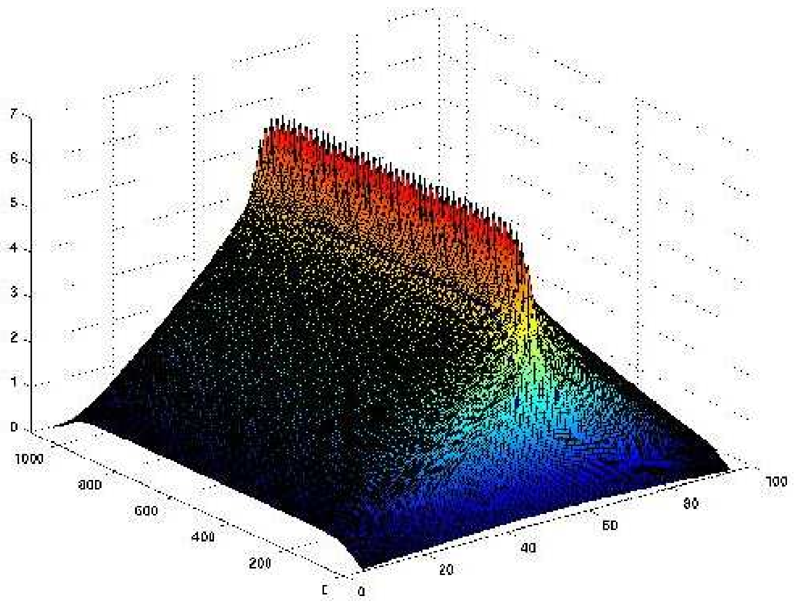


FIG. 2.11 – Solution interpolée du cas de couche géologique simplifiée. Grilles : (110 x 10) et (2040 x 40) mailles. On note les 50 pics correspondant au 50 sources.

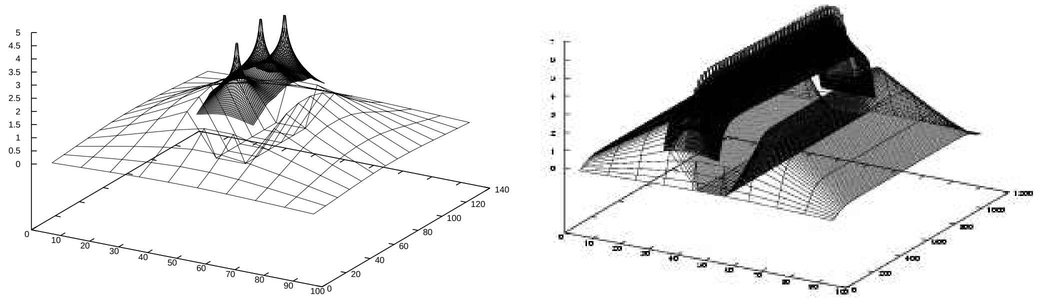


FIG. 2.12 – Solution du cas de couche géologique simplifiée. Gauche : illustration avec 3 sources pour mieux visualiser les maillages et les pics. Droite : cas avec les 50 sources. Grilles : (110 x 10) et (2040 x 40) mailles.

programmes qui mettent en œuvre de façon efficace la méthode de décomposition de domaines que nous venons d'introduire. L'implémentation parallèle et les expérimentations en 3D constituent précisément le sujet du prochain chapitre.

Chapitre 3

Couplage de codes et parallélisme avec OcamlP3l

Ce chapitre est dédié à la mise en œuvre informatique de la décomposition de domaines et aux résultats en 3 dimensions illustrant cette implémentation. En temps normal, cette partie, technique et laborieuse, serait à peine abordée ou reléguée en fin de mémoire au milieu des annexes. Cependant, un travail original a été réalisé qui mérite à mon avis une description complète et en bonne place dans cet ouvrage.

Ce travail consiste en l'utilisation de l'environnement OcamlP3l pour programmer la méthode de décomposition de domaines. Nous donnerons plus loin d'amples explications sur ce qu'est OcamlP3l, sur ce qu'il peut faire et comment il peut être utilisé. Nous pouvons cependant dire dès à présent que ce système permet de programmer la structure parallèle d'un code en langage Ocaml[1] avec un nombre réduit de commandes. Nous croyons fermement que l'approche utilisant OcamlP3l possède un avenir prometteur : l'environnement OcamlP3l détient de sérieux atouts inhérents à sa nature-même. Sa plus grande qualité réside certainement dans le fait que le système OcamlP3l permet à l'utilisateur d'écrire et de déboguer une version *séquentielle* de son programme et ensuite la version parallèle est *automatiquement déduite* par une simple recompilation du programme source. D'autre part, nous devons mentionner une faiblesse du système : dans sa version actuelle, l'application qui nous intéresse, la décomposition de domaines, ne peut pas être programmée de façon *scalable*, car il existe un goulot d'étranglement des communications. Cette limitation n'a pas semblé avoir d'influence sur nos applications, pour lesquelles le nombre de sous-domaines restait relati-

[1] Rappelons que l'on prononce Ocaml avec le "O" de *O rage ! ô désespoir ! ô vieillesse ennemie !* [34, Acte I, scène IV], et "*avec le "ca" de* café *et le "mel" de* melba" [97], page xi. Voir les URL `http ://pauillac.inria.fr/ocaml/` ou `http ://www.ocaml.org/`.

vement faible —de l'ordre de 16 en général. Des recherches sont en cours pour pallier cette faiblesse qui serait une forte nuisance pour des calculs massivement parallèles. Cette restriction posée, nous pouvons passer à la description du système OcamlP3l qui permettra, nous n'en doutons pas, de montrer toute sa force.

Mais avant tout, nous tenons à préciser que l'étude qui suit est le fruit d'un an et demi d'une collaboration interdisciplinaire exemplaire entre deux chercheurs en informatique, Pierre Weis[2] et Roberto Di Cosmo[3], d'un côté, et trois numériciens de l'autre, François Clément, Arnaud Vodicka[4] et moi-même. Trois publications, [30, 31], dont un article hybride mi-informatique, mi-numérique soumis à un journal d'informatique [35], attestent du dynamisme de ce partenariat.

L'interaction entre numériciens et informaticiens a été un réel succès dans la mesure où l'échange a vraiment été réciproque. D'un côté, les demandes des numériciens, liées à leurs problèmes spécifiques, ont engendré une foison d'idées nouvelles pour le système qui sont maintenant incorporées dans la version actuelle d'OcamlP3l : la *coloration**[5] des nœuds physiques et virtuels du réseau de calcul pour spécifier leur appariement mutuel, la nouvelle notion de *parfuns**qui sont des sous-réseaux de calcul parallèle, considérés par l'utilisateur comme des fonctions, et enfin la possibilité d'initialiser localement ou globalement ces sous-réseaux, ce qui joue un rôle important dans la décomposition de domaines. Ces trois notions améliorent l'efficacité et facilitent l'écriture de programmes, ce qui constitue un pas en avant vers l'intégration en douceur du calcul parallèle dans le paradigme[6] de la programmation fonctionnelle.

Dans l'autre sens, les informaticiens ont apporté tout simplement la méthodologie et l'outil de base qui ont permis aux numériciens d'écrire un programme de couplage de codes, avec en prime le parallélisme.

[2]Projet Cristal - INRIA Rocquencourt

[3]Projet Cristal - INRIA Rocquencourt et Université de Paris 7

[4]Respectivement chercheur et ingénieur dans le projet Estime - INRIA Rocquencourt

[5]Rappelons que les termes suivis d'un astérisque sont brivement définis dans le lexique en Annexe C.1.

[6]Paradigme : modèle théorique de pensée qui oriente la recherche et la réflexion scientifique, d'après le Petit Larousse illustré, 1998.

3.1 Introduction : couplage de codes, décomposition de domaines et parallélisme

Le but final de l'étude que nous présentons ici est le *couplage de codes*. En effet, la complexité des problèmes à résoudre s'accroît, à la fois en termes de taille des problèmes, mais aussi en raison de la diversité des physiques à traiter. Il devient crucial de coupler des physiques différentes. Ainsi, dans la problématique du stockage de déchets en milieu souterrain, l'équation du transport doit-elle, dans certains cas, être couplée à des équations de chimie : on parle alors de couplage chimie–transport. Restons sur cet exemple simple. La plupart du temps des codes traitant séparément la chimie et le transport existent déjà. Un moyen d'éviter la réécriture de ces codes est de les utiliser en les couplant entre eux. Le couplage de codes entre eux est donc l'enjeu final du projet.

Un premier pas dans cette direction est le couplage d'un code avec lui-même. et une application non triviale est la décomposition de domaines. Nous présentons donc ici les résultats obtenus dans ce cadre. La méthode de décomposition de domaines utilisée est la méthode de Robin–Robin décrite au Chapitre 2 dans la Section 2.3. Un certain nombre de difficultés intrinsèques du couplage de codes ont pu être touchées du doigt au cours de cette étude. Notamment, la définition claire de l'interface, au sens informatique du terme, entre le code à coupler et le coupleur s'est révélée indispensable.

Une autre question qui est reliée au couplage de code est celle du parallélisme. Il est évidemment intéressant d'exécuter de façon parallèle les codes de calculs, quand c'est possible. Le but du parallélisme est double : d'une part il permet de répartir la charge mémoire sur plusieurs machines, et d'autre part il peut accélérer notablement les calculs. Dans le cadre de la décomposition de domaines, l'utilisation du parallélisme est naturelle et a pu être mise en œuvre. La question délicate de l'équilibrage de charges ne sera pas abordée en tant que telle, mais reste une préoccupation qui reviendra tout au long de cette partie, notamment au moment des applications numériques.

Nous avons choisi de coupler les codes en utilisant le langage Ocaml et plus particulièrement l'environnement OcamlP3l qui nous a fourni *automagiquement* toute la tuyauterie nécessaire pour les communications entre les codes, avec au passage le parallélisme pour le prix d'une simple recompilation !

Nous devons dire dès maintenant que, par manque de temps, nous n'avons effectué aucune confrontation avec le standard MPI, [59]. Nous pensons qu'il serait intéressant de comparer l'efficacité des codes parallèles, mais également les temps de développement en programmant avec OcamlP3l puis avec MPI. Nous pensons qu'à algorithme égal, l'implémentation avec MPI en C++ serait sans doute *un peu* plus rapide, puisqu'il n'y aurait pas d'interface avec un autre langage informatique. Par contre nous sommes persuadés que le temps de développement et surtout de débogage serait à coup sûr *beaucoup* plus important. C'est d'ailleurs principalement cette perspective de devoir perdre quelques mois à déboguer un code parallèle MPI qui nous a découragés de l'écrire, *alors que le code* OcamlP3l *fonctionnait déjà...*

Nous commençons par un bref survol du code à coupler dans la Section 3.2. Ce code, écrit en C++, résout un problème sur un sous-domaine et est donc appelé en toute originalité *solveur de sous-domaine*. La Section 3.3 est consacrée à une courte présentation du langage de programmation Ocaml qui a pour but de mettre au clair certaines notions qui sont utilisées dans la partie suivante. Dans la Section 3.4, nous décrivons le système OcamlP3l dans sa généralité puis nous donnons un exemple d'utilisation avec le programme qui opère le couplage entre codes : ici encore, nous ferons preuve d'esprit d'initiative en le baptisant *coupleur*. Enfin, la Section 3.5 est dédiée aux résultats numériques obtenus en faisant interagir les solveurs de sous-domaine au moyen du coupleur. Ces résultats sont de deux sortes : nous avons validé la méthode de décomposition de domaines et testé le parallélisme sur un cas académique ; ensuite, nous nous sommes confrontés à un problème “industriel” fourni par l'Andra. Une courte conclusion clôt définitivement ce chapitre à la Section 3.7.

Nous précisons qu'une description complète du contenu des sections se trouve au début de chacune d'entre elle.

3.2 Solveur de sous-domaine : le code LifeV

Nous expliquons rapidement dans cette section le solveur de sous-domaine. La Section 3.2.1 donne une bibliographie succinte concernant les méthodes d'éléments finis mixtes et mixtes hybrides. Il y est également mentionné des problèmes spécifiques qu'ont ces méthodes avec les hexaèdres déformés, [75]. Ensuite dans les Sections 3.2.2 et 3.2.3, nous fournissons quelques indications sur la manière dont nous avons programmé le solveur de sous-domaine. Enfin, la courte Section 3.2.4 contient quelques remarques personnelles sur le langage C++, dans lequel le solveur de sous-domaine a été écrit, et propose quelques pistes bibliographiques sur le sujet.

Le solveur de sous-domaine est un programme que nous avons inséré dans un projet plus vaste dénommé LifeV : LifeV[7] est une librairie d'éléments finis en

[7] Voir l'URL `http ://www.lifev.org/`.

3 dimensions —LI*brary of* F*inite* E*lements*— écrite en C++ et qui vise, à son origine tout au moins, des applications plutôt orientée vers les écoulements sanguins. Cette librairie est le fruit d'un partenariat étroit qui dure depuis quelques années entre différents laboratoires de recherche européens : notamment au sein de l'Inria Rocquencourt, du Politecnico de Milan en Italie, et de l'Ecole Polytechnique Fédérale de Lausanne en Suisse[8].

Le lien avec les milieux poreux, qui nous préoccupent ici, peut sembler ténu au premier abord. En fait, une collaboration s'est mise en place car certains écoulements sanguins à travers des filtres, dont le but est de séparer les différents constituants du sang, peuvent se modéliser comme des écoulements en milieux poreux anisotropes. C'est ainsi que j'ai commencé à travailler avec Jean-Frédéric Gerbeau et son étudiant Mohamed Bel Hadj[9].

3.2.1 Eléments finis mixtes et mixtes hybrides

Ainsi, avons-nous programmé ensemble en C++ la méthode des éléments finis mixtes hybrides, pour des éléments de Raviart–Thomas–Nédélec d'ordre le plus bas, notés RT_0, sur un problème elliptique d'écoulement avec un tenseur de perméabilité pouvant être anisotrope, et dans un maillage général[10] constitué d'hexaèdres déformés. Nous ne détaillerons pas ici les éléments finis mixtes ni la technique d'hybridation. Nous devons cependant dire quelques mots sur l'hybridation des éléments finis mixtes. Elle permet de transformer, après quelques manipulations algébriques, le système matriciel issu de la méthode des éléments finis mixtes, qui est symétrique mais non-défini pour les problèmes elliptiques et qui a pour inconnues les vitesses et les pressions, en un système symétrique défini positif dont les inconnues sont des multiplicateurs de Lagrange qui sont introduits pour réaliser l'hybridation et qui s'interprètent comme des traces de pression sur les faces des éléments.

Une littérature abondante existe à ce sujet : l'hybridation a été originellement introduite dans [43], puis a été analysée en détails dans [16]. Sur les éléments finis mixtes de Raviart et Thomas, citons [82] ; pour les éléments 3D, il convient d'ajouter le nom de Nédélec aux deux précédents, [79]. Nous citons également les deux ouvrages [83] et [23], qui permettent d'avoir une vision globale de la question. Mentionnons enfin l'article [29] qui offre une très bonne approche physique et pédagogique des éléments finis mixtes et mixtes hybrides.

Comme nous travaillons sur des hexaèdres déformés, nous devons également dire un mot du travail de Naff, Russell et Wilson [75], [77] ou [76]. Ces au-

[8]Voir l'URL `http ://iacs.epfl.ch/cmcs/Haemodel.php3`.
[9]Projet Bang - Inria Rocquencourt.
[10]C'est-à-dire non-structuré.

teurs montrent en effet que la convergence de la méthode des éléments finis mixtes n'est pas assurée sur des hexaèdres convexes ayant des déformations en 3 dimensions : *les vitesses constantes ne vivent pas toujours dans l'espace d'approximation.* Ceci est lié aux propriétés de la transformation de Piola, [23], qui est utilisée habituellement dans les méthodes mixtes pour envoyer une fonction vectorielle du cube unité sur un hexaèdre quelconque ; or, cette transformation de Piola transforme obligatoirement une fonction vectorielle vivant dans RT_0 sur le cube de référence, en une fonction dont les flux ont une dépendance *linéaire* sur chaque face de l'hexaèdre d'arrivée[11]. Le problème est que les flux d'une fonction constante dans certains hexaèdres déformés peuvent avoir un comportement *quadratique*, [75]. L'un des deux exemples fournis dans les références citées est celui de la pyramide régulière à base carrée, dont le sommet a été tronqué. Dans un tel élément, les fonctions de RT_0 transformées par Piola ne peuvent représenter un flux uniforme de façon exacte.

L'effet est est essentiellement lié aux hexaèdres. Un problème similaire de dégradation de la convergence du terme en divergence de la vitesse peut apparaître pour des quadrilatères convexes quelconques en 2 dimensions, [15]. Il est dû essentiellement au fait que la fonction transformant le cube de référence en un hexaèdre quelconque est *trilinéaire*, alors que les fonctions transformant un simplexe de référence en un simplexe quelconque en toute dimension sont *linéaires*. Nous notons que la fonction transformant un carré de référence en un quadrangle en 2D est *bilinéaire*.

Une solution à ce problème préconisée dans [77] est de se placer dans le cadre des méthode CVMFE —*Control-Volume Mixed Finite Element*— et de prendre des fonctions de forme *quadratiques* pour la vitesse et non plus linéaires. Ceci permet d'améliorer notablement la précision dans la plupart des cas. Faute de temps, nous n'avons pas exploré plus avant la question et n'avons pas tenté de programmer les fonctions de forme recommandées.

3.2.2 Implémentation des éléments finis mixtes

Ces quelques lignes expliquent le travail de programmation que nous avons réalisé pour obtenir un code qui résolve l'écoulement sur un seul domaine (2.2).

Nous sommes partis de la librairie LifeV et nous sommes servis des outils de gestion de maillage, de traitement des conditions aux limites, d'assemblage, de manipulations de tableaux, d'inversion matricielle, etc., qui existaient déjà. Nous avons utilisé le formalisme de mise en œuvre des éléments finis pour ajouter les éléments de Raviart-Thomas-Nédélec sur des hexaèdres, [82] et [79] de plus

[11] Plus exactement, la composante k du flux dépend linéairement de la variable x_k, $k = 1, 2, 3$.

bas degré aux éléments finis déjà implémentés[12]. Je tiens à préciser que le code était suffisamment bien conçu initialement pour avoir permis de programmer assez facilement ces éléments finis mixtes, qui sont relativement différents des éléments de type Lagrange, puisque leurs degrés de liberté sont vectoriels et vivent sur les faces.

La manière dont nous avons programmé les éléments finis mixtes devrait normalement laisser la possibilité de développer très facilement des éléments finis mixtes d'ordre élevés. En particulier, l'élimination des inconnues de vitesse et de pression qui a lieu pendant la phase d'hybridation, est faite avec les librairies Blas[13] —*Basic Linear Algebra Subprograms*— [47], et Lapack[14] —*Linear Algebra PACKage*— [11], et reste valable pour tous les éléments finis mixtes hybrides. L'inversion du système matriciel est opérée grâce à la librairie Aztec, [94], qui offre un choix de méthodes itératives pouvant être utilisées en parallèle, ce que nous n'avons pas cherché à faire, le parallélisme provenant directement de la décomposition de domaines et d'OcamlP3l.

3.2.3 Du solveur sur un domaine au solveur de sous-domaine

Programmer de façon efficace une méthode de décomposition de domaines nécessite plus qu'un code résolvant un problème sur un domaine, il faut que ce code soit un véritable solveur de sous-domaine. Expliquons-nous : de façon générale, un code d'éléments finis qui traite un problème stationnaire et linéaire sur un domaine opère suivant le schéma : il lit un maillage, lit les paramètres (conditions aux limites, coefficients, etc.), puis à partir de ces données forme la matrice du problème et le second membre. Ensuite il inverse le système linéaire et écrit la solution sur le disque et la trace éventuellement.

Le solveur de sous-domaine agit de façon sensiblement différente : il doit réaliser l'opérateur d'interface (2.47) de la Section 2.3.9. Pour ceci, il ne fait qu'une seule fois la phase d'initialisation pendant laquelle il lit le maillage, les conditions aux limites, forme —et factorise— la matrice. Ensuite, le solveur est une *fonction* qui prend un vecteur d'interface en argument, fait un calcul —consistant essentiellement en une inversion de système matriciel— et renvoie un vecteur d'interface en sortie. Nous reviendrons plus en détails sur la manière dont fonctionne le solveur de sous-domaine dans la Section 3.5.3, où nous décrivons complètement le coupleur qui coordonne les solveurs de sous-domaine.

[12]Citons les éléments $P1$, $P1$ bulle, $P2$, et d'autres, pour les tetraèdres et $Q1$ pour les hexaèdres.

[13]Voir l'URL `http ://www.netlib.org/blas/`.

[14]Voir l'URL `http ://www.netlib.org/lapack/`.

Nous précisons à ce propos que notre philosophie a été de découpler complètement les sous-domaines : un sous-domaine ne connaît que lui-même ; il ignore complètement que d'autres sous-domaines peuvent éventuellement exister. En revanche, un sous-domaine Ω_i sait différencier deux sortes de bords : les bords extérieurs, qui appartiennent en fait à Γ_i et les bords d'interface, qui constituent $\mathcal{S}_i^h$, voir la Section 2.2 pour les notations. Dans cette optique, seul le coupleur connaît l'ensemble des sous-domaines et la manière dont ils sont reliés entre eux. Disons un mot de l'interfaçage, c'est-à-dire de la manière dont les communications sont effectuées avec le coupleur : les communications passent par les canaux d'entrée et de sortie standards du solveur de sous-domaine, voir la Section 3.5.3. Les vecteurs d'interface qui transitent entre un sous-domaine et le coupleur sont précédés d'un en-tête qui le caractérise entièrement Par exemple, un vecteur d'interface vivant sur le maillage d'interface $\mathcal{S}_{ij}^h,\ (i,j) \in \mathcal{N}$ entre Ω_i et Ω_j et qui est vu depuis le sous-domaine Ω_i, est précédé d'un en-tête qui contient dans l'ordre : un entier représentant le numéro unique de l'interface $\mathcal{S}_{ij}^h$[15], le numéro i du sous-domaine, le numéro j du voisin, et la taille du vecteur d'interface.

Après avoir fini de programmer la méthode des éléments finis, nous avons donc passé quelques mois à coder en C++ le solveur de sous-domaine. Pour la résolution matricielle directe, nous avons utilisé la méthode LU fournie par la librairie UMFPack, [42].

3.2.4 A propos du C++

Le langage de programmation orienté objets C++ est maintenant largement répandu dans la communauté numérique. Par conséquent, il n'est sans doute pas utile de consacrer trop de temps à lui faire de la publicité, d'autant plus que, comme nous le verrons à la Section 3.3.2, la programmation objet est beaucoup moins sûre que la programmation fonctionnelle. Mon expérience de la programmation objet aurait tendance à confirmer cette assertion, sous la forme d'une remarque que ne démentiront certainement pas les programmeurs de C++ : *il est extrêmement facile de mal programmer en* C++. Il est nécessaire d'avoir une très bonne connaissance de la programmation pour éviter de faire n'importe quoi. Ceci peut, à mon avis, s'interprèter comme une faiblesse de ce langage très permissif.

Néanmoins, nous sommes parvenus à faire ce que nous voulions dans ce langage —après bien des phases de débogages, certes—, et nous demeurons

[15]Une bijection est choisie entre l'ensemble de ces numéros d'interface $\mathcal{S}_{ij}^h,\ (i,j) \in \mathcal{N}$ inclus dans $\mathbb{N}$, et l'ensemble $\mathcal{N}$ inclus dans $\mathbb{N}^2$.

relativement satisfaits du résultat ainsi que du moyen.

Pour finir, présentons quelques ouvrages —goutte d'eau au milieu de l'océan des publications consacrées au C++— qui nous ont servi et que nous avons trouvés bien faits : le passeport pour le C++ [60] est un bon livre plutôt à l'usage des étudiants, donnant un bon aperçu du C++. Le livre de Stroustrup [92] est la référence incontournable, très complète bien qu'un peu indigeste. Nous concluons avec un livre original sur le C++ : [66] ; son intérêt majeur est de commencer directement par l'utilisation de la librairie de modèles standard —*Standard Template Library* ou STL—, et de fournir un accès rapide aux possibilités avancées du langage. Il renverse ainsi les habitudes de l'enseignement du C++ et permet sans doute à un débutant de démarrer assez rapidement.

Nous refermons cette parenthèse sur le C++, qui est considéré comme un terrain connu, et commençons l'exploration des contrées désertiques avec Ocaml.

3.3 Présentation du langage Ocaml

Nous nous sommes largement inspirés de deux sources pour cette partie : la page ouèbe de Pierre Weis [96] et le livre de cours de Xavier Leroy et Pierre Weis [97][16]. Nous nous sommes également servi du gros ouvrage sur Ocaml [25].

Dans la Section 3.3.1, nous résumons l'essentiel à savoir sur le langage Ocaml. Ensuite, dans la Section 3.3.2, quelques mots sont dits concernant la programmation fonctionnelle. Un survol des traits saillants d'Ocaml est fait dans 3.3.3. Dans la Section 3.3.4, des exemples simples permettant de se familiariser avec la syntaxe, illustrent les notions introduites. Les Sections 3.3.5 et 3.3.6 sont dédiées à des exemples d'utilisation de deux fonctions Ocaml qui possèdent des versions parallèles dans le système OcamlP3l : respectivement `map` et `fold`. Enfin, la Section 3.3.7 permet de toucher du doigt une petite subtilité qui sera utile pour la description d'OcamlP3l : la notion d'initialisations locale et globale.

3.3.1 Introduction : l'essentiel sur Ocaml

Nous présentons dans cette section quelques caractéristiques du langage Ocaml qui nous serviront pour décrire l'environnement OcamlP3l. Il est en effet

[16]Outre sa clarté et sa rigueur, ce livre sera apprécié pour son humour, démontrant ainsi définitivement et royalement qu'informatique et détente peuvent être fortement corrélées. Je conseille notamment la lecture du Chapitre 7, où les auteurs décrivent comment programmer un psychanalyste informatique...

indispensable d'être familier avec un certain nombre de concepts pour pouvoir comprendre ce qui est expliqué dans la Section 3.4. Nous pensons que cette présentation d'Ocaml est justifiée par la méconnaissance de ce langage dans le milieu numérique et les forts a priori négatifs qui lui sont parfois associés. Une exception toutefois mérite d'être mentionnée : Chandrasekaran et Gu ont développé camlFloat[17], une interface en Ocaml pour les librairies Lapack [11], et Blas [47], qui peut être appelé par un programme, ou être utilisé interactivement, créant ainsi un environnement similaire à Matlab, mais avec accès à la puissance d'Ocaml. Les auteurs semblent s'en servir avec succès, voir par exemple [26] et les références sur la page ouèbe de camlFloat.

Un lecteur pressé qui a déjà des connaissances sur Ocaml pourra sans doute sauter cette partie. Au besoin, il pourra se reporter au petit lexique en Annexe C.1, où certains termes sont explicités, et aux exemples simples de la Section 3.3.4. Ce qu'il doit savoir sur Ocaml peut être résumé en quelques lignes : Ocaml est un langage de programmation fonctionnel*, sûr, fortement typé, avec des types polymorphes*, qui permet également de coder de façon impérative*, et qui a une gestion automatique de la mémoire. Une explication de ces qualificatifs, en compagnie d'autres, est fournie dans la Section 3.3.3.

Je tiens à souligner que le *typage fort* est l'atout majeur que je perçois dans Ocaml. Alors que le programmeur ne déclare jamais les types de ses variables et fonctions, tout est typé. Le compilateur infère les types à partir du programme source et vérifie leur concordance. Cette vérification systématique et automatique peut être vue comme une contrainte par un débutant, dérouté par le fait que son programme ne compile pas à cause de ce qu'il peut considérer de petites fautes de type. En fait, le typage fort se révèle en fait particulièrement agréable : il permet d'éviter et de corriger de nombreuses erreurs dès la compilation et apporte un confort et une sécurité indéniables dans la programmation.

De manière générale, la propriété fondamentale du langage Ocaml est sa "sûreté". Essayons en quelques lignes de le faire sentir : par exemple, les tableaux et vecteurs sont systématiquement initialisés ; les défauts d'accès sont impossibles, les données sont au contraire toujours valides, c'est-à-dire que le programmeur ne peut pas déborder d'un tableau ou créer un pointeur qui pointe sur n'importe quoi ; Ocaml permet de composer facilement des structures de données *sans effet de bord**[18]. Ainsi, la fonction `map` —*muliple apply*— permet d'appliquer une même fonction à tous les éléments d'un tableau, comme nous le verrons sur un exemple à la Section 3.3.5. Nous nous servons de la version parallèle de cette fonction, notée `mapvector`*, qui est fournie par le système OcamlP3l. Un autre exemple avec la fonction `fold`*, dont le pendant parallèle se nomme `reduce-`

[17]Voir l'URL `http ://www.math.ucsb.edu/ lyons/camlFloat/`.

[18]Voir ci-dessous la Section 3.3.2 et le lexique en Annexe C.1.

`vector`* est détaillé à la Section 3.3.6.

3.3.2 Un mot sur la programmation fonctionnelle

La programmation *fonctionnelle**[19] est un style de programmation basé sur la manipulation de fonctions. Dans un langage fonctionnel, les fonctions *calculent* le résultat souhaité au sens des calculs mathématiques, c'est-à-dire par simplification successives d'une expression.

On parle de programmation fonctionnelle par opposition à la programmation *impérative**, qui est celle dont les numériciens programmant en Fortran, en C ou en C++ ont l'habitude. La programmation impérative est basée sur la modification de l'état mémoire de l'ordinateur : le programmeur manipule des tableaux ou des vecteurs qu'il remplit et qu'il transforme avec des boucles et des affectations au cours de l'exécution du code. Des *effets**, ou *effets de bord*, sont réalisés en permanence.

Bien que la programmation fonctionnelle ne soit pas dans les habitudes des numériciens et paraîsse étrange au premier abord, elle s'avère en fait très naturelle, car très proche des mathématiques. Elle confère indubitablement une certaine assurance dans la programmation, parce qu'elle est *sûre* : il est possible de *prouver* des théorèmes sur des programmes dans le cadre fonctionnel, et cela est notoirement plus difficile, voire impossible, dans le cadre de la programmation orientée objets.

Je suis persuadé qu'il est possible de faire de très belles choses dans un code numérique en combinant la programmation impérative et la programmation fonctionnelle, et le langage Ocaml semble vraiment l'outil adapté.

3.3.3 Bref aperçu du langage Ocaml

Caml est un langage de programmation, développé et distribué par l'INRIA depuis 1984. Il existe en fait deux dialectes de Caml : Caml Light et Objective Caml, que nous utilisons et qui se contracte en Ocaml. Caml Light est un sous-ensemble d'Objective Caml, qui est plus spécialement adapté à l'enseignement et à l'apprentissage de la programmation. En plus du coeur du langage de Caml Light, Objective Caml comporte un puissant système de modules, des objets et un compilateur optimisant. Cette précision faite, nous commettons un abus systématique dans ce mémoire en employant Ocaml indifféremment pour parler d'Objective Caml en tant que tel, et du *langage* Caml. Qu'on veuille bien nous pardonner cette simplification.

[19]Se référer à l'Annexe C.1 pour une définition un peu plus détaillée.

Comme nous n'expliquerons pas aussi bien ce qu'est le langage Ocaml qu'un de ses concepteurs, nous laissons la plume à Pierre Weis, [96]. Des précisions que nous avons jugées utiles ont été ajoutées afin de rendre la lecture plus aisée pour un néophyte en Ocaml.

Sûreté Le langage Ocaml est très sûr. Le compilateur fait de nombreuses vérifications à la compilation des programmes et pendant l'exécution. De nombreuses erreurs de programmation deviennent ainsi impossibles en Ocaml : confusions de types de données, accès erronés à l'intérieur des données par exemple. En effet, tous ces points sont vérifiés et gérés automatiquement par le compilateur, ce qui garantit l'intégrité parfaite des données manipulées par les programmes.

Ocaml est typé statiquement*, c'est-à-dire que la vérification de la concordance des types est réalisée au moment de la compilation. Cependant, il est inutile d'ajouter des informations de type dans les programmes (comme en Ada, en Pascal ou en C) : les annotations de typage sont automatiquement calculées par le compilateur.

Types de données Il existe en Ocaml de nombreux types de données prédéfinis :

- types de base : entiers, flottants, booléens, caractères, chaînes de caractères.
- types de données plus complexes : n-uplets, tableaux, listes, ensembles, tables de hachage, files, piles, flux de données.

Au-delà de ces types prédéfinis, Ocaml propose de puissants moyens de définir de nouveaux types : types enregistrements, types énumérés, et types sommes généraux. Les types sommes sont une généralisation des types unions, à la fois simple, sûre et facile à maîtriser. Ils permettent la définition de types de données qui présentent des valeurs hétérogènes repérées par des *constructeurs de valeurs*.

Au gré du programmeur, tous ces types sont définissables *concrètement* (les constructeurs sont disponibles à l'extérieur du module) ou *abstraitement* (l'implémentation est restreinte au module de définition et les constructeurs sont invisibles de l'extérieur).

Ce mécanisme autorise un contrôle fin du degré d'*encapsulation* des données manipulées par les programmes, ce qui est indispensable pour la programmation à grande échelle.

Fonctions Ocaml est un langage de programmation fonctionnel : il n'y a pas de restriction à la définition et à l'usage des fonctions, qu'on peut librement passer en argument ou retourner en résultat dans les programmes.

Gestion mémoire automatisée et incrémentale Ocaml offre une gestion automatique de la mémoire : l'allocation et la libération des structures de données est implicite (il n'y a pas de primitives de manipulation explicite de la mémoire comme "new" ou "free" ou "dispose"), et laissée à la charge du compilateur. On obtient ainsi une programmation bien plus sûre, puisqu'il n'y a jamais de corruption inattendue des structures de données manipulées.

De plus le gestionnaire mémoire opère en parallèle avec l'application, sans jamais l'arrêter de façon notable (récupération mémoire incrémentale).

Traits impératifs Ocaml offre la panoplie complète des traits de la programmation impérative, en particulier les tableaux modifiables en place, les boucles et les variables affectables, les enregistrements avec champs physiquement modifiables.

Compilateur rapide, code exécutable rapide Ocaml propose un compilateur de fichiers, et la compilation séparée est assurée par un système de modules. De surcroît, le compilateur Ocaml comporte une option qui maximise la vitesse de compilation, la portabilité des programmes obtenus, et minimise la taille des exécutables (compilation en code-octets).

Le compilateur d'Objective Ocaml comporte en plus une option "optimisante" qui privilégie la vitesse d'exécution (compilation en code natif) : le compilateur optimisant d'Objective Ocaml produit des programmes dont la vitesse d'exécution est digne des meilleurs compilateurs disponibles actuellement.

Interactivité Ocaml offre également un système interactif (une boucle de lecture-évaluation-impression des résultats), qui est très pratique pour apprendre le langage ou essayer et corriger des bouts de programmes : il n'y a pas besoin d'utiliser forcément des fichiers, ni d'ajouter des ordres d'impression dans les programmes puisque les résultats sont imprimés automatiquement par le système interactif.

Forte capacité de traitement symbolique Ocaml vous propose la "programmation orientée par filtrage" : cette puissante méthode de programmation est une généralisation de l'analyse de cas traditionnelle[20], qui est maintenant disponible pour tous les types de données du langage. Le mécanisme de filtrage est un moyen concis et élégant de "tester et de nommer" les données en une seule opération. Le compilateur Ocaml tire avantage de ce trait unique pour faire de nombreuses vérifications sémantiques sur le code qui lui est soumis :

[20]du genre `if, then, else`, ou encore `switch(), case`

le *vérificateur de filtrage* du compilateur est capable de détecter les branches inutiles des analyses de cas (“ce cas ne se présentera jamais à l’exécution”) et, plus étonnant encore, *de détecter les cas oubliés* (“ce cas n’est pas envisagé par votre programme”). Ainsi, le vérificateur de filtrage met souvent le doigt sur de subtiles erreurs qui se glissent dans les programmes. De surcroît, le vérificateur de filtrage est capable d’*apporter la preuve* de couverture exhaustive des cas des programmes qui utilisent le filtrage.

Le filtrage apporte un confort inégalé dans le traitement symbolique des données.

Le vérificateur de filtrage procure un niveau de sécurité dans la programmation et un degré de qualité inégalés des programmes qui manipulent des données symboliques.

Traitement des erreurs Ocaml possède un mécanisme général d’exceptions, pour traiter ou corriger les erreurs ou les situations exceptionnelles.

Mise au point des programmes Plusieurs méthodes de mise au point des programmes s’offrent à vous en Ocaml :

- le système interactif offre une méthode élémentaire mais très simple et rapide pour tester des (petites) fonctions : on vérifie simplement les résultats obtenus sur quelques exemples tapés directement dans la boucle d’interaction.
- dans les cas plus complexes, le système interactif permet également à très peu de frais de suivre la progression des calculs avec le mécanisme de *trace des appels de fonctions.*
- enfin le *débogueur symbolique avec retour arrière* permet de suivre très finement le déroulement de l’exécution, de l’arrêter à tout moment pour examiner l’état courant des variables et des fonctions en attente, et même de *revenir en arrière* dans les calculs pour reprendre l’exécution au moment où un évènement intéressant se produit.

Polymorphisme *

Ocaml est doté d’un puissant typage “polymorphe” : certains types peuvent rester indéterminés, représentant alors “n’importe quel type”.

Ainsi, les fonctions et procédures qui sont d’usage général s’appliquent à n’importe quel type de données, sans exception (par exemple les routines de tri s’appliquent à tout type de tableaux).

Méthode d’évaluation Ocaml est un langage “strict”*, par opposition aux langages paresseux*. Ceci signifie que les arguments d'une fonction sont complètement évalués avant de lui être appliqués. Cependant la pleine fonctionnalité permet de créer des suspensions et donc de coder l'évaluation paresseuse de données potentiellement infinies.

Programmation en vraie grandeur Les programmes Ocaml sont formés d'unités de compilation que le compilateur compile séparément. Cette organisation est parfaitement compatible avec l'utilisation d'outils traditionnels de gestion de projets (comme l'utilitaire make d'Unix). Le système de module du langage est puissant et sûr (toutes les interactions entre modules sont statiquement vérifiées par le contrôleur de types). Les modules d'Objective Ocaml peuvent comporter des sous-modules (à un degré d'emboîtement quelconque) et les fonctions des modules dans les modules sont autorisées (ce qui permet de définir des modules paramétrés par d'autres modules).

Programmation orientée objets Objective Caml propose des *objets* qui permettent d'utiliser le style *orienté objets* dans les programmes Ocaml. Fidèle à la philosohie du langage, cette extension orientée objets obéit au paradigme du "typage fort" : en conséquence, aucune méthode ne peut être appliquée à un objet qui ne pourrait y "répondre" ("les méthodes sont toujours bien comprises"). Encore une fois, cette vérification systématique du compilateur évite de nombreuses erreurs. Ceci offre au programmeur Ocaml, outre un confort insoupçonné dans l'écriture de ses programmes orientés objets, un niveau inégalé de qualité des programmes qu'il produit.

Puissantes bibliothèques De nombreuses bibliothèques et contributions sont disponibles en Ocaml, en particulier des primitives de dessin indépendantes de la machine (libgraph), une arithmétique rationnelle exacte en multi-précision (camlnum), et de nombreuses interfaces avec des technologies bien connues : générateurs d'analyseurs lexicaux et syntaxiques avec camllex et camlyacc. Sous Unix, on dispose aussi d'un débogueur avec retour arrière, d'un navigateur dans les fichiers sources (camlbrowser), d'une interface graphique à l'aide de Tk/Tcl (camltk) et d'une interface poussée avec le système (libunix).

3.3.4 Quelques lignes d'Ocaml

Les exemples de cette section ont été tirés du site internet de présentation du langage Ocaml, [96]. Je me suis à nouveau permis d'ajouter aux commentaires de Pierre Weis quelques remarques supplémentaires, afin, d'une part, de clarifier certaines choses qui pourraient être mal comprises, et, d'autre part, dans le but d'en mettre d'autres en valeur qui pourraient passer inaperçues.

Fonctions élémentaires Avec le système interactif, nous définissons la fonction carré et la fonction factorielle récursive. Puis nous essayons ces nouvelles fonctions sur quelques exemples. Le langage détermine automatiquement les types des arguments des fonctions sans que l'utilisateur ait besoin de les déclarer explicitement. Le système interactif renvoie à la fin de chaque commande le type de la fonction qui a été définie. Les fonctions `carré` et `fact` sont des fonctions qui prennent en entrée un argument de type entier et retournent en sortie un entier.

```
#let carré (x) = x * x;;
carré : int -> int = <fun>
#let rec fact (x) =
  if x <= 1 then 1 else x * fact (x - 1);;
fact : int -> int = <fun>
#fact (5);;
- : int = 120
#carré (120);;
- : int = 14400
```

Gestion automatique de la mémoire Toutes les opérations d'allocation et de libération de la mémoire sont complètement automatiques. Donnons l'exemple des listes.

Les listes sont prédéfinies en Ocaml, la liste vide est notée `[]`, et le constructeur d'ajout d'un élément à une liste est `::` (opérateur binaire et infixe).

```
#let l = 1 :: 2 :: 3 :: [];;
l : int list = [1; 2; 3]
#[1; 2; 3];;
- : int list = [1; 2; 3]
#5 :: l;;
- : int list = [5; 1; 2; 3]
```

Polymorphisme : le tri des listes Le tri par insertion est défini à l'aide de deux fonctions récursives, en utilisant le filtrage sur les listes. Le constructeur de type polymorphe `'a`, qu'on lit “alpha”, permet de travailler avec n'importe quel type. La liste de type `'a list` peut donc être tour à tour une liste d'entiers ou de chaînes de caractères[21].

```
#let rec tri = function
   | [] -> []
   | x :: l -> insert x (tri l)

and insert e = function
  | [] -> [e]
  | x :: l -> if e < x then e :: x :: l else x :: insert e l;;
tri : 'a list -> 'a list = <fun>
insert : 'a -> 'a list -> 'a list = <fun>
#tri [2; 1; 0];;
- : int list = [0; 1; 2]
#tri ["oui"; "ok"; "sûr"; "ouais"; "certain"];;
- : string list = ["certain"; "ok"; "ouais"; "oui"; "sûr"]
```

Programmation impérative La programmation impérative est la programmation à laquelle tout numéricien codant en Fortran, en C ou en C++ est habitué. Il est donc possible de travailler en Ocaml de manière purement impérative. L'algorithme de Bi-CGStab a par exemple été reprogrammé en Ocaml de façon habituelle.

Les polynômes sont représentés comme des vecteurs ; nous les ajoutons en créant d'abord le polynôme somme et en remplissant ensuite ses cases à l'aide de deux boucles "for".

```
#let ajoute_polynômes p1 p2 =
   let result = make_vect (max (vect_length p1) (vect_length p2)) 0 in
   for i = 0 to vect_length p1 - 1 do result.(i) <- p1.(i) done;
   for i = 0 to vect_length p2 - 1 do result.(i) <- re-
sult.(i) + p2.(i) done;
   result;;
ajoute_polynômes : int vect -> int vect -> unit
#ajoute_polynômes [| 1; 2 |] [| 3; 4 |];;
- : int vect = [|4; 6|]
```

[21]Un habitué du C++ pourrait trouver une ressemblance avec les modèles ou Templates. L'idée est similaire, bien que le typage soit incomparablement plus strict et plus exact dans Ocaml.

Les accumulateurs sont définis en Ocaml à l'aide de cellules modifiables appelées références. L'expression `ref init` renvoie une nouvelle cellule, dont le contenu initial est `init`, `!cell` renvoie le contenu actuel de `cell`, et `cell := val` met dans `cell` la valeur `val`.

Définissons `fact` avec un accumulateur et une boucle "for" :

```
#let fact n =
   let result = ref 1 in
   for i = 1 to n do
    result := i * !result
   done;
   !result;;
fact : int -> int = <fun>
#fact 5;;
- : int = 120
```

Pleine fonctionnalité Il n'y a pas de restriction sur les fonctions, qui peuvent donc être d'ordre supérieur : une fonctionnelle, c'est-à-dire une fonction de fonctions, a donc un sens, de même qu'une fonction de fonctionnelles, et ainsi de suite...

