N° d'ordre : 4493 ANNÉE 2011

UNIVERSITÉ DE RENNES 1

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

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Baptiste POIRRIEZ

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Systèmes Aléatoires
Composante universitaire ISTIC

Étude et mise en œuvre d'une méthode de sous-domaines pour la modélisation de l'écoulement dans des réseaux de fractures en 3D

Thèse soutenue à Rennes le 20 decembre 2011

devant le jury composé de :

Jean ROBERTS
Directrice de recherche, INRIA, Rocquencourt
Présidente

Patrick AMESTOY
Professeur, INPT-ENSEEIHT-IRIT, Toulouse
Rapporteur

Frédéric NATAF
Directeur de recherche, UPMC, Paris
Rapporteur

Jean-Raynald de DREUZY
Chargé de recherche avec HDR, CNRS, Rennes
Examinateur

Christian PEREZ
Directeur de recherche, INRIA, Lyon
Examinateur

Jocelyne ERHEL
Directrice de recherche, INRIA
Directrice de thèse

Remerciements

Je tiens à remercier tous ceux qui, au cours de ces années de thèses, m'ont soutenu. Mes remerciements vont en premier lieu à Jocelyne Erhel. Je ne lui ai pas rendu la vie facile tous les jours, mais par ses qualités humaines et scientifiques, elle a su me pousser à donner le meilleur de moi même. Je remercie aussi chaleureusement les membres, passés et présents, de l'équipe SAGE. Bernard Philippe, Édouard Canot, Julia Charrier, Mohamad Muhieddine, Souhila Sabit, Mestapha Oumouni, Sinda Khalfallah, Géraldine Pichot, Nadir Soualem, Caroline de Dieuleveult et Aurélien Le Gentil m'ont tous, d'une façon ou d'une autre, par leur aide technique, scientifique ou simplement leur amitié au quotidien, permis d'aller au bout de ce projet. Ils m'ont aidé à dépasser les moments de doute et de découragement. En particulier Mohamad et Géraldine, qui m'ont supporté comme voisin de bureau.

Ces travaux m'ont aussi permis de découvrir l'hydrogéologie. Grâce aux membres de Géosciences Rennes avec qui nous avons travaillé, cette découverte fut riche et pédagogique. Je tiens à remercier particulièrement Jean-Raynald de Dreuzy. Il m'a lui aussi, par ses questions scientifiques et ses suggestions, poussé vers l'avant.

D'une façon générale, j'ai eu la chance d'être accueilli dans un projet dont les membres ont su rendre le travail en équipe agréable. J'ai ainsi eu, pendant quatre ans, la chance d'avoir des conditions de travail exceptionnelles. Les relations humaines de qualité ont construit ce cadre propice au développement scientifique et personnel. Je me rends compte jour après jour, de la chance que cela représente.

Je ne peux non plus oublier les enseignants et collègues de l'ISTIC et de l'INSA de Rennes. J'ai pu, grâce à eux, découvrir le métier d'enseignant dans de bonnes conditions. Ils m'ont aidé à ne pas chuter lors de mes premiers pas face aux étudiants. Merci pour leurs conseils et leur amitié.

Je veux enfin remercier Patrick Amestoy et Frédéric Nataf pour leur relecture attentive de mon manuscrit et pour l'intérêt qu'ils ont montré pour mes travaux. Je remercie également Jean-Raynald de Dreuzy, Christian Perez et Jean Roberts d'avoir accepté de faire partie de mon jury de soutenance.

Cette thèse n'aurait pas vu le jour sans le soutien quotidien et indéfectible de mon épouse, Anne. Je sais que ce ne fut pas facile tous les jours d'être mariée à un thésard. Merci d'avoir cru en moi et en nous.

Table des matières

Table des figures

Liste des tableaux

Liste des notations

Paramètres de génération des réseaux de fractures :

α	Coefficient de la loi puissance de distribution des tailles de fractures
l	Longueur de la fracture considérée
d	Densité surfacique des fractures dans le domaine
L	Longueur des côtés du domaine

Numérotation des fractures et des systèmes associés :

Ω_f	Domaine délimité par la fracture f
A_{ii}	Bloc interne du sous-domaine i
$A_{00}^{(i)}$	Bloc intersection du sous-domaine i
S_i	Complément de Schur local associé au sous-domaine i
R_{i0}	Restriction de l'espace associé à toutes les intersections vers l'espace associé aux intersections du sous-domaine i

Taille des systèmes :

n_i	nombre d'arêtes interne dans le sous-domaine i
n_0	Nombre d'arêtes intersections dans le domaine complet,
$n_0^{(i)}$	Nombre d'arêtes intersections dans le sous-domaine Ω_i
n_{ii}	Nombre d'arêtes internes dans le sous-domaine Ω_i
n_i	Taille de la matrice B_i (nombre total d'arêtes dans le sous-domaine Ω_i)
n	Nombre de sous-domaines

Introduction

Contexte scientifique

Les ressources souterraines fournissent une part importante de l'eau douce nécessaire à nos sociétés. L'industrie et l'agriculture sont de forts consommateurs d'eau et ces activités présentent parfois un risque de polution. L'eau douce est par ailleurs nécessaire à l'homme, comme boisson et pour un usage domestique. C'est donc une ressource vitale qu'il faut protéger.

L'équipe SAGE est en lien avec une équipe de Géoscience Rennes pour étudier les phénomènes d'écoulement et de transport dans les milieux souterrains. La plate-forme logicielle H2OLab regroupe différents outils permettant de modéliser des transferts de fluides dans des milieux poreux et dans des réseaux de fractures. Les domaines d'applications des modèles étudiés sont, entre autres, la protection des nappes phréatiques, la gestion des ressources en eau souterraine et l'étude de stockage de déchets à longue durée de vie. Ces problèmes sont caractérisés par une forte hétérogénéité géologique et par la faible quantité de données précises sur la composition du domaine étudié.

Nous nous sommes intéressés plus particulièrement au problème de la simulation de l'écoulement dans les fractures et à la résolution du système linéaire associé. La présence de fractures dans le milieu change radicalement la perméabilité du terrain, les fractures étant des chenaux où l'écoulement se fait en priorité. La distribution et le positionnement des fractures ne peuvent pas être déterminés exactement. De même, la répartition des flux dans les chenaux ne peut être déduite d'une observation simple du terrain. Ces difficultés conduisent à l'utilisation de modèles stochastiques, où les données incertaines sont remplacées par des variables aléatoires suivant des lois établies à partir des observations de terrain. Les résultats obtenus par les simulations sont validés par une confrontation avec des données issues de mesures expérimentales (captage, étude de puits, ...). Ces simulations reposent sur des méthodes de quantification d'incertitudes, par exemple la méthode de Monte-Carlo. Il faut donc que les simulations puissent être réalisées de façon systématique, sans avoir à traiter des échecs liés à la génération du problème ou à sa résolution. Le nombre de simulations nécessaire pour une étude étant élevé, de l'ordre du millier, il faut que chaque simulation soit le plus rapide et le plus fiable possible. Un grand nombre d'échecs induisant une charge en calcul inutilisée, ces derniers doivent donc être évités au maximum.

Les réseaux de fractures discrets générés aléatoirement présentent une géométrie complexe qu'il faut mailler pour utiliser un schéma de discrétisation. Afin de pouvoir étudier des domaines de plus grande taille qu'avec une approche purement 3D, les fractures sont maillées en 2D, en s'appuyant sur leurs frontières et sur les intersections. Or les inter-

sections entre fractures peuvent créer des angles de faible ouverture et faire échouer un logiciel de génération de maillage. De plus, ces configurations dans les maillages pénalisent la résolution en rendant le système numériquement instable. Si aucune précaution n'est prise, la présence de ces angles fait ainsi échouer un grand nombre de simulations.

Plan

Une méthode originale de discrétisation géométrique a été mise au point dans l'équipe SAGE, permettant un maillage des fractures en 2D. Cette méthode utilise une discrétisation préliminaire en 3D. Les arêtes obtenues pour les frontières entre fractures sont ensuite projetées dans le plan des fractures, permettant de supprimer ces angles.

Mais cette étape doit être complétée par une phase de correction locale. En effet, la projection engendre, dans la majorité des cas, une géométrie 2D des fractures qui ne satisfait pas les conditions requises par le logiciel de maillage. Par exemple, le domaine à mailler peut ne pas être connexe. Nous avons mis en place une procédure afin de supprimer ces causes d'échecs. Une détection préalable des arêtes posant problème est réalisée, la géométrie est modifiée localement et le maillage est effectué.

Une fois cette méthode mise en place, nous l'avons validée numériquement. Nous avons vérifié, sur un grand nombre de cas tests, que la convergence du schéma n'était pas affectée par les corrections locales. Les étapes de discrétisation en 3D, de projection et de correction permettent de réaliser des simulations en grand nombre, malgré des conditions de maillage difficile. La génération du système, la correction et la validation de la méthode sont détaillées dans le chapitre 1.

Nous avons ensuite étudié comment se comportent des solveurs directs et itératifs lorsque la taille du problème augmente. Le solveur direct, basé sur une factorisation LU, a une complexité qui suit une loi puissance. Il cesse rapidement d'être compétitif et échoue, sur la machine utilisée, par manque de mémoire, à partir d'environ un million d'inconnues. Le solveur multigrille algébrique (AMG), est efficace mais instable. Dans certains cas il a une convergence lente, n'arrivant parfois pas à converger dans le nombre d'itérations imparti. Ces systèmes, qui ne sont pas prévisibles simplement avec une étude préalable du réseau, représentent 30% des cas. Le solveur itératif utilisant la méthode du gradient conjugué préconditionné (PCG) est stable. Il a une complexité linéaire en fonction de la taille des systèmes, mais c'est le plus lent des trois solveurs testés, lorsque AMG ou le solveur direct sont efficaces. Cette étude est présentée dans le chapitre 2.

Afin de dépasser les limites des solveurs précédents, nous avons étudié une technique de décomposition en sous-domaines. Les fractures présentant naturellement des connexions de type interface, c'est une méthode du complément de Schur primal que nous avons choisie. Cette méthode consiste à résoudre le problème à l'interface des sous-domaines puis à étendre la solution sur le domaine complet. Elle utilise un solveur direct sur les problèmes internes aux sous-domaines, qui sont des problèmes de petite taille, et un algorithme itératif de Gradient Conjugué pour résoudre le problème à l'interface. Cette méthode hybride permet ainsi de cumuler les atouts des deux approches, en alliant rapidité de la résolution sur les petits systèmes et robustesse de la convergence.

Le système à l'interface est préconditionné en utilisant le préconditionnement de Neumann-Neumann, qui résoud des systèmes locaux à chaque sous-domaine. Le solveur qui sert à effectuer les résolutions locales sur chaque sous-domaine utilise une factorisation de Cholesky du système local. Une approche originale de cette factorisation, exploitant la structure en bloc de la matrice associée au système local, permet de gagner en temps et en mémoire lorsque le nombre de sous-domaines augmente. L'augmentation du nombre de

sous-domaines permet par contre de réduire la taille des systèmes locaux et d'augmenter le parallélisme. Lorsque le nombre de sous-domaines augmente, le Gradient Conjugué préconditionné converge par contre plus lentement. Il est donc nécessaire de pouvoir dépasser la limite inhérente à Neumann-Neumann, ce qui peut se faire avec un préconditionnement global. Nous avons choisi une méthode de déflation basée sur un sous-espace, construit à partir de la définition des sous-domaines. Cette méthode de Schur avec préconditionnements est étudiée d'un point de vue théorique dans le chapitre 3. Le chapitre 4 présente la mise en œuvre de cette méthode. Enfin, le chapitre 5 présente son application aux réseaux de fractures. Ce chapitre contient aussi une comparaison entre les solveurs présentés au chapitre 2 et la méthode du complément de Schur.

Chapitre 1

Simulation de l'écoulement dans les réseaux de fractures discrets

Comme on a pu le voir dans l'introduction, il n'est pas possible de faire des observations directes sur les milieux fracturés. Les données que nous pouvons obtenir sont donc limitées qualitativement et quantitativement. Afin de contourner ce problème, nous utilisons une approche stochastique. Il est donc nécessaire de pouvoir réaliser de multiples simulations de façon systématique. Si l'on se contente de mailler les ellipses modélisant les fractures, le grand nombre d'intersections entre fractures donne un nombre important d'angles de faible ouverture. Si rien n'est fait pour éviter ces angles, certaines simulations, en nombre non négligeable, échouent à l'étape du maillage. Dans d'autres cas, le maillage généré est de mauvaise qualité et le résultat de la simulation très imprécis. Comme on cherche à effectuer une analyse stochastique, il n'est pas possible de voir un nombre important de simulations échouer.

Nous utilisons un modèle où chaque fracture Ω_f est représentée par une ellipse. Le champ de perméabilité 2D est fixé et est noté K. Les inconnus sont la charge hydrolique h et la vitesse de Darcy v. Le domaine considéré est un cube, avec des conditions aux limites du perméamètre classique. Les bords latéraux (Γ_N) sont des frontières de Neumann homogènes et les bords supérieur et inférieur (Γ_D) sont des frontières de Dirichlet. Les ellipses qui intersectent les bords du domaine sont tronquées. Nous considérons que l'écoulement se situe uniquement dans les fractures, la matrice étant considérée comme imperméable, ce qui se traduit par des conditions de Neumann homogènes sur les contours (Γ_f) non tronqués des fractures. De plus, pour assurer la cohérence au niveau des intersections, nous imposons la continuité des flux et de la charge sur les intersections entre fractures. Nous obtenons au final le système d'équations suivant :

$$\begin{aligned} v &= -K\nabla h \text{ dans } \Omega_f, \\ \nabla.v &= s \text{ dans } \Omega_f, \end{aligned}$$

Les conditions aux limites et celles de continuité s'expriment sous la forme :

$$\begin{aligned} h &= h_D \text{ sur } \Gamma_D, \\ v.n &= q_N \text{ sur } \Gamma_N, \\ v.n &= 0 \text{ sur } \Gamma_f, \\ h_{k,f} &= h_k, \text{ sur } S_k, \forall f \in F_k, \\ \sum_{f \in F_k} v_{k,f}.n_{k,f} &= 0 \text{ sur } S_k \end{aligned}$$

où S_k est un segment d'intersection et F_k l'ensemble des fractures ayant S_k en commun.

Le système linéaire correspondant est obtenu en utilisant une méthode des éléments finis mixtes hybrides. Lors de la modélisation des réseaux de fractures, il est donc nécessaire de limiter le nombre de triangles ayant une faible ouverture. Afin de garantir ce faible nombre, une méthode utilisant une projection préalable sur une grille tri-dimensionnelle a été développée et est présentée dans la thèse de H. Mustapha [54, 55]. Les bords des fractures et les intersections entre fractures sont discrétisés sur une grille régulière en 3D. Pour chaque fracture, les centres des voxels correspondant sont projetés dans le plan de la fracture. Ces points servent de base pour déterminer le maillage. Les intersections et les limites des fractures sont ainsi discrétisées sous forme "d'escalier" dans les plans des fractures, assurant l'absence de petits angles entre ces différents éléments.

Cette méthode donne de bons résultats mais présente un inconvénient majeur : après la phase de projection, il arrive que certaines arêtes de bord de fracture soient connectées à plus de deux arêtes. Le réseau obtenu n'est alors pas maillable, ou certaines matrices élémentaires peuvent être singulières. Cette situation est suffisamment fréquente pour poser problème lors d'une étude stochastique. Il faut effectuer une correction. Ce chapitre présente les corrections que nous avons dû mettre en place ainsi que les résultats que nous avons obtenus.

Dans la première partie, nous aborderons plus en détail le problème hydro-géologique étudié et le modèle physique associé. Nous présenterons ensuite les résultats que nous avons obtenus en faisant varier plusieurs paramètres de génération (Tab : 1.3, page 20), montrant ainsi la stabilité de la méthode.

Il s'agit d'un article paru dans la revue Siam Journal on Scientific Computing [30].

FLOW SIMULATION IN THREE-DIMENSIONAL DISCRETE FRACTURE NETWORKS

Jocelyne Erhel, Jean-Raynald de Dreuzy, and Baptiste Poirriez

SIAM J. SCI. COMPUT. Vol. 31, No. 4, pp. 2688–2705

Abstract. In fractured rocks, fluid flows mostly within a complex arrangement of fractures. Both the fracture network structure and its hydraulic properties are determined at first order by the broad range of fracture lengths and densities. To handle the observed wide variety of fracture properties and the lack of direct fracture visualization, we develop a general and efficient stochastic numerical model for discrete fracture networks (DFNs) in a three-dimensional (3D) computational domain. We present an original conforming mesh generation method addressing the penalizing configurations stemming from close fractures and acute angles between fracture intersections. Flows are subsequently computed by using a mixed hybrid finite element (MHFE) method. The lack of direct fracture knowledge is treated by Monte-Carlo simulations requiring simulations with a large number of networks with various characteristics. We analyze the complexity in size and in time for the computation of flow in 3D DFNs meshed with our method and compare with the complexities for 2D rectangular domains meshed with a regular grid. We find out that complexity in size is similar whereas complexity in time is slightly larger for DFNs than for 2D regular domains.

Key words. discrete fracture network (DFN), mesh generation, mixed hybrid finite element method (MHFE)

DOI. 10.1137/080729244

1.1 Introduction

Numerical simulations in hydrogeology aim at finding the solution of the flow (and transport) equations in a complex, only partly identified, geologic system. Here we consider rocks where groundwater flow is channeled in fractures. Our model is based on discrete fracture networks (DFNs) and follows a stochastic approach.