Nous définissons d'abord la fonction `sigma` qui renvoie la somme des résultats de l'application d'une fonction `f` donnée aux éléments d'une liste :

```
#let rec sigma f = function
   | [] -> 0
   | x :: l -> f x + sigma f l;;
sigma : ('a -> int) -> 'a list -> int = <fun>
```

Les fonctions temporaires peuvent se définir comme des valeurs fonctionnelles anonymes à l'aide de la construction `function` :

```
#sigma (function x -> x * x) [1; 2; 3];;
- : int = 14
```

Combinée au polymorphisme, la pleine fonctionnalité permet de définir la composition des fonctions. On remarquera que les types sont corrects au sens de la composition mathématique de fonctions : `g` va de `'c` dans `'a`, et `f` va de `'a` dans `'b`, les ensembles de départ et d'arrivée sont donc concordants.

```
#let compose f g = function x -> f (g (x));;
compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

```
#let carré_o_fact = compose carré fact;;
carré_o_fact : int -> int = <fun>
#carré_o_fact 5;;
- : int = 14400
```

3.3.5 Utilisation de map

La fonction map permet d'appliquer une fonction à toutes les composantes d'un tableau . Voici une fonction calculant le carré d'un tableau d'entiers :

```
# let a = [| 2 ; 4 ; 6|];;
val a : int array = [|2; 4; 6|]
# let carré_tab v = Array.map carré v ;;
val carré_tab : float array -> float array = <fun>
# carré_tab a ;;
- : int array = [|4; 16; 36|]
# let add1 x = x + 1;;
val add1 : int -> int = <fun>
# let carré_o_add1_tab v = Array.map ( compose carré add1 ) v ;;
val carré_o_add1_tab : int array -> int array = <fun>
# carré_o_add1_tab a ;;
- : int array = [|9; 25; 49|]
```

3.3.6 Un peu de pliage avec fold

Les fonctions fold_left et fold_right permettent de "replier" un tableau en un scalaire en accumulant une fonction sur toutes ses composantes. Voici une fonction calculant la norme l^2 d'un vecteur de réels, avec la fonction fold_left :

```
# let f x y = sqrt (x *. x +. y *. y);;
val f : float -> float -> float = <fun>
# let norm v = Array.fold_left f 0. v;;
val norm : float array -> float = <fun>
# let a = [| 4.; 3.|];;
val a : float array = [|4; 3|]
# norm a;;
- : float = 5
```

3.3.7 Initialisation locale et globale

Nous illustrons d'un exemple simple le fonctionnement de l'initialisation locale ou globale, qui servira pour l'implémentation en OcamlP3l de la méthode de décomposition de domaines, voir la Section 3.4.5. Définissons deux fonctions, `f` et `g`. La fonction `f` fait une "initialisation locale", ici une impression à l'écran. L'impression n'est pas effectuée au moment de la définition de `f`, mais à chaque fois qu'on appelle `f` avec son argument `()`. En revanche, la fonction `g` a une "initialisation globale" : elle imprime du texte à l'écran uniquement lors de sa définition.

Nous notons que la fonction `f`, comme la fonction `g` d'ailleurs, dépend de deux arguments de types respectifs `unit` et `int`. Le type `unit` est un ensemble contenant un seul élément noté `()`, d'où son nom. Par conséquent, les fonctions `f1 = f ()` et `g1 = g ()` sont des fonctions *partielles* prenant un argument de type `int` et retournant un `int`. On note que `f1` n'imprime son texte à l'écran qu'une seule fois, au moment de sa définition. Ceci est normal puisque la définition de la fonction partielle nécessite l'évaluation de la commande d'impression, et qu'ensuite, la fonction partielle n'est plus qu'une fonction incrémentant un entier.

```
# let f () = print_string "Appel de la fonction f!! " ;
             fun x -> x + 1;;
val f : unit -> int -> int = <fun>
# let g = print_string "Appel de la fonction g!! " ;
          fun () -> fun x -> x + 1;;
Appel de la fonction g!! val g : unit -> int -> int = <fun>
# g ();;
- : int -> int = <fun>
# let g1 = g ();;
val g1 : int -> int = <fun>
# g () 4 ;;
- : int = 5
# g1 4 ;;
- : int = 5
# f () ;;
Appel de la fonction f!! - : int -> int = <fun>
# let f1 = f () ;;
Appel de la fonction f!! val f1 : int -> int = <fun>
# f () 77;;
Appel de la fonction f!! - : int = 78
```

```
# f1 23 ;;
- : int = 24
```

3.4 Le système OcamlP3l

Nous présentons dans cette section les principales caractéristiques de l'environnement OcamlP3l. Pour l'essentiel, nous nous sommes inspirés de l'article [35] dont nous reproduisons en français des parties consacrées au système OcamlP3l. Nous pensons en effet qu'il mérite une large diffusion dans le monde numérique. Peut-être ce mémoire y contribuera-t-il modestement.

Nous commençons dans la Section 3.4.1 par définir ce qu'est le système OcamlP3l. Les trois sémantiques*, séquentielle, parallèle et graphique, d'OcamlP3l sont ensuite introduites dans la Section 3.4.2. La Section suivante 3.4.3 est consacrée à la description des propriétés des squelettes*— *skeletons*— et en particulier leur interprétation en tant que fonction agissant sur des flux de données. Dans la Section 3.4.4, la liste des différentes sortes de squelettes d'OcamlP3l permet de faire entrer en scène une première fois ces squelettes. Ils sont entièrement détaillés dans la Section 3.4.5 ; leur syntaxe et leurs types montrent la première des trois modifications importantes apportées par les chercheurs en informatique pour satisfaire nos besoins spécifiques pour la décomposition de domaines : la transformation des squelettes en *générateurs* de squelettes, capables d'assurer une initialisation *locale* des fonctions. La seconde nouveauté importante est la construction du squelette `parfun`, dont l'utilité et les caractéristiques sont montrées dans la Section 3.4.6. En un mot, ce `parfun` facilite grandement la programmation des algorithmes parallèles en autorisant la création de sous-réseaux parallèles. Il manque un dernier mot-clef pour compléter la panoplie d'OcamlP3l : le `pardo`, auquel est dédiée la Section 3.4.7. La structure générale que doit posséder un programme OcamlP3l est également décrite dans cette Section 3.4.7. La dernière nouveauté, dont nous nous sommes servis dans les simulations numériques, est la notion de *couleur**, que détaille la Section 3.4.8. Les couleurs permettent d'introduire un degré de réglage manuel dans l'affectation automatique des tâches aux différents nœuds d'un calculateur parallèle ; ceci s'avère particulièrement intéressant pour l'équilibrage des charges. Enfin, car tout a une fin, la Section 3.4.9 termine cette longue liste en expliquant comment utiliser OcamlP3l dans la pratique ; un script montre d'ailleurs la facilité d'emploi du système.

3.4.1 Introduction : description générale du système

Dans un système de programmation parallèle basée sur des squelettes [33, 40, 38], un ensemble de *squelettes* est fourni au programmeur. Un squelette est une fonction de second ordre, c'est-à-dire une fonctionnelle du premier ordre*, représentant l'un des modèles courants d'un code parallèle. Le programmeur doit utiliser les squelettes pour donner la structure parallèle désirée à son application et utilise un langage purement séquentiel pour exprimer les portions séquentielles de son code parallèle, qui deviendront des paramètres pour les squelettes. Il n'est pas possible de donner des activités parallèles au programme autrement qu'avec les squelettes : on ne peut pas créer explicitement des processus, ni les synchroniser, ni y mettre fin. Il n'existe pas de notion de mémoire partagée, ni de primitives de communication. En fait, il n'y a aucune indication de l'exécution du programme sur une architecture parallèle. Ceci décharge l'utilisateur de toute responsabilité quant à l'écriture de code créant des processus parallèles, affectant les tâches, synchronisant les processus sur le matériel utilisé, établissant des réseaux de communication (canaux, localisation de la mémoire partagée, etc.) ou mettant en place réellement les communications entre processus. Toutes ces tâches parfaitement génériques et donc automatisables qui sont nécessaires pour mettre en œuvre le squelette sur le matériel, sont à la charge du compilateur et du support d'exécution de l'environnement de programmation par squelettes.

OcamlP3l est un environnement de programmation qui permet d'écrire des programmes parallèles en Ocamlavec les modèles de squelettes du langage p3l[22]. Il montre une fusion possible entre la programmation parallèle et la programmation fonctionnelle. De plus il propose des fonctionnalités avancées comme le débogage logique séquentiel d'un code parallèle —c'est-à-dire le débogage fonctionnel du programme parallèle via l'exécution de l'architecture d'un code parallèle sur une machine séquentielle—, et le typage fort, qui est utile à la fois pour l'enseignement de la programmation parallèle et pour la construction d'applications réelles.

Dans OcamlP3l, comme dans tous les systèmes basés sur des squelettes, l'utilisateur décrit la structure parallèle du code au moyen d'un ensemble de squelettes. Une des caractéristiques spécifiques d'OcamlP3l est que la sémantique de ces squelettes n'est pas figée. Le système permet à l'utilisateur de compiler son code sans aucune modification des sources, en choisissant parmi diverses sémantiques, comme nous allons le voir immédiatement.

[22] Voir l'URL `http ://www.di.unipi.it/.susanna/p3l.hmtl`.

3.4.2 Trois sémantiques fortement reliées

Comme c'est décrit dans l'article originel [39], il existe trois sémantiques possibles pour tout programme OcamlP3l.

sémantique séquentielle le programme est compilé et lié avec une implémentation séquentielle des squelettes. De cette façon, l'exécutable obtenu peut tourner sur une seule machine, comme un seul processus, et ainsi être débogué de manière usuelle avec les outils courants pour les programmes séquentiels,

sémantique parallèle le programme est compilé et lié avec une implémentation parallèle des squelettes. L'exécutable obtenu est un programme SPMD générique —*Single Program Multiple Data*, un seul programme, plusieurs données—, et peut être déployé sur une machine parallèle, une grappe de PC ou un réseau de stations de travail,

sémantique graphique le programme est compilé et lié avec une implémentation graphique des squelettes, de sorte que l'exécutable obtenu, quand il est lancé, affiche une représentation du réseau de calcul parallèle qui est mis en place quand on exécute la version parallèle.

Du point de vue de l'utilisateur, ces trois sémantiques différentes sont obtenues simplement en compilant le même programme source avec trois options différentes du compilateur.

Bien évidemment, le but est de garantir que les exécutions séquentielle et parallèle sont en accord : quel que soit le programme créé par l'utilisateur, les deux sémantiques doivent produire exactement les mêmes résultats. La force d'Ocaml est qu'il est possible d'essayer de le *démontrer*, prouvant ainsi un *théorème*.

3.4.3 Les squelettes en tant que fonctions sur des flux

Les squelettes d'OcamlP3l sont *compositionnels**, c'est-à-dire que l'on peut toujours les combiner entre eux pour en former un autre. Les squelettes sont des combinateurs* —*combinators*— qui forment une algèbre de fonctions et fonctionnelles, appelée le *langage squelettique*[23].

Pour être précis, un squelette est une *fonction sur des flux*[24] —*stream processor*—, c'est-à-dire une fonction qui transforme un flux de données d'entrée en un flux de données de sortie. Ces fonctions peuvent être composées entre

[23] Pourquoi ne pas utiliser un adjectif quand il existe ? Aussi qualifiera-t-on de *squelettique* tout ce qui se rapporte à un squelette OcamlP3l.

[24] Le flux ici a le sens informatique de flot d'informations envoyées et reçues par un processus, et n'a pas de lien avec les flux physiques qui sont calculés avec les équations de Darcy.

elles arbitrairement, définissant de cette manière le comportement parallèle du programme.

Cette vision des squelettes en tant que combinateurs est intéressante notamment parce que si l'on montre que chacun des combinateurs est “correct”, cela *prouvera* en même temps que n'importe quel programme que pourrait concevoir un utilisateur fait ce qu'il est censé faire.

L'interprétation des squelettes comme des fonctions sur des flux est évidente quand on observe leurs types. De plus la nature compositionnelle des squelettes apparaît également de façon claire dans leur implémentation :

Pour l'implémentation de la sémantique parallèle , un squelette est mis en œuvre en tant que processeur de flux —*stream processors*— éventuellement paramétrée par des fonctions d'initialisation ou d'autres processeurs de flux.

Pour l'implémentation de la sémantique séquentielle , le système définit un type abstrait des flux de données : le type polymorphe* `'a stream`. L'implémentation séquentielle des squelettes est alors définie comme un ensemble de fonctions sur ces flux.

3.4.4 Les combinateurs squelettiques dans OcamlP3l

Dans la version actuelle d'OcamlP3l, les combinateurs, ou briques de base, du langage squelettique sont de cinq sortes :

- les squelettes de *tâches parallèles* qui modélisent le parallélisme d'activités *indépendantes* affectant des données d'entrée différentes. Dans cet ensemble sont regroupés les combinateurs `pipe`*et `farm`*, qui correspondent aux squelettes de tâches parallèles apparaissant à la fois dans p3l et dans d'autres modèles de squelettes [33, 40, 41].
- les squelettes de *données parallèles* qui modélisent le calcul parallèle sur différentes parties de la même donnée. Dans cet ensemble, OcamlP3l regroupe `mapvector`*et `reducevector`*. Le squelette `mapvector` modélise l'application parallèle d'une fonction générique f à toutes les composantes d'une structure vectorielle de donnée, tandis que le squelette `reducevector` modélise un calcul parallèle qui accumule —*fold**— les éléments d'un vecteur avec un opérateur binaire commutatif et associatif ($\oplus$).
- les squelettes *d'interface de données* qui opèrent l'injection et la projection entre le monde séquentiel et le monde parallèle : `seq`*convertit une fonction séquentielle en un nœud du réseau de calcul parallèle, et inversement `parfun`*convertit un réseau de calcul parallèle en une fonction agissant sur des flux.

- le squelette de *délimitation du champ d'exécution parallèle*, le combinateur `pardo`*, qui doit encapsuler tout le code qui invoque un `parfun`.
- le squelette de *contrôle*, le combinateur `loop`*, qui permet l'exécution répétitive d'une expression squelettique donnée. (`loop` n'est pas une entité parallèle en soi).

3.4.5 Syntaxe des squelettes, sémantique et types

Nous décrivons brièvement ici la syntaxe, la sémantique informelle et les types affectés à chaque combinateur du langage squelettique.

Les combinateurs comme générateurs de squelettes (nouveauté)

Tout d'abord, nous allons expliquer pour quelle raison les types Ocaml des squelettes sont un peu plus complexes que ce qu'on aurait pu penser au premier abord. En effet, ces types sont en quelque sorte pollués par rapport à ce qu'on attendrait, par l'ajout de types `unit`. Cette complexité additionnelle n'est évidemment pas gratuite : elle a été rendue nécessaire pour pouvoir implémenter les fonctionnalités nouvelles qui étaient indispensables pour faire fonctionner de façon efficace l'application qui nous intéresse : la décomposition de domaines.

Pour expliquer la raison de ce type `unit`, nous nous restreignons au squelette le plus simple, le combinateur `seq`. Comme nous l'avons mentionné plus haut, `seq` encapsule toute fonction Ocaml f pour en faire un processus séquentiel qui applique f à toute donnée venant d'un flux de données d'entrée. Quand on écrit `seq f`, toute fonction Ocaml de type `f : 'a -> 'b` [25] est immergée dans un processus séquentiel. De cette manière, il serait "naturel" pour `seq` d'avoir le type
`('a -> 'b) -> 'a stream -> 'b stream.` [26]
Cependant, en OcamlP3l, `seq` est déclaré comme

```
seq : (unit -> 'a -> 'b) -> unit -> 'a stream -> 'b stream
```

ce qui signifie que la fonction argument `seq` prend un argument supplémentaire de type `unit`. En effet, dans les applications réelles, les fonctions employées par l'utilisateur peuvent avoir besoin de données locales qui doivent être créées à

[25] La fonction `f` prend un argument de type quelconque `'a` en entrée et retourne un argument de type quelconque `'b`, a priori différent de celui d'entrée. Voir la définition du polymorphisme dans le lexique de l'Annexe C.1.

[26] Dans cette hypothèse, `seq` retournerait une fonction agissant sur des flux : `seq f` prendrait en entrée un flux de données ayant le *même* type `'a` que l'argument d'entrée de `f`. Idem pour pour le flux de sortie. Par exemple, si `f` est de type `int -> float`, `seq f` serait de type `int stream -> float stream`.

un instant ou à un autre. Il est nécessaire de permettre à l'utilisateur de décrire finement quand et comment ces données doivent être initialisées ou copiées.

Dans le cadre de la méthode de décomposition de domaines, il est clair que la phase d'initialisation est fondamentale : on lit le maillage du sous-domaine, on construit la matrice d'éléments finis associée et on la factorise. Cette phase est coûteuse en temps et en mémoire : il est indispensable qu'elle ne soit effectuée qu'une fois pour toute au tout début de la méthode itérative. Sans cet argument supplémentaire `unit`, à chaque appel du produit matrice vecteur dans la méthode itérative —voir l'Algorithme 2—, il aurait fallu refaire cette phase d'initialisation pour chaque sous-domaine. En effet, l'ajout de cet argument `unit` permet d'initialiser *localement* les solveurs de sous-domaine.

Cette modification a donc été faite par Roberto Di Cosmo et Pierre Weis sur notre demande. Ils ont ainsi utilisé des techniques de programmation fonctionnelle pour initialiser ou allouer des données globalement et/ou localement à une fonction.

Ce qui suit est assez technique. Pour faciliter la compréhension, nous avons inséré un exemple simple illustrant les mécanismes des initialisations globale et locale dans la Section 3.3.7.

- *initialisation globale* : les données sont initialisées une fois pour toute et sont ensuite dupliquées dans chaque copie de la fonction sur les flux (*stream processor*) qu'un squelette `farm` ou `mapvector` peut lancer ; c'était déjà possible avec les *précédentes versions* d'OcamlP3l puisqu'on pouvait écrire
```
let f =
  let localdata = do_huge_initialisation_step () in
  fun x -> compute (localdata, x);;
...
farm (seq f, 10)
```
- *initialisation locale* : les données sont initialisées par chaque fonction sur les flux, *après* que la copie aura été effectuée par un squelette `farm` ou `mapvector` ; c'était tout simplement impossible dans les précédentes versions d'OcamlP3l ; avec le nouveau schéma, c'est à présent facile :
```
let f () =
  let localdata = do_huge_initialisation_step () in
  fun x -> compute (localdata, x);;
...
farm (seq f, 10)
```
quand le squelette `farm` crée dix copies de `seq f`, chaque copie est créée en passant () au combinateur `seq`, qui à son tour passe () à f, produisant

l'allocation d'une copie différente de *localdata* pour chaque instance.
Il faut noter que l'ancien comportement, c'est-à-dire une unique initialisation partagée par toutes les copies, peut toujours être aisément obtenu, et peut aussi être combiné avec d'autres initialisations locales si besoin est :

```
let f =
  let localdata = do_huge_initialisation_step () in
  fun () -> fun x -> compute (localdata, x);;
...
farm (seq f, 10)
```

Les deux expressions sont différentes parce que l'expression `seq f` effectue systématiquement l'appel $f()$. Dans le premier cas, l'initialisation n'est pas faite —elle sera donc locale—, tandis que dans le second elle est réalisée.

Pour résumer, le paramètre additionnel `unit` donne au programmeur la possibilité de décider si une initialisation locale de ses données doit être partagée par toutes les copies de ses fonctions ou non. En d'autres termes, les combinateurs dans la version actuelle d'OcamlP3l peuvent être vus comme des "squelettes retardés" ou des "usines à squelettes" qui produisent *une instance* d'un squelette chaque fois qu'un argument () leur est passé.

Typage et sémantique des squelettes

Nous pouvons à présent détailler les autres squelettes :

Le squelette farm calcule en parallèle une fonction f sur les différents éléments qui viennent sur son flux d'entrée.
D'un point de vue fonctionnel, étant donné un flux d'éléments $x_1, \ldots, x_n$, et une fonction f, l'expression $\texttt{farm}(f, k)$ opère le calcul $f(x_1), \ldots, f(x_n)$. Le parallélisme intervient en faisant calculer par k processus indépendants la fonction f sur différents éléments du flux d'entrée.
Si f a pour type `(unit -> 'a stream -> 'b stream)`, et k est de type `int`, alors $\texttt{farm}(f, k)$ est de type `unit -> 'a stream -> 'b stream`.

Le squelette pipeline est noté au moyen de l'opérateur infixe `|||` ; il calcule en parallèle les différentes phases d'une composition de fonctions sur les différents éléments du flux d'entrée.
Fonctionnellement, $f_1 \,|||\, f_2 \ldots |||\, f_n$ calcule $f_n(\ldots f_2(f_1(x_i))\ldots)$ sur tous les arguments x_i sur le flux d'entrée. Le parallélisme est obtenu dans ce cas en utilisant n processus parallèles indépendants. Chaque processus exécute une fonction f_i sur les arguments d'entrée qui sont les résultats du processus s'occupant du calcul de f_{i-1} et retourne sa sortie au processus qui calcule f_{i+1}.

Si f_1 a pour type `(unit -> 'a stream -> 'b stream)`,
et f_2 a pour type `(unit -> 'b stream -> 'c stream)`,
alors f_1`|||`f_2 est de type `unit -> 'a stream -> 'c stream`.

Le squelette map est appelé `mapvector` ; il calcule en parallèle une fonction sur toutes les composantes d'un vecteur, engendrant le —nouveau— vecteur de résultat[27].
Par conséquent, pour chaque vecteur X du flux d'entrée, `mapvector`(f, n) calcule la fonction f sur toutes les composantes de X, en utilisant n processus parallèles distincts qui calculent f sur des composantes distinctes du vecteur.
8 Si f a pour type `(unit -> 'a stream -> 'b stream)`, et n est de type `int`, alors `mapvector`(f, n) est de type `unit -> 'a array stream -> 'b array stream`.

Le squelette reduce est appelé `reducevector` ; il accumule une fonction sur toutes les composantes d'un vecteur pour renvoyer un scalaire, de la même manière qu'une fonction Ocaml *fold**. L'exemple le plus simple est sans doute la fonction qui renvoie la norme l^2 d'un vecteur de flottants, voir la Section 3.3.6.
Donc `reducevector`(f, n) calcule $x_1 f x_2 f \dots f x_n$ à partir du vecteur $x_1, \dots, x_n$, pour chaque vecteur du flux d'entrée. Le calcul est effectué en utilisant n processus parallèles différents qui calculent f.
Si f a pour type `(unit -> 'a * 'a stream -> 'a stream)`, et n est de type `int`, alors `reducevector`(f, n) est de type
`unit -> 'a array stream -> 'a stream`.

C'est le squelette `mapvector` que nous avons utilisé pour notre application de méthode de décomposition de domaines. Nous avons en effet besoin de garder *en ordre* les composantes du vecteur d'interface ; une synchronisation —implicite— existe au moment de la formation du vecteur de résultat. On note par ailleurs que le squelette `farm` n'est pas adapté dans ce cas puisqu'il considère les données dans son ensemble, sans ordre.

3.4.6 Construction du squelette `parfun` (nouveauté)

Dans le système p3l d'origine, un programme est clairement stratifié en deux couches : il y a un *chapeau* squelettique —*skeleton cap*—, qui peut être composé d'un nombre arbitraire de combinateurs, mais dès que l'on sort de ce chapeau, on retourne dans le code séquentiel grâce à l'utilisation du combinateur `seq` et il

[27] `mapvector` est une extension parallèle de la fonction Ocaml —nécessairement séquentielle— `map`*, voir la Section 3.3.5.

n'est pas possible pour le code séquentiel d'appeler un squelette. En deux mots, l'entrée d'un programme p3l est obligatoirement une expression squelettique, et aucune expression utilisant un squelette n'est autorisée ailleurs dans le code. Cette stratification est raisonnable dans le système p3l, où le but est de construire *un seul* réseau de calcul de flux décrit par le chapeau squelettique. Cependant elle entraîne des limitations :

- cela casse l'uniformité, car même si les squelettes *ressemblent* à de fonctions ordinaires, elles *ne peuvent pas* être employée comme des fonctions normales, en particulier au sein d'un code séquentiel,
- comme notre application l'illustre, beaucoup d'algorithmes sont constitués de simples boucles, dont certaines parties sont parallélisables, d'autres non ; le fait de forcer le programmeur à mettre tout le parallélisme dans le chapeau squelettique l'obligerait à réécrire tout l'algorithme d'une façon fort peu naturelle. Prenons l'exemple de Bi-CGStab utilisé comme accélérateur de la méthode de décomposition de domaines. C'est essentiellement une boucle, dont la seule partie parallèle est l'appel du produit matrice vecteur. Dans le cadre p3l décrit ici, il faudrait réécrire tout Bi-CGStab avec des squelettes dans le chapeau[28].
- Comme dans l'application, une opération parallélisable peut être utilisée à plusieurs endroits dans l'algorithme : le chapeau squelettique de p3l ne permet pas au programmeur de spécifier que des parties du réseau de calcul de flux de donnée peuvent être partagées entre différentes phases du calcul, ce qui est essentiel pour éviter de gaspiller les ressources informatiques.

Pour surmonter ces difficultés et limitations, la version `1.9` d'OcamlP3l introduit le squelette `parfun` qui est le *dual* du squelette `seq`. On peut envelopper une expression de squelettes complète dans un `parfun` et obtenir une fonction régulière agissant sur les flux, utilisable sans limitation dans toute partie séquentielle de code : un squelette encapsulé dans un `parfun` se comporte exactement comme une fonction normale qui reçoit un flux en entrée et renvoie un flux en sortie. Cependant, dans la sémantique parallèle, le combinateur `parfun` est doté d'une interprétation parallèle, de telle sorte que la fonction encapsulée est réellement implémentée comme un réseau parallèle, réseau dont le combinateur `parfun` fournit l'interface. Comme beaucoup d'expressions avec un parfun peuvent survenir dans un programme OcamlP3l, il peut exister plusieurs réseaux parallèles séparés au moment de l'exécution. Ceci implique que, à la différence de p3l, le modèle OcamlP3l de calculs requiert un *programme principal* séquentiel : ce programme principal a la responsabilité de l'échange de communications entre les différents squelettes encapsulés par des `parfun`.

[28]Pour çà, il aurait fallu travailler du chapeau...

On pourrait s'attendre à ce que `parfun` ait un type `(unit -> 'a stream -> 'b stream) -> 'a stream -> 'b stream` : étant donné une expression squelettique de type `(unit -> 'a stream -> 'b stream)`, `parfun` renvoie une fonction agissant sur des flux de type `'a stream -> 'b stream`.

En fait, le type réel de `parfun` introduit un niveau supplémentaire de fonctionnalité : l'argument n'est plus une expression squelettique, mais une fonctionnelle qui renvoie un squelette :

```
val parfun :
  (unit -> unit -> 'a stream -> 'b stream) ->
  'a stream -> 'b stream
```

Ceci est nécessaire pour garantir que le squelette englobé dans une expression `parfun` ne sera lancé et instancié que par le programme principal, et non par l'une des multiples copies du binaire SPMD qui sont en cours d'exécution. Il faut cependant que ces copies puissent évaluer les squelettes de `parfun` ; le programme principal créera effectivement les squelettes nécessaires en appliquant son argument fonctionnel, tandis que les copies génériques jetteront la fonctionnelle, évitant ainsi d'instancier les squelettes.

3.4.7 Délimiteur de champ parallèle : `pardo`

Typage de `pardo`

Enfin, le combinateur `pardo` définit le champ au sein duquel le programmeur peut utiliser les expressions encapsulées par un `parfun`.

```
val pardo : (unit -> 'a) -> 'a
```

`pardo` prend quelque chose en argument et renvoie le résultat de son évaluation. De la même manière que pour le combinateur `parfun`, ce retard supplémentaire est nécessaire pour assurer que l'initialisation du code sera prise en charge exclusivement par le programme principal et non par les copies génériques SPMD qui participent au calcul parallèle.

Règle de délimitation de champ parallèle

Les règles de délimitation des champs s'articulent autour de trois points :

- des fonctions définies par le combinateur `parfun` doivent être *définies avant* le combinateur `pardo`, celui-ci ne pouvant apparaître qu'une seule fois,

- ces fonctions définies par des `parfun` ne peuvent être *exécutées* qu'*au sein* du corps du paramètre fonctionnel du combinateur `pardo`,
- aucun `parfun` ne peut intervenir à l'intérieur d'un combinateur `pardo`.

Structure d'un programme OcamlP3l

Du fait de cette règle de délimitation des champs, la structure générale d'un programme OcamlP3l a l'apparence suivante :

```
(* (1) Functions defined using parfun *)
let f = parfun(skeleton expression)
let g = parfun(skeleton expression)

(* (2) code referencing these functions under abstractions *)

let h x = ... (f ...) ... (g ...) ...

...
(* NO evaluation of code containing a parfun
   is allowed outside pardo *)

(* (3) The pardo occurrence where parfun encapsulated
       functions can be called. *)
pardo
 (fun () ->
    (* NO parfun combinators allowed here *)

    (* code evaluating parfun defined functions *)
    ...
    let a = f ...
    let b = h ...
    ...
 )
(* finalization of sequential code here *)
```

Au moment de l'exécution, dans le modèle séquentiel, chaque copie générique attend juste les instructions du nœud principal ; celui-ci évalue d'abord les arguments des combinateurs `parfun` pour construire une représentation des squelettes utilisés. Ensuite, dès qu'il rencontre le combinateur `pardo`, le nœud principal initialise tous les réseaux parallèles de calcul, particularisant les copies

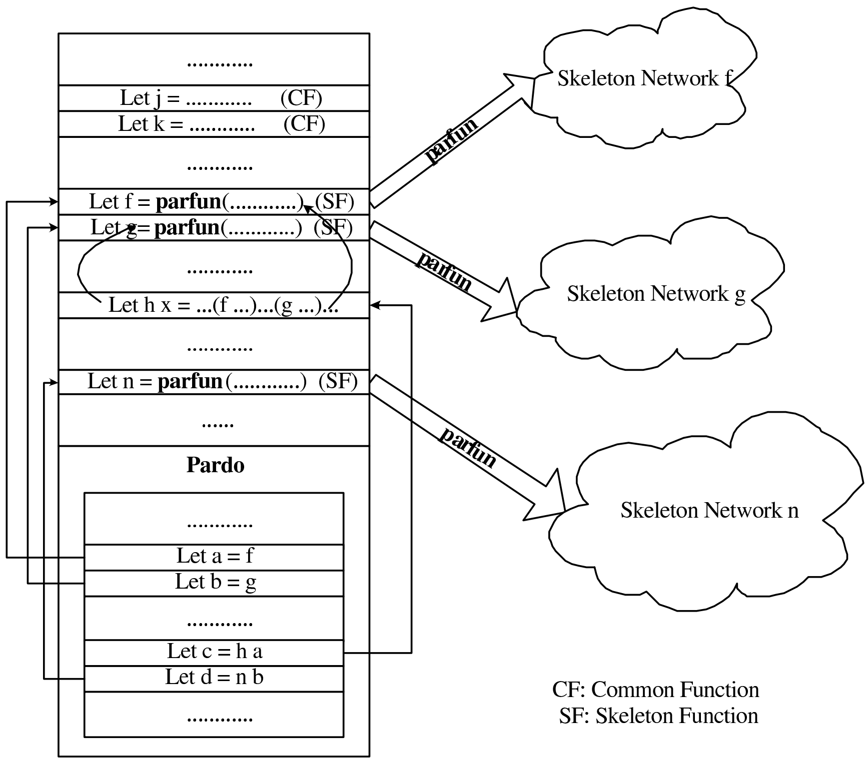


FIG. 3.1 – `parfun` et `pardo` : la structure générale.

génériques, comme décrit en détails dans [39], puis il connecte ces réseaux aux interfaces séquentielles définies dans les `parfun`s et enfin exécute son code séquentiel en appliquant `() :unit` à sa fonction argument. Le schéma est décrit à la Figure 3.1. Les réseaux de squelettes ne sont initialisés qu'une fois, mais peuvent être invoqués plusieurs fois au cours de l'exécution du `pardo`.

3.4.8 Equilibrage de charge : les couleurs (nouveauté)

Dans le système OcamlP3l, les expressions de combinateurs gouvernent la forme du réseau de processeurs virtuels qui est entièrement construit pendant la compilation. L'affectation de ces nœuds virtuels aux processeurs physiques est donc a priori à la charge du système OcamlP3l. Cette affectation automatique n'est en général pas réaliste, et nous décrivons ici la nouvelle notion de *couleur**—*color*—, qui donne la possibilité au programmeur de caractériser les expressions squelettiques afin de spécifier des requêtes sur l'association entre nœuds virtuels et physiques. Ces requêtes vont guider la stratégie automatique. Ceci transfère le gros des difficultés à l'utilisateur, qui devra tenir compte des spécificités de son programme ainsi que celles de ses moyens informatiques.

Prenons comme exemple l'expression
`farm` (`seq` $(fun\ x \rightarrow x\ *\ x), 16)$
qui correspond à un réseau comprenant un nœud émetteur, un nœud récepteur et 16 nœuds ouvriers agricoles (*farm*) qui calculent la fonction carré. Il existe de nombreuses manières de faire correspondre un ensemble de nœuds virtuels avec des nœuds physiques. Les questions de savoir quelle est la meilleure application et comment obtenir du compilateur qu'il la réalise ne sont pas évidentes. Sans couleur, l'affectation des tâches aux machines physiques est entièrement automatique et sans contrôle de l'utilisateur.

Une solution simple consiste à envoyer un par un les nœuds virtuels aux nœuds physiques, et quand ces derniers sont tous pris, il suffit de recommencer avec le premier. Or ceci donne généralement de mauvais résultats, puisque cela ne tient absolument pas compte de l'équilibrage des charges. Dans la pratique, ni les tâches d'un côté, ni les ressources informatiques de l'autre, ne sont équivalentes. L'utilisateur peut en revanche souvent estimer les deux. Il convient donc de lui laisser la possibilité d'apparier nœuds virtuels et physiques, ou tout au moins de le laisser guider cet appariement. La notion de *couleur* offre cette possibilité en permettant de classer séparément les capacités des processeurs physiques et le poids des nœuds virtuels.

Une couleur est un paramètre entier de type `int` optionnel qui est ajouté à une expression OcamlP3l dans le programme source d'une part, et dans la ligne de commande appelant le programme compilé d'autre part.

Coloration des nœuds virtuels

Pour la coloration des nœuds virtuels, il suffit d'employer la syntaxe régulière d'Ocaml pour les arguments optionnels —i.e. ajout du préfixe `~`— avec le mot-clé *col*. Ainsi, par exemple, l'expression `farm ~col:k (f, n)` signifie que tous les nœuds virtuels au sein de cette structure fermière sont classés dans le groupe de rang k. Le champ d'application d'une couleur couvre tous les nœuds virtuels qui sont à l'intérieur de la structure coloriée. A moins d'une spécification explicite, la couleur d'une expression interne est simplement héritée de la couche externe. La couche la plus extérieure est créditée d'une valeur par défaut valant 0, ce qui signifie “aucune requête particulière”.

Pour les combinateurs `farm`, `mapvector` et `reducevector`, en sus des couleurs du combinateur lui-même, il existe un paramètre de couleur optionnelle supplémentaire dénommé *colv*. Une spécification *colv*[ ] est une liste de couleurs —de type `int list`—, qui donne une couleur à chacune des structures ouvrières qui sont les arguments du combinateur. Ainsi, l'expression OcamlP3l

```
mapvector ~col:2 ~colv:[ 3; 4; 5; 6 ] (seq f, 4)
```

est une expression squelettique `mapvector` avec un émetteur et un récepteur de rang 2 et quatre nœuds ouvriers (quatre copies de `seq f`) avec les rangs respectifs 3, 4, 5, et 6.

Coloration des nœuds physiques

Nous avons peint les nœuds virtuels, maintenant c'est au tour des nœuds physiques. La couleur des processeurs physiques est déterminée par la ligne de commande au moment de l'exécution du programme. On peut écrire :

```
prog.par -p3lroot ip1:port1#color1 ip2:port2#color2 ... \
                  ip_i:port_i#color_i ...
```

où `ip_i :port_i#color_i` indique l'adresse ip (ou le nom) du nœud physique `i` participant au calcul, son port et sa couleur. Le port et la couleur sont tous deux optionnels. Quand le port n'est pas spécifié, le port par défaut `p3lport`[29] est utilisé. De même, quand il n'est pas fait mention de la couleur, la couleur par défaut 0 est utilisée.

Appariement des couleurs

Maintenant que nous avons colorié les processeurs physiques et les nœuds virtuels, il reste à les associer entre eux. Si les deux groupes de couleurs sont en bijection, l'appariement est facile. Mais dans le cas général, ce n'est pas le cas : d'abord, le nombre de nœuds virtuels n'est pas toujours égal au nombre de processeurs physiques ; de plus, dans des expressions OcamlP3l complexes, il est difficile et laborieux d'énumérer tous les nœuds virtuels dans chaque classe. Il est donc nécessaire de fournir un algorithme d'affectation des couleurs.

L'algorithme qui est implémenté est basé sur l'idée qu'*une couleur représente non pas la puissance **exacte** requise par le nœud virtuel, mais la **plus basse** acceptable*. Par exemple, une tâche de couleur 5 signifie qu'un processeur de couleur 5 est nécessaire, mais s'il existe un nœud physique de couleur 6 qui est libre, pourquoi ne pas l'utiliser ? Dans la pratique, les nœuds virtuels sont triés dans l'ordre décroissant de leurs couleurs afin de refléter leur ordre de priorité pour choisir leur processeur physique. Les nœuds virtuels dont les couleurs sont les plus élevées seront les premiers à choisir leurs nœuds physiques. Ensuite, pour chaque nœud virtuel, est dressée la liste des nœuds physiques de couleur plus grande que ou égale à la couleur du nœud virtuel. Parmi ces processeurs

[29] La valeur par défaut est fixée à `p3lport` = 4000 dans la version actuelle d'OcamlP3l.

physiques retenus, l'algorithme associe au nœud virtuel celui qui a la plus petite charge de travail, c'est-à-dire le moins de nœuds virtuels déjà affectés.

Cet algorithme fournit une méthode d'appariement totalement automatique, avec une part de liberté pour le réglage manuel. Il faut cependant noter qu'une couleur désigne une classe de calcul qualitativement, sans relation avec une estimation quantitative de la puissance de la machine ou du poids de la tâche à réaliser. Il ne faut donc pas espérer une grande précision, ni une grande efficacité d'un tel algorithme.

Néanmoins, l'approche par les couleurs est suffisamment simple et précise pour permettre des améliorations significatives dans le cas de la méthode de décomposition de domaines, comme nous le verrons sur nos expérimentations numériques de la Section 3.6.

Remarque. Il faut préciser que nous nous sommes rendus compte à l'usage que l'algorithme d'appariement ci-dessus est difficile d'emploi dans le cas de la décomposition de domaines. De manière générale, quand les sous-domaines sont différents et que la grappe d'ordinateurs est hétérogène elle-aussi, l'équilibrage n'est pas chose simple. Il s'avère que l'algorithme décrit ci-dessus ne semble pas adapté à ce type de problèmes : un usage sans précaution peut faire affecter au processeur le plus performant *plusieurs* des tâches les plus lourdes ; quand la taille mémoire est le facteur limitant, cela peut tuer le calcul au moment des factorisations. L'utilisation des couleurs passe alors par l'affectation d'une même couleur à un processeur et à un sous-domaine pour obtenir un appariement de "un pour un". Ceci peut devenir fastidieux quand le nombre de sous-domaine devient important.

Cette remarque peut sembler limiter la portée de l'automatisme des couleurs. En fait, l'avantage reste quand même certain puisque l'utilisateur peut ne pas se préoccuper des nœuds virtuels de communication. De plus, le fait de pouvoir affecter manuellement une tâche à un processeur nous a permis de gagner dans certains cas des temps de calcul importants, voir la Section 3.6.1. Enfin, il reste possible de changer l'algorithme d'appariement des couleurs : une fois celles-ci mises en place, il est envisageable d'élargir le choix des algorithmes d'appariement et de laisser l'utilisateur décider de l'algorithme qui convient à son application.

La Figure 3.2 résume les types des combinateurs dans leurs formes actuelles au sein de la version `1.9` d'OcamlP3l. Les types tiennent compte des paramètres optionnels de couleur.

3.4.9 Utilisation du système

Le codage, le débogage et l'exécution d'un programme OcamlP3l sont simples et élégants.

Comme nous l'avons mentionné ci-dessus, OcamlP3l propose trois sémantiques proches : les sémantiques séquentielle, parallèle et graphique. Dans la pratique, le programmeur peut compiler tout programme OcamlP3l avec l'une des trois options de l'utilitaire de compilation `ocamlp3lcc` et obtenir ainsi trois codes exécutables différents, mais reliés, sans modifier le code source.

- Avec l'option `-seq` le fichier source est compilé en un code tournant sur une seule machine. En invoquant la commande
  `ocamlp3lcc -seq sourcename.ml`
  on crée un fichier exécutable appelé `sourcename.seq` qui peut être lancé sur une seule machine, testé et débogué avec le débogueur habituel d'Ocaml comme n'importe quel autre programme séquentiel.
- Avec l'option `-gra`, un fichier exécutable est créé qui permet de dessiner le réseau de squelettes qui est spécifié par le programme. En tapant
  `ocamlp3lcc -gra sourcename.ml`
  on obtient le fichier `sourcename.gra` dont l'exécution fait apparaître une fenêtre graphique dans laquelle sont dessinés les réseaux parallèle mis en place par le programme.
- Enfin, la version parallèle du code est obtenue avec l'option `-par`. La commande de compilation
  `ocamlp3lcc -par sourcename.ml`
  produit l'exécutable SPMD générique `sourcename.par` qui va être déployé sur tous les nœuds participant au calcul. Une copie de cet exécutable est lancée sur tous les processeurs sollicités et attend des informations de configuration qui lui sont envoyées par un nœud racine (*root node*) qui se charge d'exécuter le code séquentiel encapsulé dans le `pardo`. Le nœud racine n'est rien de plus qu'une copie particulière du même exécutable qui est simplement lancée avec l'option spécifique `-p3lroot` sur la ligne de commande. De plus, le nœud racine reçoit en arguments toutes les informations requises concernant les nœuds impliqués dans le réseau de calcul : leur adresse ip ou leur nom, leur port, leur couleur, et éventuellement d'autres arguments d'exécution optionnels. En résumé, la ligne de commande pourrait s'écrire
  `sourcename.par -p3lroot` *`nodes_info_list`*

  Un exemple a déjà été présenté à la Section 3.4.8 dans la partie consacrée à la coloration des nœuds physiques.

La mise en place de ces trois sémantiques peut aider énormément le développement d'applications parallèles : le but final reste l'obtention du code parallèle, mais pour y arriver dans la philosophie OcamlP3l, la programmation et en particulier le débogage s'effectuent avec un prototype *séquentiel*. Le programmeur peut ainsi se concentrer sur les aspects algorithmiques de la *fonction* à calculer et ensuite passer directement au calcul parallèle, en s'appuyant sur l'équivalence entre sémantiques parallèle et séquentielle. En effet, un programme OcamlP3l qui est compilé et exécuté sans erreur en mode séquentiel devrait en principe toujours s'exécuter en mode parallèle avec le même résultat. Le programmeur évite avec cette optique toute la phase de débogage de l'application parallèle qui peut s'avérer complexe au point de s'arracher les cheveux.

Le fait de développer en séquentiel et d'exécuter en parallèle constitue l'une des caractéristiques les plus significatives et les plus puissantes d'OcamlP3l. Roberto Di Cosmo et Pierre Weis comptent travailler à la démonstration d'une preuve *formelle* de l'équivalence entre les deux sémantiques qui permettrait de montrer l'équivalence entre les modes séquentiels et parallèles de *tous* les codes que l'utilisateur pourrait créer.

Enfin, la sémantique graphique est une aide utile pour donner au programmeur une idée intuitive de la structure parallèle de son code à partir des sources mêmes. Cette information peut servir notamment à l'affectation des nœuds physiques aux nœuds virtuels ou plus directement à la conception et à l'analyse de l'application parallèle.

D'autres outils existent également qui visent à faciliter la programmation dans le système OcamlP3l. Par exemple, `ocamlp3lrun` permet de distribuer automatiquement les copies du code parallèle à un ensemble de machines, `ocamlp3lps` permet d'imprimer les PIDs de tous les processus exécutant des programmes OcamlP3l sur le réseau... Bien d'autres existent encore.

Cependant, l'exécution reste très simple et l'utilisateur peut très bien écrire un petit script destiné à son application sur la configuration du matériel qui est à sa disposition. Nous présentons ainsi à la Figure 3.3 le script que nous avons employé pour lancer notre calcul en parallèle sur 8 nœuds d'une grappe de PC, dont la première machine est plus lente que les autres.

```
type color = int

val seq :
  ?col:color ->
  (unit -> 'a -> 'b) -> unit -> 'a stream -> 'b stream
val ( ||| ) :
  (unit -> 'a stream -> 'b stream) ->
  (unit -> 'b stream -> 'c stream) ->
  unit -> 'a stream -> 'c stream
val loop :
  ?col:color ->
  ('a -> bool) * (unit -> 'a stream -> 'a stream) ->
  unit -> 'a stream -> 'a stream
val farm :
  ?col:color ->
  ?colv:color list ->
  (unit -> 'a stream -> 'b stream) * int ->
  unit -> 'a stream -> 'b stream
val mapvector :
  ?col: color ->
  ?colv:color list ->
  (unit -> 'a stream -> 'b stream) * int ->
  unit -> 'a array stream -> 'b array stream
val reducevector :
  ?col:color ->
  ?colv:color list ->
  (unit -> ('a * 'a) stream -> 'a stream) * int ->
  unit -> 'a array stream -> 'a stream
val parfun :
  (unit -> unit -> 'a stream -> 'b stream) ->
  'a stream -> 'b stream
val pardo : (unit -> 'a) -> 'a
```

Fig. 3.2 – Les types des combinateurs OcamlP3l

```
#!/bin/sh

NODES="clus-101 clus-102 clus-103 clus-104 \
       clus-105 clus-106 clus-107 clus-108"
PAR="./coupling_subdomains.par"

echo -n "Launching P3L amorphous nodes on the cluster:"
for NODE in $NODES; do
    echo -n " $NODE"
    ssh $NODE $PAR 1> log-$NODE 2> err-$NODE &
    case $NODE in
        clus-101) COLORED_NODES="$COLORED_NODES $NODE#1";;
        *) COLORED_NODES="$COLORED_NODES $NODE#2";;
    esac
done

echo "Starting computation with $COLORED_NODES..."
$PAR -p3lroot $COLORED_NODES 1> log-root 2> err-root

echo "Finished."
```

Fig. 3.3 – Script simple destiné à lancer un programme parallèle OcamlP3l.

3.5 Utilisation d'OcamlP3l pour la décomposition de domaines

3.5.1 Genèse d'un partenariat et échelles de temps

Je ne résiste pas au plaisir de présenter les premiers tests qui ont servi de base pour l'expérimentation d'OcamlP3l dans le cadre des méthodes de décomposition de domaines. Ces tests constituent le résultat des sept premiers mois de collaboration entre Pierre Weis, Roberto Di Cosmo, François Clément, Arnaud Vodicka et moi. On s'étonnera sans doute au premier abord de ces résultats : programmer la méthode de Schwarz additif, sans doute la méthode de décomposition de domaines la plus simple qui soit, fut-elle en parallèle, avec des différences finies sur une grille structurée régulière ne semble pas d'un attrait folichon. Mettre sept mois à plusieurs pour réaliser cet exploit pourrait tenir de la gabegie pure et simple.

En fait, un long délai, quatre mois environ, a été nécessaire à la compréhension mutuelle entre informaticiens et numériciens, qui parlent deux langues radicalement différentes[30]. Le plus grand pas a été fait quand les chercheurs en informatique ont saisi les problèmes qui nous intéressaient et proposé des outils pour les traiter et quand, de notre côté, nous avons commencé à percevoir les possibilités qu'offraient les méthodes proposées par l'équipe d'informaticiens.

Ensuite, la prise en main des squelettes d'OcamlP3l, la modification du code très simple de différences finies et l'implémentation des outils de base pour utiliser OcamlP3l dans le cadre d'un couplage dans une méthode de décomposition de domaines ont pris aussi un temps non négligeable, approximativement trois mois. Dans le même intervalle, des corrections étaient apportées au squelette `loop` d'OcamlP3l. La première étape aboutit donc à l'algorithme de Schwarz additif en deux dimensions, avec des domaines structurés et en parallèle après sept mois de labeur, voir la Section 3.5.2 et également [31] pour plus de détails.

La deuxième phase a commencé quand nous nous sommes rendus compte que la démarche avait donné satisfaction dans le cas simple précédent. Nous nous sommes donc lancés dans l'algorithme de Robin–Robin non conforme décrit en Section 2.3 en trois dimensions et en parallèle avec un code d'éléments finis réaliste. En moins de quatre mois, nous avons obtenu les premiers résultats, [30]. Là encore, un gros travail a été nécessaire. Du côté des informaticiens, trois nouveautés importantes ont dû être mises en place à notre demande : la possibilité d'initialiser localement les squelettes, voir la Section 3.4.5, les `par-`

[30] A titre d'illustration, je dirais ceci : les trois premières réunions ont été passées à essayer de s'entendre autour de la notion de maillage...

`funs`, voir la Section 3.4.6, et les `colors`, voir la Section 3.4.8. De notre côté, une assez longue période a été consacrée à la transformation du code éléments finis en un solveur de sous-domaine, voir la Section 3.2. Une phase de tests et d'amélioration du programme opérant la projection entre les interfaces, voir la Section 2.4 s'est avérée indispensable. De plus, pour coder la méthode de Robin–Robin plus complexe que le point fixe de l'algorithme de Schwarz, il a fallu enrichir notablement le code Ocaml de couplage. Ce coupleur est décrit en détail à la Section 3.5.4.

La troisième phase est la récompense des deux précédentes : après le long apprentissage et la mise en place des outils, en modifiant légèrement le coupleur et le solveur de sous-domaine, nous avons programmé la méthode de Neumann–Neumann *en parallèle* (sans solveur grossier, [67]) *en quatre jours*.

La suite logique est de coupler en parallèle des codes différents, traitant des *physiques différentes*. Il y a un bon espoir de pouvoir faire ceci rapidement car *sûrement*, c'est-à-dire en pouvant se baser sur un débogage et des tests séquentiels et passer de façon sûre à l'exécution parallèle. Un bémol doit cependant être mis : il faut supposer que les codes à coupler auront des formes adaptées au couplage, en particulier leurs entrées et sorties, l'interface au sens informatique, doivent être compatibles avec le coupleur. Cette phase de modification éventuelle des codes peut en effet s'avérer longue et complexe.

La conclusion qu'on peut tirer de cet historique de la collaboration entre chercheurs en informatique et numériciens est qu'elle a été à la fois exemplaire, dans le sens où elle a abouti à des résultats intéressants pour les deux groupes, et banale dans sa forme : le plus dur reste la compréhension mutuelle et la germination des idées, qui nécessitent toutes deux du temps et de la volonté, et n'offrent en retour immédiat que peu de choses palpables.

Pour finir, je précise que la programmation en OcamlP3l des codes de couplages a été essentiellement assurée par Arnaud Vodicka et François Clément.

Le reste de cette Section est divisée de la façon suivante : nous montrons les résultats du premier programme mettant en œuvre l'algorithme de Schwarz dans la Section 3.5.2. La Section 3.5.3 est consacrée à la description des codes à coupler et de l'algorithme de couplage. Enfin, le coupleur écrit en OcamlP3l est fourni, accompagné de quelques commentaires, dans la Section 3.5.3.

3.5.2 Algorithme de Schwarz additif en deux dimensions

Une description complète de cette étude peut être trouvée dans le rapport technique [31]. Nous ne reproduisons ici que les résultats les plus marquants.

Dans ces premières expériences numériques, nous avons utilisé l'algorithme de Schwarz additif introduit au XIXème siècle, [89]. On pourra se reporter par exemple à [81] ou [67] pour une présentation détaillée de cet algorithme. Nous mentionnons ici que c'est une méthode de décomposition de domaines avec recouvrement, qui peut s'écrire comme un point fixe sur des variables d'interface de type Dirichlet. Nous ne discuterons pas plus avant cet algorithme, nous remarquons juste qu'il est relativement facile à programmer sur des grilles structurées, avec une décomposition en sous-domaines structurée elle-aussi, voir un exemple sur la Figure 3.4. C'est pourquoi il constituait un point de départ suffisamment simple pour les premiers tests d'OcamlP3l.

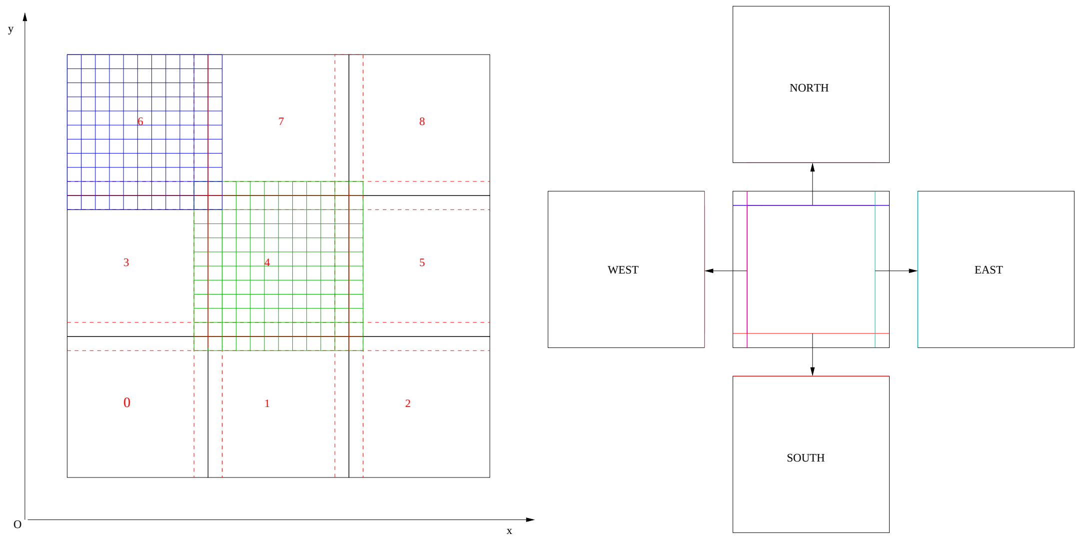


FIG. 3.4 – Gauche : le domaine Ω décomposé en 9 sous-domaines se chevauchant sur une couche de 2 mailles. Droite : les communications entre un sous-domaine et ses quatre voisins éventuels.

Les tests qui sont présentés à la Figure 3.5 ont été réalisés sur le carré unité $\Omega = [0,1] \times [0,1]$ divisé en $5{\times}5$ sous-domaines carrés (ou rectangulaires sur le bord). Tous les sous-domaines ont des maillages réguliers et structurés en carrés. Le chevauchement entre deux sous-domaines tient sur quatre mailles. L'évolution des itérés successifs dans l'algorithme de point fixe est dessiné sur les Figures 3.5. La condition initiale est la fonction identiquement nulle. La solution analytique du problème global est connue et est très simple : $p(x,y) = x(1-x)y(1-y)$. Elle est tracée également de manière à visualiser la convergence des itérés vers cette solution.

Dans la partie en bas à gauche de la Figure 3.5, est dessinée l'accélération due au parallélisme : le *speed-up*. Dans cette expérience, la décomposition reste fixe à 25 sous-domaines. Deux paramètres varient : le maillage de chaque sous-domaine est raffiné et le nombre de processeurs varie de 1 à 10. Pour chaque degré de raffinement, le speed-up augmente linéairement jusqu'à atteindre environ 7 pour 10 processeurs, ce qui est raisonnable, compte-tenu du fait que l'implémentation n'était pas du tout optimisée.

Ces résultats très satisfaisants nous ont encouragés à poursuivre avec une application plus réaliste.

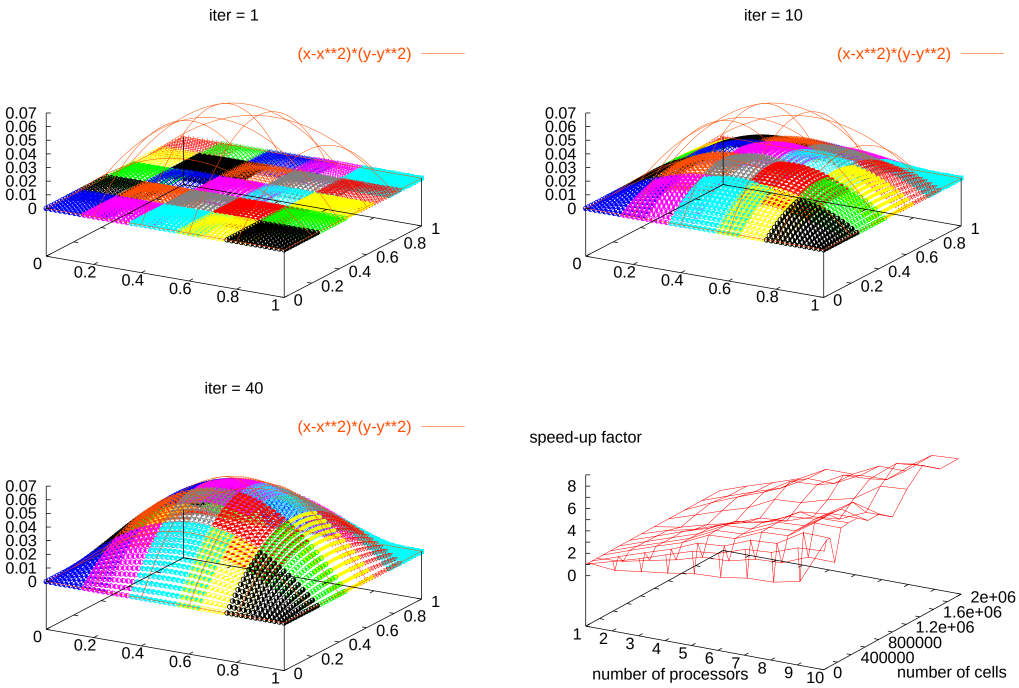


Fig. 3.5 – Un calcul avec 25 sous-domaines (chaque sous-domaine est un carré possédant sa couleur). La solution initiale (à gauche en haut), la solution calculée après 10 itérations (à droite en haut) et la solution après 40 itérations (en bas à gauche). La solution analytique est tracée en lignes rouges. En bas à droite : le facteur d'accélération pour 10 processeurs avec des grilles raffinées : d'un maillage 100×100 jusqu'à une grille de 1400×1400 mailles.

3.5.3 Algorithme de Robin–Robin non conforme

Codes à coupler

Pour opérer le couplage, trois codes écrits en C++ sont utilisés. Nous ferons largement appel aux notations introduites dans le Chapitre 2. En particulier nous utiliserons les noms des maillages d'interface de la Section 2.2.4, les espaces d'approximation de la Section 2.3.3 et surtout les opérateurs d'interface de la Section 2.3.9.

Le code build_interface_meshes prend en entrée le nom du fichier de maillage 3D du sous-domaine $\Omega_i,\ i \in I_n = \{1, 2, \ldots, n\}$. Ce programme lit le fichier sur le disque, calcule tous les maillages d'interface à deux dimensions $\mathcal{S}^h_{ij},\ j \in \mathcal{N}_i$ et écrit ces maillages dans des fichiers sur le disque. Il ne renvoie rien en sortie.
Par conséquent, il n'a besoin d'aucune redirection d'entrée ou de sortie.

Le code build_projection_matrix prend en entrée les deux noms de fichiers des deux maillages d'interface 2D $\mathcal{S}^h_{ij}$ et $\tilde{\mathcal{S}^h_{ij}} = \mathcal{S}^h_{ji}$ vivant sur la même interface géométrique entre deux sous-domaines $\Sigma_{ij},\ (i, j) \in \mathcal{N}$. Le programme lit ces fichiers sur le disque, calcule et renvoie la projection L^2 correspondante $P_{i \to j}$, voir les équations (2.50, 2.62). Cette projection est stockée sous la forme d'une matrice creuse, que nous appellons avec beaucoup d'originalité matrice de projection. La projection d'un vecteur d'un maillage d'interface à son voisin s'exprime dans ce formalisme comme un produit matrice vecteur creux. Nous n'en disons pas plus là-dessus : cette remarque vise juste à expliquer le nom du programme.
Ce code a besoin de la redirection de sa sortie standard.

Le code solve_on_a_subdomain prend en entrée le nom du fichier de maillage 3D du sous-domaine $\Omega_i,\ i \in I_n$. Ce programme réalise l'opérateur de Robin–Robin S_i défini en (2.47).
Il commence par une phase de initialisation dans laquelle il lit le fichier de maillage sur le disque, il calcule la matrice du problème de Robin (2.48) et ensuite il factorise celle-ci.
Il entre ensuite dans une boucle sans fin, attendant constamment un mot-clé.

Quand arrive le mot-clé `"init"`, il calcule et retourne le terme de Robin sur l'interface $S_i(0, q|_{\Omega_i}) \in \Lambda_{h,i}$, qui est utilisé dans le second membre de l'équation de point fixe (2.59) ; il sert donc à obtenir le second membre algébrique b du système linéaire (2.60).

Quand arrive le mot-clé `"loop"`, il prend en entrée le vecteur d'interface $x_{h,i} \in \Lambda_{h,i}$, puis il calcule et retourne le terme de Robin $S_i(x_{h,i}, 0) \in \Lambda_{h,i}$ qui est utilisé dans le produit matrice vecteur (2.59).

Quand arrive le mot-clé `"final"`, il prend en entrée le vecteur d'interface $x_{h,i} \in \Lambda_{h,i}$, puis il calcule et retourne le terme de Robin $S_i(x_{h,i}, q|_{\Omega_i}) \in \Lambda_{h,i}$, après avoir écrit sur le disque la solution $(\mathbf{u_{h,i}}, p_{h,i})(x_{h,i})$ du problème (2.48).

Ce programme a besoin des redirections à la fois de son entrée standard et de sa sortie standard.
De plus, la phase d'initialisation durant laquelle est opérée la factorisation de la matrice du sous-domaine est coûteuse à la fois en temps de calcul et en espace mémoire. Dans certains cas, elle peut occuper la majeure partie du temps de résolution. Il est donc indispensable de ne la faire qu'une seule fois, avant le début de la méthode itérative. C'est pourquoi ce programme doit être *initialisé localement*, avec un générateur de squelettes qui permette ceci, voir la Section 3.4.5. En outre, il faut pouvoir le rappeler à la demande et pour cela il faut stocker ses canaux d'entrée et de sortie.
Il est enfin fondamental d'équilibrer ces programmes lors de leur exécution parallèle en fonction de la taille du sous-domaine dont chaque solveur de sous-domaine se charge. C'est à ce niveau qu'interviennent la coloration des tâches et des ressources informatiques à disposition pour les réaliser. Cette remarque était également valable pour les deux précédents programmes, mais la coloration s'avère importante ici du fait que ce sont les solveurs de sous-domaine qui réalisent dans la pratique presque la totalité du travail.

Nous avons utilisé la méthode de Krylov non symétrique Bi-CGStab [95] pour résoudre le système linéaire (2.60) de la méthode de décomposition de domaines de Robin–Robin. Un algorithme itératif du type Bi-CGStab ne requiert que trois fonctions agissant sur des vecteurs : le produit axpy et le produit scalaire dot[31] qui peuvent être tirés de la librairie BLAS—Basic Linear Algebra Subprograms— et le produit matrice vecteur, noté ici aax, qui calcule l'action de la matrice $\mathcal{A}^{RR}$ du système (2.60) sur un vecteur x. Le produit aax est celui qui est détaillé dans l'Algorithme 2.

[31] axpy calcule le vecteur $a.x + y$ à partir du scalaire a et des deux vecteurs x et y ; dot calcule le produit scalaire $\langle x, y \rangle$ à partir des vecteurs x et y.

Le choix de Bi-CGStab, plutôt que GMRes par exemple, s'explique par la plus grande simplicité de programmation du premier par rapport au second. Il nous est en effet apparu plus facile, pour les premiers tests, de reprogrammer intégralement en Ocaml une méthode itérative pour pouvoir l'utiliser avec OcamlP3l. Il faut cependant noter que l'algorithme Bi-CGStab *n'a pas* été programmé avec des squelettes OcamlP3l, mais de façon standard en Ocaml. Seul le produit aax fait intervenir ces squelettes, mais *encapsulés dans des* **parfuns**, ce qui permet de le traiter comme une fonction Ocaml normale, bien qu'il contienne un réseau de calcul, voir la Section 3.4.6.

Il reste possible également de changer de solveur itératif à moindre frais, en gardant intact le produit aax. Cependant, il pourrait paraître dommage de perdre de l'énergie à réécrire une méthode itérative dans la mesure où elles sont toutes déjà codées dans d'autres langages. En fait, il n'existe a priori aucune raison pour ne pas appeler à partir d'Ocaml un de ces programmes existant. Nous ne l'avons pas testé jusqu'à présent, mais c'est prévu dans la suite.

Algorithme de couplage détaillé

Nous pouvons maintenant décrire l'algorithme complet tel qu'il est mis en œuvre dans le code OcamlP3l.

Initialisation

- construire en parallèle pour chaque sous-domaine $\Omega_i, i \in I_n$, les maillages d'interface $\mathcal{S}_{ij}^h,\ j \in \mathcal{N}_i$.
- calculer en parallèle pour chaque maillage d'interface $\mathcal{S}_{ij}^h,\ (i,j) \in \mathcal{N}$, la projection $P_{i \to j}$.
- factoriser en parallèle pour chaque sous-domaine $\Omega_i, i \in I_n$, les matrices des problèmes de Robin locaux (2.48).
- définir le produit aax qui applique en parallèle la matrice de la méthode de Robin–Robin $\mathcal{A}^{RR} = \mathrm{Id}_{\Lambda_h} - sPS(\cdot, 0)$, voir (2.60), à un vecteur x.
- calculer en parallèle le second membre $b = sPS(0, q)$.
- choisir une condition initiale, par exemple $x^0 = 0$.
- choisir un algorithme pour résoudre le système global, par exemple Bi-CGStab.

Itération

- lancer l'algorithme avec le produit matrice vecteur parallèle aax, le second membre b et la condition initiale x^0.
- nommer $x_h^\star \in \Lambda_h$ la solution de la méthode itérative.

Conclusion

- résoudre en parallèle les problèmes de Robin (2.48) dans chaque sous-domaine $\Omega_i, i \in I_n$, avec la donnée d'interface $x_h^\star|_{\Omega_i}$ et le second membre $q|_{\Omega_i}$, et stocker les solutions $(\mathbf{u_{h,i}}, p_{h,i})(x_h^\star|_{\Omega_i})$.

3.5.4 Implémentation du coupleur en OcamlP3l

Le module Ocaml Ddec est dédié à la décomposition de domaines. Son interface, contenant les types des fonctions, est donné dans l'annexe C.2. En particulier, il décrit des types pour le vecteur d'inconnue x qui aident à la mise en œuvre des opérateurs de restriction R_i et R_{ij} définis en (2.52, 2.51) et de prolongement, transposés des précédents. Sont également déclarées les fonctions axpy et dot. Des types semblables sont aussi construits pour l'opérateur de projection P de l'équation (2.55). Nous ne disons rien sur l'implémentation de Bi-CGStab qui est complètement habituelle.

Le code faisant le couplage est si court et si simple qu'il peut être présenté en entier ici.

```
(* we have emphasized the OcamlP3l skeletons *)
let spawn_it command cin cout =
  match !cin, !cout with
  | Some ic, Some oc -> ic, oc
  | _ -> Printf.printf "Calling %s\n" command;
    let ic, oc = Unix.open_process command in
    cin := Some ic; cout := Some oc;
    ic, oc;;

let encapsulate_mapvector colv prog n =
  let body _ = mapvector ~colv:colv (seq prog, n) in
  let network = parfun body in
  (fun v ->
     let s = network (P3lstream.of_list [v]) in
     List.hd (P3lstream.to_list s));;

let build_all_interface_meshes =
  let colv = Ddec.colv_sorting_subdomains in
  let prog =
    (fun _ -> fun i ->
       let file = Ddec.filename_of_3D_mesh_number i in
       let command = "build_interface_meshes "^file in
       Sys.command command) in
  encapsulate_mapvector colv prog Ddec.number_of_processors;;
```

```
let build_all_projection_matrices =
  let colv = Ddec.colv_sorting_connectivity_table in
  let prog =
    (fun _ -> fun i -> fun j ->
       let file_ij = Ddec.filename_of_2D_mesh_number i j
       and file_ji = Ddec.filename_of_2D_mesh_number j i in
       let command =
         "build_projection_matrix "^file_i^" "^file_j in
       let ic = Unix.open_process_in command in
       read_projection_matrix ic) in
  encapsulate_mapvector colv prog Ddec.number_of_processors;;

let solve_on_all_subdomains =
  let colv = Ddec.colv_sorting_subdomains in
  let prog _ =
    let cin = ref None and cout = ref None in
    (fun (i, tagged_x_i) ->
       let file_i = Ddec.filename_of_3D_mesh_number i in
       let command = "solve_on_a_subdomain "^file_i in
       let ic, oc = spawn_it command cin cout in
       Ddec.print_tagged_internal_boundary_values
         oc tagged_x_i;
       flush oc;
       let xi_i = Ddec.read_internal_boundary_values ic in
       Ddec.iter_vector_of xi_i) in
  encapsulate_mapvector colv prog Ddec.number_of_processors;;

pardo (fun () ->
  Printf.printf "Beginning of 3D coupling\n";
  (* Initialization *)
  let n = Ddec.number_of_subdomains in
  let v = Array.init n (fun i -> i) in
  build_all_interface_meshes v;
  let projection_of =
    let projection_matrices =
      let v = Array.of_list Ddec.connectivity_table in
      build_all_projection_matrices v in
```