1.1.1 Motivation

The interest of fractured rocks for groundwater flow has grown in the last two decades in several respects, ranging from the storage of high-level nuclear wastes to water resources [19]. As fractures act as fast flow conduits, they determine the medium hydraulic properties like the flow pattern and the permeability. For rocks of very small permeability like crystalline rocks, the sole flow bearing structures are the fractures. Flow properties may be dominated by a few large fractures, by a dense network of small fractures, or by a combination of fractures of very different sizes [23], [24].

The nature of the flow pattern depends directly on the fracture length distribution and on the density of fractures. For dense fracture networks, flow will be well spread in the medium and could be advantageously modeled by continuous porous-like methods where fractures are implicitly taken into account by the permeability repartition of the equivalent porous medium. For sparse fracture networks, flow will be more likely channeled in the most permeable and accessible fractures and could be better modeled by discrete approaches

where fractures are explicitly represented [45]. Flow properties depend also on the spatial variability of fracture aperture, which can be represented by a discrete approach. This method also provides ways to study flow and transport processes by numerical simulations [57] and may in turn give upscaling rules for permeability measurements, the flow pattern salient characteristics, and indications on hydraulic properties that cannot be directly observed on field tests.

Natural fractures occur on a broad range of scales. Because fractures cannot be observed in 3D, information on their characteristics comes from boreholes and outcrops. To address uncertainty in DFNs, stochastic approaches have been developed. The fracture shape is generally modeled by ellipses or polygons of varying aspect ratios [16], [46], [59]. Variable permeability can be modeled by a random distribution. We present in this paper a stochastic discrete approach working for a wide variety of fracture networks and adapted for both field and phenomenological studies.

1.1.2 Previous work

DFNs have a rather complex 3D geometry, since they are not 3D volumes but a set of 2D domains, with various orientations and intersecting each other. In flow computations, the rock matrix can be considered as impervious and flow is only simulated in the fractures. Simulation methods based on finite element methods face two problems. First, the number of cells in the mesh grows rapidly with the system size like in 3D models. Second, the 2D domains corresponding to the fractures display connection configurations difficult to mesh like small angles and points close to each other. There are two scales to consider in DFN : the 3D scale of the network and the 2D scale of each fracture. Two types of simulation methods were developed using either the two-scale structure in 2D fractures and 3D network fractures or a mesh generation of the full system at the network scale.

The first developed method consists of obtaining analytical relations between flows and heads within disk-shaped fractures through image theories, and of combining these analytical relations in a system of equations, giving the heads at the network scale [46]. Simpler approaches rely also on the two-scale fracture network structure. The mesh structure at the fracture scale is simplified by a network of monodimensional pipes between the fracture intersections and the fracture center, and the mesh structure at the network scale becomes a network of monodimensional pipes [16], [59]. This method solves the mesh generation and system size problems and respects the topological structure of the network. Its drawbacks are potential unrealistic flow patterns in the fracture, the difficult choice of the pipe transmissivities, and the impossibility to evaluate the uncertainties. Definition of pipe locations and transmissivities within the fractures has been more precisely quantified with use of flow solutions with the boundary element method within the fracture plan [25]. In this method, like in the method of [46], flow solving at the fracture scale provides relations between flows and heads that are used in a second step at the network scale. The boundary element method is, however, limited to homogeneous fractures and has only been applied to small fracture networks so far.

The second kind of simulation method consists of generating a mesh at the network scale and in using a finite element method for solving the flow equation. This strategy is implemented in several types of software like Rockflow and Fracman. It provides ways to introduce transmissivity heterogeneities within the fractures and to quantify uncertainties by reducing the characteristic mesh scale. Several solutions have been proposed for solving the problem of the connection configurations difficult to mesh or detrimental to numerical schemes. These configurations have been solved manually in each network through local

modifications of the mesh [44], or removed from the network by small modifications of the fracture network structure [51].

1.1.3 Our contribution

The latter approach can be integrated in a stochastic framework where several samples of DFNs are automatically generated and follow prescribed probability distributions. To our knowledge, most existing models do not allow broad fracture length distributions and broad fracture shape distributions occurring in natural fracture networks. In this article, we present a new flow simulation method based on an original mesh generator for general DFN structures.

Network generation is the first step of a global stochastic methodology. An original feature is the possibility to apply power-law fracture length distributions with a large range of possible exponents, correctly modeling both the natural broad range of scales and the variety of natural fractured sites and leading to a large number of fractures. It is also possible to define an heterogeneous permeability field, in order to take into account the spatial variability of fracture apertures.

In this paper, we focus on simple steady state flow. We assume that the rock matrix is impervious ; classical equations are written in each fracture. In contrast to [55], we write also interface conditions for all intersections, in order to get a consistent system of equations. To solve these equations, a mixed finite element method can be used, where the difficulty is to express fluxes at the intersections. We use a hybrid method, different from [51], [72], but following [42], [43], where we replace fluxes by pressures at the edges. This approach is a very convenient and easy way to express mass continuity at the interfaces.

However, this method relies on mesh generation, providing conforming mesh structures. In general, a direct application of mesh generation does not succeed, because each fracture contains many intersections in all directions. In most cases, the quality of the mesh is poor, with small angles in the triangles [11], [55]. The objective of our mesh generation is to remove difficult connection configurations in the fractures. The idea is to apply a two-level discretization of the fracture intersections and boundaries. Intersections and boundaries are first referenced on a 3D regular grid of voxels (small cubes) common to all fractures. Then voxels to which elements of a fracture belong are projected back on the fracture plane. This discretization-projection of intersections and boundaries removes small angles and small segments so that fractures can be meshed with a high quality [11], [55]. However, for many generated networks, some boundary points are connected to more than two edges. In order to run stochastic simulations, a post-projection step is mandatory in order to remove these points.

When a mesh can be generated for many DFNs, it is possible to run many numerical experiments. In contrast to [55], we generate tests in a systematic way, using a wide range of parameters, several samples of Monte-Carlo simulations, and a wide range of mesh steps. These numerous tests are used to validate the method, to analyze convergence, and to study space and time complexity.

The paper is organized as follows. Section 1.2 describes the physical and numerical model. Then we develop our method to generate the mesh in section 1.3, describing our two-level process and our post-projection adjustments. Section 1.4 is devoted to software description and section 1.5 gives some experimental results.

TABLE 1.1 – Random distributions for 3D discrete fracture networks. A random fracture is defined by four random variables : length, shape, position, and orientation.

Characteristic	Random distribution
length	power law
shape	uniform or constant
position	uniform
orientation	uniform

1.2 Model

Our model can be decomposed in three parts, related to three scientific fields : geology, hydraulics, and numerical analysis. We present these three components below.

1.2.1 Geological characteristics

The model presented in this study is made up of ellipses identified by their length, shape, orientation, and position. The length is equal to their major axis and the shape noted e is defined by the ratio between their major axis and minor axis. The stochastic model addresses the issue of uncertainty in the data by generating fracture networks where the characteristics follow given probability distributions. The broad natural fracture length distribution can be correctly modeled by a power-law distribution such as

$$p(l)dl = \frac{1}{a-1}\frac{l^{-a}}{l_{\min}^{-a+1}}dl, \tag{1.1}$$

where $p(l)dl$ is the probability of observing a fracture with a length in the interval $[l, l+dl]$, l_{min} is the smallest fracture length, and α is a characteristic exponent [13], [22]. Values of the power-law length exponent α extrapolated from outcrop observations range between 2.5 and 5, traducing a wide variety of network structures. For α below 3, the network structure is dominated by the few largest fractures, whereas for α larger than 4, it is dominated by the smallest fractures. This original feature of our model leads to configurations with a large number of fractures spanning a large range of lengths. Without loss of generality, the fracture orientation and position distributions are taken as uniform, and ellipse shapes are either uniformly distributed or constant. The stochastic model is summarized in Table 1.1.

For simulation purposes, we introduce a cubic computational domain of characteristic size L in which the fracture networks are studied. The largest fracture length is of the order of L and the smallest fractures are of length close to l_{min}. The important parameter in the simulations is the scale range $[l_{min}, L]$ characterized by the scale ratio L/l_{min}.

The global computational domain of the model is the network Ω, an open set composed of all the fractures and all the intersections between fractures. The boundary of Ω is composed of the borders of the cube and the edges of the ellipses. Let NF be the number of fractures. Each fracture is an open set $\Omega_f, f = 1, \dots NF$, an ellipse which is truncated if necessary by the faces of the cube. The boundary of a fracture is composed of the ellipse edge, the intersections with the cube faces, and the intersections with other fractures. Let NI be the number of intersections between fractures. Each intersection is a segment $S_k, k = 1, \dots NI$, and we denote by F_k the set of fractures with S_k on the boundary. Thus,

TABLE 1.2 – Parameters for 3D discrete fracture networks : exponent of power-law distribution, density of fractures, and scale ratio.

Parameter	Meaning	Range
α	power law	$[2.5,\ 5]$
d	density	$d \geq 1$
$L/l_{\min}$	scale ratio	$L/l_{\min} \geq 1$

we can write

$$\Omega = \bigcup_{f=1}^{NF} \Omega_f \cup \bigcup_{k=1}^{NI} S_k.$$

Fracture density is described by reference to the percolation threshold characterized by the critical number of fractures NF_c for which connection between the cube borders is reached. For power-law distributed fracture length, the critical density depends on the system size L [14], [15]. As this study focuses on connected fracture networks for flow simulations, we define the density as the ratio $d = NF/NF_c$. For example, $d = 2$ means that the system contains twice as many fractures than at threshold. Parameters of the model are summed up in Table 1.2.

Fracture characteristics have a critical influence on the fracture network topology. For example, the number of intersections NI increases with the fracture density d and decreases with the power-law exponent α. For ensuring a large applicability of our method, we cover in experiments a wide range of parameters, leading to 18 types of networks. Figure 1.1 displays four examples of networks used in this study.

1.2.2 Flow equations

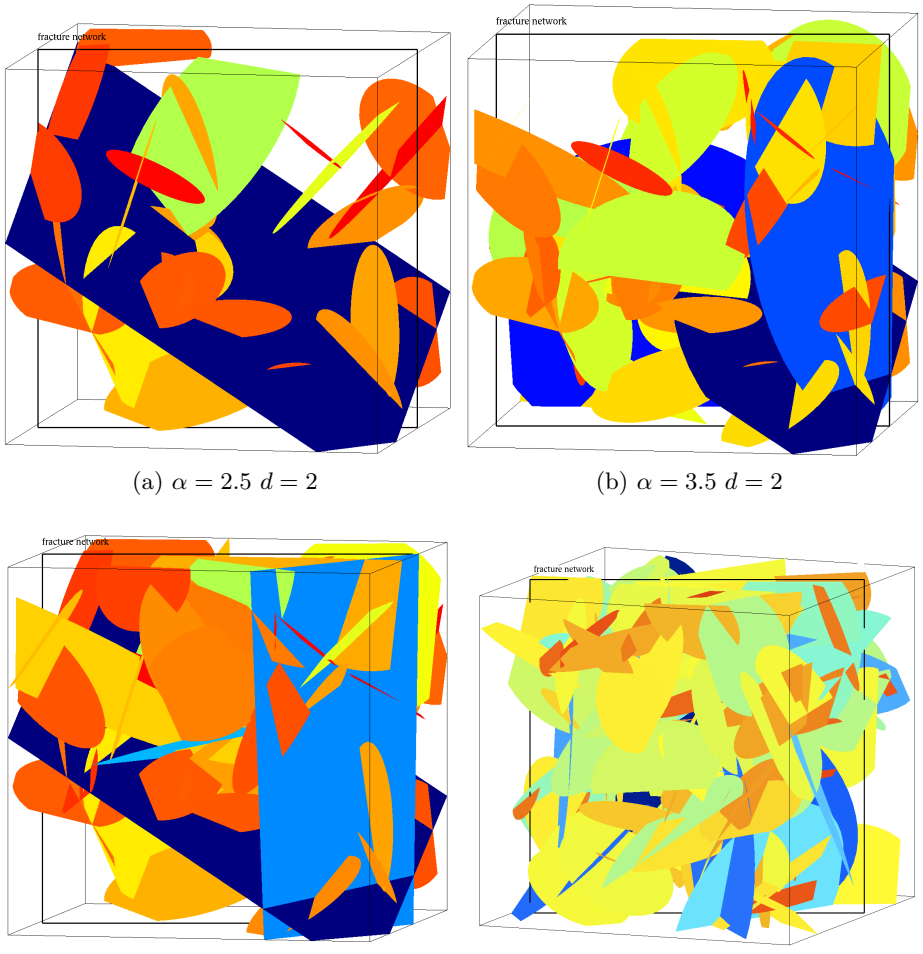
Classical laws governing the flux in a porous medium are mass conservation and Darcy law. The main physical data is the permeability of the geological domain. In our model, the rock matrix surrounding the fractures is considered as impervious. Permeability of fractures can be highly variable. Indeed, they are closely related to apertures which vary spatially and can vary in time. In our current model, we consider a deterministic heterogeneous permeability field. In the future, we plan to consider also a random permeability field.

Since the rock matrix is impervious, we assume that mass conservation and Darcy laws are satisfied in each fracture, and we have to define conditions at the intersections. We first assume that there is no longitudinal flux in the intersections. If longitudinal flux is allowed, different interface conditions must be defined [50]. Then, we assume the continuity of the hydraulic head and the transversal flux at each intersection. These assumptions are quite classical and close the system of equations [72].

In order to write the laws in each fracture, we must define a projection from the 3D cube onto the fracture plane. Let (x, y, z) be the coordinates in the 3D cube and (x_f, y_f) the projected coordinates in the fracture Ω_f. The governing laws in each fracture are written

$$\begin{cases} v = -K\nabla h \text{ in } \Omega_f, \\ \nabla . v = s \text{ in } \Omega_f, \end{cases} \tag{1.2}$$

where the Darcy velocity v and the hydraulic head h are unknown, K is a given 2D permeability field, and s is a given source term. These equations are written in the fracture plane, using the 2D coordinates (x_f, y_f).



(a) $\alpha = 2.5$ $d = 2$ (b) $\alpha = 3.5$ $d = 2$

(c) $\alpha = 2.5$ $d = 3$ (d) $\alpha = 4.5$ $d = 3$

FIGURE 1.1 – Examples of networks with various parameters : power-law exponent, density.

We assume that boundary conditions on the cube faces are either Dirichlet or Neumann. Let Γ_D be the part of the cube boundary with Dirichlet condition ($\Gamma_D \neq \emptyset$) and Γ_N be the part with Neumann condition. We also assume a Neumann zero flux condition on each ellipse edge Γ_f of fractures Ω_f.

Boundary conditions are written

$$\begin{cases} h = h_D \text{ on } \Gamma_D, \\ v.n = q_N \text{ on } \Gamma_N, \\ v.n = 0 \text{ on } \Gamma_f, \end{cases} \tag{1.3}$$

where h_D and q_N are prescribed values.

Continuity conditions in each intersection are written

$$\begin{cases} h_{k,f} = h_k, \text{ on } S_k, \forall f \in F_k, \\ \sum_{f \in F_k} v_{k,f}.n_{k,f} = 0 \text{ on } S_k, \end{cases} \tag{1.4}$$

where $h_{k,f}$ is the trace of the head and $n_{k,f}$ is the normal unit vector on the boundary S_k of the fracture Ω_f.

We assume that the set of equations (1.2)–(1.4) is well-posed in adequate functional spaces (it has a unique solution which depends continuously on the data).

1.2.3 Numerical method

Now, we have to define a spatial approximation in order to solve the physical model. We apply a mixed finite element (MFE) method for the following reasons. In general, finite element methods allow one to deal with complex geometries and to use locally refined meshes. In a mixed finite element method, the unknowns are hydraulic head and velocity ; thus the velocity field is a good approximation, and useful for subsequent transport problems. Also this method guarantees both local and global mass conservation ; therefore, MFE methods are very well-suited for solving flow problems. We do not define here the functional spaces and the variational weak mixed formulation, but refer to [51], [42], [43] for more details.

The first step is to define the mesh of the network. We require that the mesh is 2D in each fracture, in order to guarantee mass conservation in each fracture plane. We also rule that the mesh is conforming in each intersection. This means that the discretization of an intersection and of the boundaries is uniquely defined and that the discrete intersection is projected in the fracture plane of all fractures containing it. This constraint allows an easy definition of the discrete continuity conditions and allows us to apply classical results for MFE methods.

At the network level, boundaries and intersections are uniquely discretized with 3D coordinates. Then they are projected onto each fracture plane with 2D local coordinates, and each fracture can be meshed in 2D.

We assume that the 2D meshes are done with triangles. We use the hybrid version of the MFE method (MHFE method) where unknowns are mean head h_E in each triangle and mean head Φ_I on each edge [42]. For stationary flux equations, the MHFE method leads to a symmetric positive definite matrix, whereas the MFE method leads to a symmetric indefinite matrix. Also, in the context of our network of fractures, it is easy to handle intersection conditions with the MHFE method. Indeed, continuity of head-on intersection is trivial (only one unknown per edge) ; continuity of flux on intersection is easily accomplished by eliminating the velocity variable. In particular, it is possible and also sufficient

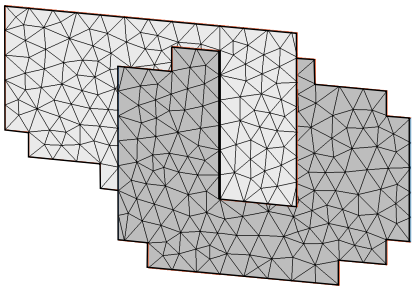


FIGURE 1.2 – Two intersecting fractures with a triangular mesh : border edges, interior edges, and intersection edges.

to define locally in each element a discrete velocity field, whereas global unknowns are hydraulic heads h_E and Φ_I.

Let NT be the number of triangles and NE be the number of edges. For each triangle E, the three edges are locally numbered $i = 1, 2, 3$ and globally numbered I. For each edge I, the set of elements E containing I is denoted by T_I. The number of elements containing I varies : an edge on a boundary (cube or ellipse) belongs to only one triangle ; an edge inside a fracture belongs to two triangles, and an edge on an intersection belongs generally to two fractures and four triangles. The different cases are illustrated by Figure 1.2, showing two fractures with their intersection.