```
        Ddec.projection_with projection_matrices in
      let f_of =
65      (fun v ->
          let xi = solve_on_all_subdomains v in
          let xi_tilde = projection_of xi in
          permutation_of xi_tilde) in
      let aax =
70      (fun x ->
           let v = Ddec.iter_vector_of x in
           let xi_double_tilde = f_of v in
           (Ddec.axpy (-.1.) xi_double_tilde x)) in
      let b =
75      let v = Ddec.init_vector_of_size n in
        f_of v in
      let x0 = Ddec.zero_structure_values n in
      let algorithm = Bicgstab.algorithm Ddec.axpy Ddec.dot in
      (* Iteration *)
80    let x_star = algorithm aax b x0 in
      (* Finalization *)
      let v = Ddec.final_vector_of x_star in
      solve_on_all_subdomains v;
      Printf.printf "Ending of 3D coupling\n"
85 );;
```

L'exécution du code avec la sémantique graphique permet d'obtenir un dessin montrant la structure parallèle du code, voir la Figure 3.6. On observe immédiatement les trois réseaux de calcul parallèle indépendants que le système met en place.

Nous donnons quelques commentaires sur le code source `coupling.ml` ci-dessus.

La fonction `spawn_it` (lignes 2–8) a pour type `string -> in_channel option ref -> out_channel option ref -> in_channel * out_channel`. Elle permet de lancer des processus une fois en conservant les canaux connectés à leurs entrée et sortie standards. Ensuite, elle garde la possibilité d'injecter de l'information dans ces canaux ou d'en récupérer.

La fonction `encapsulate_mapvector` (lignes 10–15) est de type `color array -> (unit -> 'a -> 'b) -> int -> 'a array -> 'b array`. Elle retourne un réseau de calcul sur des flux qui encapsule un squelette de `mapvector`. Ceci permet de définir le calcul en parallèle d'une fonction sur toutes les composantes d'un vecteur en n'importe quel point du code séquentiel, et en particulier dans toute fonction Ocaml.

Les fonctions `build_all_interface_meshes` (lignes 17–24), `build_all_projection_matrices` (lignes 26–36) et `solve_on_all_subdomains` (lignes 38–51) correspondent aux trois codes à coupler de la Section 3.5.3. Ils sont définis grâce à l'usine à créer des squelettes `encapsulate_mapvector`, avec des couleurs pour équilibrer leurs charges. Chacune de ces trois fonctions utilise des techniques ad'hoc pour connecter ou non leur entrée et leur sortie standard. Seule la dernière fait usage de la nouveauté qui permet une initialisation *locale*, voir la Section 3.4.5, car l'argument `_`, de type quelconque, est donné en argument dès le début (ligne 40).

L'algorithme proprement dit est implémenté à l'intérieur du délimiteur de champ `pardo` (lignes 53–85). Il fait appel aux trois réseaux de calcul qui ont été définis auparavant. Les deux premiers ne sont utilisés qu'une fois pendant la phase d'initialisation. Le dernier est employé constamment dans la boucle du Bi-CGStab. La fonction `f_of` (lignes 63–67) définit la fonction $f = sPS(\cdot, q)$ de l'équation (2.58) dont nous cherchons le point fixe. Elle est appelée dans le corps du produit matrice vecteur `aax` (lignes 68–72) et pour le calcul du second membre `b` (lignes 73–75).

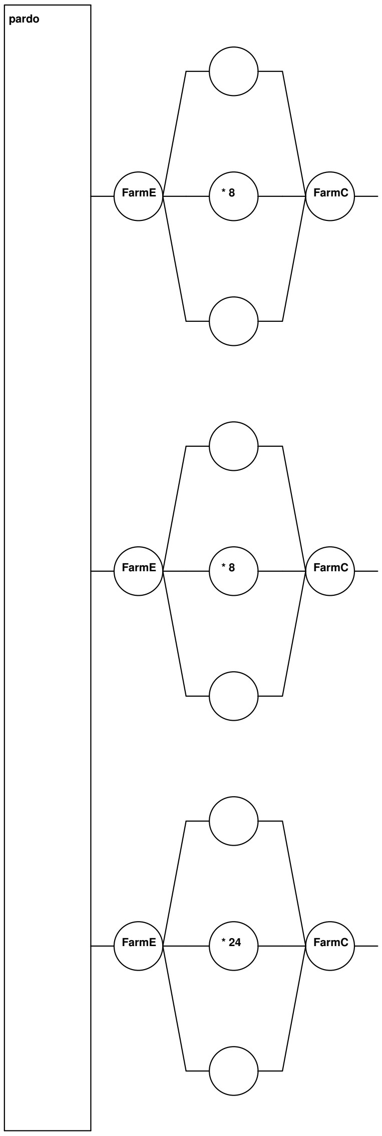


FIG. 3.6 – La structure parallèle du code `coupling.ml`. Configuration avec 8 sous-domaines et 24 interfaces. Voir la décomposition dans la Figure 3.7.

3.6 Simulations numériques 3D avec OcamlP3l

Nous avons testé dans la Section 2.5 la méthode de décomposition de domaines de Robin–Robin non conforme sur deux expériences académiques en deux dimensions, l'une pour mettre en évidence les propriétés de convergence de la méthode, l'autre pour se rapprocher des problèmes de stockage. Nous testons dans cette section cette même méthode de Robin–Robin non conforme en trois dimensions et en parallèle dans deux types de situations : dans la Section 3.6.1, nous cherchons à évaluer les performances du couplage avec OcamlP3l sur un cas académique. Dans la Section 3.6.2, nous confrontons le couplage OcamlP3l et la méthode de décomposition de domaines de Robin–Robin à la dure réalité d'une simulation physiquement réaliste ; nous résolvons l'écoulement d'un problème fourni par l'Andra.

Tous les tests ont été réalisés sur la grappe de l'Inria Rocquencourt qui est composée de 16 bi-processeurs Intel Xeon, possédant 2 Go de mémoire et plus de 2 GHz de fréquence d'horloge. La grappe n'est pas homogène : 13 nœuds ont une fréquence de 2.8 GHz et 3 nœuds sont plus lents, avec une fréquence de 2.0 GHz ou 2.2 GHz. Les communications entre les nœuds sont effectuées via un réseau Gigabit dédié.

3.6.1 Tests de performance sur un problème académique 3D

Nous mesurons dans cette section le temps CPU requis pour résoudre un problème simple donné, avec la méthode de Robin–Robin non conforme étudiée à la Section 2.3, et en utilisant le coupleur OcamlP3l parallèle présenté à la Section 3.5.

Objectifs de l'expérience

Le but est ici d'évaluer les performances de nos codes et en particulier du coupleur parallèle écrit en OcamlP3l. En effet, Ocaml a dans le monde numérique la réputation d'être lent et nous voulions savoir si cette lenteur posait problème ou non.

Les résultats obtenus qui sont résumés dans les Tableaux 3.2 et 3.3 semblent tout à faits raisonnables à notre avis. Nous n'avons pas mis en évidence de retards particuliers qui seraient dus spécifiquement à Ocaml dans ces expériences : un temps de latence dû au parallélisme existe bien, mais les phases les plus lentes, qui sont des étapes de calcul comme les factorisations ou les inversions dans chaque sous-domaine, sont du ressort du monde C ou C++. Nous n'avons pas cherché à estimer les temps de communication, qui ne semblaient pas des facteurs limitants dans ces simulations.

Enfin, nous ne pouvons pas réellement quantifier les résultats car il nous manque des comparaisons avec un code parallèle utilisant d'autres outils parallèles, comme MPI, [59]. Nous n'avons pas effectué de tests avec MPI pour des raisons qui ont été évoquées plus haut, en Section 3.1, mais la confrontation entre MPI et OcamlP3l reste une question intéressante et pourrait être réalisée dans une autre étude.

Présentation de l'expérience

Nous résolvons le problème modèle (2.2) dans le domaine $\Omega = [0,1]\times[0,2]\times[0,5]$, avec une perméabilité scalaire et constante $\mathbf{K} \equiv 1$. Nous choisissons un second membre $(q,\overline{p}) = (q,0)$, tel que la solution en pression est la fonction régulière $p^\star(x,y,z) = x(1-x)y(2-y)z(5-z)$. Une comparaison entre la solution numérique approchée et la solution exacte peut ainsi être effectuée. Un exemple de cette solution numérique peut être visualisée sur la Figure 3.7.

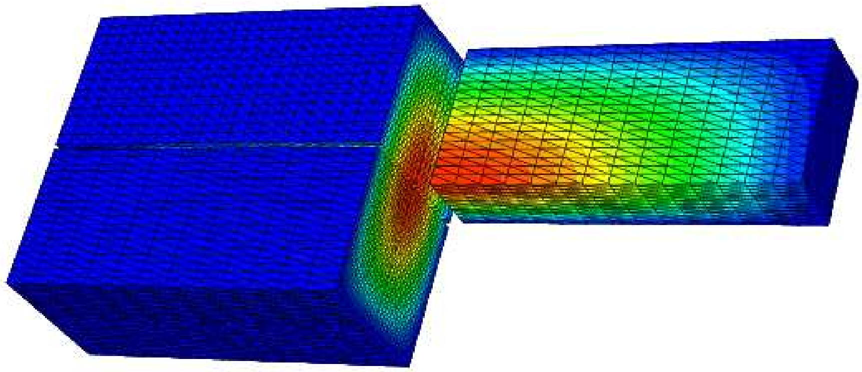


FIG. 3.7 – Champ de pression pour un calcul sur 8 sous-domaines non concordants. Pour des raisons de visualisation, nous montrons une vue éclatée de cinq des huit sous-domaines. Il faut noter que les maillages ne se raccordent pas aux interfaces. Les grandes (resp. petites) valeurs de la pression sont représentées en rouge (resp. en bleu).

Les problèmes de Robin discrets de chaque sous-domaine (2.48) sont résolus par hybridation des éléments finis mixtes, comme cela a été expliqué à la Section 3.2, voir par exemple [23] ou [29]. Dans chaque sous-domaine, l'hybridation produit des systèmes linéaires symétriques définis positifs, de taille en $\mathcal{O}(N_m)$, où N_m est le nombre de mailles, pour des maillages hexaèdriques 3D. Ces systèmes linéaires sont résolus grâce à un solveur linéaire direct LU fourni par la librairie UMFPack V4.1, [42][32] .

Les maillages utilisés sont constitués d'hexaèdres réguliers. Diverses manières de décomposer le domaine global Ω de façon structurée sont explorées. On note N_x (respectivement N_y, N_z) le nombre de sous-domaines selon l'axe Ox (resp. Oy, Oz). Dans la méthode de décomposition de domaines, les coefficients de Robin sont pris constants, tous égaux à 5, $\alpha_{ij} = 5, (i, j) \in \mathcal{N}$. La convergence de l'algorithme Bi-CGStab est atteinte quand la norme infinie du résidu est réduite d'un facteur 10^{10}.

Résultats de convergence

Nous commençons par montrer dans le Tableau 3.1 le comportement de la norme L^2 de l'erreur en fonction du degré de raffinement, afin de valider les codes. L'erreur calculée est la différence entre la pression exacte $p^\star$ et la pression $p_h \in N_h$ obtenue par la simulation numérique. La norme L^2 de cette erreur est calculée par intégration approchée par une formule de quadrature de Gauss à huit points. Cette norme *n'est pas* calculée par une différence prise au centre des mailles et on ne s'attend donc pas à une super convergence avec cette norme. En revanche, on doit observer théoriquement une décroissance en $\mathcal{O}(h)$, où h est le pas du maillage, cf. Théorème 2.

Le maillage est raffiné successivement de la façon suivante : le maillage global d'origine $\mathcal{T}^{h_1}$ est une grille régulière constituée de $(32 \times 40 \times 40) = 51\,200$ mailles ; son pas est noté h_1. Les maillages suivants $\mathcal{T}^{h_i}, i = 2, 3, 4$, sont obtenus en divisant le pas h_1 par un facteur i, et chaque maille est divisée en i^3 mailles. Pour conserver une charge constante par processeur, quand le maillage global est raffiné et le nombre de mailles est multiplié par i^3, nous décomposons le domaine en i^3 sous-domaines.

Conformément à ce que nous attendions, la norme L^2 de l'erreur en pression

[32]Un solveur direct de type Choleski aurait définitivement été plus performant dans ce cas, car il aurait profité de la symétrie du système. De cette manière, il n'aurait fallu stocker qu'une moitié de la matrice factorisée et les inversions auraient été plus rapides. Ce sont des raisons techniques mineures qui nous ont fait choisir de façon temporaire un solveur LU. Les temps de calcul sont donc ici lourdement grevés par le type de solveur choisi, mais nous nous intéressons aux performances *liées au parallélisme* dans la résolution globale. Or celles-ci sont peu influencées par la rapidité du solveur direct : le passage de LU à Choleski diminuerait sans doute le temps d'un calcul, mais ne changerait guère l'extensibilité ou le speed up.

décroît linéairement avec le pas du maillage, voir le Tableau 3.1. Ceci constitue un indice sérieux tendant à prouver que cette méthodologie et notre code fonctionnent correctement.

Quelques remarques supplémentaires doivent être faites sur la manière dont les calculs ont été menés. Pour le calcul avec 8 sous-domaines, un sous-domaine est affecté à un nœud de la grappe. Pour un sous-domaine comportant 51 200 mailles, la factorisation LU nécessite approximativement $1,4$ GO de mémoire sur les 2 GO disponibles, ce qui représente 69% de la mémoire totale disponible sur un nœud. Il n'y a donc pas eu besoin d'utiliser la mémoire auxiliaire par manque de mémoire —pas de *swap*. Le calcul est dans ce cas relativement rapide : le surcoût en temps par rapport au calcul avec un seul sous-domaine est de l'ordre de 20%.

En revanche, pour le calcul avec 27 sous-domaines, deux sous-domaines doivent être souvent affectés à un seul bi-processeur. La factorisation, qui a besoin de $2,8$ GO, a dû s'effectuer en swappant, retardant considérablement cette étape. A la fin de la factorisation, une partie de la mémoire est libérée, faisant passer le besoin en mémoire par sous-domaine à $0,8$ GO. Les calculs ont donc pu se finir assez rapidement car, deux sous-domaines tenant en mémoire sur un bi-processeurs, les 27 processus tournaient à plein régime sur 27 nœuds. Au final, un calcul sur un total de $1,4$ million de mailles, pour plus de $4,2$ millions d'inconnues, a pu se terminer en 50 minutes sur notre grappe de 16 bi-processeurs.

Enfin, le calcul avec 64 sous-domaines est tout simplement impossible en utilisant la méthode directe LU d'UMFPack, car elle est trop grande gourmande en espace mémoire. Nous avons donc réalisé ces calculs avec un solveur itératif dans chaque sous-domaine : un gradient conjugué préconditionné fourni par la librairie Aztec, [94]. Même si l'initialisation est beaucoup moins chère puisque la factorisation n'a plus lieu, chaque itération demande beaucoup plus de temps pour l'inversion des systèmes dans les sous-domaines. Ceci ralentit énormément la méthode de décomposition de domaines. Les calculs se sont malgré tout achevés après deux heures quarante minutes, pour un problème de $3,2$ millions de mailles et plus de 10 millions d'inconnues.

Résultats d'extensibilité

Nous présentons des résultats d'extensibilité dans deux cas, quand les maillages d'interface sont concordants dans le Tableau 3.2, et quand les maillages d'interface ne sont pas concordants dans le Tableau 3.3. Le principe de ce test d'extensibilité est le suivant : on garde constante la charge par processeur et on augmente le nombre de processeurs. Le but est de conserver autant que possible un temps de calcul constant, alors que la charge totale de travail s'est multipliée et des communications ont été nécessaires pour coupler le problème global. Dans le contexte de la décomposition de domaines, on associe un processeur à chaque sous-domaine, et le domaine global est divisé en 1, puis 2, puis 4, ... sous-domaines, ayant chacun approximativement le même nombre de mailles. Le calcul est fait successivement avec 1, puis 2, puis 4, ... processeurs.

(N_x, N_y, N_z)	SD	CPU	Iter	# unkn	# cells	h	$\frac{\lVert p^\star - p_h \rVert_{L^2}}{\lVert p^\star \rVert_{L^2}}$
$1 \times 1 \times 1$	1	$17'20''$	0	157 760	51 200	h_1	4.304%
$2 \times 2 \times 2$	8	$21'08''$	8	1 262 080	409 600	$h_1/2$	2.154%
$3 \times 3 \times 3$	27	$52'19''$	33	4 259 520	1 382 400	$h_1/3$	1.436%
$4 \times 4 \times 4$	64	$161'46''$	32	10 096 640	3 276 800	$h_1/4$	1.077%

TAB. 3.1 – Convergence de la solution discrète vers la solution exacte dans le cas conforme. Tous les sous-domaines sont identiques à une translation près et se raccordent sur leurs interfaces. Chacun possède 51 200 mailles rectangulaires. Le système linéaire de chaque sous-domaine a 157 760 inconnues. La norme L^2 de l'erreur relative en pression est fonction du nombre de sous-domaines. Le temps CPU est donné en minutes ($'$) et secondes ($''$). On donne aussi la configuration de la décomposition en sous-domaines, le nombre d'itérations de Bi-CGStab pour converger, le nombre *total* de mailles dans le domaine global, le nombre *total* d'inconnues dans le domaine global et le pas de discrétisation h en espace en fonction du pas h_1 du calcul avec un seul sous-domaine.

Chaque sous-domaine possède approximativement 50 000 mailles. La charge totale pour, mettons, 16 sous-domaines est alors environ 16 fois plus grande que pour un seul sous-domaine, mais comme le calcul est lancé en parallèle sur 16 machines, on espère le faire dans le même temps que pour un sous-domaine, ou pendant un temps légèrement plus long.

Plusieurs remarques peuvent être faites au vu des résultats obtenus.

Tout d'abord, il existe toujours un surcoût en temps CPU entre un calcul avec un sous-domaine, c'est-à-dire sans parallélisme, et un calcul avec plusieurs sous-domaines. Ceci est normal dans la mesure où un calcul parallèle requiert en plus par rapport à un calcul séquentiel la création d'un réseau de calcul, des itérations de Bi-CGStab, des communications, une phase de projection, etc.

(N_x, N_y, N_z)	# SD	CPU	Iter	total # cells	T_n/T_1
$1 \times 1 \times 1$	1	$17'20''$	0	51 200	1
$1 \times 1 \times 2$	2	$21'21''$	6	102 400	1.23
$1 \times 1 \times 4$	4	$21'51''$	18	204 800	1.26
$1 \times 1 \times 8$	8	$20'20''$	18	409 600	1.17
$1 \times 1 \times 16$	16	$21'47''$	22	819 200	1.26
$1 \times 2 \times 2$	4	$17'59''$	8	204 800	1.04
$1 \times 2 \times 4$	8	$20'20''$	8	409 600	1.17
$1 \times 2 \times 8$	16	$21'59''$	21	819 200	1.27
$1 \times 4 \times 4$	16	$22'41''$	23	819 200	1.31
$2 \times 2 \times 2$	8	$21'08''$	8	409 600	1.22
$2 \times 2 \times 4$	16	$23'03''$	20	819 200	1.33

TAB. 3.2 – Test d'extensibilité dans le cas conforme. Tous les sous-domaines sont identiques à une translation près, et se raccordent sur leurs interfaces. Chacun possède 51 200 mailles rectangulaires. Le temps CPU est fonction du nombre de sous-domaines, qui est égal au nombre de processeurs utilisés. Le temps CPU est donné en minutes ($'$) et secondes ($''$). On donne aussi la configuration de la décomposition en sous-domaines, le le nombre d'itérations de Bi-CGStab pour atteindre la convergence, le nombre *total* de mailles dans le domaine global, et le rapport T_n/T_1 des temps de calcul pour n sous-domaines et pour 1 sous-domaine.

(N_x, N_y, N_z)	# SD	CPU No Col	CPU Col	# Iter	total # cells
$1 \times 1 \times 1$	1	$17'20''$	$17'20''$	0	51 200
$1 \times 1 \times 2$	2	$44'29''$	$26'25''$	13	100 604
$1 \times 1 \times 4$	4	$30'50''$	$25'12''$	16	208 046
$1 \times 1 \times 8$	8	$32'36''$	$25'03''$	18	386 092
$1 \times 1 \times 16$	16	$41'08''$	$29'13''$	23	804 654
$1 \times 2 \times 2$	4	$39'33''$	$29'14''$	19	201 878
$1 \times 2 \times 4$	8	$50'53''$	$27'57''$	28	398 916
$1 \times 2 \times 8$	16	$44'28$	$34'13''$	32	816 904
$1 \times 4 \times 4$	16	$37'05''$	$31'28''$	26	785 496
$2 \times 2 \times 2$	8	$37'37''$	$33'25''$	30	421 691
$2 \times 2 \times 4$	16	$45'05''$	$31'00''$	31	812 194

TAB. 3.3 – Test d'extensibilité dans le cas non conforme. Tous les sous-domaines sont différents et ne se raccordent pas sur leurs interfaces. Chacun possède environ 50 000 mailles rectangulaires (à $\pm 10\,000$ mailles près). Le temps CPU est fonction du nombre de sous-domaines, qui est égal au nombre de processeurs utilisés. Le temps CPU est donné en minutes ($'$) et secondes ($''$) dans deux cas : dans la colone "No Col", l'option des couleurs n'est pas employée, et dans la colonne "Col", la coloration des tâches et des nœuds physiques a servi à équilibrer la charge de calcul. On donne aussi la configuration de la décomposition en sous-domaines, le le nombre d'itérations de Bi-CGStab pour atteindre la convergence et le nombre *total* de mailles dans le domaine global.

En revanche, quand le nombre de sous-domaines augmente au-delà de deux, le temps de calcul reste approximativement constant et le nombre d'itérations de Bi-CGStab augmente lentement. Cela dépend du cas-test : quand les maillages d'interface sont raccordés, la phase de projection est beaucoup plus rapide et moins coûteuse en termes de mémoire que dans le cas où les maillages ne se raccordent pas à l'interface. De plus, la convergence de Bi-CGStab est retardée quand les maillages sont non concordants. Ceci explique pourquoi le temps CPU est plus grand dans ce dernier cas. Ceci peut aussi expliquer pourquoi la configuration de la décomposition en sous-domaines joue un rôle plus important dans le cas non conforme que dans le cas raccordé : plus il y a d'interfaces, plus il y a de projections et plus coûteuse est chaque itération.

Troisièmement, la phase d'initialisation pendant laquelle est opérée la factorisation LU des matrices de problèmes de Robin (2.48), s'est avérée la partie la plus coûteuse du calcul global à la fois en termes de temps de calcul et en termes de mémoire : pour un maillage de 51 200 mailles, la factorisation nécessite, on l'a vu, approximativement $1,4$ GO de mémoire, ce qui est en-dessous de la taille mémoire disponible sur un nœud. Dans le cas concordant, il n'y a donc pas eu d'utilisation de la mémoire auxiliaire et le nœud le plus lent a retardé tous les autres.

Cependant dans le cas non concordant, les sous-domaines ne sont pas tous identiques : certains possèdent environ 40 000 mailles tandis que d'autres ont plus de 60 000 mailles. Les processeurs s'occupant des sous-domaines les plus raffinés ont swappé pendant la factorisation. Dans ces conditions, il est devenu important d'équilibrer correctement la charge de calcul avec les ressources informatiques disponibles. Les couleurs, voir Section 3.4.8, se sont révélées utiles pour cet équilibrage : on a pu affecter les sous-domaines les plus petits aux processeurs les plus lents, pendant que les autres nœuds se chargeaient du reste. Nous montrons dans la Table 3.3 une comparaison entre des calculs utilisant les couleurs et d'autres sans les couleurs, et donc avec une distribution des charges arbitraire (pas forcément la pire). On observe un gain qui peut s'élever à 25% ou même 40% du temps de calcul.

Enfin, comme nous l'avons remarqué en introduction du Chapitre 3, il existe un goulot d'étranglement de communication dans ce code : il n'est pas *scalable*. Ceci est dû à la fois à la méthode de décomposition de domaines telle qu'elle est implémentée, et à la manière dont sont traitées les communications par OcamlP3l. En effet, toutes les communications sont centralisées et par conséquent quand le nombre de sous-domaines augmente, les communications qui sont envoyées au nœud racine (*root*) qui gère la centralisation, deviennent une charge de plus en plus importante. Ceci peut engendrer une surcharge importante due aux communications à chaque itération. Cependant, ces expériences n'ont pas mis en évidence cette limitation, du fait que le nombre de sous-domaines est resté relativement petit. Mais pour des tests plus poussés avec un grand nombre de sous-domaines, il pourrait devenir critique de surmonter ce problème. C'est pourquoi les développeurs d'OcamlP3l ont déjà commencé à envisager une structure permettant de permettre des communications directement entre deux nœuds du réseau.

3.6.2 Simulation d'un écoulement 3D réaliste

Nous présentons dans cette Section les résultats d'écoulement obtenus sur un modèle 3D réaliste issu de l'Andra, [80]. Une simplification a été opérée par rapport au problème "réel" traité par l'Andra : nous avons considéré que les sous-domaines sont entièrement homogènes, ce qui n'est pas entièrement exact. Nous décrivons donc intégralement le modèle étudié ici. Dans la suite, nous nous conformerons à l'usage établi à l'Andra et qualifierons de *modèle oxfordien* tout ce qui se rapporte à ce problème.

Ce problème est difficile à résoudre numériquement essentiellement pour deux raisons : premièrement, la géométrie du domaine se caractérise par une très forte anisotropie, le domaine ayant l'apparence d'une galette, comme on peut le voir à échelle réelle sur la partie gauche de la Figure 3.8. Deuxièmement, les couches géologiques sont très hétérogènes entre elles ; les coefficient de perméabilité sautent de plusieurs ordres de grandeurs d'une couche à l'autre.

Description du modèle

Nous résolvons le problème d'écoulement (2.2) dans le domaine Ω présenté sur les Figures 3.8, 3.9 et 3.10. Il est constitué de plusieurs couches géologiques, elles-mêmes divisées en deux zones : une zone dite "régionale" et une zone "locale" où les perméabilités sont plus faibles. Le modèle comprend 6 strates hydrogéologiques dont nous donnons la liste de haut en bas : le Tithonien, le Kimméridgien, l'Oxfordien supérieur L2a-L2b, l'Oxfordien Hp1-Hp4, l'Oxfordien C3a-C3b et le Callovo-Oxfordien, parfois raccourci en "C-Ox". L'Andra étudie la possibilité de stocker des déchets dans cette dernière couche, le Callovo-Oxfordien, car c'est un type d'argile très peu perméable. L'emplacement du laboratoire de Bure où sont faites des expériences in-situ est indiqué sur la Figure 3.9.

Géométrie du domaine Ω Les couches inférieures sont des prismes à base hexagonale et d'épaisseurs constantes. Seuls le Tithonien et le Kimméridgien ont des géométries plus complexes. L'hexagone servant de base peut être vu sur la Figure 3.9. Les coordonnées de ses sommets $O_i, i = 1, \ldots, 6$, sont données dans la Table 3.4. On doit noter que le domaine a une extension horizontale de l'ordre de 40 kilomètres et une hauteur de l'ordre de 500 mètres, soit environ cent fois plus petite, cf. Figure 3.8, à gauche. Pour toutes les autres visualisations et pour certains calculs, une mise à l'échelle est réalisée : les altitudes sont mutlipliées par un facteur 100. Cette homothétie d'axe Oz et de facteur 100 définit un nouveau domaine Ω' tel que :

$$\Omega' := \left\{(x, y, z') \in \mathbb{R}^3 \ / \ (x, y, z) \in \Omega \text{ and } z' = 100z\right\}. \tag{3.1}$$

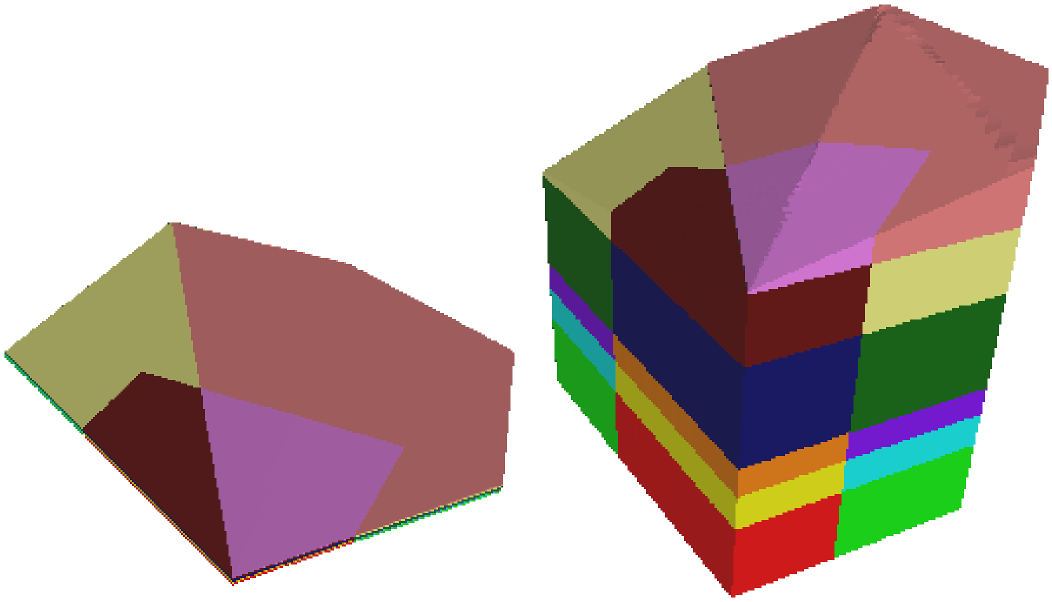


Fig. 3.8 – Vues extérieures du domaine 3D du modèle oxfordien. Gauche : domaine Ω à échelle réelle. Droite : domaine Ω', obtenu par dilatation du domaine Ω d'un facteur 100 selon l'axe Oz. Visualisation grâce à Medit (outil de visualisation de maillage de l'Inria, [52]). On observe les différents sous-domaines dessinés avec des couleurs distinctes. Les sous-domaines sont stratifiés par couches horizontales. Dans chaque couche géologique, deux sous-domaines sont accolés : le premier, à l'avant sur la Figure, est la zone dite "locale" qui s'encastre dans le second, la zone "régionale", dont on voit les faces latérales à gauche et à droite du premier. Le sous-domaine rouge contient le site de stockage.

Le Tithonien est encadré par un toit (frontière supérieure) et un plancher (frontière inférieure) divisés en plusieurs morceaux de plans horizontaux et inclinés. Cette formation est affleurante, et coiffe la quasi-totalité du domaine Ω. La projection du toit du Tithonien sur un plan horizontal est le pentagone $O_1O_3O_4O_5O_6$.

Le Kimméridgien possède un plancher horizontal et un toit divisé en trois morceaux de plans dont un est horizontal. Cette formation affleure dans la partie Sud du domaine, délimitée par les points O_1, O_2, O_3, à l'emplacement où le Tithonien n'existe pas.

Comme nous l'avons dit plus haut, toutes les couches se divisent en deux parties. La zone "locale" est un prisme de base pentagonale $S_1S_2S_4S_5O_3$, voir la Figure 3.9. La zone "régionale" est le complémentaire de l'autre zone.

Point	O_1	O_2	O_3	O_4	O_5	O_6
X (Lambert1 km)	790	$817,5$	$832,5$	816	798	790
Y (Lambert1 km)	1074	1074	1099	1111	1108	$1091,5$
Topographie (m)	145	145	251	341	341	341
Subs. Tithonien (m)	145	$--$	251	251	251	251
Subs. Kimméridgien (m)	145	145	145	145	145	145
Subs. Oxfordien L2a-L2b (m)	-20	-20	-20	-20	-20	-20
Subs. Oxfordien Hp1-Hp4 (m)	-70	-70	-70	-70	-70	-70
Subs. Oxfordien C3a-C3b (m)	-130	-130	-130	-130	-130	-130
Subs. Callovo-Oxfordien (m)	-265	-265	-265	-265	-265	-265

TAB. 3.4 – Cotes des points délimitant le domaine Ω du modèle oxfordien. Le substratum ("Subs."), qui modélise la frontière inférieure, est donné en mètres, tandis que les coordonnées horizontales sont exprimées en kilomètres.

Perméabilités Les perméabilités que nous avons utilisées dans nos calculs sont données dans la Table 3.5. Nous supposons que dans chaque couche, les zones régionale et locale sont homogènes et ont une perméabilité $\mathbf{K}$ qui s'écrit

$$\mathbf{K} = \begin{bmatrix} K_h & 0 & 0 \\ 0 & K_h & 0 \\ 0 & 0 & K_v \end{bmatrix},$$

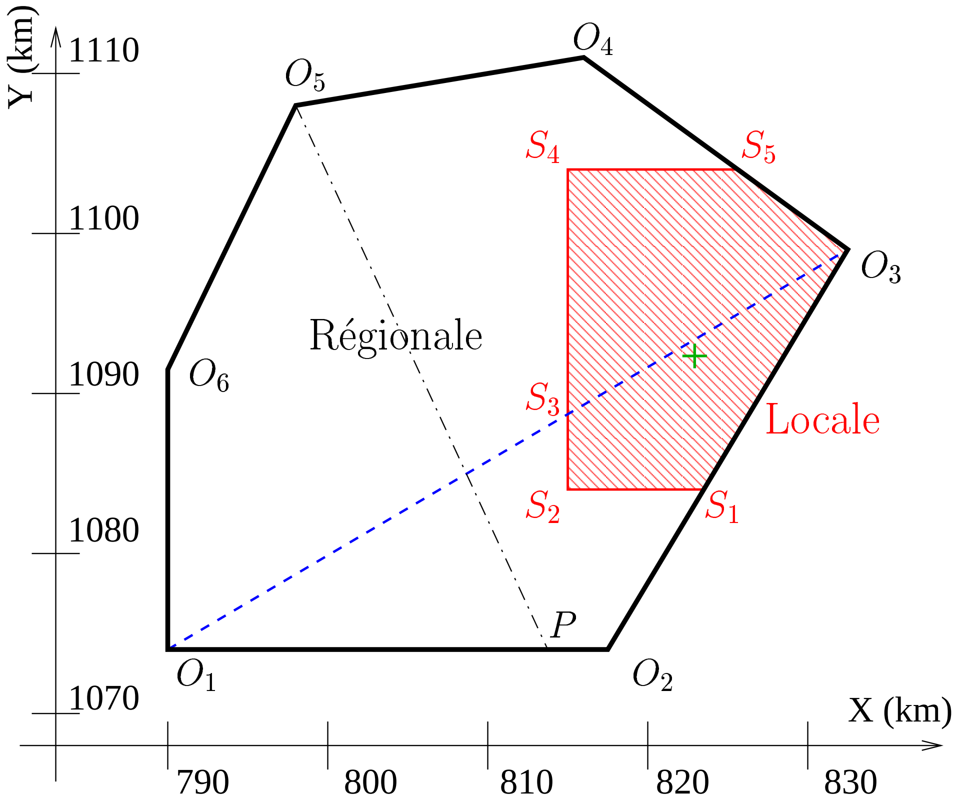


FIG. 3.9 – Vue de dessus de la géométrie du modèle oxfordien. On distingue les deux zones, régionale et locale (hachurée en rouge), possédant des propriétés physiques différentes. L'emplacement du laboratoire de Bure est positionné au niveau de la croix verte. Le trait discontinu bleu O_1O_3 marque la limite Sud de l'affleurement du Tithonien. La ligne O_5P montre le plan de coupe de la Figure 3.10.

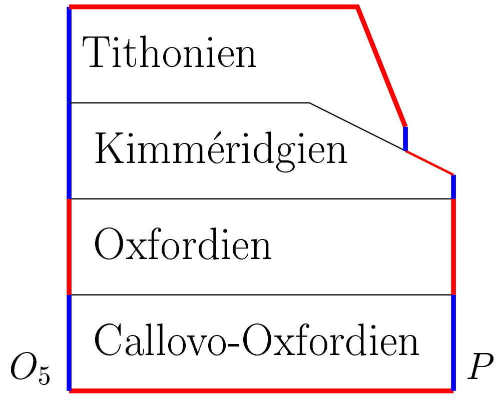


FIG. 3.10 – Schéma simplifié d'une coupe verticale selon O_5P du modèle oxfordien, cf. Figure 3.9. Les quatre strates hydrogéologiques principales sont indiquées. L'oxfordien se scinde en trois sous-couches. Sur les bords qui sont dessinées en rouge, des conditions de type Dirichlet sont imposées. Sur les bords bleus, ce sont des condtions de type Neumann homogène (flux nul). Les échelles réelles ne sont évidemment pas respectées.

où K_h est le coefficient de perméabilité horizontale, et K_v la perméabilité verticale. Les deux valeurs sont exprimées en mètre par seconde[33]. Le tenseur de perméabilité est généralement scalaire, la seule exception étant le Callovo-Oxfordien au sein duquel la perméabilité verticale est 100 fois plus faible que la perméabilité horizontale.

Ces perméabilités ont deux caractéristiques principales : d'une part, les valeurs sont extrêmement petites, et d'autre part elles sont particulièrement hétérogènes. Le saut entre les zones locale et régionale du Kimméridgien est de 7 ordres de grandeur !

Conditions aux limites Des conditions aux limites de type Neumann homogène sont affectées à certaines parois latérales des couches géologiques. Celles-ci sont définies dans le Tableau 3.5. Partout ailleurs, des conditions de type Dirichlet sont imposèes ; elles sont lues par des fichiers de données fournis par l'Andra. Les charges imposées varient entre 470 au plancher du Callovo-Oxfordien et 150 au toit du Tithonien. On peut voir sur la coupe verticale 2D de la Figure 3.10 comment sont appliquées les conditions aux limites.

[33]On a gardé cette unité. On pourrait également travailler en m/an ou en km/an qui ont plus de sens pour un calcul de transport.

Couche	zone	SD	$\Delta z(m)$	$K_h(m/s)$	$K_v(m/s)$	Neumann hom.
Tithonien	Rég.	12	$0 \leq 90$	$3E-5$	$3E-5$	$S_5O_4O_5O_6O_1S_3$
Tithonien	Loc.	11	$0 \leq 90$	$3E-5$	$3E-5$	$S_3O_3S_5$
Kimmér.	Rég.	10	$0 \leq 100$	$1E-11$	$1E-11$	$S_5O_4O_5O_6O_1O_2S_1$
Kimmér.	Loc.	9	$0 \leq 100$	$3E-4$	$3E-4$	$S_1O_3S_5$
Ox. L2a-L2b	Rég.	8	165	$2E-7$	$2E-7$	O_2S_1
Ox. L2a-L2b	Loc.	7	165	$1E-9$	$1E-9$	$S_1O_3S_5$
Ox. Hp1-Hp4	Rég.	6	50	$6E-7$	$6E-7$	O_2S_1
Ox. Hp1-Hp4	Loc.	5	50	$8E-9$	$8E-9$	$S_1O_3S_5$
Ox. C3a-C3b	Rég.	4	60	$1E-9$	$1E-9$	O_2S_1
Ox. C3a-C3b	Loc.	3	60	$1E-11$	$1E-11$	$S_1O_3S_5$
C-Ox.	Rég.	2	135	$1E-11$	$1E-13$	$S_5O_4O_5O_6O_1O_2S_1$
C-Ox.	Loc.	1	135	$1E-11$	$1E-13$	$S_1O_3S_5$

TAB. 3.5 – Caractéristiques physiques de chaque sous-domaine : couche géologique (Tithonien, Kimméridgien, Oxfordien, Callovo-Oxfordien), zone (régionale ou locale), numéro du sous-domaine, épaisseur de la couche, coefficients de perméabilité selon l'horizontale et la verticale, et bords où sont imposées les conditions aux limites de type Neumann homogène. Ces bords sont toujours des parois latérales (verticales) des sous-domaines. Ils sont donc complètement déterminés par la liste des points qui les limitent dans un plan horizontal : les points $O_i, i = 1, \ldots, 6$, ou $S_j, j = 1, \ldots, 5$, de la Figure 3.9.

Découpage en sous-domaines L'idée est ici de découper le domaine *en suivant la géologie*. Nous divisons donc le domaine en 12 sous-domaines : deux sous-domaines par couche, un sous-domaine recouvrant chaque zone, voir la Table 3.5 ; de cette façon, les sous-domaines sont tous homogènes. Nous comptons ensuite sur la méthode de décomposition de domaines pour maîtriser l'impact des sauts de perméabilité sur la raideur du problème.

Maillages et décomposition Le maillage initial du domaine Ω complet était fourni par l'Andra. Il est constitué de 384 582 hexaèdres. La topologie des couches, cf. Figure 3.10, et les conditions imposées par la géométrie du site de stockage, voir la Figure 3.11, ont contribué à la déformation importante de certaines mailles. Dans les couches les plus basses qui sont géométriquement des prismes, les déformations sont restées 2D : les mailles sont des quadrangles déformés extrudés. Mais la couche du Tithonien contient des mailles qui ont des déformations 3D ; pour ces mailles hexaédriques déformées, la convergence de la méthode des éléments finis mixtes n'est plus assurée, comme l'ont fait remarquer Naff, Russell et Wilson dans [75], voir pour plus de détails la Section 3.2.1. Il serait nécessaire dans ce cas de prendre les fonctions de forme vectorielles préconisées dans [75], ou bien de changer complètement le fusil d'épaule et d'utiliser un maillage tétraédrique. Nous n'avons pas eu le temps de développer plus avant cette intéressante question, que nous laissons donc à la relève.

En outre, le maillage est bien évidemment caractérisé par une très forte anisotropie, induite par celle du domaine lui-même.

La décomposition du maillage global de Ω en sous-domaines *qui suivent la physique*, est une tâche laborieuse et ingrate. Il n'existe pas à ma connaissance de code qui permette de réaliser cette délicate opération. Elle a donc été faite dans ce cas précis pratiquement à la main par Amel Sboui[34]. La décomposition a été faite de façon conforme : les sous-domaines sont tous raccordés aux interfaces. Il ne nous a pas semblé justifié de perdre la conformité qui existait dans le maillage initial. On peut visualiser une vue de dessous des deux maillages du Callovo-Oxfordien sur les Figures 3.12 et 3.13. Une vue de dessus du maillage complet est présentée sur la Figure 3.11.

Le fait de décomposer le domaine en suivant la géologie pose quand même un problème important : les sous-domaines sont homogènes en termes de perméabilité mais pas en terme de taille des maillages. Il n'y a en effet aucune raison pour que les couches géologiques aient toutes des maillages de même taille. Dans la pratique, elles ont des maillages très différents : le plus petit sous-domaine

[34]du projet Estime - INRIA Rocquencourt, que je remercie chaleureusement au passage.

contient un peu plus de 10 000 mailles, alors que le plus gros en contient plus de 70 000, voir la Table 3.6. La charge n'est donc pas du tout équilibrée dans ce cas et le parallélisme reste limité du fait que le plus gros sous-domaine retarde tous les autres. Deux remèdes sont susceptibles d'aider à répartir la charge de façon équilibrée : on peut redécouper les gros sous-domaines en plus petits sous-domaines pour homogénéiser les tailles des maillages ; ceci a le désavantage de multiplier les sous-domaines et repose la question de la décomposition de maillage, même si on peut dans ce cas utiliser des partitionneurs automatiques. L'autre possibilité consiste à affecter plusieurs petits sous-domaines à un même nœud physique, quand la mémoire est suffisante. Il est aussi envisageable de combiner ces deux techniques. Nous n'irons pas plus loin sur ce sujet car nous ne nous sommes pas attachés à résoudre ici ce problème d'équilibrage.

Solution et méthode de résolution

Le problème d'écoulement (2.2) est résolu après décomposition du domaine Ω en 12 sous-domaines ayant une perméabilité homogène et qui sont raccordés aux interfaces. La méthode employée est l'algorithme de décomposition de domaines de Robin–Robin étudié à la Section 2.3. Nous utilisons à nouveau le coupleur OcamlP3l parallèle présenté à la Section 3.5 et validé à la Section 3.6.1. La convergence de l'algorithme Bi-CGStab est atteinte quand la norme infinie du résidu est réduite d'un facteur 10^{10}.

Les calculs ont été effectués avec 12 nœuds de la grappe de PC de l'Inria Rocquencourt, voir leur description dans l'introduction de la Section 3.6 : chaque sous-domaine est affecté à un unique nœud. Nous utilisons encore une fois les couleurs d'OcamlP3l pour que les trois processeurs les plus lents reçoivent les trois sous-domaines les plus petits, voir la Section 3.4.8 pour la définition des couleurs.

Comme nous l'avons vu plus haut, la décomposition étant faite en suivant la géologie, la charge est mal répartie et le sous-domaine numéro 1 qui est le plus gros retarde systématiquement tous les autres processeurs. Pour dire un dernier mot sur les performances, assez faibles dans cette expérience, la phase de factorisation dure environ 7 à 8 minutes ; chaque itération de Bi-CGStab prend approximativement 7 secondes. Ainsi, la résolution du problème nécessitant 557 itérations a duré 1 heure 22 minutes ; pour le problème dont la résolution s'est faite en 207 itérations, il a fallu attendre un peu moins de 36 minutes.

On peut voir une représentation de la solution en pression sur la Figure 3.14.

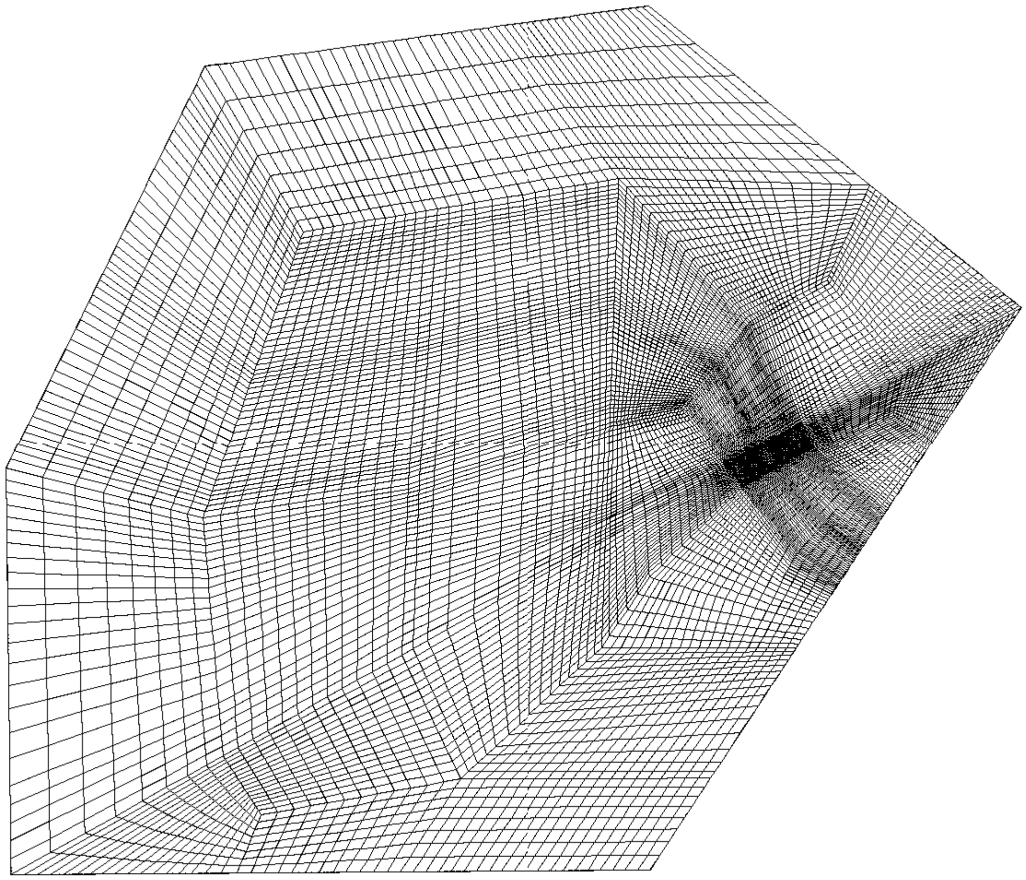


FIG. 3.11 – Vue de dessus du maillage du modèle oxfordien. Visualisation avec Medit, [52].

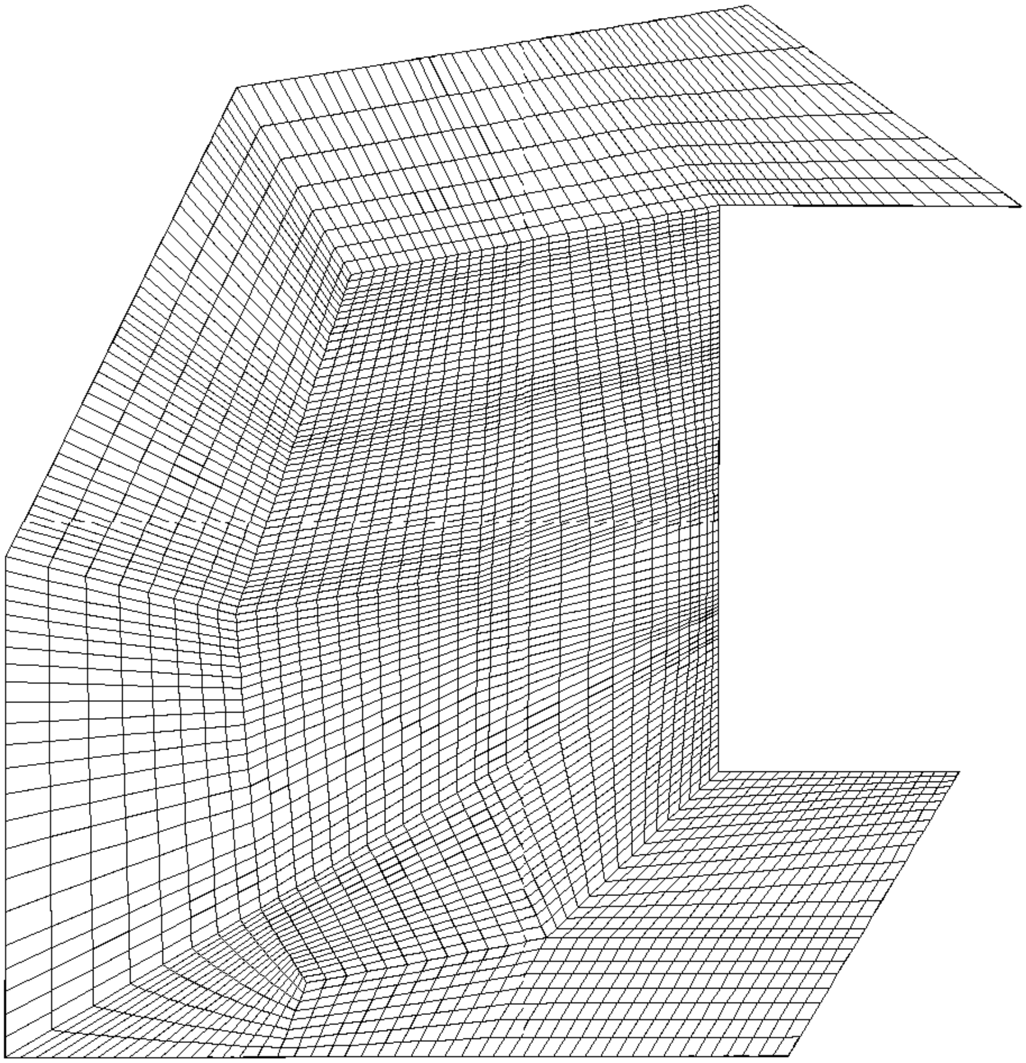


FIG. 3.12 – Vue de dessous du maillage de la zone régionale du modèle oxfordien. Visualisation avec Medit, [52].

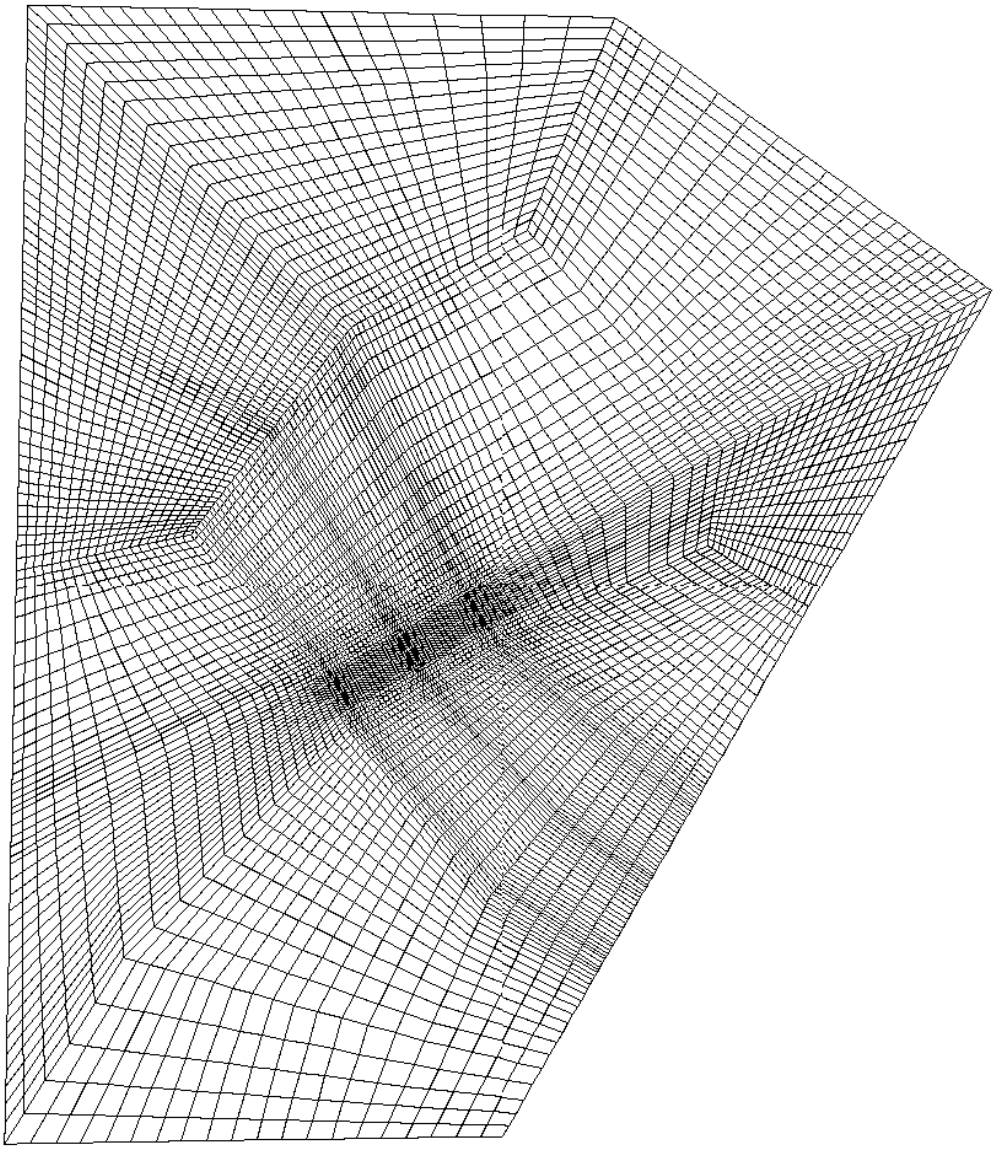


FIG. 3.13 – Vue de dessous du maillage de la zone locale du modèle oxfordien. Echelle différente de la Figure 3.12. Visualisation avec Medit.

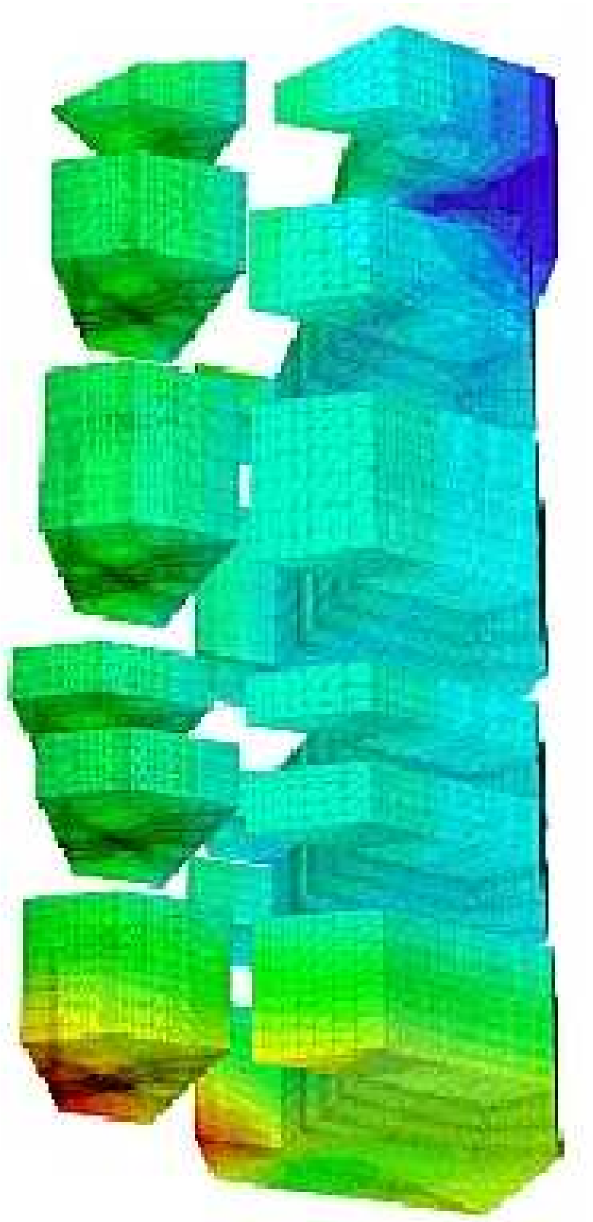


FIG. 3.14 – Vue éclatée des 12 sous-domaines 3D du modèle oxfordien avec représentation de la solution en pression. Les fortes charges sont dessinées en rouge (valeur maximale ≈ 470), et les faibles charges en bleu (valeur maximale ≈ 150). Les échelles de la géométrie ne sont pas respectées.

Couche	zone	SD	# mailles	CPU Facto.
Tithonien	Rég.	12	14 130	7″
Tithonien	Loc.	11	19 005	15″
Kimmér.	Rég.	10	23 862	18″
Kimmér.	Loc.	9	44 160	92″
Ox. L2a-L2b	Rég.	8	27 839	32″
Ox. L2a-L2b	Loc.	7	51 520	156″
Ox. Hp1-Hp4	Rég.	6	11 931	2″
Ox. Hp1-Hp4	Loc.	5	22 080	9″
Ox. C3a-C3b	Rég.	4	19 885	11″
Ox. C3a-C3b	Loc.	3	36 800	57″
C-Ox.	Rég.	2	39 770	82″
C-Ox.	Loc.	1	73 600	447″

TAB. 3.6 – Caractéristiques numériques de chaque sous-domaine : nom de la couche géologique (Tithonien, Kimméridgien, Oxfordien, Callovo-Oxfordien), type de zone (régionale ou locale), numéro du sous-domaine, nombre de mailles et temps de calcul nécessaire pour la factorisation en secondes.

Etude de convergence en fonction des coefficients de Robin

Nous discutons ici les résultats de convergence obtenus avec les différents coefficients de Robin $\alpha_{ij}, (i,j) \in \mathcal{N}$ qui ont été expérimentés[35]. La liste de ces coefficients est fournie à l'équation (3.2). Dans tous les cas, sur chaque interface géométrique Σ_{ij}, le coefficient de Robin est constant : $\alpha_{ij} = \alpha_{ji} = Cst(i,j), (i,j) \in \mathcal{N}$. Nous notons avec un exposant $k = 0, 1, 2, 3$ les suites des coefficients de Robin $\alpha^k := (\alpha^k_{ij})_{(i,j)\in\mathcal{N}}$ indexées par les numéros d'interface entre deux sous-domaines.

$$
\begin{aligned}
\alpha^0_{ij} &= \frac{2\mathbf{K_i}\mathbf{K_j}}{(\mathbf{K_i}+\mathbf{K_j})L^{ij}_\Omega} = \frac{\mathrm{MoyH}\,(\mathbf{K_i},\mathbf{K_j})}{L^{ij}_\Omega} && (i,j)\in\mathcal{N},\\
\alpha^1_{ij} &\equiv 1 && (i,j)\in\mathcal{N},\\
\alpha^2_{ij} &= \frac{2\mathbf{K_i}\mathbf{K_j}}{\mathbf{K_i}+\mathbf{K_j}} = \mathrm{MoyH}\,(\mathbf{K_i},\mathbf{K_j}) && (i,j)\in\mathcal{N},\\
\alpha^3_{ij} &= \frac{2\mathbf{K_i}\mathbf{K_j}}{(\mathbf{K_i}+\mathbf{K_j})L^{ij}_\Omega}\,\frac{L^{\max}_\Omega}{L^{\min}_\Omega} = \frac{\mathrm{MoyH}\,(\mathbf{K_i},\mathbf{K_j})}{L^{ij}_\Omega}\,\frac{L^{\max}_\Omega}{L^{\min}_\Omega} && (i,j)\in\mathcal{N},
\end{aligned}
\tag{3.2}
$$

[35]Voir la définition de la notation $\mathcal{N}$ dans la Section 2.2.2

où nous rappelons, voir la Section 2.3.8, que L_Ω^{ij} est la longueur caractéristique du domaine Ω selon la direction normale à l'interface Σ_{ij}, et $L_\Omega^{\max}$ et $L_\Omega^{\min}$ sont respectivement les longueurs maximale et minimale du domaine Ω. La définition de la moyenne harmonique $\mathrm{MoyH}(\cdot)$ est rappelée en Annexe B à l'équation (B.4). Nous notons que les coefficients α^1 et α^2 n'ont pas la bonne dimension, au sens physique du terme : tandis que les autres coefficients de Robin, α^0 et α^3 sont homogènes au rapport d'une perméabilité par une longueur, ce qui est correct, en revanche α^1 est sans dimension et α^2 a la dimension d'une perméabilité. Nous verrons que les résultats les plus probants sont donnés avec les coefficients qui sont conformes à la physique : α^0 et α^3.

Pour fixer les idées, nous donnons les valeurs prises pour les longueurs caractéristiques : $L_\Omega^{ij} = L_{\Omega'}^{ij} = 40\,000$ si l'interface $\Sigma_{ij}, (i,j) \in \mathcal{N}$, est verticale. Si l'interface Σ_{ij} est horizontale, alors $L_\Omega^{ij} = 500$ et $L_{\Omega'}^{ij} = 50\,000$. Le coefficient de forme utilisé pour α^3 s'écrit

$$\frac{L_\Omega^{\max}}{L_\Omega^{\min}} = \frac{40\,000}{500} = 80, \qquad \frac{L_{\Omega'}^{\max}}{L_{\Omega'}^{\min}} = \frac{40\,000}{50\,000} = 0,8 \approx 1.$$

Nous remarquons que le facteur de forme du domaine Ω', $L_{\Omega'}^{\max}/L_{\Omega'}^{\min}$ est proche de 1 et sera considéré égal à 1 dans la suite. Ceci permet de voir que dans le cas du domaine Ω', deux des coefficients de Robin sont égaux : $\alpha^0 = \alpha^3$; il est par conséquent inutile de faire deux calculs différents.

Disons dès maintenant que les résultats les plus probants ont été obtenus avec les coefficients α^0, donné à l'équation (3.2) ; la courbe de convergence est montrée en rouge dans la Figure 3.17. Avec toutes les autres valeurs de α testées, la méthode stagne sur le modèle oxfordien. Le choix correct du coefficient de Robin est crucial pour la méthode de décomposition de domaines de Robin–Robin non conforme.

Nous cherchons à mettre en évidence les effets des deux difficultés numériques principales de ce problème : l'anisotropie et les sauts de perméabilité.

Anisotropie Pour isoler les effets de l'anisotropie, nous fixons dans ce paragraphe les perméabilités à 1 dans tous les sous-domaines : $\mathbf{K_i} \equiv 1, i = 1, 2, \ldots, 12$. Nous résolvons deux problèmes d'écoulement *distincts* : le premier sur le domaine "réel" Ω, qui est très applati, et le second sur le domaine étiré Ω', qui a des dimensions du même ordre dans toutes les directions. Les deux maillages sont identiques, à l'homothétie d'un facteur 100 selon Oz près.

Les courbes de convergence sont montrées sur la Figure 3.15 et les résultats sont résumés dans le Tableau 3.7.

Il faut noter que la prise en compte des dimensions du domaine accélère grandement la convergence pour le domaine étiré Ω'. Pour le domaine applati Ω, le choix le plus simple, $\alpha^1 = 1$ donne étrangement de meilleurs résultats que α^0. Cependant la prise en compte du facteur de forme donne la convergence la plus rapide.

La conclusion qu'on semble devoir tirer est que le choix du coefficient de Robin $\alpha = \alpha^3$ est le plus approprié dans ce cas.

Coef. Robin ($\mathcal{O} = \Omega$ ou Ω')	$\mathbf{K} \equiv 1, \Omega'$	$\mathbf{K} \equiv 1, \Omega$	$\mathbf{K}$ réel, Ω'	$\mathbf{K}$ réel, Ω
$\alpha^0 = \dfrac{\mathrm{MoyH}\,(\mathbf{K_i}, \mathbf{K_j})}{L^{ij}_{\mathcal{O}}}$	128	436	207	557
$\alpha^1 \equiv 1$	371	156	Diverge	Stagne
$\alpha^2 = \mathrm{MoyH}\,(\mathbf{K_i}, \mathbf{K_j})$	cf. α^1	cf. α^1	Stagne	Stagne
$\alpha^3 = \dfrac{\mathrm{MoyH}\,(\mathbf{K_i}, \mathbf{K_j})}{L^{ij}_{\mathcal{O}}} \dfrac{L^{\max}_{\mathcal{O}}}{L^{\min}_{\mathcal{O}}}$	cf. α^0	82	cf. α^0	Stagne

Tab. 3.7 – Résumé des résultats pour les quatre problèmes étudiés en fonction des coefficients de Robin. Quand la méthode converge, on donne le nombre d'itérations nécessaires pour atteindre la convergence.

Sauts de perméabilité Nous analysons à présent les résultats du problème avec les perméabilités "réelles", telles qu'elles ont été énoncées dans le Tableau 3.5. Encore une fois, nous résolvons deux problèmes *différents* : l'un est posé sur Ω et l'autre sur Ω'. Ceci permet de voir l'effet conjugué de l'anisotropie et des sauts de perméabilité.

Le problème est bien évidemment beaucoup plus raide. Comme nous l'avons mentionné plus haut, le seul coefficient de Robin qui a permis de converger est α^0. Les courbes de convergence sont montrées sur les Figures 3.16 et 3.17. Les résultats sont également résumés dans le Tableau 3.7.

Il faut remarquer que, contrairement à ce que laissait espérer l'expérience précédente, le coefficient α^3 ne permet pas de converger dans ce cas. On note que la décroissance sur les 5 premiers ordres de grandeur est aussi rapide que celle de α^0, mais ensuite Bi-CGStab se met à stagner, voir la Figure 3.17.

Conclusions et perspectives sur ce modèle

Nous avons résolu l'écoulement du modèle 3d "oxfordien", tiré d'un problème réaliste de l'industrie, grâce à la méthode de décomposition de domaines

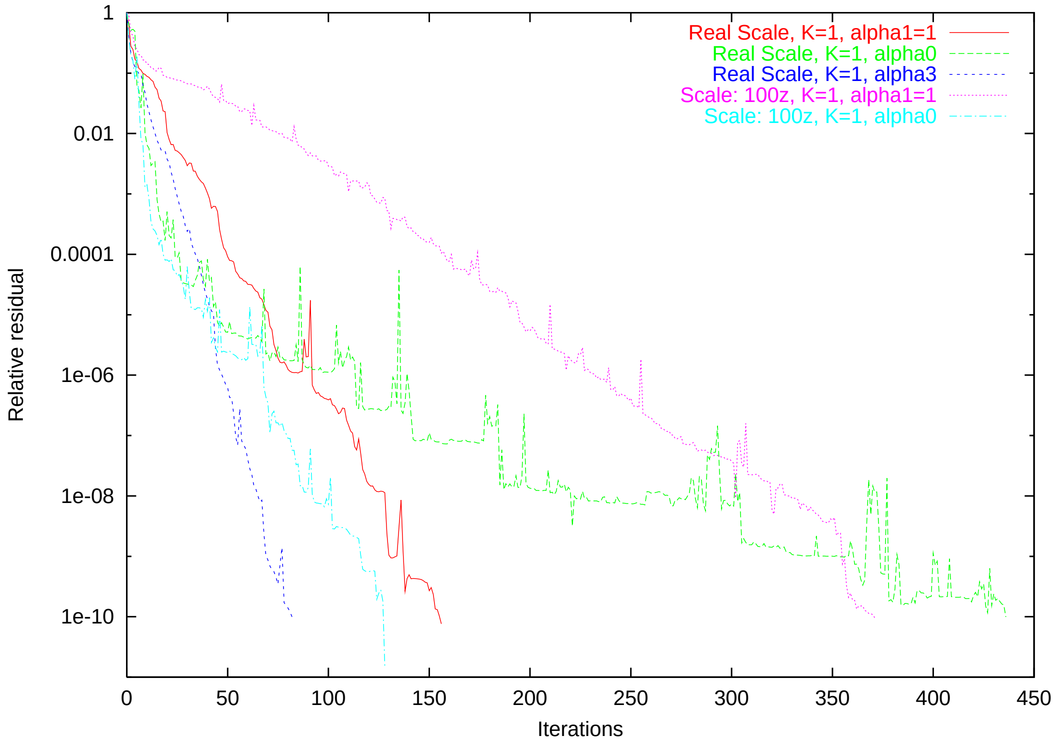


FIG. 3.15 – Courbes de convergence dans le cas où les perméabilités **K** sont prises constantes égales à 1 dans tous les sous-domaines. Résidu relatif du Bi-CGStab en échelle logarithmique dans l'intervalle $[10^{-11}; 1]$, en fonction des itérations. comprises dans l'intervalle $[0; 450]$. Les trois premières courbes, rouge, verte et bleu foncé, montrent la convergence de la méthode respectivement pour $\alpha = \alpha^1 \equiv 1$, $\alpha = \alpha^0$, et $\alpha = \alpha^3$, et résultent d'un calcul avec le domaine plat Ω. Les deux dernières courbes, rose et bleu clair, sont issues de calculs sur le domaine Ω' fortement étiré selon l'axe vertical, avec respectivement $\alpha = \alpha^1 \equiv 1$ et $\alpha = \alpha^0$.

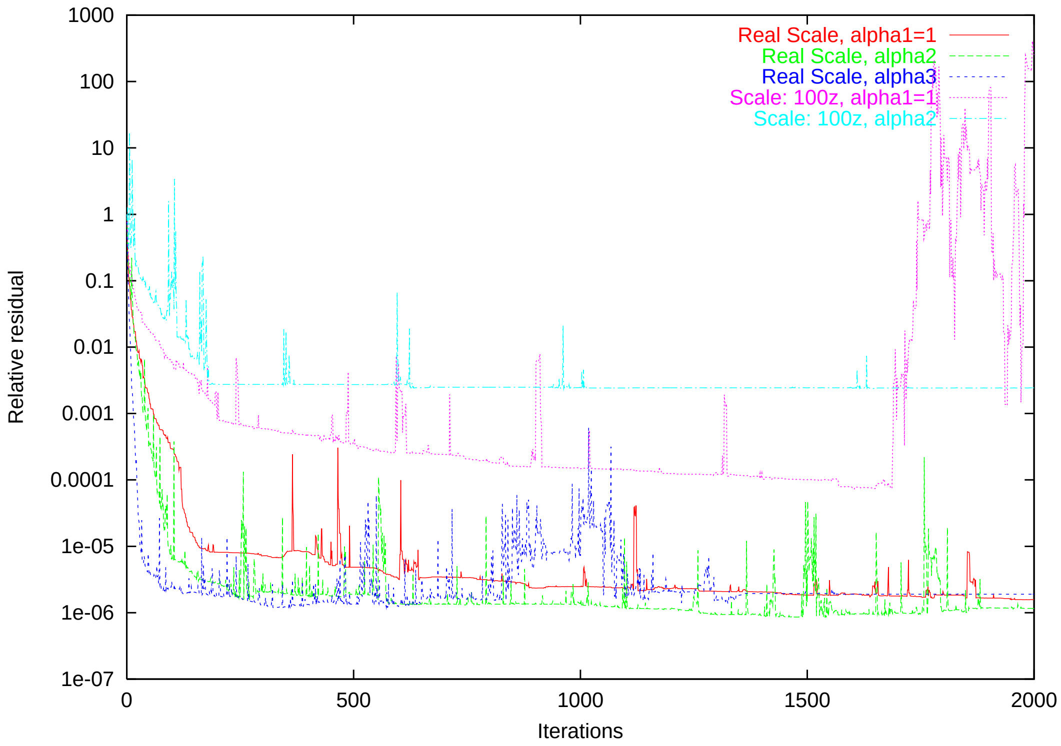


FIG. 3.16 – Courbes de non-convergence pour trois types de coefficients de Robin Résidu relatif du Bi-CGStab en échelle logarithmique dans l'intervalle $[1E-7; 1E+3]$, en fonction des itérations comprises dans l'intervalle $[0; 2000]$. Les trois premières courbes, rouge, verte et bleu foncé, montrent la stagnation de la méthode respectivement pour $\alpha = \alpha^1 \equiv 1$, $\alpha = \alpha^2$, et $\alpha = \alpha^3$, et résultent d'un calcul avec le domaine plat Ω. Les deux dernières courbes, rose et bleu clair, sont issues de calculs sur le domaine Ω' fortement étiré selon l'axe vertical respectivement pour $\alpha = \alpha^1 \equiv 1$ et $\alpha = \alpha^2$; la méthode itérative stagne quand $\alpha = \alpha^2$ et diverge quand $\alpha \equiv 1$: la courbe rose reste au-dessus de $1E + 20$ à partir de 3000 itérations.

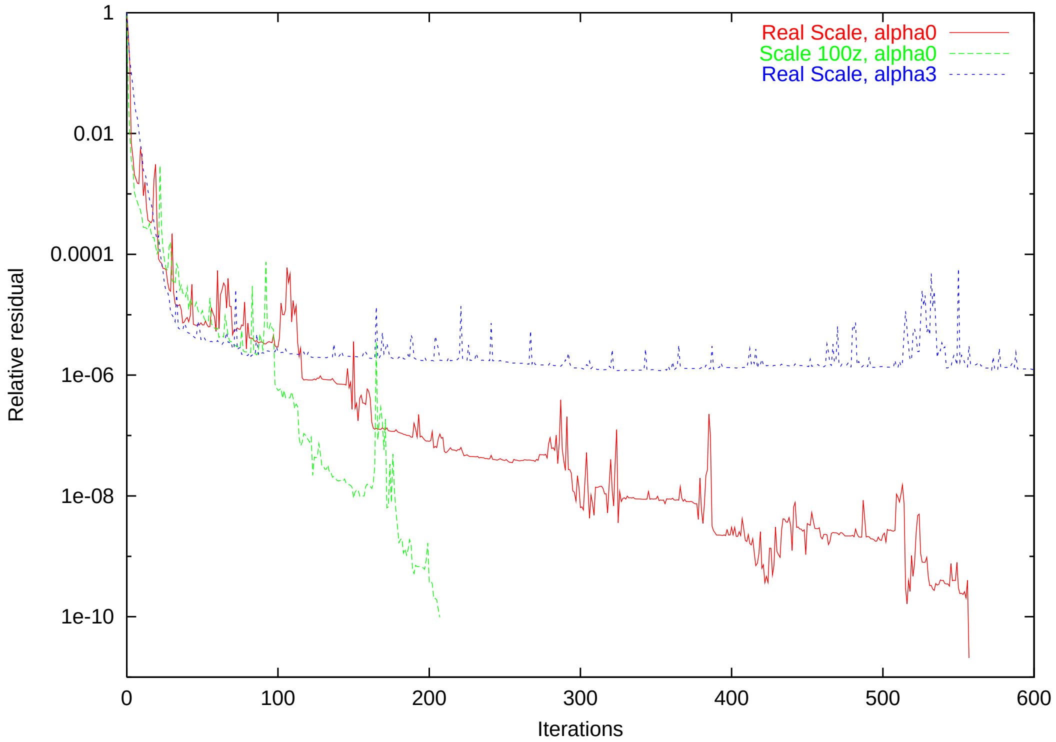


FIG. 3.17 – Courbes de convergence dans le cas où les coefficients de Robin sont : $\alpha = \alpha^0$. Résidu relatif du Bi-CGStab en échelle logarithmique dans l'intervalle $[10^{-11}; 1]$, en fonction des itérations comprises dans l'intervalle $[0; 600]$. La courbe tracée en rouge est le résultat d'un calcul qui a duré 1 heure 22 minutes et été réalisé avec les dimensions *réelles* du domaine Ω —la galette. La courbe tracée en vert est le résultat d'un calcul de 36 minutes sur le domaine Ω' fortement étiré selon l'axe vertical. On a également tracé en bleu un agrandissement de la courbe de stagnation de la Figure 3.16, pour le cas $\alpha = \alpha^3$ et un calcul sur Ω.

de Robin–Robin, et en utilisant le programme de couplage écrit dans l'environnement OcamlP3l. Ce modèle présente deux difficultés numériques principales qui sont liées à l'anisotropie du domaine et aux forts sauts de coefficients.

Nous avons mis en évidence le rôle fondamental du choix du coefficient de Robin pour permettre à la méthode de converger. Une étude expérimentale de différents coefficients, toujours constants par interface, a été menée sur ce modèle. Finalement, une valeur simple du coefficient de Robin qui est basée sur la physique du milieu poreux et la géométrie du domaine, et qui est homogène en termes de calcul dimensionnel, a permis de donner des résultats tout à fait acceptables. Nous espérons que ce coefficient particulier

$$\alpha^0 = \frac{\mathrm{MoyH}\left(\mathbf{K_i}, \mathbf{K_j}\right)}{L_\Omega^{ij}}$$

pourra être utilisé avec succès dans d'autres configurations. Il serait également intéressant dans un futur proche de mettre en œuvre d'autres techniques pour calculer des coefficients de Robin plus ou moins "optimaux".

Des tests, qui ne sont pas montrés ici, ont été réalisés avec des sous-domaines dont la perméabilité n'est pas homogène, mais au contraire saute sur plusieurs ordres de grandeur. Dans ces conditions, nous n'avons pas réussi à faire converger du tout la méthode, faute d'un choix correct du coefficient de Robin. Nous croyons donc qu'il est important de faire la décomposition du domaine en suivant la physique, c'est-à-dire les couches géologiques. Ceci permet à moindres frais de trouver un coefficient de Robin constant par interface, simple et adapté. En outre, d'autres considérations nous poussent à opérer en découpant suivant la physique : d'autres méthodes de décomposition de domaines permettent de réduire notablement l'influence des sauts de coefficients quand la décomposition est faite couche par couche : je pense notamment au préconditionneur de Neumann-Neumann, cf. par exemple [67], que nous comptons bien confronter aussi à ce modèle.

Cette décomposition en se laissant guider par la physique est naturelle, mais possède un défaut majeur, outre le fait qu'elle n'est pas toujours aisée à réaliser : celui de ne pas créer de sous-domaines homogènes en tailles, comme nous l'avons vu plus haut. Elle provoque donc un déséquilibrage de la charge qui réduit sérieusement les performances d'un calculateur parallèle. Cette question sera sans doute résolue dans la suite en redécoupant les plus gros sous-domaines. Nous envisageons par exemple de partitionner le plus gros sous-domaine qui contient le site de stockage —le sous-domaine numéro 1 extrait de la couche du Callovo-Oxfordien— et nous pourrons faire un raffinement local autour du site de stockage, comme cela a été étudié sur l'exemple simpliste 2D de la Section 2.5 et ainsi profiter pleinement de la présente méthode de décomposition de domaines *non-conforme*.

Enfin, des simulations de transport de polluant constituent évidemment le but final de ce modèle et devront être réalisées. Mais ceci est une autre histoire...

3.7 Conclusion

En guise de conclusion, je voudrais parler de ce qui reste certainement le point d'orgue de notre collaboration : la première exécution en parallèle sur notre grappe de PC. La première fois où le code a fonctionné correctement en séquentiel sur notre station de travail Intel/PC sous Linux, nous avons décidé de le tester immédiatement en parallèle sur la nouvelle grappe de nœuds Intel/PC également sous Linux : nous avons recompilé en parallèle le coupleur avec l'option `-par` sur la station de travail, nous avons copié les fichiers exécutables sur la grappe (le code OcamlP3l et les trois codes à coupler), ainsi que les fichiers de données. Nous avons lancé l'exécution. Et ça a marché ! Sans avoir besoin de déboguer quoi que ce soit en parallèle, en un quart d'heure nous sommes passés du séquentiel au parallèle.

Nous refermons ce long chapitre où nous avons essayé d'expliquer le fonctionnement d'OcamlP3l, afin de convaincre de la puissance de cette approche du parallélisme. Nous avons également montré deux résultats de simulations numériques 3D, obtenus grâce à OcamlP3l. Nous allons nous intéresser à présent à la modélisation des fractures qui peuvent survenir dans les milieux poreux.

Chapitre 4

Modélisation des grandes fractures en milieu poreux

Cette partie présente les résultats d'une étude que Jean Roberts, Jérôme Jaffré[1] et moi-même avons menée et qui concerne la modélisation des grandes fractures en milieu poreux. Nous reproduisons ici l'article qui a été tout récemment accepté pour publication [63] ; un rapport de recherche INRIA en a également été tiré [62] et un résultat partiel a été présenté lors d'une conférence [64]. Nous n'avons pas traduit le papier en français ; qu'on nous pardonne cette entorse aux Belles Lettres[2]. Elle est due principalement à un manque de temps certain et une certaine paresse, qui constituent, je n'en disconviens pas, de piètres excuses. L'éventuel ami de la Littérature qui s'égarerait dans ces pages se réconfortera peut-être à la pensée que ce chapitre est écrit dans un anglais fort correct[3], et que je n'aurais probablement pas changé une virgule en faisant son adaptation en français.

Maintenant, comme on dit sous le Dôme : "let the show begin !"

4.1 Introduction

We are concerned with flow in a fractured porous medium. Our study is carried out at a scale for which the fractures can be modeled individually. The fractures have a small width and are treated as interfaces between subdomains. We also assume that the fractures are filled with debris and that flow in the

[1] Projet Estime, Inria Rocquencourt.

[2] qui s'en remettront, j'en suis persuadé...

[3] Je puis l'affirmer en toute modestie puisque je n'en porte pas la responsabilité : merci Jean !

fractures respects Darcy's law.

We distinguish two types of fractures : fractures which have a permeability higher than that in the surrounding medium and those in which the permeability is lower than in the surrounding medium. In a medium with the former type of fracture the fluid has a tendancy to flow into the fracture and then to flow along the fracture. In this case, one should not expect the Darcy velocity to be identical on the two sides of the fracture. Consequently, as the fracture is treated as an interface, the normal component of the velocity need not be continuous across this interface. An earlier model described by Alboin, Jaffré and Roberts in [9] and [10] was based on the assumption that the permeability in the fracture was large and a jump in the normal component of the velocity across the interface was permitted.

However, when the fracture has a lower permeability, the fluid naturally tends to avoid the fracture, which represents in fact a geological barrier. Two geological layers separated by such a barrier have little communication. Thus, it is understandable that the pressure need not be the same on the two sides of the fracture. In this case the pressure is not continuous across the fracture-interface.

We present in this paper a model that generalizes the earlier model (see [9], [10]) so that it can handle both large and small permeability fractures. Indeed, the earlier model assumed the continuity of the pressure across the interface. This is no longer the case with the model presented here.

The model is derived through a process of averaging across the fracture. This process is carried out for the flow equation written in mixed form. One thereby obtains a flow equation along the interface, that is coupled with flow equations in the neighboring subdomains. The main difference between this model and the previous one is that here nonstandard Robin type conditions are imposed at the interface. The Robin coefficient is proportional to the ratio of the permeability in the fracture to the fracture width. A parameter is introduced yielding a family of models. Existence and uniqueness of the solution of the mixed weak formulation of the problem is proved for certain values of the parameter. An error estimate is obtained for a particular choice of the parameter. Some numerical experiments show the quality of the results.

The simplest form of our model was presented in [64]. Others have also treated fractures as interfaces : in (see [18]), Helmig et. al. represented the fracture by lower dimensional finite elements. A model presented by Angot in [12] is based on Robin boundary conditions at the interface and assume the continuity of the flux across the fracture. A model similar to ours for a certain value of the parameter was studied by Flauraud et al. in [51]. A survey of fractures and fracture network can be found in [6].

4.2 Description of the problem

We suppose that Ω is a convex domain in $\mathbb{R}^n$, $n = 2$ or 3, and we denote by $\Gamma = \partial\Omega$ the boundary of Ω. We suppose that the flow in Ω is governed by a conservation equation together with Darcy's law relating the gradient of the pressure p to the Darcy velocity $\mathbf{u}$:

$$\begin{array}{rcll} \operatorname{div}\mathbf{u} & = & q & \text{in } \Omega \\ \mathbf{u} & = & -\mathbf{K}\boldsymbol{\nabla}\, p & \text{in } \Omega \\ p & = & \overline{p} & \text{on } \Gamma, \end{array} \tag{4.1}$$

where p is the pressure, $\mathbf{u}$ the Darcy velocity, $\mathbf{K}$ the hydraulic conductivity (or permeability) tensor, q a source term and $\overline{p}$ the given pressure on the boundary Γ. We suppose that $\mathbf{K}$ is diagonal and that each diagonal entry K_{jj}, $j = 1, 2, ...n$ is positive and bounded above and away from 0 :

$$0 < K_{min} \leq K_{jj} \leq K_{max} \qquad j = 1, 2, ...n \qquad \text{a.e. in } \Omega.$$

We suppose (see Figure 4.1) that the fracture, Ω_f, is a sub-domain of Ω, that there is a hyperplane γ and a unit vector $\mathbf{n} = \mathbf{n_1} = -\mathbf{n_2}$ normal to γ such that

$$\begin{array}{rcl} \Omega_f & = & \{x \in \Omega : x = s + r\mathbf{n} \text{ for some } s \in \gamma \\ & & \quad \text{and some } r \text{ in the interval } (-\frac{d(s)}{2}, \frac{d(s)}{2})\} \end{array}$$

where $d(s)$ denotes the thickness of the fracture at $h \in \gamma$.

We also assume that $\overline{\Omega}_f$ separates Ω into two connected subdomains

$$\Omega \backslash \overline{\Omega}_f = \Omega_1 \cup \Omega_2, \qquad \Omega_1 \cap \Omega_2 = \emptyset.$$

We denote by Γ_i the part of the boundary of Ω_i in common with the boundary of Ω, $i = 1, 2, f$

$$\Gamma_i = \partial\Omega_i \cap \Gamma, \quad i = 1, 2, f,$$

and we denote by γ_i the part of the boundary of Ω_i in common with the boundary of the fracture Ω_f, $i = 1, 2$

$$\gamma_i = \partial\Omega_i \cap \partial\Omega_f \cap \Omega, \quad i = 1, 2.$$

Let η be the outward unit normal vector field on Γ.

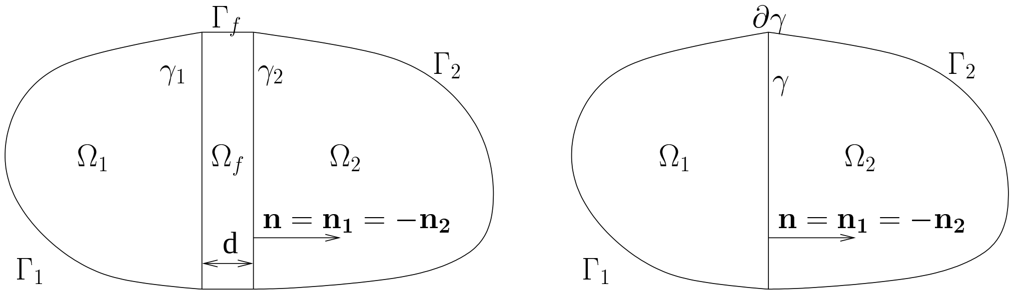


FIG. 4.1 – Left : the domain Ω with the fracture Ω_f. Right : the subdomains Ω_1 and Ω_2 separated by the fracture considered as an interface γ.

If we denote by $p_i, \mathbf{u}_i, \mathbf{K}_i$, and q_i the restrictions of $p, \mathbf{u}, \mathbf{K}$, and q respectively to $\Omega_i,\ i = 1, 2, f$, and by $\overline{p}_i$ the restriction of $\overline{p}$ to $\Gamma_i,\ i = 1, 2, f$ we can rewrite the above problem (4.1) as a transmission problem :

$$\begin{array}{rcll} \operatorname{div} \mathbf{u}_i &=& q_i & \text{in } \Omega_i, \quad i = 1, 2, f, \\ \mathbf{u}_i &=& -\mathbf{K}_i \boldsymbol{\nabla} p_i & \text{in } \Omega_i, \quad i = 1, 2, f, \\ p_i &=& \overline{p}_i & \text{on } \Gamma_i, \quad i = 1, 2, f, \\ p_i &=& p_f & \text{on } \gamma_i, \quad i = 1, 2, \\ \mathbf{u}_i \cdot \mathbf{n} &=& \mathbf{u}_f \cdot \mathbf{n} & \text{on } \gamma_i, \quad i = 1, 2. \end{array} \tag{4.2}$$

4.3 Derivation of the model

In the model presented here the fracture is treated as an interface between the domains Ω_1 and Ω_2. The model is obtained by averaging along the line segments $[\mathbf{s} - d(\mathbf{s})\mathbf{n}, \mathbf{s} + d(\mathbf{s})\mathbf{n}]$, $\mathbf{s} \in \gamma$, normal to γ. Treatment of the conservation equation is straightforward : there results a conservation equation on the surface γ with a source term representing flow into the fracture from the subdomains. Darcy's law is a vector equation and averaging the components tangential to γ yields a Darcy law in γ relating the tangential component of the gradient of the averaged pressure to the tangential component of the averaged Darcy velocity. The system of these two equations in γ is then of the same form as the system in each of the subdomains, only we have here a source term representing the exchange between γ and the subdomains. The remaining equation involving the normal components of the vectors in Darcy's law, must be exploited to obtain boundary conditions along γ for the systems in Ω_1 and Ω_2 ; however, averaging this equation yields only a formula for the difference of the trace along γ of the pressure in Ω_1 and that of the pressure in Ω_2. Several possibilities exist for closing the system.

4.3.1 Averaging across the fracture

First decompose $\mathbf{u_f}$ as $\mathbf{u_f} = \mathbf{u}_{f,\mathbf{n}} + \mathbf{u}_{f,\tau}$ with $\mathbf{u}_{f,\mathbf{n}} = \mathbf{u_f} \cdot \mathbf{n}\,\mathbf{n}$ (recall that that $\mathbf{n} = \mathbf{n_1} = -\mathbf{n_2}$). Let $\boldsymbol{\nabla}_\tau$ and div_τ denote the tangential gradient and divergence operators and $\boldsymbol{\nabla}_\mathbf{n}$ and $\mathrm{div}_\mathbf{n}$ the normal gradient divergence operators.

Averaging the conservation equation

With the above notation, the first equation of (4.2) for $i = f$ may be rewritten as

$$\mathrm{div}_\mathbf{n}\, \mathbf{u}_f + \mathrm{div}_\tau\, \mathbf{u}_f = q_f \qquad \text{in } \Omega_f. \tag{4.3}$$

Integrating in the direction normal to the fracture one obtains

$$\mathbf{u}_f \cdot \mathbf{n}|_{\gamma_2} - \mathbf{u}_f \cdot \mathbf{n}|_{\gamma_1} + \mathrm{div}_\tau\, \mathbf{U}_f \;=\; Q_f \quad \text{on } \gamma, \tag{4.4}$$

where $\mathbf{U}_f = \displaystyle\int_{-d/2}^{d/2} \mathbf{u}_{f,\tau}\, d\mathbf{n}$ and $Q_f = \displaystyle\int_{-d/2}^{d/2} q_f\, d\mathbf{n}$. Then using the continuity of the fluxes across γ_1 and γ_2, the last equation of (4.2) for $i = 1$ and 2, we may write

$$\mathrm{div}_\tau\, \mathbf{U}_f \;=\; Q_f + (\mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2}) \quad \text{on } \gamma. \tag{4.5}$$

This is the conservation equation on γ with the additional source term $\mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2}$.

Averaging Darcy's law

The second equation of (4.2) for $i = f$ may be written

$$\begin{aligned} \mathbf{u}_{f,\tau} &= -\mathbf{K}_{f,\tau} \boldsymbol{\nabla}_\tau p_f \quad \text{in } \Omega_f \\ \mathbf{u}_{f,\mathbf{n}} &= -\mathbf{K}_{f,\mathbf{n}} \boldsymbol{\nabla}_\mathbf{n} p_f \quad \text{in } \Omega_f. \end{aligned} \tag{4.6}$$

Again, integrating in the direction normal to the fracture, one obtains from the first equation of (4.6)

$$\mathbf{U}_f = -\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau P_f \quad \text{on } \gamma, \tag{4.7}$$

where $P_f = \dfrac{1}{d}\displaystyle\int_{-d/2}^{d/2} p_f\, d\mathbf{n}$ and where we have assumed that $\mathbf{K}_{f,\tau}$ is constant along the segments $[\mathbf{s} - d(\mathbf{s})\mathbf{n}, \mathbf{s} + d(\mathbf{s})\mathbf{n}]$. This is Darcy's law in the $(n -$

1)−dimensional domain γ. Together (4.5) and (4.7) give a flow equation in γ with a source term representing the flow from the subdomains Ω_1 and Ω_2 into the fracture. The remaining equation, the second equation of (4.6) must now be used to give boundary conditions along γ for the systems in Ω_1 and Ω_2 which allow for a pressure difference from one side of γ to the other.

Integrating the remaining equation, the second equation (4.6), in the direction normal to the fracture one obtains

$$\int_{-d/2}^{d/2} \mathbf{u}_{f,\mathbf{n}} \cdot \mathbf{n}\, d\mathbf{n} \;=\; -\mathbf{K}_{f,\mathbf{n}}(p_{f\,|\gamma_2} - p_{f\,|\gamma_1}). \tag{4.8}$$

The integral $\int_{-d/2}^{d/2} \mathbf{u}_{f,\mathbf{n}} \cdot \mathbf{n}\, d\mathbf{n}$ has not been computed but can be approximated using the trapezoidal rule :

$$\int_{-d/2}^{d/2} \mathbf{u}_{f,\mathbf{n}} \cdot \mathbf{n}\, d\mathbf{n} \approx \frac{d}{2}(\mathbf{u}_f \cdot \mathbf{n}|_{\gamma_2} + \mathbf{u}_f \cdot \mathbf{n}|_{\gamma_1}) = \frac{d}{2}(\mathbf{u}_2 \cdot \mathbf{n}|_{\gamma_2} + \mathbf{u}_1 \cdot \mathbf{n}|_{\gamma_1}),$$

where we have used the continuity along γ_1 and γ_2 of the fluxes, the fifth equation of (4.2). Now using the continuity along γ_1 and γ_2 of the pressures, the fourth equation of (4.2), the second equation of (4.6) is approximated by

$$\frac{1}{2}(\mathbf{u}_2 \cdot \mathbf{n}|_{\gamma_2} + \mathbf{u}_1 \cdot \mathbf{n}|_{\gamma_1}) = -\mathbf{K}_{f,\mathbf{n}} \frac{p_{2\,|\gamma_2} - p_{1\,|\gamma_1}}{d}, \tag{4.9}$$

or using the notation

$$\alpha_f = \frac{2\mathbf{K}_{f,\mathbf{n}}}{d}, \tag{4.10}$$

$$-\,\mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \alpha_f\, p_{1\,|\gamma_1} = -\,\mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \alpha_f\, p_{2\,|\gamma_2}.$$

This gives an equation for the pressure difference across the fracture.

Boundary conditions along γ, I

To close the system and obtain boundary conditions on γ for the problems in Ω_1 and Ω_2 one last equation is necessary. As (4.9) gives the difference between the pressures across γ, a natural choice is to obtain an equation for the sum of the pressures on γ by supposing that the average pressure across the fracture P_f is also the average of the pressures on the boundaries γ_1 and γ_2. Thus to calculate $p_{2\,|\gamma_2}$ and $p_{1\,|\gamma_1}$, we use the two equations

$$\begin{aligned} p_{2\,|\gamma_2} - p_{1\,|\gamma_1} &= \frac{d}{2K_f}\;(\mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} - \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1}) \\ p_{2\,|\gamma_2} + p_{1\,|\gamma_1} &= 2P_f. \end{aligned} \tag{4.11}$$

Summing and subtracting the two equations in (4.11), one gets

$$\begin{aligned} -1/2\,\mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} + \alpha_f\, p_1|_{\gamma_1} &= -1/2\,\mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} + \alpha_f\, P_f \\ -1/2\,\mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} + \alpha_f\, p_2|_{\gamma_2} &= -1/2\,\mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} + \alpha_f\, P_f. \end{aligned} \tag{4.12}$$

Boundary conditions along γ, II

If we approximate the value of p_f at the center of the fracture by P_f, and the value of $(\mathbf{u}_f\cdot\mathbf{n})$ at the center of the fracture by the averaged flux, $\dfrac{\mathbf{u}_f\cdot\mathbf{n}_f|_{\gamma_1} + \mathbf{u}_f\cdot\mathbf{n}_f|_{\gamma_2}}{2}$, and then average across each half of the fracture in the same way that we averaged across the entire width of the fracture above we obtain the following 2 equations :

$$\begin{aligned} \frac{1}{4}(3\mathbf{u}_2\cdot\mathbf{n}|_{\gamma_2} + \mathbf{u}_1\cdot\mathbf{n}|_{\gamma_1}) &= -\mathbf{K}_{f,\mathbf{n}}\frac{p_{f\,|\gamma_2} - P_f}{d/2} \\ \frac{1}{4}(\mathbf{u}_2\cdot\mathbf{n}|_{\gamma_2} + 3\mathbf{u}_1\cdot\mathbf{n}|_{\gamma_1}) &= -\mathbf{K}_{f,\mathbf{n}}\frac{P_f - p_{f\,|\gamma_1}}{d/2} \end{aligned} \tag{4.13}$$

or

$$\begin{aligned} -3/4\,\mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} + \alpha_f\, p_1|_{\gamma_1} &= -1/4\,\mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} + \alpha_f\, P_f \\ -3/4\,\mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} + \alpha_f\, p_2|_{\gamma_2} &= -1/4\,\mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} + \alpha_f\, P_f. \end{aligned} \tag{4.14}$$

These equations can be subtracted to recover (4.9), or the first equation of (4.11), but addition yields not the second equation of (4.11) but (after dividing by $d/2$)

$$-K_f\frac{p_{2\,|\gamma_2} + p_{1\,|\gamma_1} - 2P_f}{d^2/4} = \frac{\mathbf{u}_2\cdot\mathbf{n}|_{\gamma_2} - \mathbf{u}_1\cdot\mathbf{n}|_{\gamma_1}}{d}. \tag{4.15}$$

The two equations of (4.14) can also be combined to obtain

$$\begin{aligned} \mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} &= -\frac{K_f}{d}(4P_f - 3p_1|_{\gamma_1} - p_2|_{\gamma_2}) \\ \mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} &= -\frac{K_f}{d}(4P_f - p_1|_{\gamma_1} - 3p_2|_{\gamma_2}). \end{aligned} \tag{4.16}$$

Boundary conditions along γ, III

The equations (4.16) suggest another possibility :

$$\begin{aligned} \mathbf{u}_1\cdot\mathbf{n}_1|_{\gamma_1} &= -\mathbf{K}_f\,\frac{p_f|_{\gamma_1} - P_f}{d/2} \\ \mathbf{u}_2\cdot\mathbf{n}_2|_{\gamma_2} &= -\mathbf{K}_f\,\frac{p_f|_{\gamma_2} - P_f}{d/2}, \end{aligned} \tag{4.17}$$

or, using the coefficient α_f defined in (4.10),

$$\begin{array}{rcl} -\mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \alpha_f \, p_1|_{\gamma_1} & = & \alpha_f \, P_f \\ -\mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \alpha_f \, p_2|_{\gamma_2} & = & \alpha_f \, P_f. \end{array} \tag{4.18}$$

From these two equations we can again recover (4.9) or the first equation of (4.11) but the second equation of (4.11) is replaced by

$$\begin{array}{l} -K_f \dfrac{p_{2\,|\gamma_2} + p_{1\,|\gamma_1} - 2P_f}{d^2/2} = \dfrac{\mathbf{u}_2 \cdot \mathbf{n}|_{\gamma_2} - \mathbf{u}_1 \cdot \mathbf{n}|_{\gamma_1}}{d} \\ \text{or} \\ p_{2\,|\gamma_2} + p_{1\,|\gamma_1} = 2P_f - \dfrac{d}{2\,K_f}(\mathbf{u}_2 \cdot \mathbf{n}|_{\gamma_2} - \mathbf{u}_1 \cdot \mathbf{n}|_{\gamma_1}), \end{array} \tag{4.19}$$

Boundary conditions along γ, the general case

The three sets of equation (4.12), (4.14), and (4.18) can be written in the form

$$-\,\xi \; \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \alpha_f \, p_1|_{\gamma_1} \; = \; -\,(1-\xi) \; \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \alpha_f \, P_f \tag{4.20}$$
$$-\,\xi \; \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \alpha_f \, p_2|_{\gamma_2} \; = \; -\,(1-\xi) \; \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} + \alpha_f \, P_f \tag{4.21}$$

for the values $\xi = 1/2, \xi = 3/4$ and $\xi = 1$ respectively.

4.4 Model problem for the fracture

We study the following model problem which is an extension of the previous work (see [10], [9]).

4.4.1 Strong formulation of the model problem

The parameter ξ is a positive real number that will be determined later on. For common applications, this parameter should be such that : $\xi \in]1/2, 1]$. We use the notation $\boldsymbol{\nabla}_\tau$ (and respectively div_τ) for the tangential gradient (and respectively tangential divergence) operators along the fracture γ. We assume that the index i of the subdomains varies in $Z/2Z$ (so that $2+1=1$). Posing $\alpha_f = \dfrac{2\mathbf{K}_{f,\mathbf{n}}}{d}$, the problem can be written

$$
\begin{array}{rcll}
\mathbf{u}_i & = & -\mathbf{K}_i \boldsymbol{\nabla}\, p_i & \text{in } \Omega_i, \quad i=1,2 \\
\operatorname{div} \mathbf{u}_i & = & q_i & \text{in } \Omega_i, \quad i=1,2 \\
\mathbf{u}_f^\tau & = & -\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau\, p_f & \text{in } \gamma \\
\operatorname{div}_\tau \mathbf{u}_f^\tau & = & q_f + (\mathbf{u_1} \cdot \mathbf{n_1}|_\gamma + \mathbf{u_2} \cdot \mathbf{n_2}|_\gamma) & \text{in } \gamma \\
-\xi\, \mathbf{u_i} \cdot \mathbf{n_i} + \alpha_f\, p_i & = & \alpha_f\, p_f - (1-\xi)\, \mathbf{u_{i+1}} \cdot \mathbf{n_{i+1}} & \text{in } \gamma, \quad i=1,2 \\
p_i & = & \overline{p}_i & \text{on } \Gamma_i, \quad i=1,2. \\
p_f & = & \overline{p}_f & \text{on } \partial\gamma.
\end{array}
\tag{4.22}
$$

This system can be seen as a domain decomposition problem, with non-standard and nonlocal boundary conditions between the subdomains. The third equation in (4.22) represents Darcy's law in the direction tangential to the fracture. The fourth equation in (4.22) models mass conservation inside the fracture. A source term $(\mathbf{u_1} \cdot \mathbf{n_1}|_{\gamma_1} + \mathbf{u_2} \cdot \mathbf{n_2}|_{\gamma_2})$ is introduced to take into account the contribution of the subdomain flows to the fracture flow. The fifth equation in (4.22) is a Robin boundary condition for the subdomain Ω_i with a dependence with the pressure in the fracture p_f and also with neighboring subdomain Ω_{i+1}'s fluxes. This last dependence only disappears when ξ is equal to 1, and this case will be seen to be important for the below domain decomposition formulation (see Section 4.7).

This fifth equation in (4.22) is equivalent to

$$
p_i|_\gamma = p_f + \frac{\xi}{\alpha_f}\mathbf{u_i} \cdot \mathbf{n_i} - \frac{1-\xi}{\alpha_f}\, \mathbf{u_{i+1}} \cdot \mathbf{n_{i+1}} \quad \text{on } \gamma, \quad i=1,2. \tag{4.23}
$$

It is in this form that Robin boundary conditions are expressed in mixed formulation. If ξ is greater than $1/2$, we can also write

$$
\mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} = \frac{\alpha_f}{2\xi - 1}(p_2|_{\gamma_2} + p_1|_{\gamma_1} - 2p_f), \tag{4.24}
$$

$$
\mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} - \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} = \alpha_f(p_2|_{\gamma_2} - p_1|_{\gamma_1}). \tag{4.25}
$$

Remarks : The difference between this model and the model with pressure continuous across the $(n-1)$-dimensional fracture (see [9], [10]) is that in the earlier model equations (4.23) are replaced by $p_{2\,|\gamma_2} = p_{1\,|\gamma_1} = p_f$. This corresponds to the assumption $\dfrac{1}{\alpha_f} = \dfrac{d}{2\,K_f} \approx 0$ that was made previously (large permeability and small width in the fracture).

Remark : The model depends actually on two physical, fracture dependent coefficients : the product $\mathbf{K}_{f,\tau}\, d$ and the ratio $\mathbf{K}_{f,\mathbf{n}}/d$. The first coefficient is related to the jump in the normal component of the velocity (non continuity of the Darcy velocity across the fracture). The second one is related to the pressure jump (non continuity of pressure across the fracture).

The coefficient $\mathbf{K}_{f,\tau}\, d$ represents the equivalent permeability for the flow along the fracture. When this coefficient is of the same order as the permeability in the other subdomains ($\mathbf{K}_{f,\tau}\, d \approx K_i, i = 1, 2$), i.e. when the fracture permeability is sufficiently large, the flow along the fracture interacts with the flow in the rocks. In this case, the jump of the Darcy velocity across the fracture ($\mathbf{u_1} \cdot \mathbf{n_1}|_{\gamma_1} + \mathbf{u_2} \cdot \mathbf{n_2}|_{\gamma_2}$) is generally not zero, because it is the contribution of the rock flow to the fracture flow. This result is in agreement with the asymptotic analysis study by Sanchez-Palencia and al. [61], where the fracture permeability was large. One example of this type of behaviour is shown in Section 4.6.1.

The coefficient $\mathbf{K}_{f,\mathbf{n}}/d$ $(= \alpha_f/2)$ represents an equivalent "resistivity" across the fracture. Let us assume that the fracture permeability is small and that the coefficient is of the same order as the permeability in the subdomains ($\mathbf{K}_{f,\mathbf{n}}/d \approx K_i, i = 1, 2$). In this case, the fluid barely flows along the fracture and the normal velocity jump is almost zero. Equations (4.24, 4.25) become, after dividing the second equation by 2, equations (4.26, 4.27), which are similar to the transmission equations of [12]. This result is also in agreement with the asymptotic analysis study by Sanchez-Palencia [88], where the fracture permeability was small. One example of this type of behaviour is shown in Section 4.6.2.

$$\begin{aligned} \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} + \mathbf{u}_1 \cdot \mathbf{n}_1|_{\gamma_1} &= 0, && (4.26)\\ \mathbf{u}_2 \cdot \mathbf{n}_2|_{\gamma_2} &= \frac{\mathbf{K}_{f,\mathbf{n}}}{d}(p_2|_{\gamma_2} - p_1|_{\gamma_1}). && (4.27) \end{aligned}$$

4.4.2 Weak formulation of the model problem

We will need the following Hilbert spaces M and $\mathbf{W}$. It is necessary to assume more regularity than the $\mathcal{H}(\mathrm{div}, *)$ regularity (used commonly for the Mixed Finite element methods), in order to take into account properly the Robin boundary conditions (see [83] pp. 589-590 for instance).

$$\begin{aligned} M &= \{r = (r_1, r_2, r_\gamma) \in L^2(\Omega_1) \times L^2(\Omega_2) \times L^2(\gamma)\}\\ \mathbf{W} &= \{\mathbf{v} = (\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_f}) \in \mathcal{H}(\mathrm{div}, \Omega_1) \times \mathcal{H}(\mathrm{div}, \Omega_2) \times \mathcal{H}(\mathrm{div}_\tau, \gamma) :\\ &\qquad \mathbf{v_i} \cdot \mathbf{n_i} \in L^2(\gamma), \quad i = 1, 2\} \end{aligned}$$

equipped with the norms

$$
\begin{aligned}
\|r\|_M^2 &= \sum_{i=1}^{2} \|r_i\|_{0,\Omega_i}^2 + \|r_f\|_{0,\gamma}^2, \\
\|\mathbf{v}\|_W^2 &= \sum_{i=1}^{2} (\|\mathbf{v_i}\|_{0,\Omega_i}^2 + \|\mathrm{div}\ \mathbf{v_i}\|_{0,\Omega_i}^2) + \|\mathbf{v_f}\|_{0,\gamma}^2 + \|\mathrm{div}_\tau\ \mathbf{v_f}\|_{0,\gamma}^2 + \sum_{i=1}^{2} \|\mathbf{v_i}\cdot\mathbf{n_i}\|_{0,\gamma}^2.
\end{aligned}
$$

Let the bilinear forms $\alpha_\xi : \mathbf{W}\times\mathbf{W} \to \mathbb{R}$ and $\beta : \mathbf{W}\times M \to \mathbb{R}$ be defined by

$$
\begin{aligned}
\alpha_\xi(\mathbf{u},\mathbf{v}) &= \sum_{i=1}^{2} \left(K_i^{-1}\mathbf{u_i},\, \mathbf{v_i}\right)_{\Omega_i} + \left((K_{f,\tau}\ d)^{-1}\mathbf{u_f},\, \mathbf{v_f}\right)_\gamma \\
&+ \sum_{i=1}^{2} \left(\frac{1}{\alpha_f}[\xi\mathbf{u_i}\cdot\mathbf{n_i} - (1-\xi)\mathbf{u_{i+1}}\cdot\mathbf{n_{i+1}}],\, \mathbf{v_i}\cdot\mathbf{n_i}\right)_\gamma, \\
\beta(\mathbf{u},r) &= \sum_{i=1}^{2} (\mathrm{div}\ \mathbf{u_i},\, r_i)_{\Omega_i} + (\mathrm{div}_\tau\ \mathbf{u_f},\, r_f)_\gamma - \left(\sum_{i=1}^{2}\mathbf{u_i}\cdot\mathbf{n_i},\, r_f\right)_\gamma
\end{aligned}
$$

Let the linear forms $L_q : M \to \mathbb{R}$ and $L_d : \mathbf{W} \to \mathbb{R}$ be defined by

$$
\begin{aligned}
L_q(r) &= \sum_{i=1}^{2} (q_i,\, r_i)_{\Omega_i} + (\, q_f,\, r_f)_\gamma\,, \\