We use the classical Raviart–Thomas basis functions $w_{E,j}$, defined in each 2D fracture and each triangle E, on each local edge j :

$$v_E = \sum_{j=1}^{3} v_{E,j} w_{E,j}.$$

Now, Dirichlet boundary conditions are given by $\Phi_I = h_D, I \in \Gamma_D$ and Neumann boundary conditions on the cube edge are given by $v_{E,i} = q_N, I \in \Gamma_N$, whereas Neumann boundary conditions on an ellipse edge are given by $v_{E,i} = 0, I \in \Gamma_f$, where E is the element containing I, with a local number i.

In order to eliminate the velocity and to derive a hybrid formulation, we write locally Darcy's law and express the velocity with hydraulic heads. Darcy's law $v = -K\nabla h$ is integrated $\nabla.v = s$ in each element E, using each basis function as a test function in the variational formulation ; thus we get

$$\int_E K^{-1} v.w_{E,i} = -\int_E \nabla h.w_{E,i}.$$

The first term can be rewritten

$$\int_E K^{-1} v.w_{E,i} = \sum_j B_{i,j}^{(E)} v_{E,j} \text{ with } B_{i,j}^{(E)} = \int_E K^{-1} w_{E,j}.w_{E,i}.$$

Using the Green formula and the properties of the basis functions, the second term is rewritten :

$$-\int_E \nabla h.w_{E,i} = \int_E h\nabla.w_{E,i} - \int_{\partial E} h w_{E,i}.n_{\partial E} = h_E - \Phi_I.$$

Finally, we get the local Darcy's law in each element :

$$v_E = C^{(E)}(h_E u - \Phi_E), \tag{1.5}$$

where v_E is now the vector of the three components $v_{E,j}$, where $C^{(E)} = B^{(E)-1}$, Φ_E is the vector composed of the hydraulic heads at the three edges, and $u = (1,1,1)^T$. The matrices $B^{(E)}$ and $C^{(E)}$ are symmetric positive definite.

The discrete mass conservation equation is also obtained by integrating in each element :

$$\int_E \nabla.v = \int_E s, \text{ thus } \sum_i v_{E,i} = s_E.$$

Now, using the local Darcy's law (1.5), we get the global discrete mass conservation equation

$$Dh - R\Phi = s, \tag{1.6}$$

where h is the vector of the unknowns h_E and Φ is the vector of the unknowns Φ_I ; the right-hand side s contains the source terms and the Dirichlet boundary conditions ; the matrix D is diagonal and defined by $D_E = \sum_{i,j} C^{(E)}_{i,j}$; the matrix R is given by

$$\begin{aligned} &R_{E,I} = 0 \text{ if } I \notin E, \\ &R_{E,I} = \sum_j C^{(E)}_{i,j} \text{ if } I \in E. \end{aligned}$$

The discrete mass conservation is also written through each edge ; the flux condition gives

$$\sum_{E \in T_I} v_{E,i} = 0, I \in \Omega_f; v_{E,i} = 0, I \in \Gamma_f; v_{E,i} = q_N, I \in \Gamma_N.$$

We deal now with intersection conditions (1.4). Since continuity of head is trivial here (the unknown Φ_I is unique on each conforming edge), only flux intersection conditions need to be written. Indeed, thanks to the conforming mesh, the mass conservation condition through an edge of a fracture is easily generalized through any edge of an intersection, and continuity of flux is written

$$\sum_{E \in T_I} v_{E,i} = 0, I \in S_k.$$

Using flux condition on any edge of a fracture or an intersection and using local Darcy's law (1.5), we get the global discrete Darcy's law :

$$-R^T h + M\Phi = q \tag{1.7}$$

with

$$\begin{cases} M_{I,J} = 0 \text{ if no element contains } I \text{ and } J, \\ M_{I,J} = C^{(E)}_{i,j}, \ I \neq J, I \in E, J \in E, \\ M_{I,I} = \displaystyle\sum_{E \in T_I} C^{(E)}_{i,i}, \\ q_I = q_N \text{ if } I \in \Gamma_N, \\ q_I = 0 \text{ otherwise.} \end{cases}$$

By eliminating h, using the diagonal matrix D in (1.6), we get the linear system

$$A\Phi = b, \tag{1.8}$$

where the Schur complement matrix α and the right-hand side b are given by

$$\begin{cases} A = M - R^T D^{-1} R, \\ b = q + R^T D^{-1} s. \end{cases}$$

The head is then computed by $h = D^{-1}(R\Phi + s)$ and the velocity by (1.5).
The matrix α and right-hand side b are detailed below :

$$\begin{cases} A_{I,J} = M_{I,J} - R_{I,E} D_E^{-1} R_{J,E}, I \neq J, I \in \partial E, J \in \partial E \\ A_{I,I} = M_{I,I} - \sum_{E \in T_I} R_{I,E} {D_E}^{-1} R_{I,E}, \\ b_I = q_I + \sum_{E \in T_I} R_{I,E} {D_E}^{-1} s_E. \end{cases}$$

The linear system can be assembled from local matrices and vectors computed in each fracture. Let

$$\begin{cases} A_{I,J}^{(f)} = A_{I,J}, I, J \in \Omega_f, \\ A_{I,I}^{(f)} = \sum_{E \in T_I \cap \Omega_f} (C_{i,i}^{(E)} - R_{I,E} {D_E}^{-1} R_{I,E}), I \in \Omega_f, \\ b_I^{(f)} = q_I + \sum_{E \in T_I \cap \Omega_f} R_{I,E} {D_E}^{-1} s_E, \end{cases}$$

then $A = \sum_f A^{(f)}$ and $b = \sum_f b^{(f)}$. Therefore, it is possible to compute matrices locally in each fracture and to combine them in a global structure.

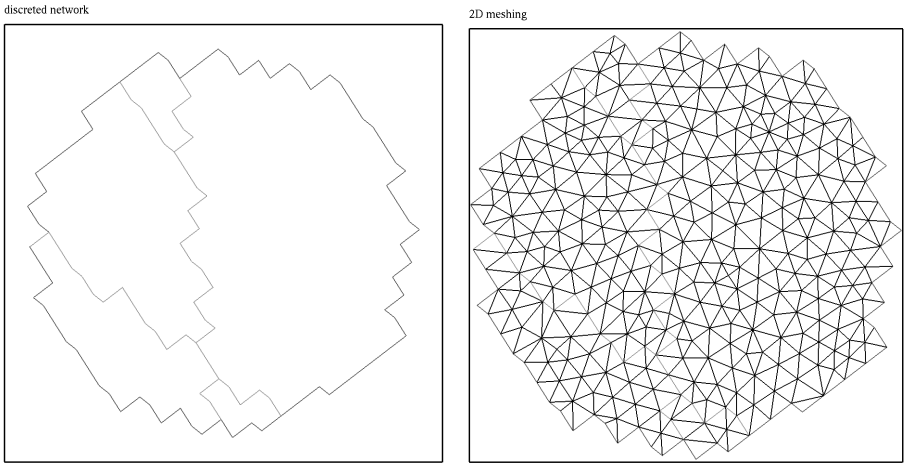
The matrix α is symmetric positive definite and is sparse. Sparse solvers, either direct or iterative, can be used to compute the solution. The size of the linear system is the number of edges in the network mesh and is roughly 1.5 times the number of triangles (about three edges for two triangles in 2D domains).

1.3 Mesh generation

In order to generate the mesh of the 3D DFN described above, we design and implement a new and original algorithm. To our knowledge, there is no existing software nor existing algorithm that would generate the mesh, which is neither 3D nor 2D nor a surface but a set of 2D meshes with a unique definition of their intersections. Because there is no specific direction of the flux, we aim at generating a mesh with a good aspect ratio, where all triangles are close to equilateral. Also, because we want to limit computational costs, we aim at reducing the number of elements in the mesh. Our algorithm is based on a two-level discretization. At the network level, each boundary and each intersection are discretized. Then the discrete boundaries and intersections of each fracture Ω_f are used to mesh the 2D domain. Therefore, we have to define a discrete projection from the 3D network onto each 2D fracture plane.

A natural approach would be to define each segment by a succession of discrete points and to project these points. However, this approach has several drawbacks. In some cases, it is not possible to generate a mesh ; if the mesh can be generated, it has a tiny mesh size and many triangles, so that the linear system is huge ; finally, it has very small angles, so that the aspect ratio is bad. For all these reasons, we define a new original approach to generate the mesh of DFNs. This approach is used in [11] and is described in [55].

The main idea is to define a 3D discretization of the boundaries and intersections. We introduce a 3D grid, with a given grid size Δx and with voxels as elementary units. The grid size is given relatively to $l_{\min}$, i.e., $\Delta x = \frac{absolute\ grid\ size}{l_{\min}}$. At the network level, boundaries and intersections are discretized by a set of voxels, instead of a set of points. Thus, at the network level, boundaries and intersections are now defined by 3D objects, which are projected onto fracture planes as before. The projected intersections and boundaries are now staircase lines with segments of length about Δx, so that small angles and small length size have disappeared. In each fracture, we can use any software to generate a 2D



(a) (b)

FIGURE 1.3 – Projecting and meshing : example of a fracture ; intersections are in light grey ; the mesh has a good aspect ratio.

mesh, with Δx as initial mesh size. The resulting mesh has a good aspect ratio and a limited number of elements.

In contrast to [72], we keep a geometrical correspondence between the intersections. Let us consider an intersection between two fractures. The projections in each fracture are different, but the edges have the same 3D antecedent. Each edge in the intersection is thus uniquely defined, and can be identified in each fracture. Continuity conditions at the intersections can still be defined for each edge : same pressure for the corresponding edge in each fracture and continuity of flux through the edge. The mixed hybrid method remains somehow conforming, and we still get local and global mass conservation.

The projected fractures are no longer convex because of the discretization in voxels. An example of DFN is meshed using our new method, and the result can be seen in Figure 1.3. In this example, the 2D projected fracture (Figure 1.3a) is connected and the boundary is well identified, with intersections inside the boundary. Each boundary or intersection edge is connected to one or two other edges, and each boundary point is connected to one or two edges. The mesh can be generated and is of good quality (Figure 1.3b).

But in some cases, this property is no longer true, as can be seen in Figure 1.4. Indeed, it may happen that two points of the 3D boundary are discretized by the same voxel. In this case, the projected fracture may have several connected components or the boundary of the open set is no longer the set of boundary edges. Thus some points are connected to three edges and some edges are connected to three other edges. Some abnormal configurations are illustrated in Figure 1.4a.

It is necessary to ensure some topological properties, each boundary or intersection edge must have only one or two neighbors and each boundary or intersection node must have only two adjacent edges. Otherwise, mesh generation can fail or elementary matrices can be singular, leading to a global singular matrix. Since these abnormal configurations may happen quite often in random 3D DFNs, it is mandatory to apply a geometrical correction. In [55], this difficulty is not described, so networks used in the tests are specific and Monte-Carlo simulations are not feasible.

In order to respect these topological properties, we apply local corrections, which mo-

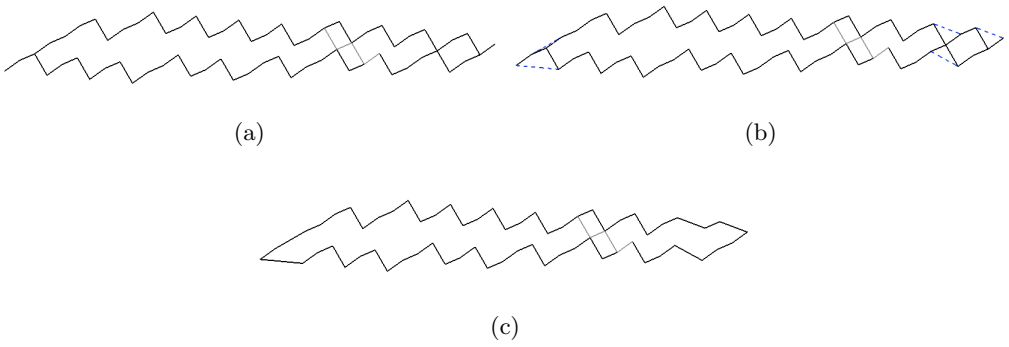


FIGURE 1.4 – Local corrections before meshing : example of a fracture. (a) points connected to 3 or 4 edges ; (b) local corrections ; (c) after correction, each point is connected to 2 edges.

dify slightly the surface of the fracture. These local corrections aim at defining correct boundary and intersections. After the corrections, each edge is connected to one or two other edges and each node is connected to one or two edges. Examples of corrections are illustrated in Figure 1.4b. After correction, the modified fracture has one connected component and the boundary is well defined, as illustrated in Figure 1.4c. Thus we can guarantee that mesh generation will not fail, that aspect ratio is good, and that each elementary matrix is not singular, so that the global matrix is not singular.

1.4 Software

We have developed a complete software, written in C++, to generate random networks of fractures and to simulate flow in these networks. The software MP_FRAC is integrated in the scientific platform H2OLAB [29] [1], which provides a user interface, a database for result compilation and analysis and visualization tools. The software MP_FRAC follows a modular and object-oriented approach, which provides a great flexibility by allowing an easy change of the algorithms and enhances the maintenance. The code is organized in three modules, as depicted in Figure 1.5 : network generation, mesh and problem generation, and flow computation.

The first module includes the random generation of the network, the definition of the geometry, the discretization of the intersections and boundaries, the global numbering of edges and points, and the projection of intersections and boundaries onto the fracture planes. This module builds the data structure "Network," which contains the geometrical data organized in a hierarchical way : geometrical information is stored at the various levels defining the network and fractures, the intersections and boundaries, the edges, and the points.

The second module makes use of the Network data structure to generate the mesh in each fracture, to number globally the edges, and to define the physical data. It builds the data structure "Mesh," which is also organized hierarchically with the same levels as Network with, in addition, the level defining the triangles. This module builds also the data structure "Pb-data," which contains the boundary conditions, the permeability field,

1. http ://h2olab.inria.fr/

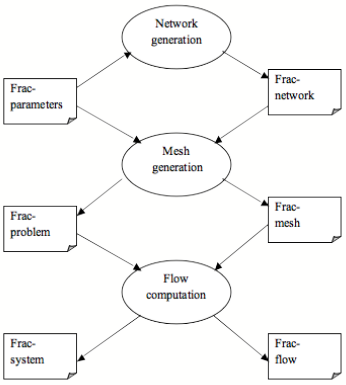


FIGURE 1.5 – Computational modules and data structures for flow computations in discrete fracture networks.

the source term, the various physical data such as the gravitational constant, the density, etc.

Finally, the third module does the computation, by first computing the local matrices in each fracture and computing the global matrix and global right-hand side ; then, it solves the sparse linear system, and computes the hydraulic head in each triangle and the velocity on each edge. This module uses previous data structures and builds the data structure "System," which contains the local matrices and right-hand sides, the global matrix and right-hand sides, the solution (hydraulic head on each edge), the hydraulic head in each triangle, and the velocity on each edge of each triangle. Then we check the result by a mass conservation verification.

The mesh generation is done by using a procedure extracted from the software FreeFem++ [40] and interfaced with a mixed finite element method. The local matrices and the velocity are computed by using adapted and rewritten procedures of the software Traces [42]. The sparse linear solver is a submodule which can be easily interfaced with free libraries. Currently, we have interfaced the direct solvers Umfpack [21] and Pspases [39], and the libraries Hypre [32] and Petsc [4].

1.5 Experimental results

In order to illustrate the efficiency and robustness of our method, we generate a large number of test cases. We check the convergence order of the method by reducing the mesh size. Finally, we do a complexity analysis.

1.5.1 Test generation

General networks have various parameters. We test our method on a set of parameters which covers a large set of DFNs, leading to a set of 18 network types. In all the tests, the permeability is chosen homogeneous and is equal to 1 everywhere. Since we use a Monte-Carlo method, for each set of parameters, we generate N_{ech} random samples, where $N_{ech} = 10$ is sufficient to validate the method. For each of the 180 networks, we vary the mesh step with 9 values, so that we get 1620 systems of meshed networks. Values are given in Table 1.3. We use the sparse direct solver Umfpack to compute the flow.

Since the solution for nonconnected networks is trivial ($v = 0$), we do not consider them.

TABLE 1.3 – Test generation : 180 networks, 1620 systems.

Parameter	Value
d	$2, 3$
$L/l_{\min}$	from 1 to 3
e	uniform
α	$2.5, 3.5, 4.5$
N_{ech}	10
Δx	from 0.01 to 0.09

We test the connection after discretization. In fact, discretization may disconnect networks when intersections critical to connectivity (bottlenecks) are smaller than the discretization scale. Among the networks, we reject 254 nonconnected networks. Our method succeeds in generating the mesh and solving the linear system for all connected networks.

For most of the discretized networks, the post-projection corrections are essential. Indeed, if we remove the corrections, the method fails to generate a mesh or to solve the linear system. For the tests considered, the method without correction succeeds in only 222 cases. We have checked that the same problems occur when all fractures are disks, by taking a constant distribution of shapes. Again, the method without corrections succeeds in less than 300 cases, whereas our method succeeds in all cases. Therefore, it is not possible to run Monte-Carlo simulations without local corrections.

1.5.2 Validation, convergence, and order

For some particular cases, like a few orthogonal fractures, we check that the computed solution is as predicted by theory or analytical solution. For all cases, once we have solved flow equations, we check systematically that the computed fluxes satisfy global mass balance and local mass balance through each edge of the mesh. These verifications are a first step in validating the method.

Then, we analyze the convergence of the numerical solution when we reduce the mesh step. With a classical 2D or 3D domain, MHFE methods are of order 1 ; a priori error estimation can be derived in L^2 norm for the velocity [27] ; when the domain is bounded, error estimation can also be written in L^1 norm. Here, we assume that the method applied to DFN with interface conditions is still of order 1 and that an error estimation can be written for the velocity at the intersections. Because we modify the geometry by projecting voxels and by local adjustments, there is no guarantee of convergence. Thus we check experimentally the behavior of numerical solutions. For each generated DFN, we realize a series of computations, each one with a different mesh step, ranging from 0.01 to 0.09. Since we do not know the exact solution, we use the simulation with the smallest mesh step as a reference to check the convergence. We use a physical criteria to check convergence, defined as the flow crossing the intersections normalized by the cumulated intersections length. Let us define $L = \sum_{k=1}^{NI} l_k$, where l_k is the length of intersection S_k. Let us define, for a velocity v,

$$\Phi(v) = \sum_{k=1}^{NI} \int_{S_k} |v| dl.$$

We compute the scalar c defined by

$$c = \frac{|\Phi(v_{\Delta x}) - \Phi(v_{ref})|}{L},$$

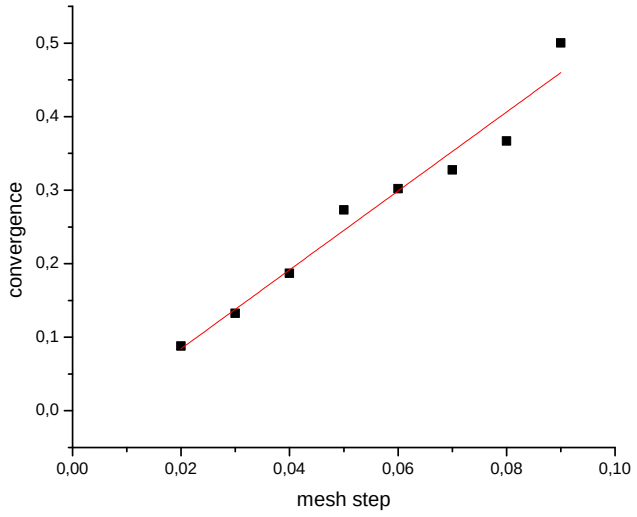


FIGURE 1.6 – Convergence analysis : relative error on flux at intersections for a given mesh step ; reference value is taken from the finest mesh. Mean values computed from all other meshed networks.

where v_{ref} is the velocity for the reference mesh. This criteria can be computed easily since

$$\Phi(v_{\Delta x}) = \sum_{I=1}^{N_{\Delta x}} |v_{E,i}| l_I,$$

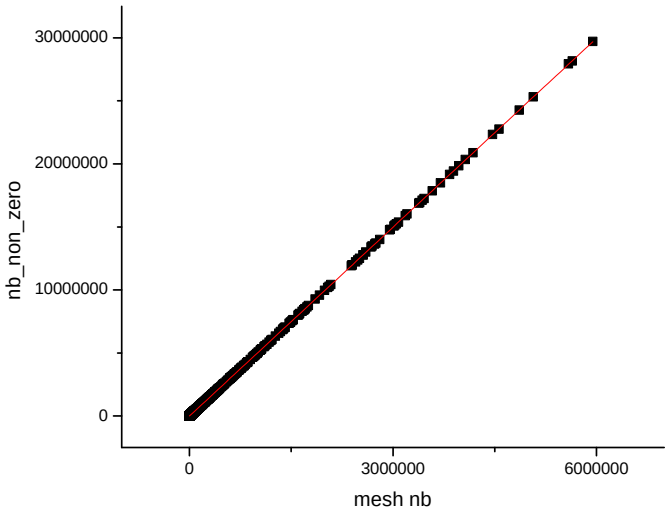
where l_I is the length of edge I and $N_{\Delta x}$ is the total number of edges at intersections.

Using all the 1366 simulations, we calculate the mean of c for each mesh step. Figure 1.6 shows these means, where one point corresponds to about 150 networks with various parameters. We observe that c has a linear convergence, as can be expected if the method is of order 1.

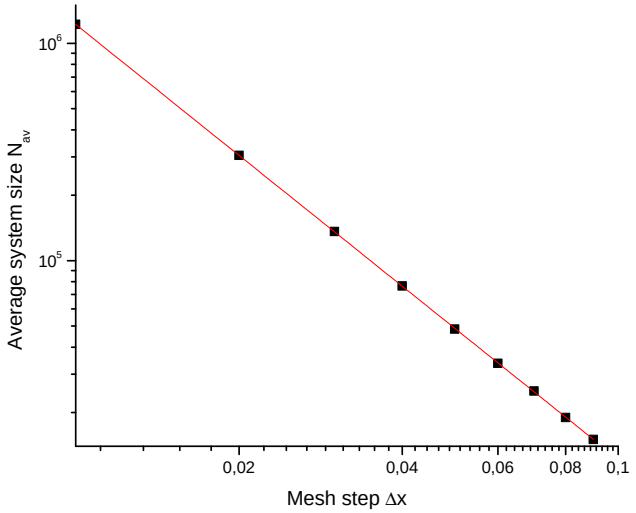
1.5.3 Sparsity and size complexity

The objective here is to analyze how the size of the linear system scales with the various parameters. In a first step, we measure the sparsity of the matrix. The number of nonzero elements on a line is the number of neighbors of the associated edge plus one for the diagonal element. In the 2D case, with a triangular mesh, each interior edge has four neighbors and border intersections have only two neighbors, leading to a number of nonzeros NNZ roughly equal to $5N$, where N is the size of the matrix. In 3D DFNs, each fracture is 2D, but a special case occurs for intersections between fractures, since the edges on an intersection between two fractures have eight neighbors. Thus NNZ could be larger than $5N$ for some networks. In Figure 1.7a, we plot NNZ versus N for all the systems. Clearly, we observe a factor $NNZ = 4.99N$, showing that the contribution of the intersections is smaller than the contribution of borders. Thus we get a sparsity complexity similar to the 2D case, although we deal with 3D DFNs.

In a second step, we analyze how the matrix size N scales with the mesh step Δx. In a classical 2D mesh, N behaves like Δx^{-2}, whereas N behaves like Δx^{-3} in 3D regular



(a) Sparsity complexity : number of nonzeros versus system size for all 1366 systems.



(b) Size complexity : average system size versus mesh step for all 1366 systems.

FIGURE 1.7 – Linear system complexity in size.

grids. For each mesh step Δx, ranging from 0.01 to 0.09, we compute the average size N_{av} of all generated networks. In Figure 1.7b, we plot N_{av} versus Δx and find a power relation with an exponent -2, showing that N behaves like Δx^{-2}, still like the 2D case.

Therefore, the size and sparsity of the discrete system arising from flow computations in 3D discrete fracture networks have a behavior similar to a system arising from a 2D domain.

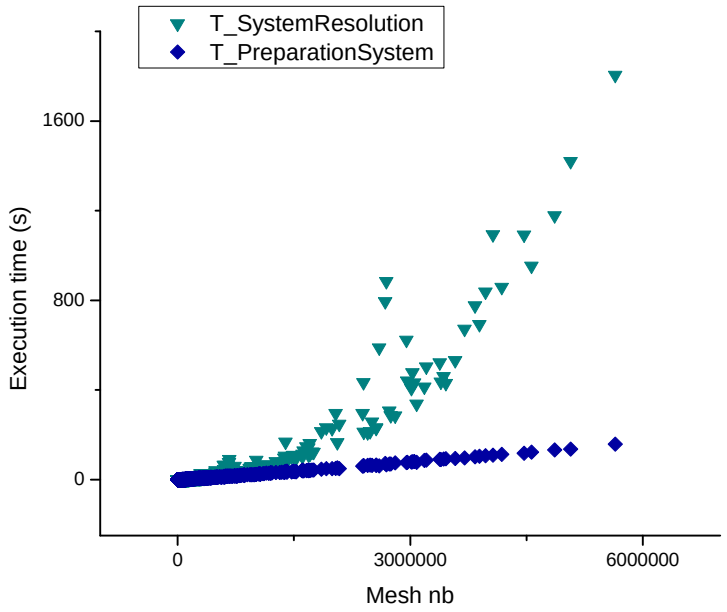
1.5.4 Time complexity

Now we analyze the complexity of the computation, by measuring the CPU time and taking the system size N as the main parameter of the system. We measure the CPU time during two phases : the first phase concerns all the preparation before solving the linear system with a sparse direct solver, which is the second phase. We recall that we use a multifrontal solver, implemented in the library Umfpack. In Figure 1.8a, we draw the CPU time versus the size for these two phases. Since we use a direct solver, the time for solving the system is not linear, but follows roughly a power law. We observe different tendencies, all of them with an exponent larger than 1.5, which is the exponent for flow computations in 2D regular grids [12]. Although the preparation phase includes more computations than a classical 2D MFE method, we can verify that the CPU time during this phase is linear in N and much smaller than the CPU time during the second phase. The details of the preparation phase are drawn on Figure 1.8b, where we have split the time into three steps. The first step deals with 3D discretization, projection, and adjustments ; the second step generates the mesh ; the third step computes the matrix and right-hand-side. The first step, which is specific to our method, is the cheapest one, whereas computing the matrix is the most expensive.

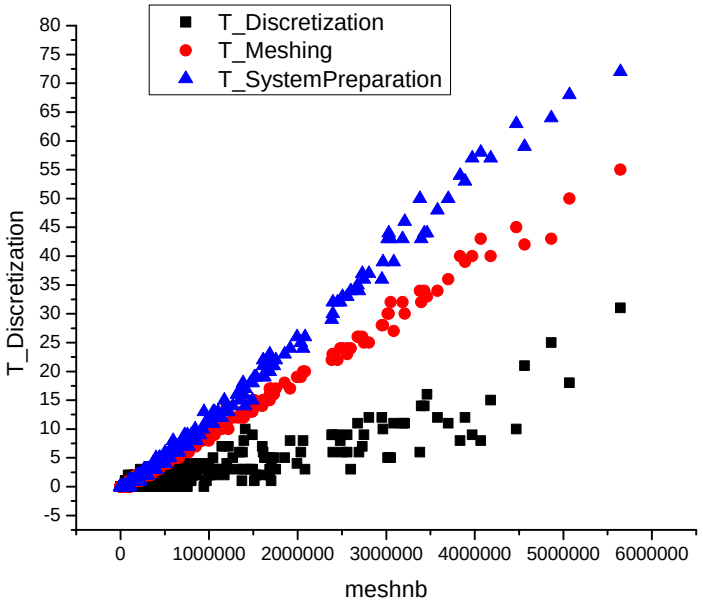
1.6 Conclusion

In this paper, we consider 3D DFNs as a stochastic approach for modeling flow in natural fractured rocks. Flow simulations aim at characterizing flow patterns in such networks. This is achieved by applying an MHFE method on a flow model where continuity of mass and flux are imposed at the intersections between the fractures. The main difficulty is to generate a mesh, which must be 2D in each fracture and must be conforming at the intersections in order to apply the numerical method. We present a two-fold method, where the network is first discretized by voxels and the projected fractures are discretized by a classical mesh generator. We show that this method yields a mesh with a good aspect ratio, but requires adjustments in order to remove abnormal configurations after projection. Thanks to our method, it is possible to generate a mesh for a wide range of random DFN ; we can vary the power-law exponent, the number of fractures, their orientation, and their shape. For all the networks tested, the mesh is of good quality, in the sense of a good aspect ratio, and a fairly large mesh size can be used. Therefore, it is possible to use uncertainty quantification methods such as Monte-Carlo simulations to study flow in fracture networks.

Although we deal with 3D structures, we show that the size complexity is similar to 2D computational domains. For any network, the size N of the linear system is the main parameter relevant to analyze CPU time ; we find out that the CPU time of the preparation phase is linear with N and that most of the time is spent in solving the sparse linear system, with a complexity larger than $O(N^{1.5})$. We plan now to further analyze this behavior in order to reduce CPU time and to tackle even larger systems. We will



(a) System preparation and linear solving



(b) System preparation detail

FIGURE 1.8 – Time complexity : CPU time versus system size for all generated networks.

run simulations with iterative solvers and will develop a parallel version. Another track of research is to relax the conforming constraint by applying a mortar methodology.

Chapitre 2

Utilisation et comparaison de solveurs linéaires directs et itératifs pour les réseaux de fractures 3D

Nous avons décrit dans le chapitre précédent une méthode permettant de générer un réseau de fractures, de le mailler et de construire le système linéaire associé de façon systématique, pour une utilisation dans un processus stochastique. Nous nous intéressons dans ce chapitre aux différents solveurs que l'on peut utiliser pour résoudre les systèmes linéaires associés.

Après une présentation théorique des solveurs, nous présentons les résultats obtenus sur un ensemble de réseaux afin d'étudier leur comportement.

2.1 Présentation des solveurs utilisés

2.1.1 Direct : factorisation de Cholesky

Les solveurs directs permettent de résoudre un système linéaire en faisant un nombre fini d'opérations flottantes, grâce à des combinaisons et des modifications des équations formant le système. La précision de ces méthodes est limitée par la précision des calculs flottants.

Lorsque la matrice A du système est symétrique définie positive, en réalisant une élimination de Gauss, on obtient une factorisation de la matrice sous forme d'un produit d'une matrice triangulaire inférieure et de sa transposée : $A = LL^T$. Il s'agit de la factorisation de Cholesky de la matrice A[53, p. 82]. A partir de cette factorisation, il est aisé d'obtenir la solution du système $Ax = b$ en effectuant successivement les résolutions de $Ly = b$ et $L^T x = y$.

Dans le cas où la matrice A est creuse, le nombre d'éléments non nuls dans la matrice L est grandement supérieur au nombre d'éléments non nuls de la matrice de départ. On parle de phénomène de remplissage. Pour diminuer ce remplissage, il faut permuter la matrice A avant la factorisation. Il est en effet possible de trouver une matrice de permutation P telle que la factorisation $PAP^T = LL^T$ présente un remplissage moins élevé que la factorisation d'origine. Plusieurs techniques peuvent être utilisées pour trouver cette permutation. Citons par exemple :

– la dissection emboîtée qui partitionne le graphe adjacent de la matrice récursivement,

en minimisant les connection entre chaque partition d'un même niveau[37] ;
- Approximate Minimum Degree (AMD)[9] : la permutation obtenue numérote en premier les éléments ayant, dans le graphe associé au facteur de Cholesky, le plus petit nombre de voisins (nœud de degré minimum). Pour accélérer la détection du minimum, à chaque itération de l'algorithme, ce n'est pas le degré réel de chaque nœud qui est utilisé, mais une approximation.

Cette méthode de résolution est la méthode privilégiée pour les systèmes de taille réduite. La complexité en temps de calcul et en mémoire augmente rapidement et rend son utilisation limitée pour les systèmes de grande taille.

2.1.2 Multi-grille algébrique

Les méthodes de multi-grilles appartiennent à la famille des méthodes multi-niveaux. Plusieurs niveaux de grille sont définis, définissant un maillage de plus en plus grossier ou un système linéaire de plus en plus petit. Des cycles sont ensuite appliqués itérativement jusqu'à l'obtention de la solution. Ces cycles sont constitués de quatre phases :
- Une phase de relaxation, qui permet d'éliminer les hauts modes de l'erreur ;
- Une phase de transfert d'information entre les niveaux de grille, pour atteindre le niveau le plus grossier, par des opérations de restrictions ;
- Une phase de solution, au niveau le plus grossier ;
- Une nouvelle phase de transfert d'information, jusqu'au niveau le plus fin, par des opérations d'interpolations.

On parle de V-cycles (Fig : 2.1).

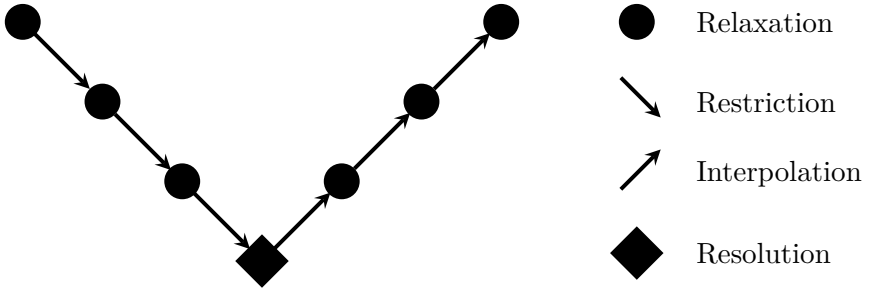


Figure 2.1 – AMG V-cycles

Dans les méthodes de multigrille géométrique, les grilles grossières sont définies en se basant sur le maillage. La version algébrique [64] n'utilise pas le maillage (auquel elle n'a pas accès) pour définir les grilles grossières, mais utilise uniquement les propriétés algébriques de la matrice. Cela nécessite d'analyser la matrice pour trouver des nœuds de faible poids vis-à-vis des connections du graphe de la matrice.

Les méthodes de multigrilles algébriques sont efficaces pour résoudre les problèmes élliptiques.

2.1.3 Gradient conjugué

La méthode du Gradient Conjugué (CG)[41] est une méthode de résolution itérative de type Krylov. C'est la méthode de Krylov qui est privilégiée quand la matrice A est une matrice symétrique définie positive. On trouve de nombreuses descriptions dans la littérature, notamment dans [31, 65]. Elle peut être associée à un préconditionnement, on parle dans ce cas du Gradient Conjugué Préconditionné (PCG). L'algorithme 1 présente la version classique de cette méthode.

Résoudre le système $Ax = b$ est équivalent à minimiser la forme quadratique $\varphi(x) = \frac{1}{2}x^T Ax - x^T b$. Le gradient de cette forme au point x vaut $r = Ax - b$. Contrairement à la méthode du gradient (méthode de la plus grande pente) qui prendrait cette direction comme direction de descente, la méthode CG impose que les directions de descentes soient conjuguées entre elles (c'est-à-dire A-orthogonales). On peut alors prouver des propriétés de convergence bien plus intéressantes.