L_d(\mathbf{v}) &= \sum_{j=1}^{2} \left(\mathbf{v_j}\cdot\mathbf{n_j},\, \overline{p}_j\right)_{\Gamma_j} + \left(\mathbf{v_f}\cdot\mathbf{n_f},\, \overline{p}_f\right)_{\partial\gamma}.
\end{aligned}
$$

With these spaces and forms, one can easily see that the weak form of (4.22) may be written as follows :

$$
\begin{array}{llll}
\mathbf{u}\in\mathbf{W},\ p\in M & & & \\
\alpha_\xi(\mathbf{u},\mathbf{v}) - \beta(\mathbf{v},p) & = & -L_d(\mathbf{v}) & \forall\mathbf{v}\in\mathbf{W} \\
\beta(\mathbf{u},r) & = & L_q(r) & \forall r\in M.
\end{array}
\tag{4.28}
$$

4.4.3 Existence and uniqueness of the solution

We assume that the right-hand side q is sufficiently regular (in M for instance) and for simplicity that there are homogeneous Dirichlet boundary conditions imposed ($L_d = 0$). The domain and the fracture are also assumed to be smooth enough. Then we can state the following existence theorem :

Theorem 1. *Assume that there exists two positive constants* $0 < K_{\min} < K_{\max}$*, such that the permeabilities in the two subdomains and the coefficients* $(\mathbf{K}_{f,\tau} d)$ *and* $(\mathbf{K}_{f,\mathbf{n}}/d) = \alpha_f/2$ *are bounded by these constants :* $0 < K_{\min} \leq K_i \leq K_{\max}, i = 1, 2,$ *and* $0 < K_{\min} \leq (\mathbf{K}_{f,\tau} d) \leq K_{\max}$*, and* $0 < K_{\min} \leq (\mathbf{K}_{f,\mathbf{n}}/d) \leq K_{\max}$*. Assume also that the parameter* $\xi > 1/2$*. Then the model problem (4.28) has a unique solution.*

Proof : It is easy to check that $\mathbf{W}$ and M are Hilbert spaces (see [83] pp. 530). The continuity of α_ξ over $\mathbf{W}^2$ and of β over $\mathbf{W} \times M$ is straighforward.

We introduce the subspace $\tilde{\mathbf{W}} = \{\mathbf{v} \in \mathbf{W} : \beta(\mathbf{v}, r) = 0 \ \ \forall r \in M\}$. To show existence and uniqueness of the solution of (4.28), it is sufficient to show that α_ξ is $\tilde{\mathbf{W}}$-elliptic and that β satisfies the inf-sup condition (see [83, 23]), that is there exist constants C_α and C_β such that

$$\inf_{\mathbf{v} \in \tilde{\mathbf{W}}} \frac{\alpha_\xi(\mathbf{v}, \mathbf{v})}{\|\mathbf{v}\|^2_{\mathbf{W}}} \geq C_\alpha, \qquad \inf_{r \in M} \sup_{\mathbf{v} \in \mathbf{W}} \frac{\beta(\mathbf{v}, r)}{\|r\|_M \, \|\mathbf{v}\|_{\mathbf{W}}} \geq C_\beta.$$

To check that α_ξ is $\tilde{\mathbf{W}}$-elliptic, we note that for $\mathbf{u} \in \tilde{\mathbf{W}}$, $\|\text{div } \mathbf{u_i}\|_{0,\Omega_i} = 0$ and $\text{div}_\tau \ \mathbf{u_f} = \mathbf{u_1} \cdot \mathbf{n_1} + \mathbf{u_2} \cdot \mathbf{n_2}$. Thus

$$\|\mathbf{u}\|^2_{\mathbf{W}} = \sum_{i=1}^{2} \|\mathbf{u}_i\|^2_{0,\Omega_i} + \|\mathbf{u_f}\|^2_{0,\gamma} + \|\sum_{i=1}^{2} \mathbf{u_i} \cdot \mathbf{n_i}\|^2_{0,\gamma} + \sum_{i=1}^{2} \|\mathbf{u_i} \cdot \mathbf{n_i}\|^2_{0,\gamma}.$$

For this $\mathbf{u} \in \tilde{\mathbf{W}}$, α_ξ writes

$$\begin{aligned} \alpha_\xi(\mathbf{u}, \mathbf{u}) \quad &= \quad \sum_{i=1}^{2} \left(K_i^{-1} \mathbf{u_i}, \mathbf{u_i}\right)_{\Omega_i} + \left((K_{f,\tau}\ d)^{-1} \mathbf{u_f}, \mathbf{u_f}\right)_\gamma \\ &\quad + \xi \sum_{i=1}^{2} \left(\frac{\mathbf{u_i} \cdot \mathbf{n_i}}{\alpha_f^{1/2}}, \frac{\mathbf{u_i} \cdot \mathbf{n_i}}{\alpha_f^{1/2}}\right)_\gamma - 2\,(1 - \xi) \left(\frac{\mathbf{u_1} \cdot \mathbf{n_1}}{\alpha_f^{1/2}}, \frac{\mathbf{u_2} \cdot \mathbf{n_2}}{\alpha_f^{1/2}}\right)_\gamma . \end{aligned} \tag{4.29}$$

The two first terms in equation (4.29) are easily estimated because the product $(\mathbf{K}_{f,\tau} d)$ and the permeabilities are bounded. Introducing $\phi_i = \alpha_f^{-1/2} \ \mathbf{u_i} \cdot \mathbf{n_i}, \ i = 1, 2$, the two other terms are equal to the quadratic form $B(\phi_1, \phi_2) = \xi \|\phi_1\|^2_{0,\gamma} - 2(1 - \xi)\,(\phi_1, \phi_2)_\gamma + \xi \|\phi_2\|^2_{0,\gamma}$. The eigenvalues of B are 1 and $2\xi - 1$. Therefore B is strictly elliptic if and only if $\xi > 1/2$. In this case, we obtain the inequality

$$\alpha_\xi(\mathbf{u}, \mathbf{u}) \geq K^{-1}_{\max}(\sum_{i=1}^{2} \|\mathbf{u_i}\|^2_{0,\Omega_i} + \|\mathbf{u_f}\|^2_{0,\gamma}) + \min\{1, 2\xi - 1\} \sum_{i=1}^{2} \left\| \frac{\mathbf{u_i} \cdot \mathbf{n_i}}{\alpha_f^{1/2}} \right\|^2_{0,\gamma} .$$

As the ratio $(\mathbf{K}_{f,\mathbf{n}}/d) = (\alpha_f/2)$ is bounded, and because $\|\sum_{i=1}^{2} \mathbf{u_i}\cdot\mathbf{n_i}\|_{0,\gamma}^2 \le 2\sum_{i=1}^{2}\|\mathbf{u_i}\cdot\mathbf{n_i}\|_{0,\gamma}^2$, for $\xi > 1/2$, we have

$$\begin{aligned}\alpha_\xi(\mathbf{u},\mathbf{u}) &\ge K_{\max}^{-1}(\sum_{i=1}^{2}\|\mathbf{u_i}\|_{0,\Omega_i}^2 + \|\mathbf{u_f}\|_{0,\gamma}^2 + \min\{1, 2\xi-1\}\sum_{i=1}^{2}\|\mathbf{u_i}\cdot\mathbf{n_i}\|_{0,\gamma}^2)\\ &\ge 1/3\; K_{\max}^{-1}\min\{1,2\xi-1\}\|u\|_{\mathbf{W}}^2.\end{aligned}$$

To see that β satisfies the inf-sup condition, given $r \in M$, we construct using the adjoint equation a $\mathbf{v}\in\mathbf{W}$ such that $\beta(\mathbf{v},r) = \|r\|_M^2$ and $\|\mathbf{v}\|_{\mathbf{W}} \le C\|r\|_M$, where C is the constant of elliptic regularity for the adjoint problem.

For $r = (r_1, r_2, r_\tau) \in M$, let $(\varphi_1,\varphi_2,\varphi_\gamma) \in H^2(\Omega_1)\times H^2(\Omega_2)\times H^2(\gamma)$ be the solution of

$$\begin{aligned}-\Delta\varphi &= \tilde{r} \quad \text{on } \Omega\\ \varphi &= 0 \quad \text{on } \Gamma,\end{aligned}$$

where $\tilde{r}\in L^2(\Omega)$ is given by $\tilde{r}_{|\Omega_i} = r_i,\ i = 1,\,2$, and

$$\begin{aligned}-\Delta_\tau\varphi_\gamma &= r_\tau \quad \text{on } \gamma\\ \varphi_\gamma &= 0 \quad \text{on } \partial\gamma.\end{aligned}$$

Pose $\mathbf{v_i} = -\boldsymbol{\nabla}\varphi_{|\Omega_i}, i = 1,2$, and $\mathbf{v}_\gamma = -\boldsymbol{\nabla}_\gamma\,\varphi_\gamma$ and note that $\operatorname{div}\mathbf{v}_i = r_i \in L^2(\Omega_i),\ i = 1,\,2,\ \operatorname{div}_\tau\mathbf{v}_\gamma = r_\tau \in L^2(\gamma)$ and $\mathbf{v_1}\cdot\mathbf{n_1} = -\mathbf{v_2}\cdot\mathbf{n_2}\in L^2(\gamma)$, because $\mathbf{v_i}\in(\mathcal{H}^1(\Omega))^d$. Thus $\mathbf{v} = (\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_\gamma)\in\mathbf{W}$ and $\beta(\mathbf{v},r) = \|\tilde{r}\|_{0,\Omega}^2 + \|r_f\|_{0,\gamma}^2 = \|r\|_M^2$. We have

$$\begin{aligned}\|\mathbf{v}\|_W^2 &= \|\tilde{r}\|_{0,\Omega}^2 + \|r_f\|_{0,\gamma}^2 + \|\nabla\varphi\|_{0,\Omega}^2 + \|\nabla_f\,\varphi_f\|_{0,\gamma}^2 + 2\|\mathbf{v_1}\cdot\mathbf{n_1}\|_{0,\gamma}^2\\ &\le (1 + C(\Omega))\|\tilde{r}\|_{0,\Omega}^2 + (1 + C(\gamma))\|r\|_{0,\gamma}^2 + C(\Omega)\|\tilde{r}\|_{0,\Omega}^2.\end{aligned}$$

□

Remark : The case in which the parameter $\xi = 1/2$, that appeared to be natural in the derivation of the model, corresponds to a stability limit for this model. In some numerical cases (for instance in the cases presented in Sections 4.6.1 and 4.6.2), the model problem gives very good results when the parameter ξ is equal to 1/2. But on some other numerical tests, instabilities appear (see for instance Section 4.6.3).

Remark : One can prove in the same way that the discrete problem has a unique solution and that it converges towards the continuous solution of the model problem.

4.5 Interpretation of the discrete model and error estimate

We have seen in Section 4.3 that one could derive the models by simply averaging the transmission problem under its strong formulation (4.2). In this section, we seek a simple error estimate for the model problem. For this purpose, we show that the discrete model problem for $\xi = 1$ and $2/3$ is actually none other than the discrete transmission problem under a specific domain decomposition formulation, with certain hypotheses over the approximation spaces and the mesh. Throughout this section, we will assume for simplicity that homogeneous Dirichlet boundary conditions are imposed on $\partial\Omega$.

4.5.1 Assumptions concerning the mesh

Let $\mathcal{T}_h = \cup\mathcal{T}_{h,i}$ be a conforming finite element partition of $\overline{\Omega} = \cup\overline{\Omega}_i$, $i = 1, 2, f$. The meshes $\mathcal{T}_{h,i}$, $i = 1, 2$ and the mesh $\mathcal{T}_{h,f}$ match on the interfaces γ_i, $i = 1, 2$. We will assume henceforth that the following Hypothesis concerning the mesh is true.

Hypothesis 1. We assume that the mesh $\mathcal{T}_h$ of the whole domain Ω posseses the following shape : in the fracture Ω_f there exists only 1 strip of rectangular (2D) or parallelepiped (3D) cells, i.e. in 3D, each cell in the fracture is a parallelepiped with one face on γ_1 and the opposite face contained in γ_2. (See figure (4.2).)

Of course this hypothesis is quite restrictive, as it forbids in 3D a mesh made exclusively of tetrahedra. However this assumption is made here only to show the link between the discrete transmission problem and the discrete model problem, therefore this restriction is only for theoretical and not computational purposes.

4.5.2 Discrete transmission multiblock problem

Let us define the approximation spaces used in this paper.

$$\mathbf{Z_i} = H(\text{div}; \Omega_i), \quad i = 1, 2, f. \quad \mathbf{Z} = \bigoplus_{i=1,2,f} \mathbf{Z_i}.$$
$$N_i = L^2(\Omega_i), \qquad i = 1, 2, f. \quad N = \bigoplus_{i=1,2,f} N_i = L^2(\Omega).$$

Let

$$\mathbf{Z_{h,i}} \times N_{h,i} \subset \mathbf{Z_i} \times N_i, \quad i = 1, 2, f,$$

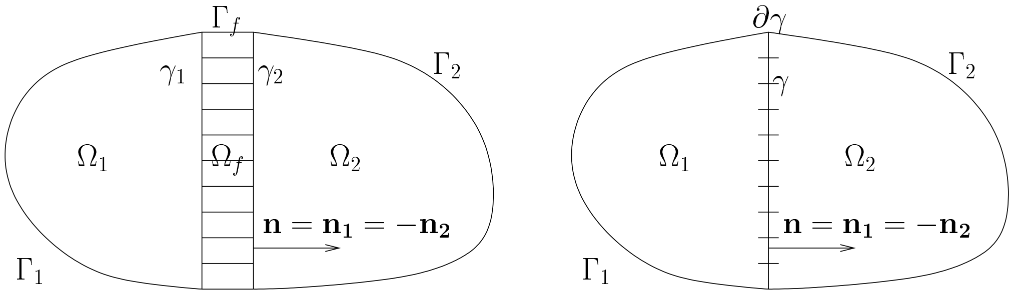


FIG. 4.2 – Left : the domain Ω with a 2D fracture Ω_f that is meshed with rectangles. Right : the 1D fracture γ is meshed with the projection of the 2D mesh on γ_1 (or γ_2).

be the usual mixed finite element approximation made of Raviart-Thomas (and Nedelec in 3D) spaces of lowest order ([82], [83], [79]). We recall that $Z_{h,i}$ consists of some linear vector functions and that $N_{h,i}$ is made up of cellwise constant scalar functions. Let

$$\mathbf{Z_h} = \bigoplus_{i=1,2,f} \mathbf{Z_{h,i}}, \qquad N_h = \bigoplus_{i=1,2,f} N_{h,i}.$$

Let

$$\Lambda_h = Q_0(\gamma) = \mathbf{Z_{h,i}} \cdot \mathbf{n_i}|_{\gamma_i} = Q_0(\gamma_i), \quad i = 1, 2, f,$$

be the interface approximation space made up of functions constant on each face. For simplicity, but with an abuse of notation, we use the same notation Λ_h for the three spaces on the interfaces γ_1, γ_2 and γ, as the spaces actually do match.

We write the discrete mixed finite element approximation for the original problem (4.1) or, equivalently, for the transmission problem (4.2) introducing interelement multipliers $(\lambda_{h,1}, \lambda_{h,2})$ (see [36], [13] and references therein). We seek $\mathbf{u_h} \in \mathbf{Z_h}, p_h \in N_h, (\lambda_{h,1}, \lambda_{h,2}) \in \Lambda_h^2$ such that

$$\begin{array}{rcll}
\left(K_i^{-1}\mathbf{u_h},\, \mathbf{v}\right)_{\Omega_i} - (\operatorname{div} \mathbf{v},\, p_h)_{\Omega_i} & = & -\left(\mathbf{v}\cdot\mathbf{n_i},\, \lambda_{h,i}\right)_{\gamma_i}, & \mathbf{v} \in \mathbf{Z_{h,i}},\ i = 1, 2, \\
\left(K_i^{-1}\mathbf{u_h},\, \mathbf{v}\right)_{\Omega_f} - (\operatorname{div} \mathbf{v},\, p_h)_{\Omega_f} & = & -\displaystyle\sum_{j=1,2} \left(\mathbf{v}\cdot\mathbf{n_f},\, \lambda_{h,j}\right)_{\gamma_j}, & \mathbf{v} \in \mathbf{Z_{h,f}}, \\
(\operatorname{div} \mathbf{u_h},\, r)_{\Omega_i} & = & (q_i,\, r)_{\Omega_i}, & r \in N_{h,i},\ i = 1, 2, f, \\
\left(\mathbf{u_{h,1}}\cdot\mathbf{n_1} + \mathbf{u_{h,f}}\cdot\mathbf{n_f},\, \mu_{h,1}\right)_{\gamma_1} & = & 0, & \mu_{h,1} \in \Lambda_h, \\
\left(\mathbf{u_{h,2}}\cdot\mathbf{n_2} + \mathbf{u_{h,f}}\cdot\mathbf{n_f},\, \mu_{h,2}\right)_{\gamma_2} & = & 0, & \mu_{h,2} \in \Lambda_h.
\end{array} \tag{4.30}$$

4.5.3 Discrete model problem

Now we write the discretization of the model problem (4.28) using the spaces of approximation defined in the previous section. Let

$$\mathbf{W_{h,\gamma}} \times \Lambda_h \subset H(\mathrm{div}_\tau;\gamma) \times L^2(\gamma),$$

be the usual Raviart-Thomas mixed finite element space of lowest order defined in the $(n-1)$-dimensional interface γ. We then define the spaces of approximation of the spaces $\mathbf{W} \times M$ set in Section 4.4.2. Let

$$\mathbf{W_h} = \bigoplus_{i=1,2} \mathbf{Z_{h,i}} \oplus \mathbf{W_{h,\gamma}}, \qquad M_h = \bigoplus_{i=1,2} N_{h,i} \oplus \Lambda_h.$$

The discrete mixed model problem becomes (with homogeneous Dirichlet boundary conditions) :

$$\begin{array}{llll} \text{Find } \mathbf{u_h^m} \in \mathbf{W_h},\ p_h^m \in M_h & & \text{such that} & \\ \alpha_\xi(\mathbf{u_h^m}, \mathbf{v}) - \beta(\mathbf{v}, p_h^m) & = & 0 & \forall \mathbf{v} \in \mathbf{W_h} \\ \beta(\mathbf{u_h^m}, r) & = & L_q(r) & \forall r \in M_h. \end{array} \tag{4.31}$$

4.5.4 Link between transmission and model problems

With these spaces and Hypothesis 1, one can simply eliminate two unknowns of the transmission problem : first the normal component of the Darcy velocity in the fracture can be computed as a function of the Darcy velocities in the neighbouring subdomains from the continuity of the fluxes accross the interfaces γ_1 and γ_2. Second one can eliminate the Lagrange multipliers that enforce the continuity of the pressure on γ_1 and γ_2. This is done with the equation expressing Darcy's law in the fracture written for a test function normal to the fracture. This can be done in two different ways yielding two different parameters ξ.

After these eliminations we obtain a new system of equations, the model problem (4.28). We illustrate this at an algebraic level in Section 4.5.5.

4.5.5 Algebraic system

The discretization of the transmission problem in a mixed weak form (4.30) yields the following symmetric system (4.32). In this section, we do not write the right hand side explicitly, because it is of no interest for our purpose here.

$$\begin{bmatrix} A_1 & B_1^\top & 0 & 0 & 0 & 0 & E_1^\top & 0 \\ B_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & A_2 & B_2^\top & 0 & 0 & 0 & E_2^\top \\ 0 & 0 & B_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & A_f & B_f^\top & D_1^\top & D_2^\top \\ 0 & 0 & 0 & 0 & B_f & 0 & 0 & 0 \\ E_1 & 0 & 0 & 0 & D_1 & 0 & 0 & 0 \\ 0 & 0 & E_2 & 0 & D_2 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ P_1 \\ U_2 \\ P_2 \\ U_f \\ P_f \\ TP_1 \\ TP_2 \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ * \\ 0 \\ * \\ 0 \\ 0 \end{bmatrix} \tag{4.32}$$

Here, $U_j, j = 1, 2, f$ (and $P_j, j = 1, 2, f$) represent the velocity (respectively pressure) unknowns in the subdomains $\Omega_j, j = 1, 2, f$. TP_1 and TP_2 are the Lagrange multipliers (trace-of-pressure unknows) living on the interfaces $\gamma_i, i = 1, 2$. The first pair of equations in (4.32) represents Darcy's law and the conservation equation in Ω_1. The matrix $E_1^\top$ ensures the Dirichlet boundary condition on the interface γ_1. The second pair of equations is the same as the previous pair, but this time in Ω_2. The third pair of equations is the same in the domain Ω_f, except here there are two matrices $D_1^\top$ and $D_2^\top$ for the source term. So far, all these equations enforce continuity of the pressure across the interfaces γ_1 and γ_2 through the Lagrange multipliers TP_1 and TP_2. The last two equations respectively enforce continuity of the velocities at the interfaces γ_1 and γ_2.

We decompose the velocities in the fracture U_f into a tangential component U_f^τ and a normal component $U_f^{\mathbf{n}} = [U_{f,1}^{\mathbf{n}} | U_{f,2}^{\mathbf{n}}]^\top$, which is itself split into a part based on the interface γ_1 and the other on γ_2. (We recall that we have assumed that there is only one strip of cells along the fracture.)

With this notation, the third pair of equations in (4.32) becomes (4.33), where the symmetric matrix A_f is also split into tangential and normal parts.

$$\begin{bmatrix} \begin{bmatrix} A_f^\tau & 0 & 0 \\ 0 & A_{f,11}^{\mathbf{n}} & A_{f,12}^{\mathbf{n}} \\ 0 & A_{f,21}^{\mathbf{n}} & A_{f,22}^{\mathbf{n}} \end{bmatrix} & \begin{bmatrix} B_f^{\tau,\top} \\ B_{f,1}^{\mathbf{n},\top} \\ B_{f,2}^{\mathbf{n},\top} \end{bmatrix} & \begin{bmatrix} 0 \\ D_{11}^\top \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ D_{22}^\top \end{bmatrix} \\ \begin{bmatrix} B_f^\tau & B_{f,1}^{\mathbf{n}} & B_{f,2}^{\mathbf{n}} \end{bmatrix} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_f^\tau \\ U_{f,1}^{\mathbf{n}} \\ U_{f,2}^{\mathbf{n}} \\ P_f \\ TP_1 \\ TP_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ * \\ 0 \\ 0 \end{bmatrix} \tag{4.33}$$

To obtain the discretization of the model problem (4.22) with $\xi = 2/3$, it suffices to eliminate the following unknows : the normal velocity in the fracture U_n and the two Lagrange multipliers TP_1 and TP_2.

It is easily seen that D_{11} and D_{22} are square and invertible. In the case of RT_0 spaces, these matrices are actually the identity. So, we first use the last pair of equations in (4.32) to compute $U^{\mathbf{n}}_{f,1}$ and $U^{\mathbf{n}}_{f,2}$ as functions of U_1 and U_2.

$$U^{\mathbf{n}}_{f,i} = D_{ii}^{-1} E_i U_i \quad i = 1, 2.$$

Plugging $U^{\mathbf{n}}_{f,i}, i = 1, 2$ into the last equation in (4.33), we get the conservation equation for the model problem.

$$B_f^{\tau} U_f^{\tau} + \sum_{i=1,2} B^{\mathbf{n}}_{f,i} D_{ii}^{-1} E_i U_i = *.$$

Next, we eliminate $TP_i, i = 1, 2$, from the second and third equations of (4.33).

$$TP_i = D_{ii}^{-\top} \sum_{j=1,2} A^{\mathbf{n}}_{f,ij} D_{jj}^{-1} E_j U_j + D_{ii}^{-\top} B^{\mathbf{n},\top}_{f,i} P_f \quad i = 1, 2.$$

We thus obtain the symmetric system (4.34). This system corresponds to the discrete model problem (4.31) when the parameter $\xi = 2/3$.

$$\begin{bmatrix} \tilde{A}_1 & B_1^{\top} & \tilde{C}^{\top} & 0 & 0 & \tilde{F}_1^{\top} \\ B_1 & 0 & 0 & 0 & 0 & 0 \\ \tilde{C} & 0 & \tilde{A}_2 & B_2^{\top} & 0 & \tilde{F}_2^{\top} \\ 0 & 0 & B_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & A_f^{\tau} & B_f^{\tau,\top} \\ \tilde{F}_1 & 0 & \tilde{F}_2 & 0 & B_f^{\tau} & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ P_1 \\ U_2 \\ P_2 \\ U_f^{\tau} \\ P_f \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ 0 \\ * \\ 0 \\ * \end{bmatrix} \tag{4.34}$$

with

$$\tilde{A}_i = A_i + E_i^{\top} A^{\mathbf{n}}_{f,ii} D_{ii}^{-1} E_i \ , \qquad \tilde{F}_i = B^{\mathbf{n}}_{f,i} D_{ii}^{-1} E_i \qquad i = 1, 2,$$

and

$$\tilde{C} = E_2^{\top} D_{22}^{-\top} A^{\mathbf{n}}_{f,21} D_{11}^{-1} E_1 \ .$$

In this system (4.34), the terms $\tilde{A}_i$ and $\tilde{F}_i^{\top}, i = 1, 2$, ensure a Robin-type boundary condition on the interface $\gamma_i, i = 1, 2$. The unusual coupling term $\tilde{C}$ disapears when using a quadrature rule that kills the extra block-diagonal terms $A^{\mathbf{n}}_{f,21} = A^{\mathbf{n},\top}_{f,12}$ in $A^{\mathbf{n}}_f$. This is what happens when the parameter ξ is equal to 1, which amounts to a finite volume modelling (see [29] for instance).

Conclusion : The two systems (4.34) and (4.32) are equivalent. The system (4.34) represents exactly the discretization of model problem (4.28) when the parameter $\xi = 2/3$ (or $\xi = 1$ with a finite volume discretization).

4.5.6 Error estimates

We have shown that the two algebraic systems (4.32) and (4.34) are equivalent, under the hypothesis concerning the spaces of approximation and the mesh. Therefore the error estimates that hold for the transmission problem also hold for the model problem, when the parameter $\xi = 2/3$.

We obtain the error estimate (4.35) below. If $\mathcal{T}_h$ is a regular family of triangulations of Ω respecting the assumption (1), there exists a constant C independant of h and d, such that if the mixed solution $(\mathbf{u}, p)$ of the original problem (4.1) is such that $(\mathbf{u}, p) \in (H^1(\Omega))^n \times H^1(\Omega)$ and $\mathrm{div}\mathbf{u} \in H^1(\Omega)$, and if $(\mathbf{u_h}, p_h)$ in $\mathbf{Z_h} \times N_h$ is the solution to the discrete problem (4.30), we have the standard error estimate

$$||\mathbf{u}-\mathbf{u_h}||_{\mathcal{H}(\mathrm{div},\Omega)}+||p-p_h||_{L^2(\Omega)} \leq C\max\{h,d\}(|p|_{1,\Omega}+|\mathbf{u}|_{1,\Omega}+|\mathrm{div}\mathbf{u}|_{1,\Omega}). \quad (4.35)$$

We denote by $(\mathbf{u_h^m}, p_h^m) = ((\mathbf{u^m_{h,1}}, \mathbf{u^m_{h,2}}, \mathbf{u^m_{h,f}}), (p^m_{h,1}, p^m_{h,2}, p^m_{h,f}))$ the solution to the discrete model problem (4.31). The solution $(\mathbf{u_h^m}, p_h^m)$ that lives in $\mathbf{W_h} \times M_h$ is extended to a function $(\tilde{\mathbf{u}}_\mathbf{h}^\mathbf{m}, \tilde{p}_h^m)$ that lives in $\mathbf{Z_h} \times N_h$. This is simply done in equation (4.36). We define the constant extension from γ to $\Omega_f : E_f : L^2(\gamma) \rightarrow L^2(\Omega_f),\ E_f(p) = p \otimes 1_{[-d/2,\ d/2]}$, where $1_{[-d/2,\ d/2]}$ is the characteristic function of the segment $[-d/2,\ d/2]$.

$$\begin{array}{ll}
\tilde{\mathbf{u}}^\mathbf{m}_{\mathbf{h},\mathbf{i}} = \mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{i}} & \text{in } \Omega_i, \quad i = 1, 2 \\
\tilde{p}^m_{h,i} = p^m_{h,i} & \text{in } \Omega_i, \quad i = 1, 2 \\
\tilde{\mathbf{u}}^\mathbf{m}_{\mathbf{h},\mathbf{f}} = \left([|\mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{i}} \cdot \mathbf{n_i}|_\gamma|] \dfrac{x}{d} + \{\mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{i}} \cdot \mathbf{n_i}|_\gamma\}; \dfrac{1}{d}\ E_f \mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{f}} \right) & \text{in } \Omega_f = \left[-\frac{d}{2}, \frac{d}{2}\right] \times \gamma, \\
\tilde{p}^m_{h,f} = E_f\ p^m_{h,f} & \text{in } \Omega_f = \left[-\frac{d}{2}, \frac{d}{2}\right] \times \gamma.
\end{array} \quad (4.36)$$

In equation (4.36), the expression $[|\mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{i}} \cdot \mathbf{n_i}|_\gamma|]$ (respectively $\{\mathbf{u}^\mathbf{m}_{\mathbf{h},\mathbf{i}} \cdot \mathbf{n_i}|_\gamma\}$) represents the jump of the normal velocity (resp. the mean velocity from Ω_1 to Ω_2) on the interface $\gamma = \partial\Omega_1 \cap \partial\Omega_2$. The extension of the velocity from γ to Ω_f is thus composed of a linear approximation in the normal direction (first component along the Ox axis) and a constant extension of the tangential component. It is not difficult to see that such an extension lives in $\mathbf{Z_h}$.

Finally we have seen in Section 4.5.5 that the algebraic solution of the discrete model problem (4.31) and of the original discrete problem (4.30) were the same, in the sense that $(\mathbf{u_h}, p_h) = (\tilde{\mathbf{u}}_\mathbf{h}^\mathbf{m}, \tilde{p}_h^m)$. So this yields the error estimate (4.37) for the discrete model problem.

$$||\mathbf{u} - \tilde{\mathbf{u}}_\mathbf{h}^\mathbf{m}||_{\mathcal{H}(\mathrm{div},\Omega)} + ||p - \tilde{p}_h^m||_{L^2(\Omega)} \leq C\max\{h, d\}(|p|_{1,\Omega} + |\mathbf{u}|_{1,\Omega} + |\mathrm{div}\mathbf{u}|_{1,\Omega}) \quad (4.37)$$

Remark : This error estimate (4.37) has a limitation : in some cases, the solution of a fractured problem has little regularity. In these cases, the constant in the estimate may become very large and delay the convergence. This is probably what happens in Section 4.6.3 where the permeability jumps are large.

Remark : This error estimate (4.37) holds as well when the parameter $\xi = 1$ because one gets the same error estimates from a finite volume discretization.

Remark : In the thin subdomain Ω_f, the cells are in general very long and narrow. This can produce large errors, but in [65], it was pointed out that the mixed finite element method behaves well even when the cells are thus stretched. Hence the estimates (4.35) and therefore (4.37) remain significant despite the thin subdomain Ω_f.

4.6 Numerical results

Some numerical results are given in this section in order to illustrate the properties of the model presented in this paper. In the first test case (see Section 4.6.1), we show that the model presented in this paper gives good results under the hypotheses that were made to derive the former model (see [9]) : velocity jumps across the fracture that can occur when the tangential permeability is large, are properly modeled. In the second test case (see Section 4.6.2), one can see that this improved model is able to properly handle a geological barrier (with a small normal permeability in the fracture). It is shown that this was not possible with the former model. The third test case (see Section 4.6.3) is more difficult : it combines the two previous classes of problems. In the third test case, the fracture has anisotropic permeabilities, thus creating a zone where neither the pressure nor the velocities is continuous across the fracture. These two very stiff test cases show an influence of the parameter ξ, and also show that the model can tackle –when the fracture width is small enough– these types of problems quite reasonably.

The discrete relative L^2 errors are computed in the following way. A direct 2D computation is performed with a mixed hybrid method on a fine mesh $\mathcal{T}_\eta$ (η being the mesh size, sufficiently small ($\leq 1/200$)). This gives a “reference” pressure $P_\eta^\star$ that we will assume to be a “good” approximation to the solution. We present this pressure for instance in Figure 4.4, where the grid is considerably coarsened for picture purposes. Note that the grid is locally refined around the fracture. We call P_h^m the solution computed with the model presented in this paper. We use a square mesh with a mesh size h ($> \eta$) in each subdomain, and a 1D mesh in the fracture (with the same mesh size h). We call $\Pi_\eta P_h^m$ its projection onto the fine mesh $\mathcal{T}_\eta$. The square of the error is then equal to

$$||P_h^m - P_\eta^\star||^2_{L_h^2(\Omega)} = \frac{\sum\limits_{C_\eta \in \mathcal{T}_\eta} (\Pi_\eta P_h^m - P_\eta^\star)^2 \text{ meas}(C_\eta)}{\sum\limits_{C_\eta \in \mathcal{T}_\eta} (P_\eta^\star)^2 \text{ meas}(C_\eta)},$$

where $\mathcal{T}_\eta$ is the fine mesh and $\text{meas}(C_\eta)$ the measure of the cell C_η.

4.6.1 First test-case : large permeability in the fracture and Dirichlet boundary conditions

The test-case is described in the Figure 4.3 (left Figure). The lengths of the domain along the Ox and Oy axes are respectively $L_x = 2$, $L_y = 1$. The permeability tensor in the fracture depends on a parameter K_f. The permeability in the other subdomains is constant and isotropic : $K = 1$. The fracture width is denoted by d. Dirichlet conditions hold on the fracture boundaries. Finally, in this test case, the permeability tensors in the fracture are given by

$$K_{f1} = K_{f2} = K_f \; Id,$$

where Id is the 2D identity matrix and K_f is a parameter greater than 1. This means that the fluid tends to flow rapidly along the fracture.

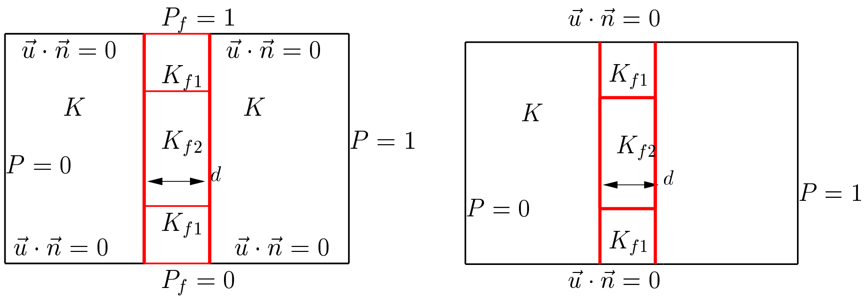


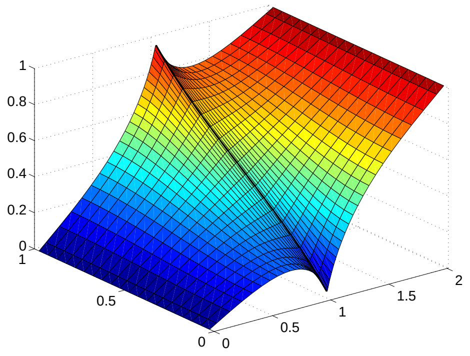
FIG. 4.3 – **Left :** Dirichlet test case with 2 permeability tensors in the fracture. **Right :** Neumann test case.

One example of a solution given by the model is shown in Figure 4.5. As a comparison one can see a reference computation performed with a locally refined mesh in Figure 4.4.

Most of the figures plotting the L^2 error that follow can be interpreted in three ways : first, for a constant mesh size h, we can see the dependence of the model when d tends to 0 (“horizontal” curves). Second, for a constant fracture width d (“vertically”), one can see the convergence of the discrete model solution towards the reference solution. Third, one can see the influence of the parameter ξ. The convergence rate was plotted for four values : $\xi = 0.51, 2/3, 1, 10$. In the Section 4.6.1 and 4.6.2, there is little influence of the parameter ξ.

For instance, in Figure 4.6, (left Figure), the L^2 errors are plotted as a function of the fracture width d, with $K_f = 1/d$, for different values of ξ and for different mesh sizes. Here the curves can be interpreted in these three ways : first, for a constant h, we can hardly see any convergence when d tends to 0. Actually, this is normal as the model very little depends on d, but depends on the product $K_f d$ that is constant in this test case. Second, for a constant d (small enough : $< 1E-2$), when h is divided by a factor 2, the L^2 error is divided by a constant factor that is close to 2. The discrete model solution converges as $\mathcal{O}(\max\{h, d\}$ toward the reference solution : this confirms the previously stated error estimate. The parameter $\xi \leq 1$ has little influence on the solution given by the model in this case. Still, one should not take ξ greater than 1 as in general it degrades the solution.

The behaviour of the solution when the fracture width varies for a fixed fracture permeability is shown in Figure 4.6, (right Figure). The L^2 error picture shows again a convergence in $\mathcal{O}(\max\{h, d\})$.



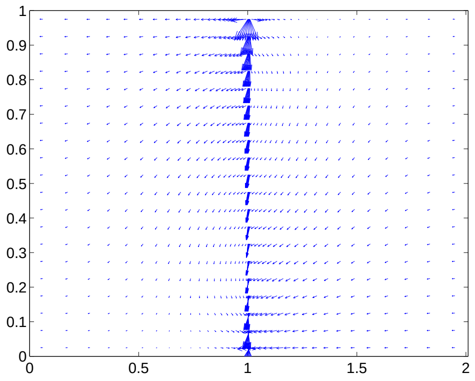


FIG. 4.4 – Test case 1. Reference pressure (left) and Darcy velocity (right) given by a locally refined mesh computation. $K_f = 100, d = 0.01$. (The grid is very coarse for picture purposes.)

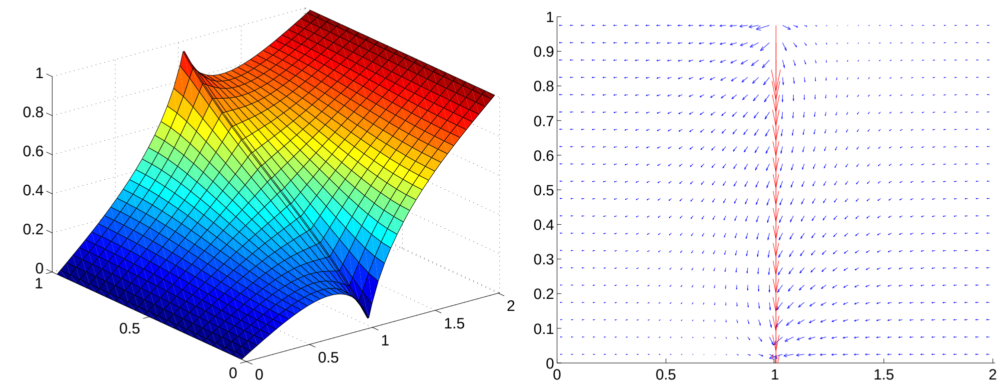


FIG. 4.5 – Test case 1. Pressure (left) and Darcy velocity (right) given by the model. $K_f = 100, d = 0.01, \xi = 2/3$.

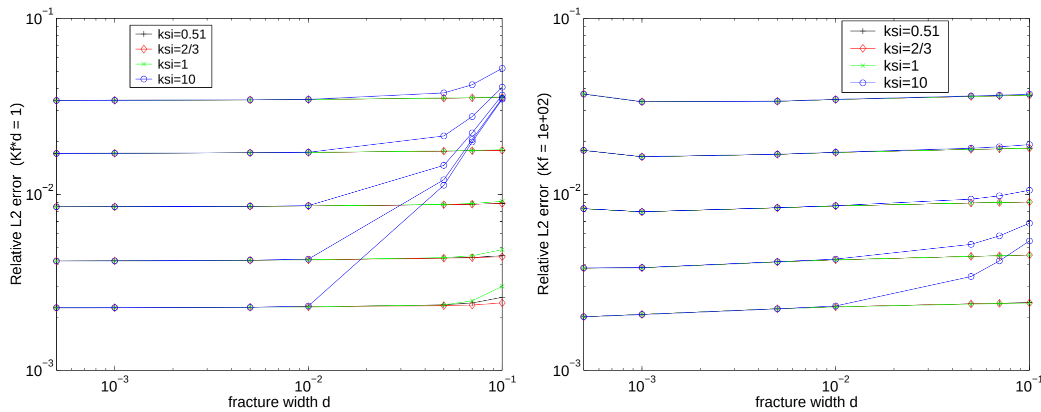


FIG. 4.6 – Test case 1. Discrete relative L^2 error variation as a function of the fracture width d, for different values of ξ. $\xi = 0.51$ in black, $\xi = 2/3$ in red, $\xi = 1$ in green, $\xi = 10$ in blue. The five sets of curves show the dependency over the mesh size. From the upper curves to the lower ones : $h = 1/10$, $h = 1/20$, $h = 1/40$, $h = 1/80$, and $h = 1/160$. **Left :** the fracture permeability varies as $1/d$. The permeability increases as the fracture width decreases : $K_f \times d = 1$. **Right :** the fracture permeability is constant : $K_f = 100$.

4.6.2 Second test-case : small permeability in the fracture and Neumann boundary conditions

The test-case is described in the Figure 4.3, (right Figure). There is a low permeability fracture in the middle of the domain. Homogeneous Neumann conditions are imposed at the upper part and lower part of the fracture. The permeability tensor in the fracture is given by :

$$K_{f1} = Id, \text{ and } K_{f2} = K_f \; Id,$$

where Id is the 2D identity matrix and K_f is a parameter smaller than 1. Obviously, the fluid tends to avoid the fracture that represents a geological barrier.

One example of a solution given by the model is shown in Figure 4.9. For comparison, a result given by the previous model (with a pressure assumed to be continuous across the fracture, see [9]) is shown in Figure 4.8. The result given by the previous model is not satisfactory. As a reference, one can see the result of a computation performed with a mesh that is refined around the fracture (see Figure 4.7).

The behaviour of the solution when the fracture width and the permeability vary in the same manner is shown in Figure 4.10, (left Figure). There is little dependence of K_f/d (which is constant in the model). The behaviour of the solution when the fracture width varies for a fixed fracture permeability is shown in Figure 4.10, (right Figure).

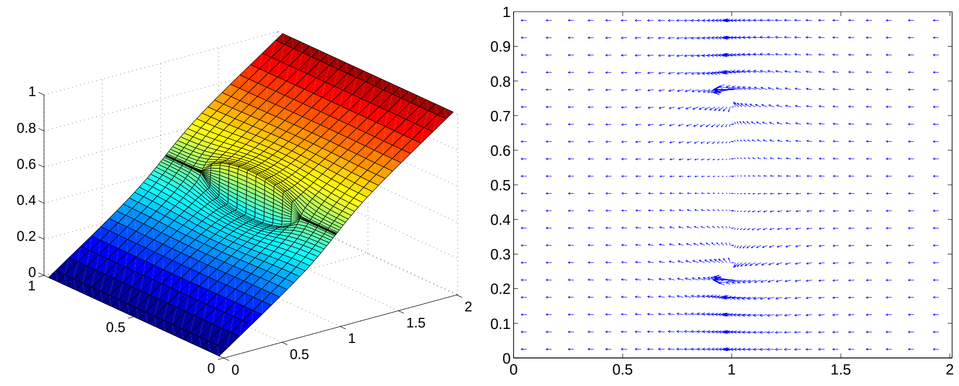


Fig. 4.7 – Test case 2. Reference pressure (left) and Darcy velocity (right) given by a locally refined mesh computation. $K_f = 2e-3, d = 0.01$.

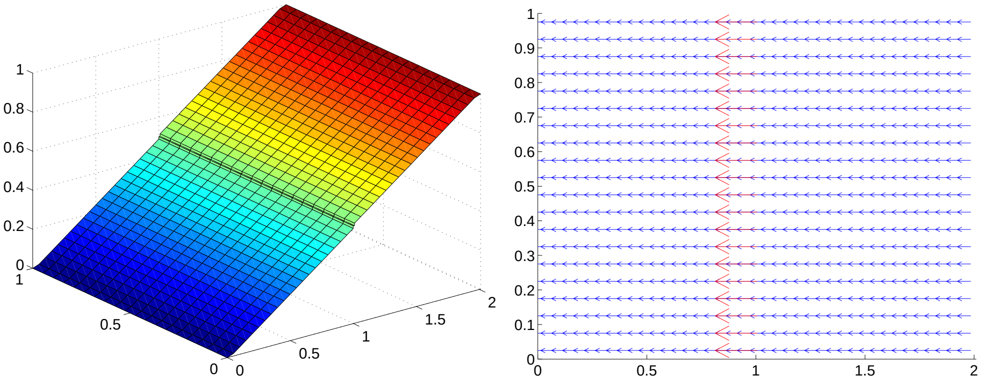


FIG. 4.8 – Test case 2. Pressure (left) and Darcy velocity (right) given by the **OLD model**. $K_f = 2e-3, d = 0.01$. This result is obviously not satisfactory.

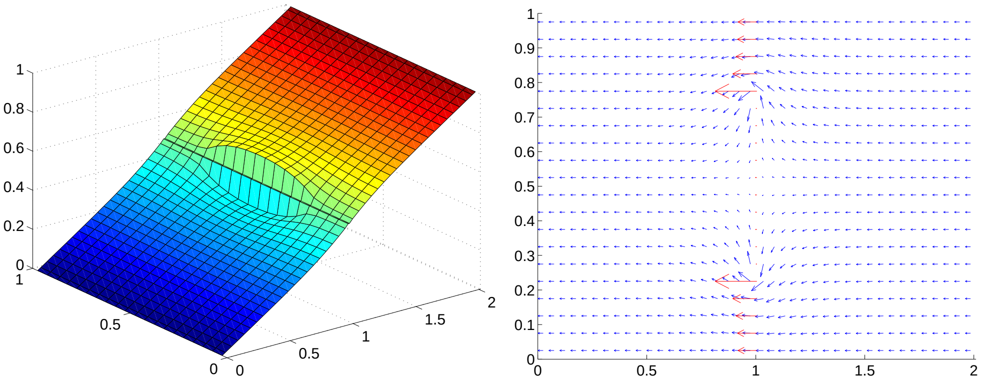


FIG. 4.9 – Test case 2. Pressure (left) and Darcy velocity (right) given by the model. $K_f = 2e-3, d = 0.01, \xi = 2/3$.

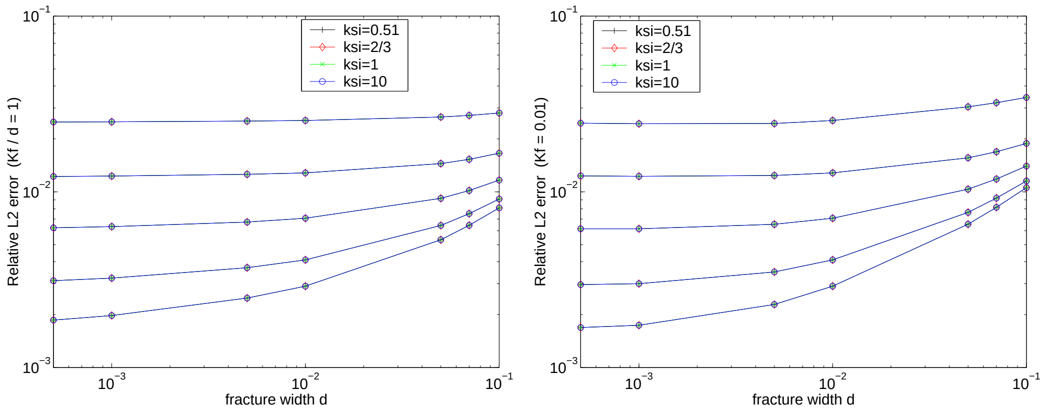


FIG. 4.10 – Test case 2. Discrete relative L^2 error variation as a function of the fracture width d, for different values of ξ. $\xi = 0.51$ in black, $\xi = 2/3$ in red, $\xi = 1$ in green, $\xi = 10$ in blue. The five sets of curves show the dependence on the mesh size. From the upper curves to the lower ones : $h = 1/10$, $h = 1/20$, $h = 1/40$, $h = 1/80$, and $h = 1/160$. **Left :** the fracture permeability varies as d. The permeability decreases like the fracture width : $K_f/d = 1$. **Right :** the fracture permeability is constant : $K_f = 0.01$.

4.6.3 Third test-case : 2 anisotropic permeabilities in the fracture and Dirichlet boundary conditions

This is the same test-case as in Section 4.6.1 (see the Figure 4.3, (left Figure)), with a modification in the permeability tensor in the fracture. Dirichlet conditions hold on the fracture boundaries and there are 2 different anisotropic permeability tensors :

$$K_{f1} = \begin{bmatrix} 1/K_f & 0 \\ 0 & K_f \end{bmatrix} \text{ and } K_{f2} = \begin{bmatrix} K_f & 0 \\ 0 & 1/K_f \end{bmatrix},$$

where K_f is a parameter greater than 1. This means that in the middle part of the fracture where the permeability is equal to K_{f2}, the fluid cannot flow along the fracture, but can easily cross it. It is the contrary in the upper and lower parts of the fracture.

Some examples of solutions given by the model for different parameters ξ are shown in Figures 4.12, and 4.13. Some reference results can be seen in Figure (4.11). One can notice that, on this test case, the model does not approach correctly the reference solution everywhere. This is especially true in the regions close to the fracture, where the permeability tensor is K_{f1} (extremities of the fracture). It might be due to the high singularities at the exits of the fractures.

All the results given by the model strongly depend on the parameter ξ, and none of them are completely satisfactory. The lowest errors are generally provided for $\xi = 0.51$, but the model with this parameter does not respect the maximum principle : pressure greater than 1 are computed. And for ξ greater than 1, the L^2 errors are very large. The best compromise seems to be $\xi = 2/3$ in this example : the maximum principle is respected and the error remains reasonable.

The behaviour of the solution is shown in Figure 4.14, when d tends to 0 with a constant product $K_f d$. The solution given by the model is almost independent of $K_f d$. In Figure 4.15, the error is plotted as a function of d, for two constant values of K_f. In all these curves, one must note that the error is relatively large, in comparison to the previous test-cases.

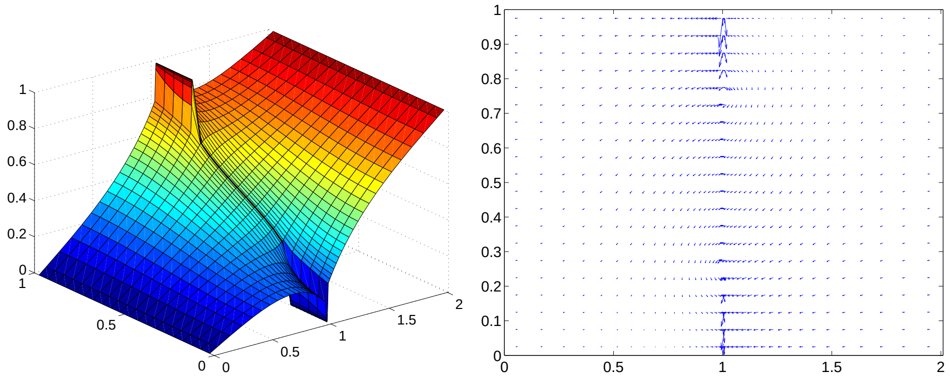


FIG. 4.11 – Test case 3. Reference pressure (left) and Darcy velocity (right) given by a locally refined mesh computation. $K_f = 200, d = 0.01$.

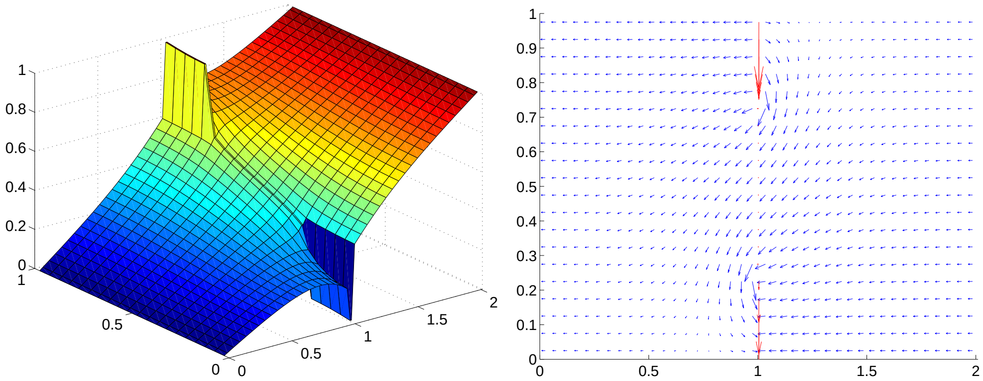


FIG. 4.12 – Test case 3. Pressure (left) and Darcy velocity (right) given by the model. $K_f = 200, d = 0.01, \xi = 2/3$.

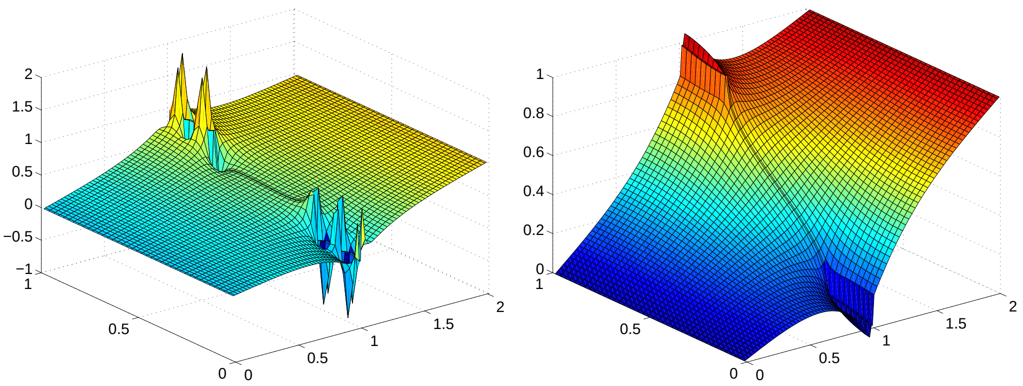


FIG. 4.13 – Test case 3. Pressure given by the model. Left : $\xi = 0.49$, the parameter ξ is smaller than the stability limit ; the model is not stable when $\xi < 1/2$; scales for the pressure : $[-1, 2]$. Right : $\xi = 0.51$; small overshoots and undershoots appear at the boundary of the fracture. $K_f = 200, d = 0.01$.

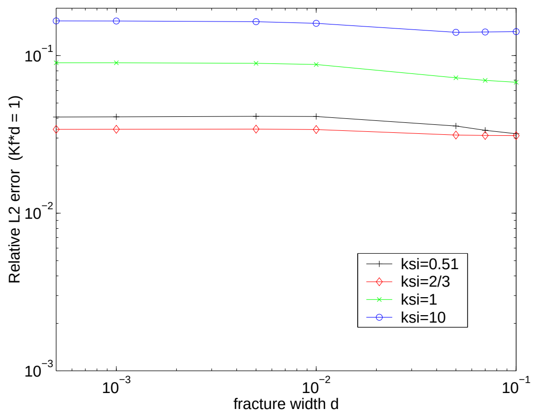


FIG. 4.14 – Test case 3. Discrete relative L^2 error variation as a function of the fracture width, the fracture permeability parameter varying as $1/d$, for different values of ξ. $\xi = 0.51$ in black, $\xi = 2/3$ in red, $\xi = 1$ in green, $\xi = 10$ in blue. The fracture permeability varies as $1/d$; the permeability parameter increases as the fracture width decreases : $K_f \times d = 1$. $nx = 160,\ h = 6.25E-3$. Mind the error scales : from 0.1% up to 20% : errors remain at a quite high level.

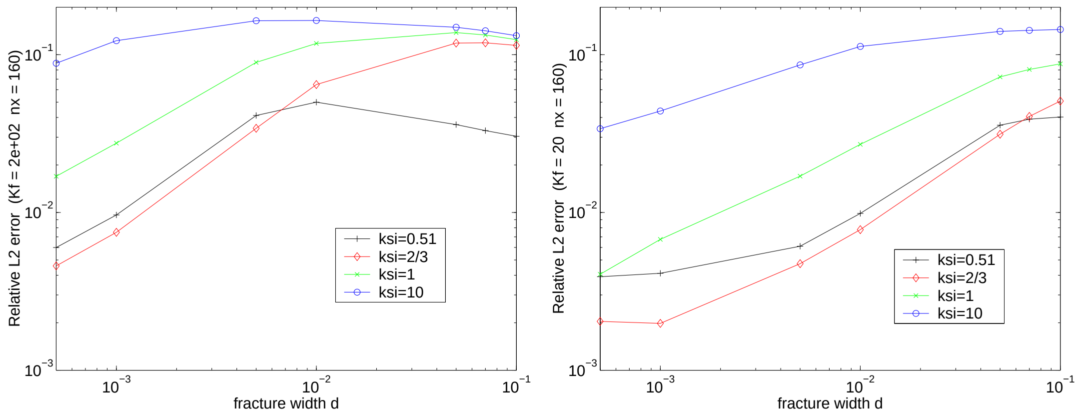


FIG. 4.15 – Test case 3. Discrete relative L^2 error variation as a function of the fracture width, for different values of ξ. $\xi = 0.51$ in black, $\xi = 2/3$ in red, $\xi = 1$ in green, $\xi = 10$ in blue. The fracture permeability parameter is constant. **Left :** $K_f = 200$. **Right :** $K_f = 20$. For both figures : $nx = 160$, $h = 6.25E-3$. Mind the error scales : from 0.1% up to 20% : errors remain at a quite high level.

4.7 Domain decomposition

In this section we present a way of solving efficiently the problem arising from the model (4.22).

4.7.1 Domain decomposition formulation

The direct mixed discretization of problem (4.22) given in equation (4.34) yields the algebraic system (4.31). As this system (4.31) is not positive definite and is very large, it is expensive to solve. The goal is then to eliminate some of the unknowns to obtain a problem easier to solve.

When the parameter $\xi = 1$, the model problem reduces to the nonlocal nonstandard positive definite interface problem (4.43). For other values of the parameter ξ, it is not clear yet what to do (see (4.40)).

Throughout this section, for simplicity, we will assume that homogeneous Dirichlet boundary conditions are imposed on $\partial\Omega$. We introduce the discrete Dirichlet to Neumann operators S_1 and S_2 in (4.38) for the subdomains Ω_1 and Ω_2 (analogous to the discrete Steklov Poincaré operator, except here they include the source term).

$$\begin{array}{rcll} S_i : \Lambda_h \times N_{h,i} & \to & \Lambda_h, & i = 1, 2, \\ S_i(\lambda_i, q_i) & = & -\mathbf{u_i} \cdot \mathbf{n}_i, & \end{array} \tag{4.38}$$

so that

$$\begin{array}{rcll} \operatorname{div} \mathbf{u_i} & = & q_i & \text{in } \Omega_i, \\ \mathbf{u_i} & = & -\mathbf{K}_i \boldsymbol{\nabla}\, p_i & \text{in } \Omega_i, \\ p_i & = & 0 & \text{on } \Gamma_i, \\ p_i & = & \lambda_i & \text{on } \gamma. \end{array} \tag{4.39}$$

In terms of these operators, the problem to be solved, (4.22), becomes

$$\begin{array}{rcl} S_1(\lambda_1, q_1) + S_2(\lambda_2, q_2) - \operatorname{div}_\tau(\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau p_f) & = & q_f, \\ \xi\, S_1(\lambda_1, q_1) + \alpha_f\, \lambda_1 - (1-\xi)\, S_2(\lambda_2, q_2) - \alpha_f\, p_f & = & 0, \\ -(1-\xi)\, S_1(\lambda_1, q_1) + \xi\, S_2(\lambda_2, q_2) + \alpha_f\, \lambda_2 - \alpha_f\, p_f & = & 0, \end{array} \tag{4.40}$$