Si A est de taille $n \times n$, la solution est atteinte en au plus n itérations. Mais la précision recherchée peut être atteinte en moins d'itérations. La vitesse de convergence dépend du conditionnement du système.

Conditionnement

Lors de la résolution avec un algorithme itératif du système linéaire $Ax = b$, on cherche une solution numérique qui converge vers la solution exacte. Avec la méthode PCG, la convergence est strictement monotone, mais il se peut que la vitesse de convergence soit très faible. On peut prévoir ce comportement grâce au spectre de la matrice. Une première indication est donnée par le conditionnement, noté $\kappa(A)$, de la matrice A.

$\kappa(A)$ est défini ainsi :

$$\kappa(A) = \| A \| \| A^{-1} \| \ .$$

Dans le cas où A est symétrique, cela donne

$$\kappa(A) = \frac{|\lambda_{max}|}{|\lambda_{min}|}$$

où $|\lambda_{max}|$ et $|\lambda_{min}|$ sont respectivement les valeurs propres maximale et minimale de A, en valeur absolue, car $\| A \|_2 = |\lambda_{max}|$. Si A est définie positive alors $\lambda > 0$.

Plus le conditionnement de A est petit, plus la convergence du Gradient Conjugué est rapide. Lorsque $\kappa(A)$ est trop élevé, on dit que le système est mal conditionné.

Préconditionnement

Comme on l'a évoqué dans le point précédent, une résolution numérique peut ne pas aboutir si le système est trop mal conditionné, l'algorithme ne convergeant pas assez vite. On va alors essayer de diminuer le conditionnement du système. Pour ce faire, on va le transformer en un autre système linéaire de même solution mais mieux conditionné.

Étant donné une matrice M, pour un préconditionnement à gauche le nouveau système est donné par :

$$M^{-1}Ax \ = \ M^{-1}b.$$

Lors du choix de la matrice M on s'assure que

- si A est symétrique et définie positive alors M le soit aussi, pour pouvoir appliquer PCG
- si A est une matrice creuse, alors M le soit aussi puisqu'on ne veut pas utiliser beaucoup plus de capacité de stockage pour M que pour A,
- M soit facile à construire, pour ne pas perdre trop de temps de calcul dans la phase de préparation,
- $M^{-1}y$ soit facile et rapide à résoudre, pour la même raison,
- le nombre d'itérations final soit presque constant, quelle que soit la taille du système.

On peut citer comme méthodes de préconditionnement classiques les méthodes basées sur des factorisations incomplètes du système, comme la factorisation de Cholesky incomplète [52]. Ce préconditionnement perd en efficacité lorsque la taille du système augmente.

Les méthodes de décomposition de domaines (Schwarz additif restreint [17] ou multiplicatif [10], Schur reccursif [67, 38]) peuvent aussi définir des préconditionnements. Il est aussi possible d'utiliser un cycle de multigrille pour préconditionner le système. A priori, avec ce préconditionnement, le nombre d'itérations est presque constant quand la taille augmente.

Algorithm 1 Gradient Conjugué Préconditionné

Compute M ▷ preprocessing
Choose x_0 ▷ Initialisation
$r_0 = b - Ax_0$
$z_0 = M^{-1}r_0$
$p_0 = z_0$
$k = 0$

repeat ▷ Iterations
  $q_k = Ap_k$
  $\alpha_k = \frac{r_k^T z_k}{p_k^T q_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k q_k$
  $z_{k+1} = M^{-1} r_{k+1}$
  $\beta_k = r_{k+1}^T z_{k+1}$
  $p_{k+1} = z_{k+1} + \frac{\beta_k}{\beta_{k-1}} p_k$
  $k = k + 1$
until convergence

2.2 Application des solveurs et comparaisons des performances

2.2.1 Interface H2oLab/Solveur

Dans la plate-forme `H2oLab`, les simulations de réseaux de fractures sont effectuées dans l'exécutable `MP_Frac_D3_Flow`. Cet exécutable contient un module qui permet d'utiliser plusieurs solveurs linéaires.

2.2.2 Bibliothèques utilisées

Nous avons utilisé des solveurs implémentés dans des bibliothèques open-source. Pour des raisons pratiques, le solveur direct que nous avons utilisé est un solveur utilisant une factorisation LU et non une factorisation de Cholesky. Il s'agit d'`UMFPACK`[20], disponible dans `SuiteSparse`[8], qui était déjà intégré à la plate-forme H2oLab. Il s'agit d'un solveur uniquement séquentiel. Nous aurions pu utiliser `Cholmod`, présent aussi dans `SuiteSparse`. Comme les résultats auraient été similaires, nous n'avons pas effectué de nouveaux jeux de tests avec ce solveur.

Les solveurs itératifs sont fournis par la bibliothèque `hypre`[7]. Nous avons utilisé BoomerAMG comme solveur multigrille algébrique et PCG préconditionné par un cycle de BoomerAMG. Comme on l'a dit précédemment, les méthodes de multigrilles algébriques sont efficaces pour traiter des problèmes elliptiques. En particulier, BoomerAMG a été utilisé avec succès pour résoudre un problème d'écoulement en milieu poreux hétérogène [28].

Il nous a paru pertinant de tester ce solveur sur le problème d'écoulement dans les réseaux de fractures, l'écoulement dans chaque fractures correspondant à un écoulement dans un milieu 2D.

2.2.3 Descriptions des réseaux utilisés pour les tests

Afin de tester l'efficacité des différents solveurs, nous avons effectué un grand nombre de simulations. Nous avons fait évoluer la taille des systèmes soit en augmentant le nombre de fractures avec le paramètre de la densité d (Tab : 2.1), soit en augmentant le nombre d'arêtes avec le paramèdre de pas de maillage δx (Tab : 2.2). Pour chaque sélection de paramètres, 10 simulations ont été effectuées avec des graines différentes pour le générateur aléatoire. L'ensemble des jeux de tests représente 180 réseaux de fractures différents et 420 systèmes linéaires, dont 417 inversibles. La variation de densité décrit en effet 180 réseaux, 30 d'entre eux étant réutilisés pour la variation du pas de maillage. Sur ces 180 réseaux, 3 ne sont pas composés d'un bloc connexe reliant les bords de type Dirichlet. De ce fait, le système linéaire associé est singulier et ces réseaux sont exclus des tests.

Paramètre	valeurs
L/l_{min}	3
α	$2.5; 3.5; 4.5$
d	$2; 3; 4; 5; 6; 7$
Δx	0.04
N_{ech}	10

Table 2.1 – Variation du nombre de fractures avec la densité d (densité) : 177 réseaux connectés

Paramètre	valeurs
L/l_{min}	3
α	$2.5; 3.5; 4.5$
d	4
Δx	$0.01; 0.02; 0.03; 0.04; 0.05; 0.06; 0.07; 0.08; 0.09$
N_{ech}	10

Table 2.2 – Variation du pas de maillage δx (maillage) : 270 réseaux connectés

Solveur	Succès
UMF	387
AMG	375
PCG	417

Table 2.3 – Nombre de simulations résolues avec succès sur les 417 systèmes générés

Le premier jeu de tests, correspondant à la variation de la densité, définit 177 systèmes dont la taille varie de 1.4×10^5 à 6.1×10^5 mailles. Le deuxième jeu, correspondant à la variation du pas de maillage, génère 270 systèmes, dont la taille varie de 6×10^4 à 5.9×10^6 mailles.

Ces tests ont tous été exécutés sur un serveur de calcul et de développement tournant sous windows server 2003. Ce serveur dispose de 16 Go de RAM, et de 16 processeurs. Cette dernière ressource n'est pas exploitée puisqu'il s'agit de tests séquentiels.

Le tableau 2.3 présente le nombre de succès obtenus avec chaque solveur testé. Nous commentons ce tableau et les résultats détaillés dans les paragraphes suivants.

2.2.4 UMFPACK

`UMFPACK` est le solveur le plus rapide pour les petits systèmes. Le temps nécessaire pour résoudre le système augmente rapidement avec la taille du système. Il n'est pas facile de déterminer la tendance, car plusieurs paramètres ont un impact sur la matrice. Il est toutefois visible sur les graphiques (Fig : 2.2) que l'évolution suit une loi puissance, avec différents exposants. Sur les systèmes les plus grands, la résolution n'a pas abouti, `UMFPACK` ayant atteint les limites de la mémoire disponible (Tab : 2.3). Le graphique correspondant pour `UMFPACK` (Fig : 2.2b) est tronqué après la dernière résolution ayant réussi. Ces systèmes correspondent aux réseaux discrétisés avec le pas de maillage le plus fin (0.01) pour chaque valeur du paramètre α. Cela représente 30 systèmes. Tous les autres systèmes ont été résolus.

2.2.5 BoomerAMG

`BoomerAMG` appliqué à ce problème n'a pas un comportement stable. En effet, comme on peut le voir sur les graphiques (Fig : 2.3), il arrive que certains systèmes soient nettement plus longs à résoudre que d'autres du même jeu de tests. Pour certains, la résolution n'aboutit pas dans le nombre limite d'itérations. `BoomerAMG` n'a pas les mêmes limitations qu'`UMFPACK` vis à vis de la mémoire, les échecs sont répartis sur l'ensemble des systèmes testés. Sur les 417 systèmes correspondants aux réseaux connectés, 42 systèmes n'ont pu être résolus (Tab : 2.3). Ces échecs ne sont pas représentés sur les graphiques. Ces situations correspondent à une cassure dans la vitesse de convergence, comme on peut le voir sur la figure (Fig : 2.4).

Lorsque tout se passe correctement, `BoomerAMG` a une complexité linéaire en temps, le nombre de V-cycles étant quasiment constant. Les graphiques (Fig : 2.5a, 2.5b) montrent le nombre de V-cycles nécessaire pour atteindre la précision souhaitée (1×10^{-12}). Le nombre de cycles maximum était fixé à 500. Dans les tests réalisés, `BoomerAMG` est plus lent que `UMFPACK` sur les petits systèmes. Mais il le devance rapidement, dès que les systèmes ont plus de 400 000 mailles et que le nombre de V-cycles n'est pas trop important.

2.2.6 PCG

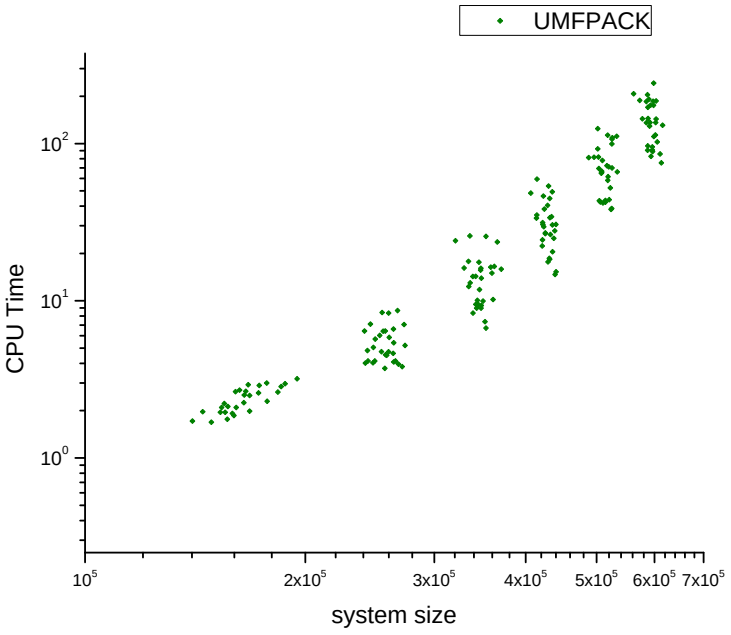
`PCG` est le plus stable des solveurs testés. Tous les systèmes ont été résolus (Tab : 2.3). Mais `PCG` préconditionné par `BoomerAMG` est le moins rapide des trois solveurs testés pour les petits systèmes. `PCG` est plus rapide que `UMFPACK` pour les systèmes de taille relativement importante (environ 5×10^5 mailles). Quand `BoomerAMG` converge correctement, il surpasse `PCG`. Il a, comme ce dernier, une convergence à peu près linéaire en temps, à nombre d'itérations presque constant.

Les graphiques (Fig : 2.7a, 2.7a) montrent l'évolution du nombre d'itérations en fonction de la taille des systèmes. Le nombre d'itérations évolue peu. La convergence est toujours atteinte en moins de 9 itérations, et il en faut entre 2 et 5 pour la majorité des systèmes. Seuls les plus grands systèmes (1.5×10^6 mailles) nécessitent plus d'itérations.

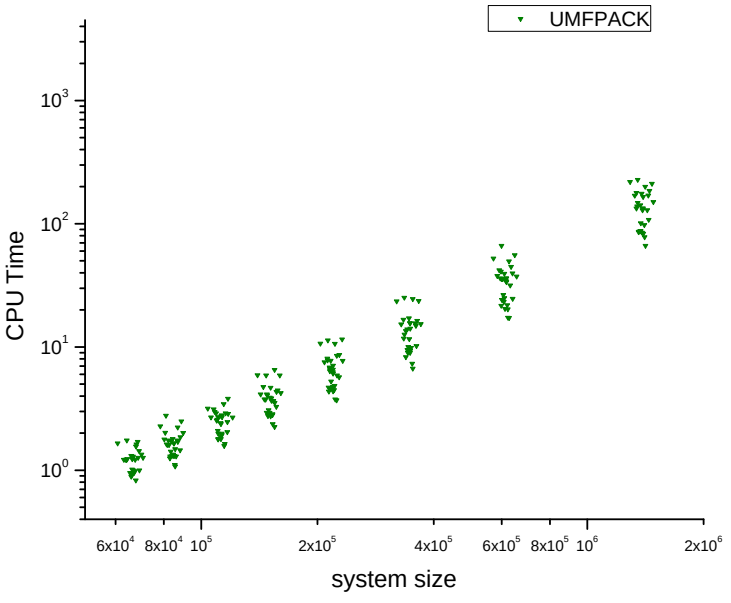
2.2.7 Conclusion

Comme on a pu le voir, l'utilisation d'un solveur direct (`UMFPACK`) est la méthode à privilégier pour les petits systèmes. Pour les systèmes de plus grande taille, il est nécessaire

de privilégier un solveur itératif. Le choix se porte sur `PCG` pour des raisons de stabilité. Il parait intéressant de chercher à cumuler la rapidité du solveur direct et la stabilité de `PCG`. Les méthodes de sous-domaines sont connues pour exploiter cet avantage. Nous nous intéressons, dans les chapitres suivants, en particulier à la méthode du complément de Schur. Elle permet, en décomposant le problème initial, de résoudre sur chaque sous-domaine un petit système en utilisant une méthode directe, et en utilisant une méthode itérative sur le problème à l'interface des sous-domaines.

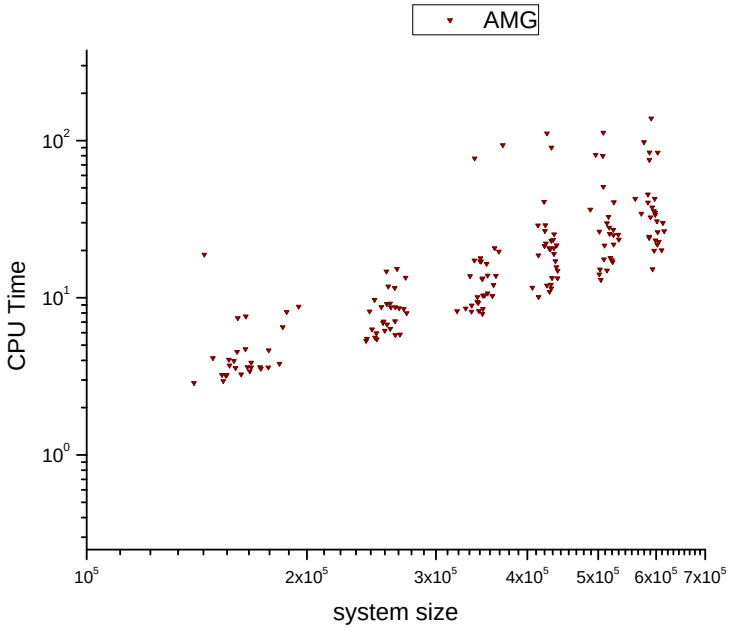


(a) Variation de la densité

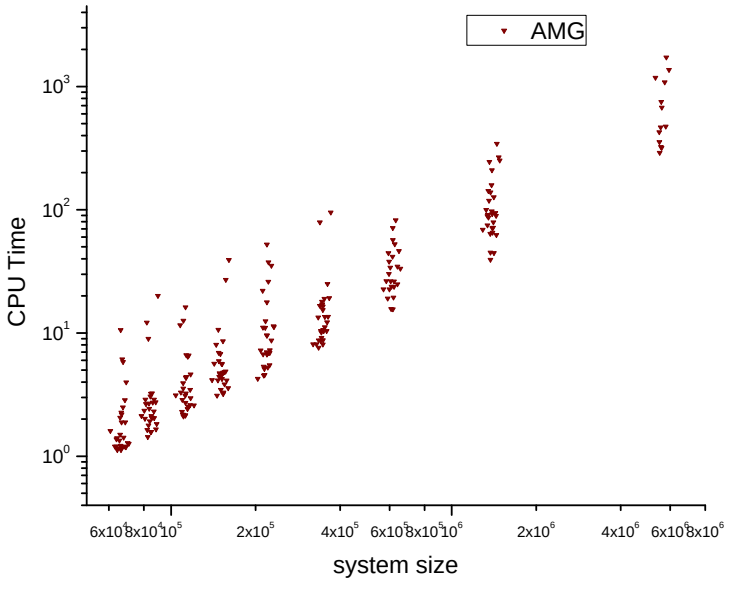


(b) Variation du pas de maillage

FIGURE 2.2 – Temps de résolution en fonction de la taille des systèmes avec `UMFPACK`.



(a) Variation de la densité



(b) Variation du pas de maillage

FIGURE 2.3 – Temps de résolution en fonction de la taille des systèmes avec `BoomerAMG`.

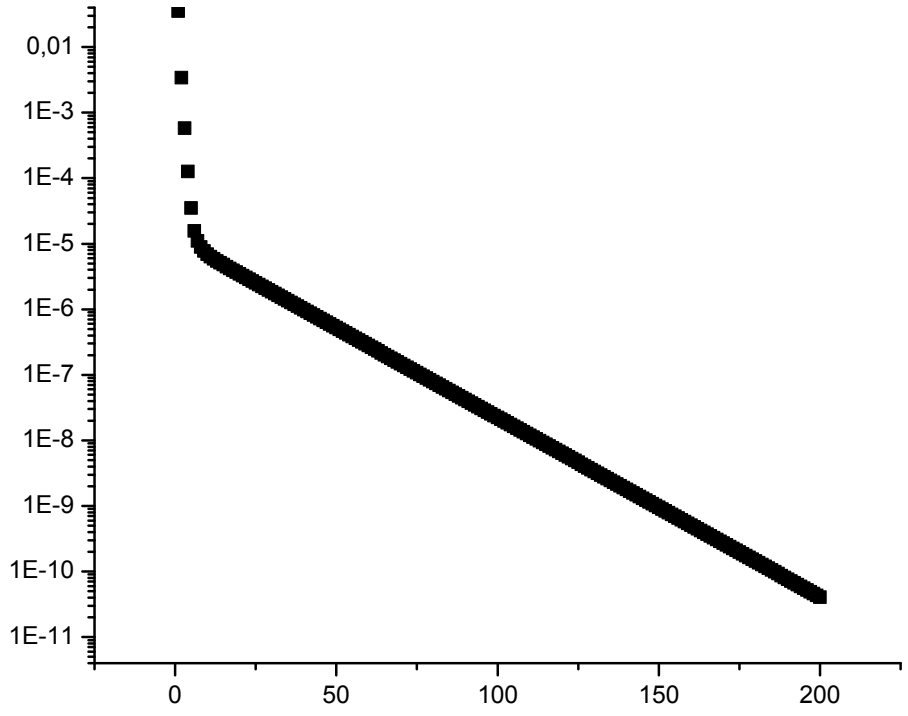
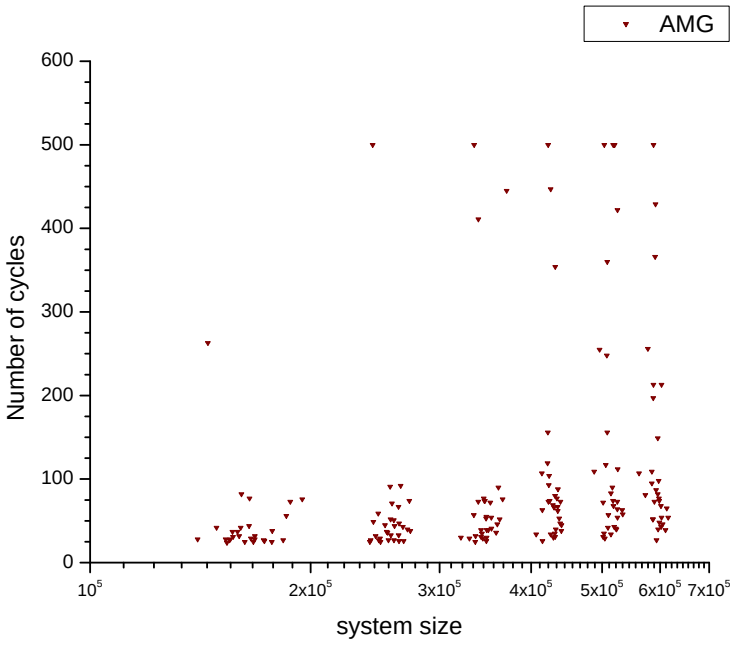
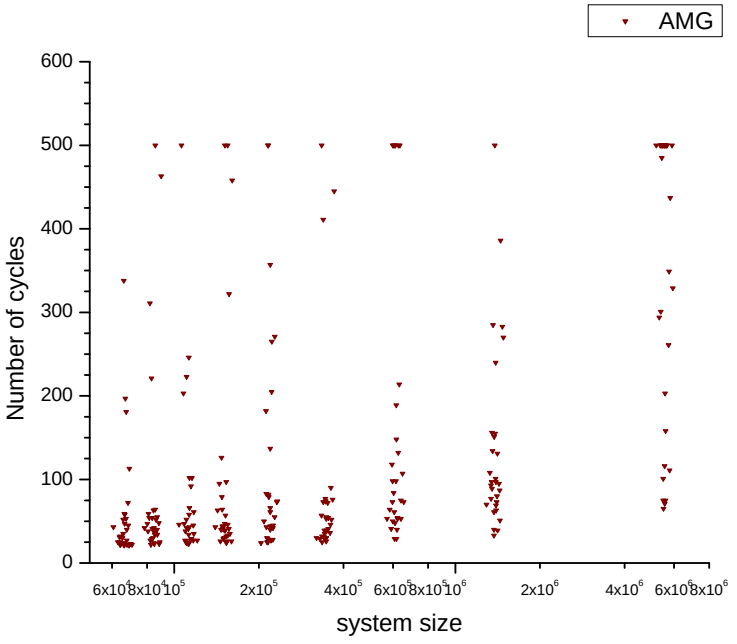


FIGURE 2.4 – Convergence lente avec `BoomerAMG`.

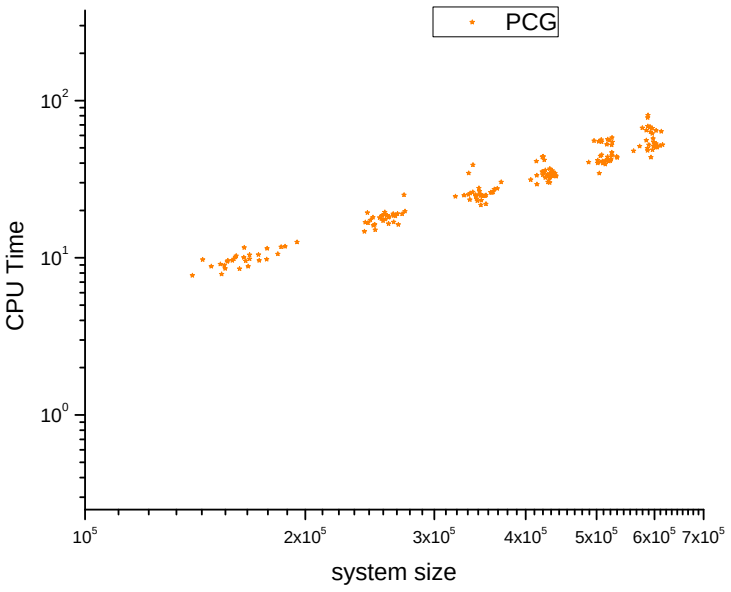


(a) Variation de la densité

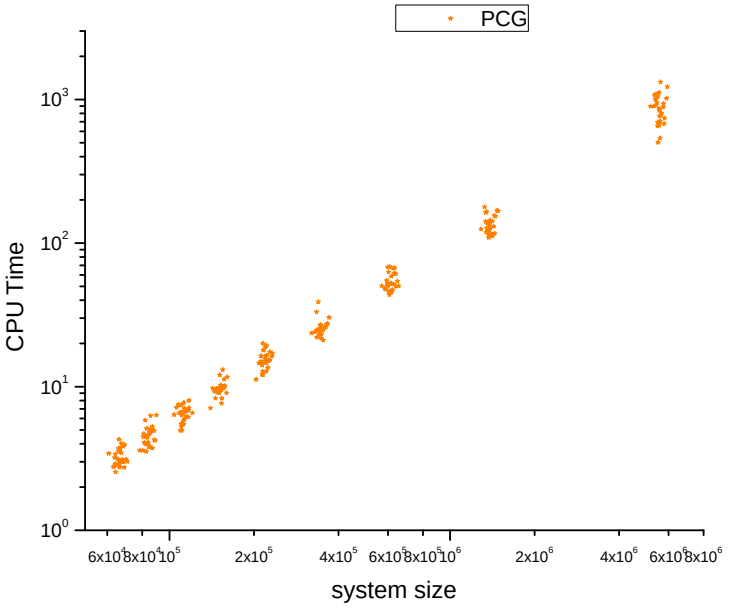


(b) Variation du pas de maillage

FIGURE 2.5 – Nombre de cycles en fonction de la taille des systèmes avec `BoomerAMG`.

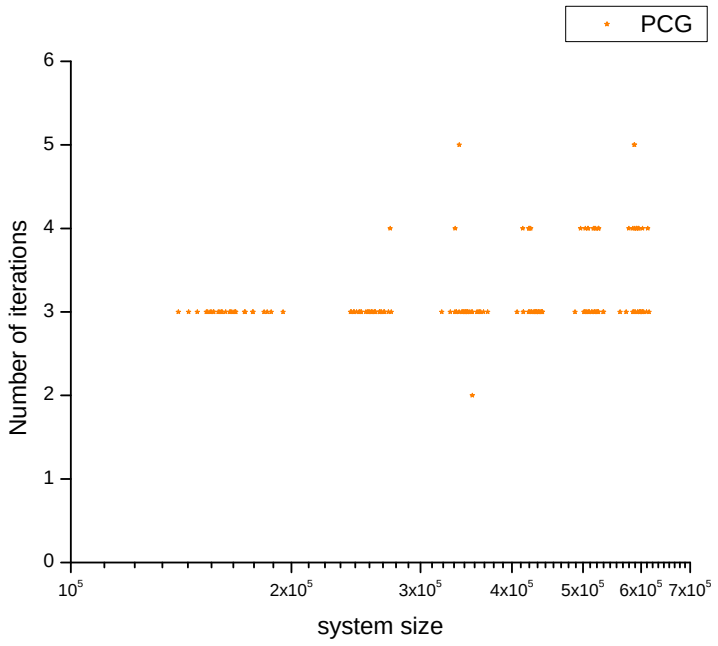


(a) Variation de la densité

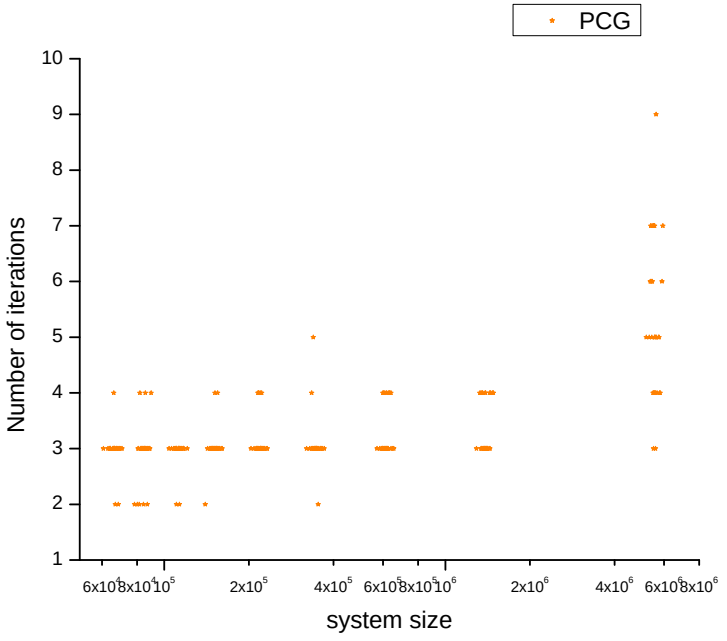


(b) Variation du pas de maillage

FIGURE 2.6 – Temps de résolution en fonction de la taille des systèmes avec `PCG`.



(a) Variation de la densité



(b) Variation du pas de maillage

FIGURE 2.7 – Nombre d'itérations en fonction de la taille des systèmes avec `PCG`.

Chapitre 3

Méthode du complément de Schur avec préconditionnement de Neumann Neumann et déflation

3.1 Introduction

Les méthodes de décomposition de domaines appartiennent à la famille algorithmique des méthodes de type “diviser pour régner”. Le principe de base de ces méthodes est de découper un problème de taille importante en problèmes de taille plus petite afin de faciliter leur résolution. L'idée est ici de décomposer le domaine sur lequel nous effectuons la simulation en plusieurs sous-domaines. Les systèmes linéaires sont alors construits localement, un par sous-domaine.

Ces méthodes présentent un intérêt certain pour l'utilisation de machines parallèles. En découpant le problème en sous-problèmes que l'on peut résoudre indépendamment, on obtient naturellement un algorithme parallèle. De plus, il est aisé d'obtenir un parallelisme à plusieurs niveaux, en utilisant un solveur parallèle pour résoudre les sous-problèmes.

Les méthodes de sous-domaines connaissent un succès certain depuis le début des années 90, comme en témoigne la grande quantité d'articles traitant du sujet, avec notamment ceux issus des conférences Domain Decomposition Methods [1], organisées depuis 1987. Des livres ont aussi été consacrés au sujet [63, 68, 70]. Les méthodes de sous-domaines peuvent être séparées en deux grandes catégories, en fonction de l'approche utilisée pour le découpage. On parlera de méthode avec ou sans recouvrement, en fonction du type de découpage. On trouve notamment dans la première catégorie les méthodes de Schwarz multiplicatif et de Schwarz additif, ainsi que les méthodes dérivées, comme l'accélération d'Aitken-Schwarz[36].

Dans la catégorie des méthodes sans recouvrement, nous trouvons les méthodes basées sur le complément de Schur ainsi que des méthodes dérivées de la première catégorie (méthode de Schwarz sans recouvrement). Les méthodes basées sur le complément de Schur consistent à réduire le problème complet à un problème à l'interface entre les sous-domaines puis à étendre la solution à l'ensemble du problème. Elles sont issues de la décomposition en sous-structures des problèmes de mécaniques [62]. Plusieurs variations de cette méthode existent. Citons notamment les méthodes duales, telle FETI [34]. Nous nous intéressons uniquement à la méthode du complément de Schur primal. Nous souhaitons appliquer cette méthode aux réseaux de fractures, qui présentent naturellement des interfaces, ce

qui nous a conduit à choisir la méthode du complément de Schur.

Dans ce chapitre, nous présentons comment utiliser conjointement le complément de Schur et le gradient conjugué pour résoudre un système linéaire. Nous décrivons aussi deux préconditionnements applicables à cette méthode.

3.2 Méthode de sous-domaines de type Schur

3.2.1 Cas de deux sous-domaines

Soit un système linéaire $Ax = b$ se décomposant sous cette forme :

$$\begin{bmatrix} A_{11} & 0 & A_{10} \\ 0 & A_{22} & A_{20} \\ A_{10}^T & A_{20}^T & A_{00} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_0 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_0 \end{bmatrix} . \tag{3.1}$$

La matrice A est supposée inversible. Les blocs A_{11} et A_{22} sont alors inversibles et le système se réduit par substitution à un système équivalent avec comme unique inconnue x_0.

$$Sx_0 = c_0 \tag{3.2}$$

avec

$$S = (A_{00} - A_{10}^T A_{11}^{-1} A_{10} - A_{20}^T A_{22}^{-1} A_{20})$$

et

$$c_0 = b_0 - A_{10}^T A_{11}^{-1} b_1 - A_{20}^T A_{22}^{-1} b_2 \tag{3.3}$$

La matrice S est appelée complément de Schur de A_{00} dans A.

Si x_0 est connue alors

$$x_1 = A_{11}^{-1}(b_1 - A_{10}x_0) \tag{3.4}$$

$$x_2 = A_{22}^{-1}(b_2 - A_{20}x_0) \tag{3.5}$$

3.2.2 Lien avec une décomposition en sous-domaines

Considérons un système d'équations provenant de la discrétisation d'une EDP elliptique. Dans cette partie nous considérons une EDP elliptique, $div(kgrad(u)) = f$ dans un domaine Ω rectangulaire. Les conditions aux limites sont fixées sur $\partial\Omega$, dont des conditions de Dirichlet sur $\partial_D\Omega \neq \emptyset$. Ω est séparé en deux sous-domaines Ω_1 et Ω_2 par la frontière Γ (Fig. 3.1). L'EDP est discrétisée avec des éléments finis mixtes hybrides. Dans le cas d'un écoulement en milieu poreux, u correspond à la charge. Les inconnues sont alors la charge aux arêtes du maillage, pour une discrétisation par éléments finis mixtes hybrides.

La renumérotation des inconnues dans le domaine Ω est faite de façon à obtenir

$$x = [x_1 \ x_2 \ x_0]^T$$

avec $x_1 \in \Omega_1$, $x_2 \in \Omega_2$ et $x_0 \in \Gamma$. Le système linéaire $Ax = b$ s'écrit avec une notation par blocs qui donne le système (3.1).

Dans ces conditions, A_{11} (resp. A_{22}, A_{00}) est une matrice qui représente la relation entre les inconnues dans Ω_1 (resp. Ω_2, Γ). Le bloc A_{10} (resp. A_{20}) correspond à la relation entre Ω_1 (resp. Ω_2) et Γ.