and it is not obvious how to solve this problem.

If $\xi = 1$, we can express separately λ_1 and λ_2 as functions of p_f. In this case, we replace the Dirichlet to Neumann operators by the discrete Robin to Neumann operators $\overline{S}_i$ defined by (4.41).

$$\begin{array}{rcll} \overline{S}_i : \Lambda_h \times N_{h,i} & \rightarrow & \Lambda_h, & i = 1, 2, \\ \overline{S}_i(\lambda_i, q_i) & = & -\mathbf{u_i} \cdot \mathbf{n}_i, & \end{array} \tag{4.41}$$

so that

$$\begin{array}{rcll} \operatorname{div} \mathbf{u_i} & = & q_i & \text{in } \Omega_i, \\ \mathbf{u_i} & = & -\mathbf{K}_i \boldsymbol{\nabla}\, p_i & \text{in } \Omega_i, \\ p_i & = & 0 & \text{on } \Gamma_i, \\ -\mathbf{u_i} \cdot \mathbf{n}_i + \alpha_f\, p_i & = & \alpha_f \lambda_i & \text{on } \gamma. \end{array} \tag{4.42}$$

Then with $\xi = 1$, problem (4.22) becomes the simpler interface problem (4.43) that depends on one scalar unknow, p_f, that lives on the interface.

$$\overline{S}_1(p_f, q_1) + \overline{S}_2(p_f, q_2) - \operatorname{div}_\tau(\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau p_f) \;=\; q_f. \tag{4.43}$$

We recall here that the fracture data are present in two different ways in this equation (4.43). First the product $\mathbf{K}_{f,\tau}\, d$ plays the role of a mean permeability for the Darcy law equation along the fracture. Also the operators $\overline{S}_i$, $i = 1, 2$, are Robin to Neumann operators with a specific Robin coefficient that depends on the quotient $\mathbf{K}_{f,\mathbf{n}}/d$.

4.7.2 Weak formulation

It is well known that the Dirichlet to Neumann operators $S_i(\cdot, 0), i = 1, 2$, defined in (4.38) are symmetric, positive semi-definite (see [36], [13]). We show that this property also holds for the operators defined in (4.41).

Lemma 1. *If the Robin coefficient is positive ($\alpha_f > 0$), the discrete Robin to Neumann operators $\overline{S}_i(\cdot, 0)$ are symmetric, positive semi-definite on Λ_h for $i = 1, 2$.*

Proof : Define the bilinear operators :

$$\begin{array}{rcll} \overline{s}_i : \Lambda_h \times \Lambda_h & \rightarrow & \Lambda_h, & i = 1, 2, \\ \overline{s}_i(\lambda, \mu) =< \overline{S}_i(\lambda, 0), \mu > & = & (\mu, \mathbf{u_{h,\,i}}(\lambda) \cdot \mathbf{n_i})_\gamma, & i = 1, 2, \end{array} \tag{4.44}$$

with $\mathbf{u_{h,\,i}}(\lambda)$ the solution of the following problem, (4.45), in $\Omega_i,\ i = 1, 2$.

Given λ in Λ_h, seek $\mathbf{u_{h,\,i}}(\lambda) \in \mathbf{Z_{h,\,i}}$ and $p_{h,i} \in N_{h,i}$, such that for $\mathbf{v} \in \mathbf{Z_{h,i}}$ and $r \in N_{h,i}$

$$\begin{array}{rcl} \left(K_i^{-1}\mathbf{u_{h,\,i}}(\lambda),\, \mathbf{v}\right)_{\Omega_i} & = & (\mathrm{div}\, \mathbf{v},\, p_{h,i})_{\Omega_i} - \left(\mathbf{v} \cdot \mathbf{n_i},\, \lambda + \dfrac{1}{\alpha_f}\, \mathbf{u_{h,\,i}}(\lambda) \cdot \mathbf{n_i}\right)_\gamma, \\ (\mathrm{div}\, \mathbf{u_{h,\,i}}(\lambda),\, r)_{\Omega_i} & = & 0. \end{array} \tag{4.45}$$

Take $\mathbf{v} = \mathbf{u_{h,\,i}}(\mu)$ in (4.45) to see that the operators $\overline{s}_i$ can be expressed as

$$\overline{s}_i(\lambda, \mu) = \left(K_i^{-1}\mathbf{u_{h,\,i}}(\lambda),\, \mathbf{u_{h,\,i}}(\mu)\right)_{\Omega_i} + \left(\frac{1}{\alpha_f^{1/2}}\mathbf{u_{h,\,i}}(\lambda) \cdot \mathbf{n_i},\, \frac{1}{\alpha_f^{1/2}}\, \mathbf{u_{h,\,i}}(\mu) \cdot \mathbf{n_i}\right)_\gamma. \tag{4.46}$$

It is now easy to see that the operators $\overline{s}_i, i = 1, 2$ are symmetric and positive semi-definite. □

4.7.3 Solving the system efficiently

In the case $\xi = 1$, a good way to solve the model problem (4.22) is to solve iteratively the linear interface equation (4.43). To do so, a good idea would be to take a standard finite volume discretization of the Laplace operator ($-\mathrm{div}_\tau(\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau \cdot)$) that yields a symmetric, positive semi-definite operator. One has to invert the operator $\overline{S}_1(\cdot, 0) + \overline{S}_2(\cdot, 0) - \mathrm{div}_\tau(\mathbf{K}_{f,\tau}\, d\, \boldsymbol{\nabla}_\tau \cdot)$ which is symmetric, positive semi-definite as a sum of such operators (actually it is positive definite because of the Dirichlet boundary conditions). Thus a simple conjugate gradient method can be applied.

This iterative method has been tested numerically and gives the expected results. Some further studies are under way to try to find a preconditioner for this method.

4.8 Conclusion

The models we presented in this paper allow the treatment at the same time and in the same model of the case in which there is a small permeability in the fracture, involving a pressure discontinuity, and the case in which there is a large permeability in the fracture, involving a velocity discontinuity. In each case, the models agree with the asymptotic analysis given by Sanchez-Palencia in [61], [88]. They also give make it possible to treat high anisotropies in the fractures (see Section 4.6.3), though the numerical results are not completely satisfactory and some further studies would probably be useful to understand clearly where the solution lives (as neither the scalar unknown is continuous at the interface, nor its normal derivative).

Some further work is under way to test a more realistic 3-dimensional problem, involving two or three intersecting fractures. These tests should involve also the transport equations. Also a preconditionner for the iterative method that was described in Section 4.7 will be studied and tested.

Acknowledgement

We would like to thank Magne Espedal[4] for useful discussions and remarks.

This work was partially supported by Andra, the French National radioactive waste management agency[5].

[4]Dpt of Math., LIMT, University of Bergen, Norway, Magne.Espedal@mi.uib.no

[5]http ://www.andra.fr/

Chapitre 5

Conclusions et Perspectives

Ou comment tout travail n'est jamais qu'un début...

Cette thèse traite des simulations multidomaines des écoulements en milieux poreux. Trois aspects différents ont été abordés. Tout d'abord, on a mené une étude concernant une méthode de décomposition de domaines avec des maillages non-raccordés utilisant des conditions d'interface de type Robin, pour des éléments finis mixtes. Une analyse expérimentale simple basée sur une simulation en trois dimensions réaliste montre l'importance du choix des coefficients de Robin dans cette méthode. D'autre part, la mise en œuvre en parallèle de la méthode de décomposition de domaines constitue un travail original : en partenariat avec des chercheurs en informatique, nous avons développé un coupleur coordonnant les sous-domaines, grâce au système parallèle OcamlP3l écrit dans le langage Ocaml. Le principal atout d'OcamlP3l réside dans le fait que la programmation et le débogage se font en *séquentiel*, l'exécutable parallèle étant automatiquement obtenu par une simple recompilation. Enfin, nous avons présenté un nouveau modèle d'écoulement dans un milieu poreux contenant des fractures d'une importance telle qu'elles doivent être modélisées explicitement. Les fractures peuvent avoir des perméabilités très grandes et/ou très faibles. Dans ce modèle, nous considérons les failles comme des *interfaces* entre sous-domaines. Une analyse théorique prouvant l'existence et l'unicité de la solution et fournissant une estimation d'erreur, est confirmée par des tests numériques.

Plusieurs développements sont envisageables dans la continuité de la thèse.

- Essayer d'utiliser l'accélération d'Aitken pour la méthode de Robin non-conforme pourrait être un sujet d'étude intéressant.
- Le choix des coefficients de Robin n'est pas encore toujours clair ; il serait judicieux de tester et d'utiliser des méthodes de détermination de ces coefficients proposées dans [78].
- Les modèles de fractures méritent de plus amples développements. Des

simulations de fractures réalistes devraient être menées : des configurations avec des fractures s'intersectant, dans un domaine à trois dimensions et des coefficients physiques sont à l'étude. Par ailleurs, un préconditionneur performant pour le problème de décomposition de domaine surgissant dans le modèle de fractures doit être testé et analysé. Des recherches concernant la modélisation du transport et des écoulements multiphasiques doivent aussi être considérées.

- Le transport constitue aussi un axe de recherche en soi. Dans le cadre de la décomposition de domaines, l'adaptation des pas d'espace et de temps aux sous-domaines paraît naturelle. C'est pourquoi la décomposition de domaines avec des pas de temps locaux sont actuellement étudiés.
- Au niveau de l'implémentation parallèle, une comparaison entre OcamlP3l et MPI devrait être mise en place tant. Il faudrait alors mettre en perspective les temps de développement dans les deux systèmes, ainsi que l'efficacité des programmes obtenus. Le système OcamlP3l devrait aussi permettre de faire communiquer les processus entre eux, sans passer par obligatoirement par un nœud central. Ceci permettrait de programmer les méthodes de décomposition de domaines de façon scalable.
- La poursuite du projet de couplage de codes est en cours. Le but est maintenant de coupler des codes différents. Par ailleurs, la programmation d'une autre méthode de décomposition de domaines est en phase d'achèvement ; la mise en place d'un cadre permettant l'automatisation des couplages est donc en train de se dessiner.
- Concernant les éléments finis mixtes, il est nécessaire d'utiliser des fonctions de formes améliorant la précision de la méthode, quand les éléments hexaèdriques sont déformés.

Annexe A

Coordonnées barycentriques

Ce chapitre est consacré à quelques rappels de géométrie élémentaire concernant les coordonnées barycentriques. Nous avons jugé utile de rédiger ces rappels dans la mesure où ils permettent d'introduire des notations qui servent ensuite dans la Section A.2.3, où nous décrivons la méthodologie utilisée pour tester l'appartenance d'un point à un triangle. Je me suis servi du cours [58] ainsi que de notes fournies par Martial Mancip, [70], en particulier pour la Section A.2.3.

Nous nous plaçons en dimension deux, $d = 2$, pour coller au plus près de la question de l'intersection de deux maillages plans, qui est abordée dans la Section 2.4. Ceci posé, tout ce que nous disons dans ce Chapitre s'étend de façon directe à toute dimension $d \geq 1$.

Nous nous plongeons donc dans l'espace affine usuel $\mathcal{A}$, de direction l'espace vectoriel euclidien $\mathbb{R}^d$, avec $d = 2$. On note O l'origine et (Ox_1) et (Ox_2) les axes des abscisses et des ordonnées. Etant donnés deux points de l'espace affine $\mathcal{A}$, X et Y, on appellera $\overrightarrow{XY}$ l'unique vecteur v de $\mathbb{R}^d$ tel que $X + v = Y$.

A.1 Définition

Nous rappellons ici une définition des coordonnées barycentriques, ainsi que quelques unes de leurs propriétés élémentaires.

Soit un triangle T en dimension $d = 2$ déterminé par $d + 1 = 3$ points $A_j = (a_{ij})_{i=1,2}$, $j = 1, 2, 3$, non-alignés, c'est-à-dire qu'ils forment une famille affinement libre. L'arête opposée au sommet A_j est appelée $\mathcal{E}_j$, voir la Figure A.1, et la droite portée par l'arête $\mathcal{E}_j$ est notée $(\mathcal{E}_j)$.

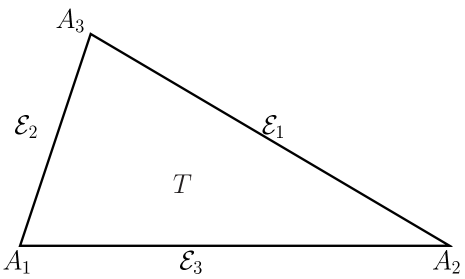


FIG. A.1 – Triangle T : notations des sommets et des arêtes.

Comme les points $A_j, j = 1, 2, 3$ forment une famille affinement libre, la matrice

$$\mathcal{M} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 1 & 1 & 1 \end{bmatrix}$$

est inversible. Tout point X de $\mathcal{A}$ de coordonnées cartésiennes $x_i,\ i = 1, 2$, est caractérisé par la donnée du vecteur $\lambda(X) \in \mathbb{R}^{d+1}$ solution du système linéaire

$$\lambda(X) = \begin{bmatrix} \lambda_1(X) \\ \lambda_2(X) \\ \lambda_3(X) \end{bmatrix} = \mathcal{M}^{-1} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}. \tag{A.1}$$

En d'autres termes, les 3 scalaires $\lambda_j(X),\ j = 1, 2, 3$, qui sont appelées les *coordonnées barycentriques* du point X par rapport aux 3 points A_j, sont définis intégralement et de façon unique[1] par le système

$$\sum_{j=1}^{3} a_{ij}\lambda_j(X) = x_i,\ i = 1, 2, \quad \text{et} \quad \sum_{j=1}^{3} \lambda_j(X) = 1.$$

Ceci s'interprète comme le fait que tout point X de l'espace affine $\mathcal{A}$ est la combinaison linéaire des points $A_j,\ j = 1, 2, 3$ —qui jouent tous le même rôle par symétrie— pondérée par les coordonnées $\lambda_j(X)$:

$$\overrightarrow{OX} = \sum_{j=1}^{3} \lambda_j(X)\ \overrightarrow{OA_j}, \quad \text{avec} \quad \sum_{j=1}^{3} \lambda_j(X) = 1.$$

[1] Les coordonnées barycentriques sont parfois définies à une constante multiplicative non nulle près. Nous avons fixé cette constante en imposant que la somme des coordonnées barycentriques est égale à 1. Dans ce cas, les coordonnées barycentriques existent et sont uniques pour toute famille affinement libre $(A_j)_{j=1,\ldots,d+1}$ et tout point X de l'espace affine $\mathcal{A}$.

En fait, il est facile de voir que n'importe quel point peut jouer le rôle de O dans l'équation ci-dessus. Soit P un point quelconque du plan $\mathcal{A}$, on peut donc écrire

$$\overrightarrow{PX} = \sum_{j=1}^{3} \lambda_j(X)\, \overrightarrow{PA_j}, \quad \text{avec} \quad \sum_{j=1}^{3} \lambda_j(X) = 1. \tag{A.2}$$

A.2 Quelques propriétés

A.2.1 Caractérisation de droites

Parmi les propriétés remarquables des coordonnées barycentriques, elles permettent de caractériser très simplement les droites parallèles aux arêtes $\mathcal{E}_j, j = 1,2,3$, du triangle T.[2]

Lemme 3. *La droite portée par l'arête $\mathcal{E}_j$ est caractérisée par*

$$(\mathcal{E}_j) = \{X \in \mathcal{A} \ / \ \lambda_j(X) = 0\}, \qquad j = 1,2,3.$$

Preuve : Pour un j dans $\mathbb{Z}/3\mathbb{Z}$, de façon à avoir $j, j+1$ et $j+2$ dans $\{1,2,3\}$, la droite portée par une arête $\mathcal{E}_j$ s'écrit par définition

$$(\mathcal{E}_j) = \left\{X \in \mathcal{A} \ / \ \overrightarrow{OX} = (1-\mu)\,\overrightarrow{OA_{j+1}} + \mu\,\overrightarrow{OA_{j+2}},\ \mu \in \mathbb{R}\right\}, \qquad j = 1,2,3.$$

Donc, en utilisant la définition (A.2) avec $P = O$, on obtient immédiatement $\{X \in \mathcal{A} \ / \ \lambda_j(X) = 0\} \subset (\mathcal{E}_j)$.

La réciproque provient de l'unicité des coordonnées barycentriques. □

Le Lemme 3 a une interprétation de géométrie analytique intéressante.

Soit la matrice Λ, inverse de la matrice $\mathcal{M}$, qu'on décompose en ses vecteurs lignes $\Lambda^j, j = 1,2,3$

$$\Lambda = \mathcal{M}^{-1} = \begin{bmatrix} \Lambda^1 \\ \hline \Lambda^2 \\ \hline \Lambda^3 \end{bmatrix}. \tag{A.3}$$

Avec ces notations, les coordonnées barycentriques du point X définies par le système (A.1) deviennent évidemment

$$\lambda(X) = \begin{bmatrix} \lambda_1(X) \\ \lambda_2(X) \\ \lambda_3(X) \end{bmatrix} = \begin{bmatrix} \Lambda^1 \\ \hline \Lambda^2 \\ \hline \Lambda^3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}.$$

[2] Ceci se généralise sans difficulté en dimension $d \geq 1$ à tout hyperplan parallèle à une hyperface du simplexe T.

Ceci, associé au Lemme 3 permet d'écrire la droite $(\mathcal{E}_j)$ de façon analytique

$$(\mathcal{E}_j) = \left\{X = [x_1, x_2]^\top \in \mathcal{A} \ / \ \Lambda^j \cdot [x_1, x_2, 1]^\top = 0\right\}, \qquad j = 1, 2, 3,$$

soit encore, introduisant les composantes $(\Lambda^j)_k, k = 1, \ldots, d+1$ du vecteur ligne (Λ^j),

$$(\mathcal{E}_j) = \left\{X = (x_j)_{j=1,\ldots,d} \in \mathcal{A} \ / \ \sum_{k=1}^{d} (\Lambda^j)_k \ x_k + (\Lambda^j)_{d+1} = 0\right\}, \quad j = 1, \ldots, d+1. \tag{A.4}$$

Pour tout $j = 1, 2, 3$, cette caractérisation (A.4) permet d'identifier le vecteur n^j de $\mathbb{R}^2$, défini par ses composantes $(n^j)_k = (\Lambda^j)_k, k = 1, 2$, comme un vecteur normal à la droite $(\mathcal{E}_j)$. On a donc montré le

Lemme 4. *Pour $j = 1, 2, 3$, toute droite D_j parallèle à l'arête $\mathcal{E}_j$ est caractérisée comme*

$$D_j // (\mathcal{E}_j) \quad \Longleftrightarrow \quad \exists\, c_j \in \mathbb{R} \text{ tel que } D_j = \{X \in \mathcal{A} \ / \ \lambda_j(X) = c_j\}.$$

□

En particulier, comme on vérifie aisément que

$$\lambda_j(A_k) = \delta_{jk} \qquad j, k = 1, 2, 3, \tag{A.5}$$

ou δ_{jk} est le symbole de Kronecker, la droite passant par A_j et parallèle à $(\mathcal{E}_j)$ s'écrit $\{X \in \mathcal{A} \ / \ \lambda_j(X) = 1\}$ pour $j = 1, 2, 3$.

A.2.2 Distance d'un point à une droite

Lemme 5. *Soit un point X fixé dans l'espace affine $\mathcal{A}$. Soit également un j dans $\{1, 2, 3\}$. La distance euclidienne d'un point $X = (x_1, x_2)$ à la droite $(\mathcal{E}_j)$ est notée $d(X, (\mathcal{E}_j))$. Elle vaut*

$$d(X, (\mathcal{E}_j)) = d(A_j, (\mathcal{E}_j))|\lambda_j(X)| = \frac{|\lambda_j(X)|}{\sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2}} = \frac{|(\Lambda^j)_1 x_1 + (\Lambda^j)_2 x_2 + (\Lambda^j)_3|}{\sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2}}$$

où les $(\Lambda^j)_k, k = 1, 2, 3$, sont les composantes du vecteur ligne Λ^j de la matrice Λ des coordonnées barycentriques définie en (A.3).

Preuve : La distance d'un point $A = (a_1, a_2)$ à un hyperplan D —une droite ici— d'équation $D: \ \alpha_1 x_1 + \alpha_2 x_2 + h = 0$, où $\alpha_i \in \mathbb{R}, i = 1, 2$, et $h \in \mathbb{R}$ s'écrit

$$d(A, D) == \frac{|\alpha_1 a_1 + \alpha_2 a_2 + h|}{\sqrt{\alpha_1^2 + \alpha_2^2}}.$$

En utilisant l'équation de la droite $(\mathcal{E}_j)$ donnée à l'équation (A.4), on obtient immédiatement le résultat. On remarque au passage que la hauteur issue du sommet A_j s'écrit, d'après l'identité (A.5),

$$h_j := d(A_j, (\mathcal{E}_j)) = \frac{1}{\sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2}}. \tag{A.6}$$

$\square$

A.2.3 Appartenance d'un point à un triangle

Le triangle T de sommets $A_j, \ j = 1, 2, 3$, est caractérisé comme

$$T = (A_1 A_2 A_3) = \left\{ X \in \mathbb{R}^2 \ / \ \lambda_j(X) \in [0, 1], j = 1, 2, 3 \right\}.$$

Ecrasement d'un triangle

On peut introduire une notion très simple d'anisotropie du maillage à partir des hauteurs du triangle :

Définition 1. On dira que le triangle T est applati dans la direction j quand sa hauteur h_j est très inférieure à ses deux autres hauteurs, c'est-à-dire quand

$$h_j = \frac{1}{\sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2}} << h_k, \qquad j, k \in \{1, 2, 3\}, \ k \neq j.$$

$\square$

Appartenance à un triangle

Ce qui suit est issu d'une note de [70].

L'appartenance d'un point X à une maille triangulaire en géométrie "exacte" est donc particulièrement simple avec les coordonnées barycentriques ; elle consiste à réaliser trois doubles tests :

$$0 \leq \lambda_j(X) \leq 1, \qquad j = 1, 2, 3.$$

Cependant, avec les erreurs d'arrondis et les anisotropies du maillage, il devient nécessaire de prendre quelques précautions. Le test d'appartenance d'un point à un triangle doit être fait *par excès* : nous voulons être sûrs d'obtenir tous les points qui appartiennent au triangle T, quitte à en avoir trop. Nous englobons donc le triangle d'une gangue de faible épaisseur, et le test d'appartenance se fait non pas par rapport au triangle T lui-même, mais par rapport à ce triangle enveloppé de sa gangue, que nous appellerons dorénavant "surtriangle", noté $\overline{T}$.

En introduisant un seuil de tolérance $\varepsilon > 0$, deux sortes de tests sont possibles :

tests relatifs On teste dans ce cas

$$-\varepsilon < \lambda_j(X) < 1 + \varepsilon, \qquad j = 1, 2, 3. \tag{A.7}$$

Multipliées par la hauteur h_j, définie en (A.6), les inéquations (A.7) deviennent, en utilisant le Lemme 5,

$$-\varepsilon\, h_j < \lambda_j(X)\, h_j = d(X, (\mathcal{E}_j)) < h_j + \varepsilon\, h_j, \qquad j = 1, 2, 3,$$

où nous avons étendu la notion de distance $d(X, (\mathcal{E}_j))$ à une notion de distance *algébrique* (≥ 0 si le point X est dans le même demi-plan délimité par la droite $(\mathcal{E}_j)$ que le sommet A_j, et ≤ 0 sinon).

L'épaisseur de la gangue du surtriangle $\overline{T}$ dépend des hauteurs h_j du triangle T. Cette gangue n'est donc pas uniforme si le triangle est fortement applati, cf. la Définition 1. J'ai qualifié ces tests de *relatifs* pour cette raison, par opposition aux tests *absolus* dans lesquels la gangue a une épaisseur fixe ε.

tests absolus On teste dans ce cas

$$-\varepsilon\ \sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2} < \lambda_j(X) < 1 + \varepsilon\ \sqrt{(\Lambda^j)_1^2 + (\Lambda^j)_2^2}, \qquad j = 1, 2, 3. \tag{A.8}$$

Multipliées par la hauteur h_j, les inéquations (A.8) deviennent cette fois

$$-\varepsilon < \lambda_j(X)\, h_j = d(X, (\mathcal{E}_j)) < h_j + \varepsilon, \qquad j = 1, 2, 3.$$

Dans ce type de tests, sont considérés à l'intérieur du triangle T tous les points qui sont distants à moins de ε de T, indépendamment des hauteurs, et donc de l'écrasement du triangle. Contrairement aux tests dits "relatifs" dans lesquels ε était sans dimension, comme $\lambda_j(X)$, le seuil ε est homogène à une distance. Il est peut-être plus aisé dans ce cas de le choisir en fonction des tailles caractéristiques du maillage.

A propos du choix du seuil ε

Pour finir, il reste à choisir correctement le seuil de tolérance, ce qui peut s'avérer délicat et important. Nous proposons comme choix du seuil de tolérance une valeur sans dimension, locale à une maille telle que

$$\varepsilon = \sum_{j=1}^{3} \left[|1 - \lambda_j (A_j)| + \sum_{j' \neq j} |\lambda_j (A_{j'})| \right]. \tag{A.9}$$

En fait ce seuil de tolérance est basé sur l'erreur commise en inversant la matrice $\mathcal{M}$ définie à l'équation (A.1), ou bien, autrement dit, sur l'erreur commise sur les identités (A.5).

Conclusion

Nous avons expérimenté les deux types de tests. Sur des maillages peu déformés, cela ne change guère les résultats. En revanche, nous avons observé de fortes différences sur des maillages en rectangles —découpés en deux triangles— fortement écrasés, comme dans la simulation numérique de la Section 3.6.2. Avec les tests "relatifs", l'Algorithme 3 de la Section 2.4 ne parvient pas dans certains cas à calculer le maillage d'intersection, tandis que le même Algorithme 3 opère correctement le calcul du maillage d'intersection en utilisant les tests absolus et le seuil donné par (A.9).

Annexe B

Problème elliptique 1D avec coefficients

Nous présentons dans cette partie la solution du problème elliptique (B.1) avec des coefficients constants par morceaux dans le cas où le domaine Ω est dans $\mathbb{R}$.

Soit donc le problème

$$\begin{array}{rcll} \operatorname{div} \mathbf{u} & = & 0 & \text{dans } \Omega = [0; L] \\ \mathbf{u} & = & -\mathbf{K} \boldsymbol{\nabla} p & \text{dans } \Omega \\ p(0) & = & P_0 & \\ p(L) & = & P_L, & \end{array} \tag{B.1}$$

où L est le diamètre du domaine Ω et P_0 et P_L sont des valeurs de Dirichlet imposées aux extrémités du domaine Ω. Les autres notations sont les mêmes que celles de l'équation (2.2) dans la Section 2.2.

Nous supposons qu'il existe une partition de Ω en $n > 1$ sous-domaines sans chevauchement $\Omega_i, i = 1, 2, \ldots, n$, tels que le sous-domaine Ω_i est de longueur d_i et possède une perméabilité *constante* $\mathbf{K_i}$, voir la Figure B.1. Nous supposons également pour fixer les idées que les sous-domaines se suivent de gauche à droite : Ω_i est à gauche de Ω_{i+1}, $i = 1, 2, \ldots, n-1$.

Il est facile de retrouver la solution du problème (B.1). Elle est telle que la vitesse est constante sur Ω et la pression est continue et linéaire par morceaux.

$$\begin{array}{rcll} \mathbf{u}(x) & = & V\, \mathbf{e_x}, & x \in \Omega, \quad V = \text{Cst}, \\ p_i(x) & = & -\dfrac{V}{\mathbf{K_i}} x + C_i, & x \in \Omega_i, \quad i = 1, 2, \ldots, n, \end{array} \tag{B.2}$$

où nous avons appellé $\mathbf{e_x}$ le vecteur unitaire selon Ox, et la vitesse V et les C_i sont des constantes déterminées par la continuité de la pression aux interfaces

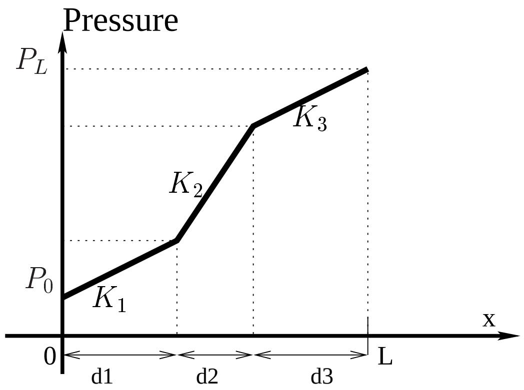


Fig. B.1 – Pression en fonction de l'abscisse dans un domaine 1D $\Omega = [0; L]$ avec des coefficients constants par morceaux $\mathbf{K_i}$.

et les conditions aux limites. En particulier, la vitesse V se calcule aisément et dépend de la moyenne harmonique des rapports de la perméabilité par le diamètre des sous-domaines

$$P_L - P_0 = -V \left(\sum_{i=1}^{n} \frac{d_i}{\mathbf{K_i}} \right). \tag{B.3}$$

Nous rappelons ici que par définition, la moyenne harmonique $\mathrm{MoyH}\left((a_j)_j\right)$ d'une suite finie de réels $(a_j)_{j=1,2,\ldots,m}$ s'écrit

$$\mathrm{MoyH}\left((a_j)_j\right) = \frac{m}{\displaystyle\sum_{j=1}^{m} \frac{1}{a_j}}. \tag{B.4}$$

Avec cette définition, la vitesse V devient

$$V = -\frac{1}{n}\, \mathrm{MoyH}\left(\frac{\mathbf{K_i}}{d_i}\right)\, (P_L - P_0). \tag{B.5}$$

Si l'on suppose de plus que les sous-domaines ont tous la même longueur $d_i = L/n$, $i = 1, 2, \ldots, n$, alors la vitesse s'exprime

$$V = -\frac{\mathrm{MoyH}\left(\mathbf{K_i}\right)}{L}\, (P_L - P_0). \tag{B.6}$$

Annexe C

Quelques précisions à propos d'Ocaml et d'OcamlP3l

La Section C.1 contient un petit lexique où certains concepts informatiques et des termes techniques d'Ocaml et d'OcamlP3l sont définis. La Section C.2 présente l'interface du module Ddec servant pour la mise en œuvre de la décomposition de domaines avec OcamlP3l.

C.1 Petit lexique de Ocaml et d'OcamlP3l

Nous donnons une brève définition de certains termes informatiques qui sont employés dans le mémoire. Certains recouvrent des nouveautés ou des spécificités d'OcamlP3l ; d'autres ont une portée plus générale mais ne doivent pas paraître familiers à la plupart des numériciens.

Couleur (option OcamlP3l) Les couleurs permettent à l'utilisateur d'indiquer à l'environnement OcamlP3l comment associer les nœuds virtuels —les processus— aux nœuds physiques —les processeurs. L'utilisateur colorie séparément des nœuds virtuels et des nœuds physiques, et un algorithme apparie ensuite les deux sortes de nœuds coloriés au moment de la compilation. Cette coloration est optionnelle, car le système se charge automatiquement de la répartition des charges, les couleurs ne servant qu'à le guider dans cette tâche. Voir la Section 3.4.8.

Combinateur Fonction qui n'a aucune variable libre, c'est-à-dire qui n'est *que* combinaison de ses arguments. Par exemple, la fonction `function z -> z + 1` n'est pas un combinateur dans la mesure où le `+` n'est pas défini en argument. Une algèbre de combinateurs est compositionnelle par construction : il existe toujours un moyen de composer des combinateurs entre eux.

L'espace des polynômes réels muni de $(+, *)$ est un bon exemple d'algèbre compositionnelle. L'espace des fonctions réelles muni de $(+, \circ)$ constitue un autre exemple mais très restrictif dans la mesure où l'espace d'arrivée d'une fonction doit coïncider avec l'espace de départ d'une seconde, pour pouvoir les composer entre elles.

Compositionnel Voir Combinateur.

Constructeur de type polymorphe Le constructeur de type polymorphe noté `'a`, qu'on lit "alpha", remplace n'importe quel type : `int`, `float`, vecteur... Voir un exemple dans le paragraphe sur le polymorphisme de la Section 3.3.4 et la définition de polymorphisme ci-dessous.

Dynamique (typage) Par opposition au typage statique, la vérification qu'un argument passé à une fonction est bien du type du paramètre formel de la fonction s'éffectue *au cours de l'éxecution* du programme.

Effet ou Effet de bord Un effet est une modification d'une case de la mémoire —tableau ou référence—, ou encore une interaction avec le monde extérieur —impression ou lecture. La programmation impérative est basée sur le principe de faire des effets, cf. la défintion d'impératif ci-dessous. Une méthode agissant sur un objet en programmation orientée objets réalise par définition un effet de bord[1]. Un effet de bord pose généralement des problèmes de sûreté, mais il peut être plus ou moins vicieux : une impression à l'écran est certes moins dangereuse que la modification d'un paramètre par une fonction qui ne le prend *pas* en argument —en utilisant le `common` du Fortran par exemple.

`farm` **(squelette OcamlP3l)** Une ferme calcule en parallèle une fonction sur les différents éléments qui viennent sur son flux d'entrée. C'est un squelette de tâches parallèles. Voir les Sections 3.4.4 et 3.4.5.

Flux ou flot voir `stream` ci-dessous.

`fold` **(fonction Ocaml)** La fonction `fold` accumule une fonction sur tous les éléments d'un tableau, pour renvoyer un scalaire. Un exemple simple de son utilisation est la norme d'un vecteur, voir la Section 3.3.6.

Fonctionnelle (substantif) Fonction de fonctions ou de fonctionnelles ; une fonctionnelle d'ordre 1 est une fonction d'ordre 2. La fonction `compose` de la Section 3.3.4 est un exemple de fonctionnelle d'ordre 1.

Fonction d'ordre supérieur Fonction dont les arguments ou les résultats sont des fonctions, qui peuvent elles-mêmes être fonctions de fonctions,

[1] Ainsi, quand j'ai découvert les effets de bord, aurais-je pu dire, à l'instar de M. Jourdain [73, Acte II, Scène 4] : "cela fait plus de six ans que je fais des effets de bord, sans que j'en susse rien."

etc... à un ordre arbitraire. Une fonction d'ordre 2 est une fonctionnelle d'ordre 1, voir ci-dessus.

Fonctionnelle (programmation) Style de programmation basé sur la manipulation de fonctions, qui s'avère par conséquent très proche des mathématiques et très naturel. Les fonctions *calculent* le résultat souhaité au sens des calculs mathématiques, c'est-à-dire par simplification successives d'une expression.

D'après [25], pages 11–12 : *"un programme est une fonction appliquée à ses arguments ; il calcule un résultat qui est retourné (quand le calcul se termine) comme sortie du programme. Il devient alors facile de composer des programmes : la sortie d'un programme devient l'argument d'entrée d'un autre, au sens de la composition de fonctions.*

La programmation fonctionnelle repose sur un modèle de calcul simple possédant trois constructions : les variables, la définition d'une fonction et son application à un argument. Ce modèle s'appelle le λ-calcul et il a été introduit par Alonzo Church en 1932. [...] *En λ-calcul, toutes les fonctions sont des valeurs manipulables.* [...] *Il lui a été ajouté des valeurs de base (tels que les entiers ou les chaînes de caractères), des opérateurs sur ces valeurs de base, des structures de contrôle et les déclarations qui permettent de nommer des valeurs ou des fonctions et, en particulier, des fonctions récursives.*

Il existe plusieurs classifications des langages fonctionnels. Pour notre part, nous les distinguons suivant deux caractéristiques qui nous paraissent prépondérantes :

- *sans effet de bord (pur) ou avec effets de bord (impur) : un langage fonctionnel pur est un langage où l'affectation n'existe pas. Tout n'y est que calcul et l'on ne s'intéresse pas à son déroulement. Des langages fonctionnels impurs, comme* Lisp *ou* ML [famille dont fait partie Ocaml]*, intègrent des traits impératifs comme l'affectation. Ils permettent d'écrire des algorithmes dans un style plus proche de langages comme* Fortran *où l'ordre d'évaluation des expressions est déterminant.*
- *typé dynamiquement ou statiquement : le typage permet de vérifier si un argument passé à une fonction est bien du type du paramètre formel de la fonction. On peut faire cette vérification à l'exécution du programme* [cas du *typage dynamique*, ..., ou bien] *avant l'exécution du programme, c'est-à-dire au moment de la compilation. On appelle cette vérification a priori le* typage statique*. Etant effectué une fois pour toutes, elle ne ralentira pas l'exécution du programme. C'est le cas du langage* ML *et de ses dialectes comme* Ocaml*. Seuls les programmes correctement*

typés, c'est-à-dire ceux acceptés par le vérificateur de types, pourront être compilés puis exécutés."

Impérative (programmation) Style de programmation habituel aux numériciens qui codent en Fortran, C ou C++. Un programme est une suite d'ordres donnés à l'ordinateur qui modifient son état mémoire. On travaille essentiellement sur des tableaux, des listes et des références, en utilisant surtout des boucles (`for`, `while`).

D'après [97], page 37 : en programmation impérative, *"un calcul est un processus évolutif, où le temps a son importance. Il s'agit de* modifier un état *: l'ordinateur commence l'exécution du programme dans un certain état initial, que l'exécution du programme modifie jusqu'à parvenir à un état final qui contient le résultat voulu. On change l'état courant par modification du contenu de la mémoire de l'ordinateur (à l'aide d'*affectations*), ou encore par interaction avec le monde extérieur : interrogation de l'utilisateur, affichage de résultats, lecture ou écriture de fichiers, bref tout ce qu'on nomme les entrées-sorties. Toutes ces opérations qui modifient physiquement le contenu des adresses mémoire sont appelées* effets.*"*

Voir les effets ci-dessus.

`loop` **(squelette OcamlP3l)** Il permet de répéter l'exécution d'une expression squelettique. Sa portée est fortement réduite depuis l'introduction des `parfun`s. C'est un squelette de contrôle. Voir la Section 3.4.4

`map` **(fonction Ocaml)** La fonction `fold` applique une fonction à tous les éléments d'un tableau et renvoie le nouveau tableau. Un exemple simple de son utilisation se trouve dans la Section 3.3.5.

`mapvector` **(squelette OcamlP3l)** Il calcule en parallèle une fonction sur toutes les composantes d'un vecteur, renvoyant le —nouveau— vecteur de résultat. C'est un squelette de données parallèles. Voir les Sections 3.4.4 et 3.4.5.

`pardo` **(délimiteur de champ OcamlP3l)** Il permet de séparer la partie du programme où sont *définies* les fonctions encapsulées par un `parfun`, de celle où on peut *exécuter* ces fonctions. Voir la Section 3.4.7.

Paresseux Dans un langage paresseux —*lazy*—, par opposition à un langage strict, les arguments d'une fonction ne sont pas complètement évalués avant de lui être appliqués. Les flux ou `streams` du langage Ocaml sont évalués paresseusement : *"lorsqu'on construit le flux des caractères provenant d'un fichier, ce dernier n'est pas lu tout entier en mémoire : le flux ne contient en mémoire que le caractère courant et va chercher le prochain caractère sur le disque lorsqu'on en a besoin. Ce comportement est éco-*

nomique en mémoire, en particulier quand le fichier est gros", [97], page 158.

`parfun` **(squelette OcamlP3l)** Il convertit un réseau de calcul parallèle en une fonction agissant sur des flux. C'est un squelette d'interface de données, dual de `seq`. Voir les Sections 3.4.4 et 3.4.6.

`pipe` **(squelette OcamlP3l)** noté `|||`. Il calcule en parallèle les différentes phases d'une composition de fonctions sur les différents éléments du flux d'entrée. C'est un squelette de tâches parallèles. Voir les Sections 3.4.4 et 3.4.5.

Polymorphisme Possibilité d'avoir plusieurs formes. En informatique, cela se traduit par la possibilité pour des programmes ou des objets d'être utilisés dans des contextes très variables. Ainsi une comparaison $\leq$ qui peut s'appliquer à des entiers, des flottants ou des chaînes de caractères est dite polymorphe. Voir Constructeur de type polymorphe.

`reducevector` **(squelette OcamlP3l)** Il accumule une fonction en parallèle sur toutes les composantes d'un vecteur pour renvoyer un scalaire, de la même manière qu'une fonction Ocaml —séquentielle— *fold*, se reporter au `fold` ci-dessus et à la Section 3.3.6. C'est un squelette de données parallèles. Voir également les Sections 3.4.4 et 3.4.5.

Sémantique Interprétation, signification d'un système formel (par opposition à syntaxique), d'après le Petit Larousse illustré, 1998. Quand on applique cette définition à l'informatique, la sémantique est la signification mathématique d'un exécutable à partir du programme source dont il est issu. Cette signification est toujours une fonction mathématique. Dans OcamlP3l, il existe trois significations différentes d'un même programme source : l'une est séquentielle, l'autre parallèle et la dernière graphique.

`seq` **(squelette OcamlP3l)** Il convertit une fonction séquentielle en un nœud du réseau de calcul parallèle C'est un squelette d'interface de données, dual de `parfun`. Voir les Sections 3.4.4 et 3.4.5.

Squelette Un squelette est une brique de base servant à construire l'édifice parallèle dans un programme OcamlP3l. C'est uniquement à travers ces 6 squelettes —`parfun`, `seq`, `loop`, `pipe`, `mapvector`, `reducevector`— que l'on peut construire un réseau parallèle en OcamlP3l. Il convient également d'ajouter `loop`, qui a un rôle particulier, et `pardo`, qui n'est pas un squelette, pour énumérer exhaustivement toute la syntaxe OcamlP3l.

Les squelettes sont des combinateurs : on peut les composer arbitrairement entre eux, et l'ensemble des squelettes forme une algèbre compositionnelle appelée le langage squelettique. Chaque squelette est une fonction sur des flux : il prend en argument un flux et retourne un flux en sortie. Enfin, dans la version actuelle, grâce à une spécifité de l'implémentation des squelettes, visible dans leurs types, on peut initialiser *globalement* ou *localement* les données dans un calcul parallèle. L'initialisation locale est particulièrement importante pour la décomposition de domaines. Voir la Section 3.4.5.

Statique (typage) Par opposition au typage dynamique, la vérification qu'un argument passé à une fonction est bien du type du paramètre formel de la fonction s'éffectue *avant* l'exécution du programme, *pendant la compilation.*

`stream` Dans le système OcamlP3l, c'est un flux —ou flot— de données, qui sert d'entrée et de sortie pour les squelettes OcamlP3l. Un `int stream` est un flux d'entiers et un `'a stream` est un flux de trucs qui ont un type `'a`.

Strict Dans un langage strict, comme Ocaml, par opposition à un langage paresseux, les arguments d'une fonction sont complètement évalués avant de lui être appliqués.

C.2 Interface du module Ddec