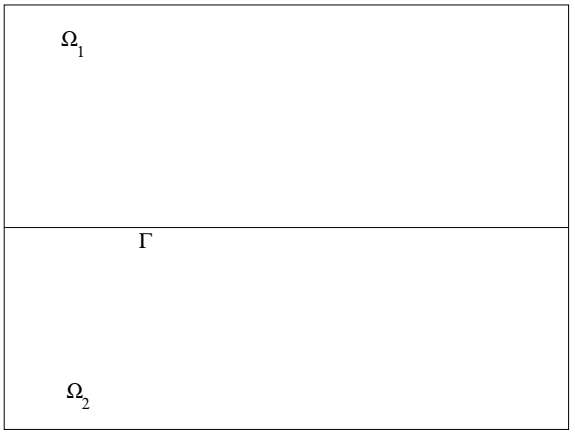


FIGURE 3.1 – Décomposition en deux sous-domaines sans recouvrement

3.2.3 Extension à n sous-domaines

La méthode de Schur est extensible à n sous-domaines Ω_i, de frontière $\partial\Omega_i$, numérotés de 1 à n. Pour chaque sous-domaine Ω_i, la frontière avec ses voisins est notée Γ_i, $\Gamma_i \subset \partial\Omega_i$ et $\bigcup_i \Gamma_i = \Gamma$. La figure 3.2 illustre ces notations.

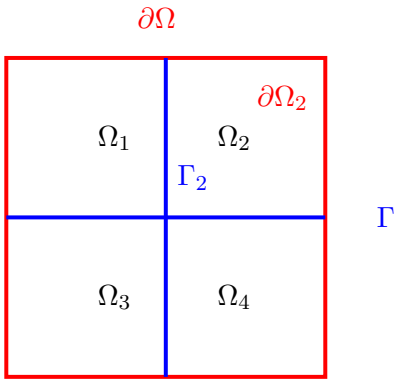


FIGURE 3.2 – Décomposition en quatre sous-domaines sans recouvrement

En numérotant les éléments de Ω en commençant par ceux propres aux sous-domaines (dans $\Omega_i \bigcup \partial\Omega_i \setminus \Gamma_i$), puis en prenant en compte les intersections de Γ, on obtient une matrice de la forme

$$\begin{bmatrix} A_{11} & & 0 & \overline{A_{10}} \\ & \ddots & & \\ & & A_{ii} & \overline{A_{i0}} \\ 0 & & \ddots & \\ & & & A_{nn} \quad \overline{A_{n0}} \\ \overline{A_{10}}^T & \overline{A_{i0}}^T & \overline{A_{n0}}^T & A_{00} \end{bmatrix} \tag{3.6}$$

Si A est inversible, alors les matrices A_{ii}, pour $i = 1 \ldots n$ le sont aussi. Le complément de Schur correspondant à cette matrice s'écrit alors

$$S = A_{00} - \sum_{i=1}^{n} \overline{A_{i0}}^T A_{ii}^{-1} \overline{A_{i0}}$$

où A_{ii} représente la relation entre les inconnues dans Ω_i, A_{00} les relations entre les inconnues sur toutes les interfaces et $\overline{A_{i0}}$ la relation entre Ω_i et les interfaces. En définissant R_i comme l'opérateur de restriction de l'espace associé à toutes les arêtes interfaces vers celui associé aux arêtes interfaces de Γ_i, on a $\overline{A_{i0}} = A_{i0}R_i$. Les matrices R_i sont de taille $n_0^{(i)} \times n_0$, $n_0^{(i)}$ étant le nombre d'arêtes intersections sur la frontière Γ_i et n_0 le nombre total d'arêtes intersections.

Avec une partition en deux sous-domaines, $\overline{A_{i0}} = A_{i0}$ et $\overline{A_{00}^{(i)}} = A_{00}^{(i)}$.

Il est possible de décomposer A_{00} en n sous-matrices $\overline{A_{00}^{(i)}}$, $i = 1 \dots n$, chaque sous-matrice correspondant à la contribution du sous-domaine i au bloc des intersections. On a alors

$$A_{00} = \sum_{i=1}^{n} R_i^T A_{00}^{(i)} R_i = \sum_{i=1}^{n} \overline{A_{00}^{(i)}}$$

On introduit la notion de complément de Schur local associé au sous-domaine i, noté S_i :

$$S_i = A_{00}^{(i)} - A_{i0}^T A_{ii}^{-1} A_{i0}.$$

Le complément de Schur s'écrit alors

$$S = \sum_{i=1}^{n} R_i^T S_i R_i \ . \tag{3.7}$$

Si A est une matrice symétrique définie positive, alors les A_{ii} le sont aussi et le complément de Schur S aussi. Les compléments de Schur locaux ne le sont pas forcément. Si S_i n'est pas inversible, Ω_i est appelé un sous-domaine flottant [70, p. 88].

3.2.4 Résolution du système de Schur

Si la matrice A est symétrique définie positive, alors son complément de Schur S est aussi une matrice symétrique définie positive. On peut alors utiliser les méthodes présentées à la section 2.1. Nous allons voir que la méthode du Gradient Conjugué est la méthode à privilégier.

Solveur direct

Afin de pouvoir résoudre le système associé au complément de Schur (3.2) avec un solveur direct, il est nécessaire de pouvoir factoriser la matrice S. Cette factorisation impose de construire explicitement S. Il faut alors inverser chaque A_{ii} ou résoudre le système $A_{ii}X = A_{i0}$, avec X une matrice rectangulaire de largeur $n_0^{(i)}$. Il faut ensuite calculer chaque S_i puis effectuer la somme (3.7) et enfin factoriser S. De plus, à cause du remplissage dans les A_{ii}^{-1} la matrice S est une matrice dense.

AMG

Pour pouvoir appliquer la méthode AMG au complément de Schur, il est nécessaire, comme pour le solveur direct, de construire S explicitement. Nous allons voir qu'en utilisant l'algorithme du Gradient Conjugué, cela n'est pas nécessaire.

3.2.5 Application du Gradient Conjugué

Dans l'algorithme 1 du chapitre 2, page 30, la matrice A n'est utilisée à chaque itération que pour le calcul de q_k, en effectuant un produit matrice-vecteur. Le produit correspondant lorsque l'on applique le Gradient Conjugué au complément de Schur est donné par l'algorithme 2.

Algorithm 2 Produit $q = Sp$

for all Ω_i, $i = 1 \dots n$ **do**
  $p_i = R_i p$
  $u_i = A_{i0} p_i$
  Solve $A_{ii} v_i = u_i$
  $y_i = A_{i0}^T v_i$
  $q_i = R_i^T (A_{00}^{(i)} p_i - y_i)$
end for
$q = \sum_{i=1}^{n} q_i$

Le produit Sp ne nécessite donc pas de calculer S de façon explicite. Effectuer le produit de S_i par un vecteur revient à résoudre un problème dans le sous-domaine Ω_i avec des conditions aux limites de Dirichlet à l'interface Γ_i. Ces problèmes étant indépendants, leur résolution peut se faire en parallèle. Il faut aussi effectuer des produits matrice/vecteur.

Comme le Gradient Conjugué appliqué au complément de Schur converge en moins de $\sum_{i=1}^{n} n_0^{(i)}$ itérations, il est préférable de ne pas calculer S.

3.3 Préconditionnement local

3.3.1 Préconditionnement de Neumann-Neumann

Le conditionnement du complément de Schur dépend du pas de maillage utilisé lors de la discrétisation du problème. Nous pouvons le voir simplement sur un exemple. En considérant un modèle de Poisson, discrétisé avec la méthode aux éléments finis standards, si la taille du maillage est h et le nombre de sous-domaines n alors $\kappa(S) = O(\frac{n}{h})$ [53, p. 631]. Il est intéressant de supprimer la dépendance à la taille du maillage en préconditionnant S.

Le préconditionnement de Neumann-Neumann (NN) [69] permet de supprimer cette dépendance. Dans le cas du modèle de Poisson, avec deux sous-domaines, on a alors $\kappa(M_{NN}^{-1} S) = O(1)$ [53, p. 618], avec M_{NN}^{-1} la matrice du préconditionnement.

L'idée directrice de ce préconditionnement est d'approcher S^{-1} par la somme des S_i^{-1}. Cette approximation n'est possible que si S_i est inversible pour tout i. Comme les sous-domaines flottants donnent des S_i non inversibles, on ne peut prendre directement la somme des inverses.

On pose

$$M_{NN}^{-1} = D \sum R_i^T S_i^{\dagger} R_i D$$

où D est une matrice diagonale servant à pondérer. Les valeurs de cette matrice correspondent à l'inverse du nombre de sous-domaines auxquels appartient l'arête correspondante. La matrice $S_i^{\dagger}$, symétrique définie positive, est définie par $S_i^{\dagger} = S_i^{-1}$ si S_i est inversible, ce qui correspond à un sous-domaine i non flottant. Si S_i n'est pas inversible,

$S_i^\dagger$ est telle que $(S_i^\dagger)^{-1}$ approche S_i. Autrement dit, $S_i^\dagger$ peut être la pseudo-inverse de S_i ou l'inverse d'une matrice proche de S_i et inversible.

Il est nécessaire de pouvoir appliquer le préconditionnement sans avoir à construire explicitement les compléments de Schur locaux S_i.

Dans l'algorithme du gradient conjugué préconditionné, appliquer le préconditionnement consiste à calculer $z = M_{NN}^{-1} r$. Cette opération est détaillée dans l'algorithme 3.

Algorithm 3 Produit $z = M_{NN}^{-1} r$

for all Ω_i, $i = 1 \dots n$ **do**
  $r_i = R_i D r$
  $z_i = S_i^\dagger r_i$
end for
$z = D \sum_{i=1}^{n} R_i^T z_i$

Soit la matrice B_i définie par

$$B_i = \begin{pmatrix} A_{ii} & A_{i0} \\ A_{i0}^T & A_{00}^{(i)} \end{pmatrix} \tag{3.8}$$

On pose $r_i = R_i D r$, $z_i = S_i^\dagger r_i$ et on a alors $z = D \sum R_i^T z_i$. Dans le cas où S_i est inversible, calculer $z_i = S_i^{-1} r_i$ revient à résoudre le système $S_i z_i = r_i$, ce qui équivaut au problème suivant dans Ω_i, avec des conditions de Neumann sur Γ_i [33, p. 42] :

$$B_i \begin{pmatrix} x_i \\ z_i \end{pmatrix} = \begin{pmatrix} A_{ii} & A_{i0} \\ A_{i0}^T & A_{00}^{(i)} \end{pmatrix} \begin{pmatrix} x_i \\ z_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix} \tag{3.9}$$

En effet la première ligne du système (3.9) donne

$$x_i = -A_{ii}^{-1} A_{i0} z_i$$

En remplaçant x_i dans la seconde ligne de (3.9) on obtient :

$$z_i = (A_{00}^{(i)} - A_{i0}^T A_{ii}^{-1} A_{i0})^{-1} r_i = S_i^{-1} r_i$$

Utiliser cette approche pour appliquer le préconditionnement de Neumann-Neumann permet de ne pas avoir à construire explicitement les compléments de Schur locaux S_i mais de résoudre un problème local dans Ω_i. L'application du préconditionnement correspond alors à l'algorithme 4.

Algorithm 4 Application de Neumann-Neumann

for all Ω_i, $i = 1 \dots n$ **do**
  $r_i = R_i D r$
  Solve $B_i \begin{pmatrix} x_i \\ z_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix}$
end for
$z = D \sum_{i=1}^{n} R_i^T z_i$

Si la matrice S_i n'est pas inversible, le système à résoudre dépend de l'approximation utilisée pour $S_i^\dagger$.

3.3.2 Neumann-Neumann et sous-domaines flottants

Pour pouvoir mettre en œuvre le préconditionnement de Neumann-Neumann, il faut pouvoir traiter le cas des sous-domaines flottants. Il est possible d'utiliser la pseudo-inverse de la matrice S_i, mais c'est une solution coûteuse. Nous avons préféré une autre approche, introduisant une erreur plus importante que la méthode de la pseudo-inverse, mais ayant une complexité plus faible.

Puisqu'ils sont singuliers, les systèmes associés aux sous-domaines flottants possèdent une infinité de solutions. Dans le cas d'un sous-domaine connexe qui n'est connecté à aucun bord de Dirichlet, la matrice A_{ii} est irréductible et la matrice B_i a un noyau de dimension 1. Pour obtenir une matrice inversible, il suffit d'imposer une condition de Dirichlet arbitraire en un point de $\Gamma \bigcap \partial\Omega_i$ [42]. La condition que l'on impose arbitrairement modifie le système local que l'on résout. Comme le calcul du préconditionnement est déjà une approximation, il n'est pas gênant de faire une erreur supplémentaire, dans la mesure où le préconditionnement reste efficace.

Pour imposer la condition, on modifie la dernière ligne non nulle de la matrice du système (3.9) ainsi que la colonne correspondante pour garder la symétrie, afin d'avoir uniquement un 1 sur la diagonale. On obtient alors l'approximation de B_i

$$\tilde{B}_i = \begin{pmatrix} A_{ii} & \tilde{A}_{i0} & 0 \\ \tilde{A}_{i0}^T & \tilde{A}_{00}^{(i)} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Le complément de Schur local associé à $\tilde{B}_i$ est noté $\tilde{S}_i$.

$\tilde{B}_i$ est inversible et une fois cette transformation effectuée, on calcule z_i par la résolution de

$$\tilde{B}_i \begin{pmatrix} x_i \\ z_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix}$$

Cette solution au problème des matrices non inversibles correspondant aux sous-domaines flottants est simple à mettre en oeuvre et ne présente pas de surcoût calculatoire. Une autre solution, utilisant une modification explicite de rang 1 de la matrice est décrite dans [26].

3.4 Interprétation physique

Nous considérons un problème elliptique sur un domaine Ω connexe. Des conditions aux limites sont appliquées sur la frontière $\partial\Omega$ du domaine. Ces conditions sont de type Neumann sur $\partial_N\Omega$ et de type Dirichlet sur $\partial_D\Omega$, avec $\partial\Omega = \partial_N\Omega \bigcup \partial_D\Omega$.

Le domaine Ω est partitionné en sous-domaines Ω_i, $i = 1 \dots n$. La frontière entre sous-domaines est notée Γ. La frontière du sous-domaine Ω_i est notée $\partial\Omega_i$. On a $\partial\Omega_i \bigcap \Gamma \neq \emptyset$, $\forall i$, $i = 1 \dots n$.

3.4.1 Produit par S_i

Lors du produit par S_i (Algorithme 2), le système $A_{ii}v_i = u_i$ est résolu. Ce système correspond à celui que l'on obtient en considérant le sous-domaine Ω_i avec les conditions aux limites héritées du système global sur sa frontière $\partial\Omega_i \setminus \Gamma$ et des conditions de Dirichlet sur $\partial\Omega_i \bigcap \Gamma$.

On retrouve ce système dans le calcul des x_i une fois x_0 connu. Ce calcul, détaillé pour deux sous-domaines dans les équations (3.4), s'écrit $x_i = A_{ii}^{-1}(b_i - A_{i0}x_0)$, $i \in \{1, 2\}$ et

revient à résoudre le système local au sous-domaine i en ayant sur Γ des conditions aux limites de type Dirichlet, de valeur x_0.

Dans le cas d'un calcul d'écoulement, x_i est une charge et b_i est un flux. Le produit par S permet de passer de la charge sur Γ au flux sur Γ.

3.4.2 Application de Neumann-Neumann

Lorsque l'on résout le système (3.9), cela revient à résoudre le système associé au sous-domaine Ω_i avec les conditions aux limites héritées du système global sur sa frontière $\partial\Omega_i \backslash \Gamma$ et des conditions de Neumann sur $\partial\Omega_i \bigcap \Gamma$. Si $\partial\Omega_i \bigcap \partial_D\Omega = \emptyset$, nous nous retrouvons donc avec un problème comportant uniquement des conditions au bord de type Neumann. Comme nous cherchons à caractériser la charge, ce système est sous-contraint et le nombre de solutions est infini. Si une solution particulière est déterminée, toute variation constante sur l'ensemble du domaine est aussi une solution.

Le nom du préconditionnement, Neumann-Neumann, correspond au cas avec 2 sous-domaines, où l'on résout un problème de Neumann sur chaque sous-domaine.

3.4.3 Gradient Conjugué

L'application du Gradient Conjugué au complément de Schur peut alors s'interpréter en termes de charges et de flux sur les intersections.

Nous avons deux conditions de continuité à respecter, pour la charge et pour le flux.

Les inconnues, correspondant au vecteur x_0, sont la charge sur Γ. En construisant le système, comme il n'y a qu'une inconnue par arête intersection, nous avons imposé la continuïté de la charge. L'application du Gradient Conjugué, en minimisant le résidu $r_0 = Sx_0 - c_0$ va permettre d'imposer la continuité des flux.

A chaque itération, de nouvelles approximations du flux r_0 et de la charge x_0 sont calculées. En appliquant le préconditionnement de Neumann-Neumann, on cherche la charge correspondant au flux r_0 que l'on vient de calculer. on obtient une nouvelle approximation de la charge, z_0. A partir de ces approximations, une direction de descente pour la charge, p_0, est calculée, qui permet, lors de l'itération suivante, de calculer les nouvelles approximations.

3.5 Factorisation de Cholesky

3.5.1 Accélération du produit par S

A chaque itération de PCG, pour effectuer le produit par S, on résout les systèmes locaux avec A_{ii}. Pour accélérer ces résolutions, on peut effectuer une factorisation de Cholesky en amont. Cette factorisation est possible quel que soit le sous-domaine concerné. La matrice A étant symétrique définie positive, tous les blocs internes A_{ii} sont eux-mêmes symétriques définis positifs. Afin de minimiser le remplissage dans le facteur de Cholesky, il faut utiliser une permutation P_{ii} (section 2.1.1, page 27), et on a $A_{ii} = P_{ii}^T L_{ii} L_{ii}^T P_{ii}$. Le produit par S correspond alors à l'algorithme 5.

3.5.2 Accélération du préconditionnement

A chaque itération de PCG, on résout aussi les problèmes locaux avec B_i si Ω_i n'est pas flottant (resp. $\tilde{B}_i$ s'il est flottant) pour appliquer le préconditionnement de Neumann-Neumann. Cette résolution peut être accélérée de la même façon, en factorisant B_i (resp.