```
 (* Module Ddec for 3D domain decomposition *)

val filename_of_3D_mesh_number : int -> string
val filename_of_2D_mesh_number : int -> int -> string

val number_of_subdomains : int
val number_of_processors : int

type interface
type connectivity_table = interface list
val connectivity_table : connectivity_table

val colv_sorting_subdomains : int array
val colv_sorting_connectivity_table : int array

type interface_values
and internal_boundary_values = interface_values array
and tagged_internal_boundary_values =
```

```
  | Init
  | Loop of internal_boundary_values
  | Final of internal_boundary_values
and structure_values = internal_boundary_values array

val zero_structure_values : int -> structure_values

val read_internal_boundary_values :
    in_channel -> internal_boundary_values
val print_internal_boundary_values :
    out_channel -> internal_boundary_values -> unit
val print_tagged_internal_boundary_values :
    out_channel -> tagged_internal_boundary_values -> unit

val init_vector_of_size :
    int -> int * tagged_internal_boundary_values array
val loop_vector_of : structure_values
  -> int * tagged_internal_boundary_values array
val final_vector_of : structure_values
  -> int * tagged_internal_boundary_values array

val axpy : float -> structure_values -> structure_values
  -> structure_values
val dot : structure_values -> structure_values -> float

val permutation_of : structure_values -> structure_values

type projection_matrix
type projection_matrices = projection_matrix array array

val projection_with :
    projection_matrices -> structure_values -> structure_values
```

Bibliographie

[1] Y. Achdou, C. Japhet, P. Le Tallec, F. Nataf, F. Rogier, et M. Vidrascu. Domain decomposition methods for non-symmetric problems. In *Eleventh International Conference on Domain Decomposition Methods (London, 1998)*, pages 3–17 (electronic). DDM.org, Augsburg, 1999.

[2] Y. Achdou, C. Japhet, Y. Maday, et F. Nataf. A new cement to glue nonconforming grids with Robin interface conditions : the finite volume case. *Numer. Math.*, 92(4) :593–620, 2002. ISSN 0029-599X.

[3] Yves Achdou, Patrick Le Tallec, Frédéric Nataf, et Marina Vidrascu. A domain decomposition preconditioner for an advection-diffusion problem. *Comput. Methods Appl. Mech. Engrg.*, 184(2-4) :145–170, 2000. ISSN 0045-7825. Vistas in domain decomposition and parallel processing in computational mechanics.

[4] Yves Achdou et Frédéric Nataf. Preconditioners for the mortar method based on local approximations of the Steklov-Poincaré operator. *Math. Models Methods Appl. Sci.*, 5(7) :967–997, 1995. ISSN 0218-2025.

[5] Yves Achdou et Frédéric Nataf. A Robin-Robin preconditioner for an advection-diffusion problem. In *Domain decomposition methods, 10 (Boulder, CO, 1997)*, pages 377–383. Amer. Math. Soc., Providence, RI, 1998.

[6] P.M. Adler et J.-F. Thovert. *Fractures and fracture network*. Kluwer, 1999.

[7] A. Agouzal, J. Baranger, J.-F. Maître, et F. Oudin. Connection between finite volumes and mixed finite element methods for a diffusion problem with nonconstant coefficients, with application to convection-diffusion. *East-West Journal on Numerical Analysis*, 3 :237–254, 1995.

[8] Pierre Alart, Mikaël Barboteu, Patrick Le Tallec, et Marina Vidrascu. Méthode de Schwarz additive avec solveur grossier pour problèmes non symétriques. *C. R. Acad. Sci. Paris Sér. I Math.*, 331(5) :399–404, 2000. ISSN 0764-4442.

[9] C. Alboin, J. Jaffré, J. E. Roberts, et C. Serres. Domain decomposition for flow in fractured porous media. In C.H. Lai, P. E. Bjorstad, M. Cross, et O. B. Widlund, editors, *Domain Decomposition Methods in Sciences and Engineering*, pages 365–373. Domain decomposition press, 1999.

[10] Clarisse Alboin, Jérôme Jaffré, Jean E. Roberts, Xuewen Wang, et Christophe Serres. Domain decomposition for some transmission problems in flow in porous media. In *Numerical treatment of multiphase flows in porous media (Beijing, 1999)*, volume 552 of *Lecture Notes in Phys.*, pages 22–34. Springer, Berlin, 2000.

[11] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et D. Sorensen. *LAPACK user's guide, 3rd edition.* SIAM, 1999.

[12] Ph. Angot, T. Gallouët, et R. Herbin. Convergence of finite volume method on general meshes for non smooth solution of elliptic porblems with cracks. In F. Benkhaldoun R. Vilsmeier et D. Hänel Eds, editors, *Proc. 2nd Int. Symposium Finite Volumes for Complex Applications, Duisberg, Germany, July 1999*, pages 215–222, 1999.

[13] Todd Arbogast, Lawrence C. Cowsar, Mary F. Wheeler, et Ivan Yotov. Mixed finite element methods on nonmatching multiblock grids. *SIAM J. Numer. Anal.*, 37(4) :1295–1315 (electronic), 2000. ISSN 1095-7170.

[14] Todd Arbogast et Ivan Yotov. A non-mortar mixed finite element method for elliptic problems on non-matching multiblock grids. *Comput. Methods Appl. Mech. Engrg.*, 149(1-4) :255–265, 1997. ISSN 0045-7825. Symposium on Advances in Computational Mechanics, Vol. 1 (Austin, TX, 1997).

[15] D.N. Arnold, D. Boffi, et R. S. Falk. Quadrilateral H(div) finite elements. Submitted to SIAM J. on Numerical Analysis, see URL http ://www.ima.umn.edu/ arnold/publications.html.

[16] D.N. Arnold et F. Brezzi. Mixed and nonconforming finite element methods : implementation, postprocessing and error estimates. *M2AN*, 19 : 7–32, 1985.

[17] N. Barberou, M. Garbey, M. Hess, M. M. Resch, T. Rossi, Toivanen J., et D. Tromeur-Dervout. Efficient metacomputing of elliptic linear and nonlinear problems. *J. Parallel Distrib. Comput.*, 63 :564–577, 2003.

[18] Peter Bastian, Zhangxin Chen, Richard E. Ewing, Rainer Helmig, Hartmut Jakobs, et Volker Reichenberger. Numerical simulation of multiphase flow in fractured porous media. In *Numerical treatment of multiphase flows in porous media (Beijing, 1999)*, volume 552 of *Lecture Notes in Phys.*, pages 50–68. Springer, Berlin, 2000.

[19] J. Bear et A. Verruijt. *Modeling Groundwater Flow and Pollution.* D. Reidel Publishing Company, Holland, 1987.

[20] C. Bernardi, Y. Maday, et A. T. Patera. Domain decomposition by the mortar element method. In *Asymptotic and numerical methods for partial differential equations with critical parameters (Beaune, 1992)*, pages 269–286. Kluwer Acad. Publ., Dordrecht, 1993.

[21] C. Bernardi, Y. Maday, et A. T. Patera. A new nonconforming approach to domain decomposition : the mortar element method. In *Nonlinear partial differential equations and their applications. Collège de France Seminar, Vol. XI (Paris, 1989–1991)*, pages 13–51. Longman Sci. Tech., Harlow, 1994.

[22] Christine Bernardi et Yvon Maday. Spectral, spectral element and mortar element methods. In *Theory and numerics of differential equations (Durham, 2000)*, pages 1–57. Springer, Berlin, 2001.

[23] F. Brezzi et M. Fortin. *Mixed and Hybrid Finite Element Methods.* Springer-Verlag, Berlin, 1991.

[24] Albert Camus. *Caligula.* Gallimard, Paris, 1958. In French.

[25] Emmanuel Chailloux, Pascal Manoury, et Bruno Pagano. *Développement d'applications avec Objective Caml.* O'Reilly, Paris, France, 2000. In French.

[26] S. Chandrasekaran et Ming Gu. A fast and stable solver for recursively semi-separable systems of linear equations. In *Structured matrices in mathematics, computer science, and engineering, II (Boulder, CO, 1999)*, volume 281 of *Contemp. Math.*, pages 39–53. Amer. Math. Soc., Providence, RI, 2001.

[27] G. Chavent et J. Jaffré. *Mathematical Models and Finite Elements for Reservoir Simulation*, volume 17 of *Studies in Mathematics and its Applications.* North Holland, Amsterdam, Amsterdam, 1986.

[28] G. Chavent, J. Jaffré, et J. Roberts. Generalized cell-centered finite volume methods : application to two-phase flow in porous media. In M.-O. Bristeau, G. Etgen, W. Fittzgibbon, J.-L. Lions, J. Periaux, et M.-F. Wheeler, editors, *Computational Science for the 21st Century*, pages 231–241. John Wiley, Chichester, England, 1987.

[29] G. Chavent et J. Roberts. A unified physical presentation of mixed, mixed-hybrid finite elements and standard finite difference approximations for the determination of velocities in waterflow problems. *Advances in Water Ressources*, 14(6) :329–348, 1991.

[30] F. Clément, V. Martin, A. Vodicka, R. Di Cosmo, et P. Weis. Domain decomposition for flow simulation around a waste disposal site : direct computation versus code coupling using ocamlp3l. In *International Conference on Supercomputing in Nuclear Applications (SNA'2003)*, September 2003.

[31] F. Clément, A. Vodicka, R. Di Cosmo, et P. Weis. Couplage de codes numériques, parallélisme et langages de haut niveau. Rapport de Recherche 4825, Inria, Rocquencourt, France, 2003.

[32] Bernardo Cockburn. An introduction to the discontinuous Galerkin method for convection-dominated problems. In *Advanced numerical approximation of nonlinear hyperbolic equations (Cetraro, 1997)*, pages 151–268. Springer, Berlin, 1998.

[33] M. Cole. *Algorithmic Skeletons : Structured Management of Parallel Computations*. Research Monographs in Parallel and Distributed Computing. Pitman, 1989.

[34] Corneille. *Le Cid*. Larousse, Paris, 1970. In French.

[35] R. Di Cosmo, P. Weis, Zheng Li, F. Clément, V. Martin, et A. Vodicka. Parallel programming with the ocamlp3l system with application to coupling numerical codes. 2003. Submitted.

[36] Lawrence C. Cowsar, Jan Mandel, et Mary F. Wheeler. Balancing domain decomposition for mixed finite elements. *Math. Comp.*, 64(211) :989–1015, 1995. ISSN 0025-5718.

[37] Lawrence C. Cowsar et Mary F. Wheeler. Parallel domain decomposition method for mixed finite elements for elliptic partial differential equations. In *Fourth International Symposium on Domain Decomposition Methods*

for Partial Differential Equations (Moscow, 1990), pages 358–372. SIAM, Philadelphia, PA, 1991.

[38] M. Danelutto, R. Di Meglio, S. Orlando, S. Pelagatti, et M. Vanneschi. A methodology for the development and support of massively parallel programs. *Future Generation Computer Systems*, 8(1–3) :205–220, July 1992.

[39] Marco Danelutto, Roberto Di Cosmo, Xavier Leroy, et Susanna Pelagatti. Parallel functional programming with skeletons : the ocamlp3l experiment. *The ML Workshop*, 1998.

[40] J. Darlington, A. J. Field, P. G. Harrison, P. H. J. Kelly, D. W. N. Sharp, et Q. Wu. Parallel Programming Using Skeleton Functions. In *PARLE'93*, pages 146–160. Springer, 1993. LNCS No. 694.

[41] J. Darlington, Y. Guo, H. W. To, et J. Yang. Parallel Skeletons for Structured Composition. In *Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM Press, July 1995.

[42] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. Technical Report TR-03-006, U. of Florida, 2003. Submitted to ACM Trans. Math.

[43] B. Fraeijs de Veubeke. Stress functions approach. In *World Congress on Finite Element Methods in Structural Mechanics 1, Bournemouth, England*, pages J.1–J.51, 1975.

[44] Bruno Després. Domain decomposition method and the Helmholtz problem. In *Mathematical and numerical aspects of wave propagation phenomena (Strasbourg, 1991)*, pages 44–52. SIAM, Philadelphia, PA, 1991.

[45] Bruno Després. Domain decomposition method and the Helmholtz problem. II. In *Second International Conference on Mathematical and Numerical Aspects of Wave Propagation (Newark, DE, 1993)*, pages 197–206. SIAM, Philadelphia, PA, 1993.

[46] Bruno Després, Patrick Joly, et Jean E. Roberts. A domain decomposition method for the harmonic Maxwell equations. In *Iterative methods in linear algebra (Brussels, 1991)*, pages 475–484. North-Holland, Amsterdam, 1992.

[47] J. J. Dongarra, J. Du Croz, I. S. Duff, et S. Hammarling. Algorithm 679 : A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16 :18–28, 1990.

[48] J. Douglas, Jr., P. J. Paes-Leme, J. E. Roberts, et Jun Ping Wang. A parallel iterative procedure applicable to the approximate solution of second order partial differential equations by mixed finite element methods. *Numer. Math.*, 65(1) :95–108, 1993. ISSN 0029-599X.

[49] Robert Eymard, Thierry Gallouët, et Raphaèle Herbin. Finite volume methods. In *Handbook of numerical analysis, Vol. VII*, pages 713–1020. North-Holland, Amsterdam, 2000.

[50] C. Fahrat et F.X. Roux. Implicit parallel processing in structural mechanics. *Comput. Mech. Adv.*, 2(1) :1–124, 1993.

[51] I. Faille, E. Flauraud, F. Nataf, S. Pégaz-Fiornet, F. Schneider, et F. Willien. A new fault model in geological basin modelling, application to finite volume scheme and domain decomposition methods. In R. Herbin et D. Kröner Eds, editors, *Proc. 3rd Int. Symposium Finite Volumes for Complex Applications, Porquerolles, France, June 2002*, pages 543–550, 2002.

[52] Pascal J. Frey. Medit : an interactive mesh visualisation software. Technical report 0253, Inria, Rocquencourt, France, Dec. 2001.

[53] Pascal J. Frey et Paul-Louis George. *Maillages, applications aux éléments finis*. Hermes Sciences Publications, Paris, 1999. In French.

[54] M. Garbey et D. Tromeur-Dervout. On some Aitken-like acceleration of the Schwarz method. *Internat. J. Numer. Methods Fluids*, 40(12) :1493–1513, 2002. ISSN 0271-2091. LMS Workshop on Domain Decomposition Methods in Fluid Mechanics (London, 2001).

[55] Marc Garbey et Damien Tromeur-Dervout. Two level domain decomposition for multi-clusters. In T. Chan, T. Kako, H. Kawarado, et O. Pironneau, editors, *Proc. Int. Conf. on Domain Decomposition Methods DD12 in Chiba, Japan, 1999*, pages 325–340. ddm.org, 2001.

[56] Pencheva Gergina et Ivan Yotov. Balancing domain decomposition for mortar mixed finite element methods. *Numer. Linear Algebra Appl.*, 10 : 159–180, 2003.

[57] Roland Glowinski et Mary Fanett Wheeler. Domain decomposition and mixed finite element methods for elliptic problems. In *First International Symposium on Domain Decomposition Methods for Partial Differential Equations (Paris, 1987)*, pages 144–172. SIAM, Philadelphia, PA, 1988.

[58] Bernard Gostiaux. *Cours de mathématiques spéciales. Géométrie affine et métrique*, volume 4. Presses Universitaires de France, 1995. In French.

[59] William Gropp, Ewing Lusk, et Anthony Skjellum. *Using MPI, 2nd edition, Portable parallel programming with the the Message-Passing Interface*. The MIP Press, Cambridge, Massachussets, London, England, 1999.

[60] Georges Hansel. *Passeport pour C++ : concepts et mise en pratique*. Vuibert, Paris, 1999. In French.

[61] Pham Huy Hung et Enrique Sánchez-Palencia. Phénomènes de transmission à travers des couches minces de conductivité élevée. *J. Math. Anal. Appl.*, 47 :284–309, 1974.

[62] J. Jaffré, V. Martin, et J. Roberts. Modelling fractures and barriers as interfaces for flow in porous media. Report 4848, Inria, Rocquencourt, France, 2003.

[63] J. Jaffré, V. Martin, et J. Roberts. Modelling fractures and barriers as interfaces for flow in porous media. *Siam Journal for Scientific Computations*, 2004. Accepted for publication.

[64] J. Jaffré, J. Roberts, et V. Martin. Generalized cell-centered finite volume methods for flow in porous media with fault. In R. Herbin et D. Kröner Eds, editors, *Proc. 3rd Int. Symposium Finite Volumes for Complex Applications, Porquerolles, France, June 2002*, pages 357–364, 2002.

[65] E.F. Kaaschieter. Mixed finite elements for accurate particle tracking in saturated groundwater flow. *Advances in Water Resources*, 18 :227–294, 1998.

[66] Andrew Koenig et Barbara E. Moo. *Accelerated C++ : practical programming by example*. Adison-Wesley, Massachussets, 2000.

[67] Patrick Le Tallec. Domain decomposition methods in computational mechanics. *Comput. Mech. Adv.*, 1(2) :121–220, 1994. ISSN 0927-7951.

[68] P.L. Lions. On schwarz alternating method 3 : A variant for nonoverlapping subdomains. In *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 202–223. SIAM, 1989.

[69] Martial Mancip. *Couplage de méthodes numériques pour les lois de conservation. Application au calcul de l'injection.* PhD thesis, INSA Toulouse, Laboratoire MIP, Haute Garonne, France, 2001. In French.

[70] Martial Mancip. Personnal communication, 2003.

[71] Jan Mandel. Balancing domain decomposition. *Comm. Numer. Methods Engrg.*, 9(3) :233–241, 1993. ISSN 1069-8299.

[72] Jan Mandel et Marian Brezina. Balancing domain decomposition for problems with large jumps in coefficients. *Math. Comp.*, 65(216) :1387–1401, 1996. ISSN 0025-5718.

[73] Molière. *Le bourgeois gentilhomme*. Larousse, Paris, 1970. In French.

[74] R. Morel. Décomposition de domaine pour l'étude des écoulements en milieu poreux. Rapport de DESS, U. Paris VI Pierre et Marie Curie, Paris, France, 2001.

[75] R. L. Naff, T. F. Russell, et J. D. Wilson. Shape functions for velocity interpolation in general hexahedral cells. *Comput. Geosci.*, 6(3-4) :285–314, 2002. ISSN 1420-0597. Locally conservative numerical methods for flow in porous media.

[76] R.L. Naff, T.F. Russell, et J.D. Wilson. Test functions for three-dimensional control-volume mixed finite-element methods on irregular grids. In et al. Bentley, L.R., editor, *Proceedings of the XIII Int. Conf. on Computational Methods in Water Resources, Calgary, Alberta, Canada*, pages 667–684, 2000. URL `citeseer.nj.nec.com/311491.html`.

[77] R.L. Naff, T.F. Russell, et J.D. Wilson. Shape functions for three-dimensional control-volume mixed finite-element methods on irregular grids. In S.M. et al. Hassanizadeh, editor, *Proceedings of the XIV Int. Conf. on Computational Methods in Water Resources, Delft, The Netherlands*, pages 359–366, 2002.

[78] F. Nataf. Quasi optimal interface conditions in domain decomposition methods. application to problems with extreme contrasts in the coefficients. Report 514, Ecole Polytechnique, Palaiseau, France, 2003.

[79] J.-C. Nedelec. A new mixed finite elements in R^3. *Numer. Math.*, 50 : 57–81, 1986.

[80] G. Pépin, B. Vialay, et D. Perraud. Cahier des charges relatif à la réalisation de calculs de transport de radionucléides avec le code castem2000. Cahier des charges, Andra, Châtenay-Malabry, France, March 2001. In French.

[81] Alfio Quarteroni et Alberto Valli. *Domain decomposition methods for partial differential equations.* The Clarendon Press Oxford University Press, New York, 1999. ISBN 0-19-850178-1. Oxford Science Publications.

[82] P.-A. Raviart et J.-M. Thomas. A mixed finite element method method for second order elliptic problems. In I. Galligani et E. Magenes, editors, *Mathematical Aspects of Finite Element Methods ; Lecture Notes in Mathematics 606*, pages 292–315. Springer-Verlag, Berlin, 1977.

[83] J.E. Roberts et J.-M. Thomas. Mixed and hybrid methods. In P.G. Ciarlet et J.L. Lions, editors, *Handbook of Numerical Analysis Vol.II*, pages 523–639. North Holland, Amsterdam, 1991.

[84] F.X. Roux, F. Magoulès, S. Salmon, et L. Series. Optimization of interface operator based on algebraic approach. In O. Widlund I. Herrera, D. Keyes et R. Yates, editors, *Fourteenth International Conference on Domain Decomposition Methods in Science and Engineering (Cocoyoc, Mexico, 2002)*, pages 297–304. UNAM, Mexico City, Mexico, 2003.

[85] F.X. Roux, F. Magoulès, et L. Series. Approximation of optimal interface boundary conditions for two-Lagrange multiplier FETI method. In *Fifteenth International Conference on Domain Decomposition Methods in Science and Engineering (Berlin, Germany, 2003)*. 2004. To be published.

[86] Y. Saad et M. H. Schultz. GMRES : a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scient. Statist. Comput.*, 7(3) :856–869, 1986.

[87] L. Saas, I. Faille, F. Nataf, et F. Willien. Domain decomposition with robin interface conditions on non-matching grids using finite volume method. In R. Herbin et D. Kröner Eds, editors, *Proc. 3rd Int. Symposium Finite Volumes for Complex Applications, Porquerolles, France, June 2002*, pages 243–250, 2002.

[88] Enrique Sánchez-Palencia. Problèmes de perturbations liés aux phénomènes de conduction à travers des couches minces de grande résistivité. *J. Math. Pures Appl. (9)*, 53 :251–269, 1974.

[89] H. A. Schwarz. Gesammelte mathematische abhandlungen. *Springer*, 2 : 133–143, 1890. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp. 272-286.

[90] P. Siegel, R. Mosé, Ph. Ackerer, et J. Jaffre. Solution of the advection-diffusion equation using a combination of discontinuous and mixed finite elements. *Internat. J. Numer. Methods Fluids*, 24(6) :595–613, 1997. ISSN 0271-2091.

[91] Barry F. Smith, Petter E. Bjørstad, et William D. Gropp. *Domain decomposition*. Cambridge University Press, Cambridge, 1996. ISBN 0-521-49589-X. Parallel multilevel methods for elliptic partial differential equations.

[92] Bjarne Stroustrup. *The C++ programming language, 3rd edition*. Adison-Wesley, Reading, Massachussets, 1997.

[93] Patrick Le Tallec, Yann-Hervé De Roeck, et Marina Vidrascu. Domain-decomposition methods for large linearly elliptic three dimensional problems. *J. of Computational and Applied Mathematics*, 34, 1991. Elsevier Science Publishers, Amsterdam.

[94] Ray S. Tuminaro, Mike Heroux, Scott A. Hutchinson, et John N. Shadid. Official Aztec user's guide, version 2.1. Technical Report SAND99-8801J, Sandia, 1999.

[95] Van der Vorst H. A. Bi-cgstab : A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13 :631–644, 1992.

[96] Pierre Weis. Caml. Software and documentation available on the Web, `http ://caml.inria.fr/`, 1996.

[97] Pierre Weis et Xavier Leroy. *Le langage Caml, 2ème édition*. Dunod, Paris, 1999. In French.

[98] Barbara I. Wohlmuth. *Discretization methods and iterative solvers based on domain decomposition*. Springer-Verlag, Berlin, 2001. ISBN 3-540-41083-X.

[99] I. Yotov. *Mixed Finite Element Methods for Flow in Porous Media*. PhD thesis, TICAM, University of Texas at Austin, 1996.

Résumé :
Cette thèse traite des simulations multidomaines des écoulements en milieu poreux. Trois aspects sont abordés. Une étude est d'abord menée concernant une méthode de décomposition de domaines avec des maillages non-raccordés, utilisant des conditions d'interface de type Robin, pour les éléments finis mixtes. D'autre part, la méthode est implémentée en parallèle à l'aide du système parallèle OcamlP3l, écrit par des informaticiens dans le langage Ocaml. En OcamlP3l, l'utilisateur programme et débogue en séquentiel, puis obtient le code parallèle par une simple recompilation. Enfin, nous présentons un nouveau modèle d'écoulement dans un mileu poreux contenant de grandes fractures, avec des perméabilités très grandes et/ou très faibles. Dans ce modèle, les fractures sont traitées comme des interfaces entre sous-domaines. Une analyse théorique prouve l'existence et l'unicité de la solution, fournit une estimation d'erreur, qui est confirmée par des tests numériques.
Mots clés : écoulement en milieu poreux, simulation multidomaine, éléments finis mixtes, décomposition de domaines non-conforme, conditions d'interface de type Robin, forts contrastes de coefficients, calcul parallèle, Ocaml, OcamlP3l, fractures, failles, barrières.

Abstract :
This thesis is mainly concerned with the multidomain simulations of flow in porous media. Three different themes are considered. First, we study a domain decomposition method with non-matching meshes using Robin type interface conditions, for the mixed finite elements. Second, this method is implemented in parallel using the parallel system OcamlP3l, written in Ocaml by computer scientists. In OcamlP3l, the user develops and debugs sequentially, and obtains the parallel code with a mere recompilation. A realistic 3D simulation is given to validate the procedure. Finally, we present a new model for flow in a porous medium containing large fractures that may have very large and/or very small permeabilities. In this model, the fractures are treated as interfaces between subdomains. Existence and uniqueness of the solution is proved and an error estimate is obtained. Some numerical experiments show the quality of the results.
Keywords : flow in porous media, multidomain simulation, mixed finite elements, non-conforming domain decomposition, Robin type interface conditions, extreme constrasts of coefficients, parallel computations, Ocaml, OcamlP3l, fractures, faults, barriers.