Algorithm 5 Produit $q = Sp$

for all Ω_i, $i = 1 \dots n$ **do**
$p_i = R_i p$
$u_i = P_{ii} A_{i0} p_i$
Solve $L_{ii} t_i = u_i$
Solve $L_{ii}^T v_i = t_i$
$y_i = A_{i0}^T P_{ii}^T v_i$
$q_i = R_i^T (A_{00}^{(i)} p_i - y_i)$
end for
$q = \sum_{i=1}^{n} q_i$

$\tilde{B}_i$) sous la forme $B_i = P_i^T L_i L_i^T P_i$ (resp. $\tilde{B}_i = P_i^T \tilde{L}_i \tilde{L}_i^T P_i$). L'application du préconditionnement se fait alors par l'algorithme 6.

Algorithm 6 Application de Neumann-Neumann

for all Ω_i, $i = 1 \dots n$ **do**
$r_i = R_i D r$
$\overline{r}_i = P_i \begin{pmatrix} 0 \\ r_i \end{pmatrix}$
if Ω_i non floating **then**
Solve $L_i \begin{pmatrix} t_i \\ y_i \end{pmatrix} = \overline{r}_i$
Solve $L_i^T \overline{z}_i = \begin{pmatrix} t_i \\ y_i \end{pmatrix}$
else
Solve $\tilde{L}_i \begin{pmatrix} t_i \\ y_i \end{pmatrix} = \overline{r}_i$
Solve $\tilde{L}_i^T \overline{z}_i = \begin{pmatrix} t_i \\ y_i \end{pmatrix}$
end if
$\begin{pmatrix} x_i \\ z_i \end{pmatrix} = P_i^T \overline{z}_i$
end for
$z = D \sum_{i=1}^{n} R_i^T z_i$

3.5.3 Mutualisation de la factorisation

Si l'on effectue les deux factorisations indépendamment, on fait deux fois la factorisation des blocs A_{ii}. Nous proposons d'éviter de faire deux fois cette factorisation en mutualisant. Pour mutualiser les factorisations, il faut factoriser la matrice B_i en s'assurant que la structure des blocs est respectée par la permutation P_i. En prenant

$$P_i = \begin{pmatrix} P_{ii} \\ P_{00}^{(i)} \end{pmatrix}$$

on obtient $B_i = P_i^T L_i L_i^T P_i$ avec

$$L_i = \begin{pmatrix} L_{ii} & 0 \\ L_{i0}^T & L_{00}^{(i)} \end{pmatrix}$$

On peut extraire la factorisation $A_{ii} = P_{ii}^T L_{ii} L_{ii}^T P_{ii}$ de celle de B_i. De la même façon, en respectant la structure de $\tilde{B}_i$ on peut extraire L_{ii} de $\tilde{L}_i$, puisque

$$\tilde{L}_i = \begin{pmatrix} L_{ii} & 0 \\ \tilde{L}_{i0}^T & \tilde{L}_{00}^{(i)} \end{pmatrix}$$

De plus, lorsque la factorisation de B_i respecte la structure, elle fournit la factorisation de S_i. En effet, comme

$$\begin{pmatrix} L_{ii} & 0 \\ L_{i0}^T & L_{00}^{(i)} \end{pmatrix} \begin{pmatrix} L_{ii}^T & L_{i0} \\ 0 & L_{00}^{(i)T} \end{pmatrix} = \begin{pmatrix} A_{ii} & A_{i0} \\ A_{i0}^T & A_{00}^{(i)} \end{pmatrix}$$

alors

$$L_{00}^{(i)} L_{00}^{(i)^T} + L_{i0} L_{i0}^T = A_{00}^{(i)}$$

et

$$A_{i0} = L_{ii} L_{i0}^T$$

d'où

$$\begin{aligned} L_{00}^{(i)} L_{00}^{(i)^T} &= A_{00}^{(i)} - A_{i0}^T L_{ii}^{-T} L_{ii}^{-1} A_{i0} \\ &= A_{00}^{(i)} - A_{i0}^T A_{ii}^{-1} A_{i0} \\ &= S_i \end{aligned}$$

On peut écrire simplement la même succession d'opérations en ajoutant les permutations P_{ii} et $P_{00}^{(i)}$. Nous les avons omises pour alléger l'écriture.

De même, dans le cas d'un sous-domaine flottant, $\tilde{L}_{00}^{(i)}$ correspond au facteur de Cholesky de $\tilde{S}_i$. Il est possible d'utiliser la factorisation de S_i (respectivement $\tilde{S}_i$) pour résoudre directement $S_i z_i = r_i$ (respectivement $\tilde{S}_i z_i = r_i$), ce qui conduit à l'algorithme 7.

Algorithm 7 Application de Neumann-Neumann avec mutualisation

for all Ω_i, $i = 1 \dots n$ **do**
  $r_i = P_{00}^{(i)} R_i D r$
  if Ω_i non floating **then**
    Solve $L_{00}^{(i)} y_i = r_i$
    Solve $L_{00}^{(i)T} z_i = y_i$
  else
    Solve $\tilde{L}_{00}^{(i)} y_i = r_i$
    Solve $\tilde{L}_{00}^{(i)T} z_i = y_i$
  end if
end for
$z = D \sum_{i=1}^{n} R_{00}^{(i)T} P_{00}^{(i)T} z_i$

La contrainte sur la permutation peut la rendre moins efficace. Il est donc nécessaire de vérifier que le remplissage supplémentaire induit par cette méthode n'annule pas le

gain obtenu par la mutualisation. L'étude de la complexité des deux versions est faite au chapitre 4, page 66.

Il est aussi possible d'utiliser cette factorisation pour calculer le produit par S_i, dans le cas d'un sous-domaines non-flottant. Utiliser la factorisation pour calculer le produit par S_i n'est intéressant que si le nombre d'éléments non nuls dans L_{ii} est supérieur au nombre d'éléments non nuls de $L_{00}^{(i)}$. Dans le cas contraire, il est plus intéressant de calculer le produit par S_i sous sa forme décomposée.

Il est à noter que, même avec la mutualisation, le complément de Schur n'est pas construit explicitement. Les sous-domaines flottants imposent une approximation qui interdit de le construire à partir de la factorisation des compléments locaux. Même en l'absence de sous-domaines flottants, il faudrait, pour le construire, effectuer le produit $L_{00}^{(i)} L_{00}^{(i)T}$ pour chaque sous-domaine, puis effectuer la somme. Il resterait encore à factoriser la matrice obtenue pour résoudre le problème à l'interface.

3.5.4 Algorithmes

Les deux possibilités pour la factorisation conduisent à deux versions de l'application du gradient conjugué préconditionné au complément de Schur S. L'algorithme 8 présente la structure générale de l'algorithme, commune aux deux versions. L'algorithme 9 contient les procédures communes et les algorithmes 10 et 11 présentent les procédures qui diffèrent pour chaque version.

Algorithm 8 Gradient Conjugué Préconditionné et complément de Schur

FACTORIZE() ▷ Pre-processing

define x_{init} ▷ Initialization
$r =$INITIALRESIDUAL(x_{init})

$p =$NEUMANNNEUMANN(r)
$\beta = r^T p$

repeat ▷ Iterations
  $q =$LOCALSCHURPRODUCT(p)
  $\delta = p^T q$
  $\alpha = \frac{\beta}{\delta}$
  $x_0 = x_0 + \alpha p$
  $r = r - \alpha q$
  $z =$NEUMANNNEUMANN(r)
  $\beta_{tmp} = \beta$
  $\beta = r^T z$
  $p = p + \frac{\beta}{\beta_{tmp}} z$
until convergence or nbit > max

$x =$EXTENSION(x_0) ▷ Solution extension

Algorithm 9 Procédures communes

procedure InitialResidual(x_{init})
  for all Ω_i, $i = 1 \dots n$ **do**
    $u_i = P_{ii}(b_i - A_{i0}R_i x_{init})$
    Solve $L_{ii}t_i = u_i$
    Solve $L_{ii}^T v_i = t_i$
    $y_i = A_{i0}^T P_{ii}^T v_i$
    $r_i = (b_0^{(i)} - A_{00}^{(i)} R_i x_{init}) - y_i$
  end for
  $r = \sum_{i=1}^{n} R_i^T r_i$
  return r
end procedure

procedure LocalSchurProduct(p)
  for all Ω_i, $i = 1 \dots n$ **do**
    $p_i = R_i p$
    $u_i = P_{ii}A_{i0}p_i$
    Solve $L_{ii}t_i = u_i$
    Solve $L_{ii}^T v_i = t_i$
    $y_i = A_{i0}^T P_{ii}^T v_i$
    $q_i = R_i^T(A_{00}^{(i)} p_i - y_i)$
  end for
  $q = \sum_{i=1}^{n} q_i$
  return q
end procedure

procedure Extension(x_0)
  for all Ω_i, $i = 1 \dots n$ **do**
    $u_i = (b_i - A_{i0}R_i x_0)$
    Solve $L_{ii}t_i = u_i$
    Solve $L_{ii}^T x_i = t_i$
  end for
  $x = \begin{pmatrix} \sum_{i=1}^{n} R_{ii}^T x_i \\ x_0 \end{pmatrix}$
  return x
end procedure

Algorithm 10 Procédures pour la version sans mutualisation

procedure FACTORIZE( )
  for all Ω_i, $i = 1 \dots n$ **do**
    extract A_{ii}
    if non floating **then**
      Compute and store P_i and P_{ii} from B_i
      Compute and store L_i and L_{ii} from B_i
    else
      Compute $\tilde{B}_i$ from B_i
      Compute and store P_i and P_{ii} from $\tilde{B}_i$
      Compute and store L_i and L_{ii} from $\tilde{B}_i$
    end if
  end for
end procedure

procedure NEUMANNNEUMANN(r)
  for all Ω_i, $i = 1 \dots n$ **do**
    $r_i = R_i D r$
    $\overline{r}_i = P_i \begin{pmatrix} 0 \\ r_i \end{pmatrix}$
    if Ω_i non floating **then**
      Solve $L_i \begin{pmatrix} t_i \\ y_i \end{pmatrix} = \overline{r}_i$
      Solve $L_i^T \overline{z}_i = \begin{pmatrix} t_i \\ y_i \end{pmatrix}$
    else
      Solve $\tilde{L}_i \begin{pmatrix} t_i \\ y_i \end{pmatrix} = \overline{r}_i$
      Solve $\tilde{L}_i^T \overline{z}_i = \begin{pmatrix} t_i \\ y_i \end{pmatrix}$
    end if
    $\begin{pmatrix} x_i \\ z_i \end{pmatrix} = P_i^T \overline{z}_i$
  end for
  $z = D \sum_{i=1}^{n} R_i^T z_i$
  return z
end procedure

Algorithm 11 Procédures pour la version avec mutualisation

procedure FACTORIZE( )
  for all Ω_i, $i = 1 \dots n$ **do**
    if non floating **then**
      Compute and store $P_i = \begin{pmatrix} P_{ii} \\ P_{00}^{(i)} \end{pmatrix}$ from B_i
      Compute L_i from B_i
    else
      Compute $\tilde{B}_i$
      Compute and store $P_i = \begin{pmatrix} P_{ii} \\ P_{00}^{(i)} \end{pmatrix}$ from $\tilde{B}_i$
      Compute L_i from $\tilde{B}_i$
    end if
    Extract and store L_{ii}
    Extract and store $L_{00}^{(i)}$
  end for
end procedure

procedure NEUMANNNEUMANN(r)
  for all Ω_i, $i = 1 \dots n$ **do**
    $r_i = P_i R_i D r$
    if Ω_i non floating **then**
      Solve $L_{00}^{(i)} y_i = r_i$
      Solve $L_{00}^{(i)T} z_i = y_i$
    else
      Solve $\tilde{L}_{00}^{(i)} y_i = r_i$
      Solve $\tilde{L}_{00}^{(i)T} z_i = y_i$
    end if
  end for
  $z = D \sum_{i=1}^{n} R_i^T P_{00}^{(i)T} z_i$
  return z
end procedure

3.6 Préconditionnement global

Plus le nombre de sous-domaines augmente, moins le préconditionnement de Neumann-Neumann est efficace. Comme on approche l'inverse d'une somme par la somme des inverses, plus le nombre de termes est important, plus l'erreur commise est importante. Mais lorsque l'on augmente le nombre de sous-domaines, la taille des systèmes pour chaque sous-domaine diminue. Le remplissage engendré par la factorisation est alors plus faible, ce qui permet d'obtenir un nombre total d'éléments non nuls plus petit qu'avec un faible nombre de sous-domaines. Le parallélisme est aussi plus important, puisque chaque système local peut être résolu indépendamment de ses voisins, à chaque itération.

Il faut donc considérer un autre préconditionnement qui permet de limiter cet inconvénient. Il existe plusieurs préconditionnements permettant d'atteindre cet objectif, en utilisant une information transversale à tous les sous-domaines. On parle de préconditionnement global.

Cette section présente trois préconditionnements utilisant une réduction de la taille du problème initial, appliqués au complément de Schur. Nous utilisons une approche algébrique de ces méthodes, basée sur l'étude de Nabben et Vuick [56, 71]. Nous définissons dans un premier temps la construction du système réduit, puis nous présentons successivement la grille grossière, le balancing et la déflation.

3.6.1 Système réduit

Tous ces préconditionnements utilisent une réduction du problème initial. Cette réduction correspond à une restriction de l'espace du domaine à un sous-espace.

Pour effectuer cette restriction il faut définir Z une base d'un sous-espace de $\mathbb{R}^{n_0}$ de dimension m, avec n_0 la taille de S et m la taille souhaitée pour le système réduit. La restriction S_c de S sur l'espace $Im(Z)$ s'écrit

$$S_c = Z^T S Z \ .$$

S est inversible et Z est une base, donc est de rang plein, donc S_c est inversible.

La solution projetée sur $Im(Z)$ est alors calculée par :

$$x_c = Z S_c^{-1} Z^T b \ .$$

Nous définissons la projection

$$P = I - S Z S_c^{-1} Z^T.$$

On a les propriétés suivantes pour P :
- $P^2 = P$
- $P^T = I - Z S_c^{-1} Z^T S$
- $PSZ = 0$
- $Z^T P = 0$
- $PS = SP^T = (PS)^T$
- $ker(PS) = Im(Z)$
- $Im(PS) = Im(Z)^{\perp}$.

3.6.2 Grille grossière additif

Ce préconditionnement correspond à un calcul de la solution réduite afin de corriger l'erreur à chaque itération.

Il s'écrit

$$M_{gg}^{-1} = M^{-1} + ZS_c^{-1}Z^T$$

M^{-1} peut éventuellement être l'identité si l'on ne veut pas combiner ce préconditionnement avec un autre.

3.6.3 Déflation

Dans [66] l'algorithme décrit est une variation du gradient conjugué appelée `AugCG`. Dans cet algorithme, la déflation est réalisée en appliquant P^TP comme préconditionnement. P^TP n'est pas inversible, mais nous pouvons choisir x_0 tel que $r_k \in Im(Z)^\perp$.

De plus,

$$Z^Tr = Z^T(Sx - b) = 0 \Leftrightarrow Pr = r$$

Si on a $Z^Tr_k = 0 \forall k \geqslant 0$, le préconditionnement peut être ramené à P^T. Pour imposer cette condition, il suffit de prendre x_0 tel que

$$Z^Tr_0 = Z^T(Sx_0 - b) = 0$$

Preuve : Nous allons montrer par récurrence que ces relations sont vraies $\forall k \geqslant 0$:

$$\begin{cases} PSp_k &= Sp_k \\ Z^Tr_k &= 0 \end{cases}$$

Nous avons, pour $k = 0$,

$$\begin{aligned} Z^Tr_0 &= 0 \\ PSp_0 &= PSP^TPr_0 \\ &= SP^TPr_0 \\ &= SP^Tr_0 \\ &= Sp_0 \end{aligned}$$

En supposant les propriétés vraies pour $k-1$, on obtient :

$$\begin{aligned} PSp_k &= PSz_k - \frac{\beta_{k-1}}{\beta_k}PSp_{k-1} \\ &= PSP^TPr_k - \frac{\beta_{k-1}}{\beta_k}Sp_{k-1} \\ &= SP^TPr_k - \frac{\beta_{k-1}}{\beta_k}Sp_{k-1} \\ &= Sp_k \end{aligned}$$

$$\begin{aligned} Z^Tr_k &= Z^Tr_{k-1} - \alpha_kZ^TSp_{k-1} \\ &= Z^Tr_{k-1} - \alpha_kZ^TPSp_{k-1} \\ &= 0 \end{aligned}$$

Pour garantir la propriété sur r_0 il suffit de prendre x_0 comme

$$\begin{aligned} x_0 &= x_{-1} + ZS_c^{-1}Z^T r_{-1} \\ &= ZS_c^{-1}Z^T(b - Sx_{-1}) \\ &= ZS_c^{-1}Z^T b + P^T x_{-1} \\ &= x_c + P^T x_{-1} \end{aligned}$$

Un choix classique est de prendre $x_{-1} = 0$. On a alors $x_0 = x_c$. On obtient un gain important au niveau de la convergence juste en imposant cette condition initiale, sans préconditionner les itérations (InitCG). D'après [66], ce gain se détériore lorsqu'il faut un grand nombre d'itérations pour atteindre la convergence. Effectuer la projection à chaque itération du gradient conjugué permet de ne pas avoir cette détérioration en corrigeant l'orthogonalité.

On peut aussi cumuler les avantages de la déflation avec un préconditionnement classique. Dans ce cas le préconditionnement à appliquer est $P^T M^{-1} P$. Avec un choix initial tel que $Z^T r_0 = 0$ on a $P^T M^{-1} P r = P^T M^{-1} r$.

3.6.4 Balancing

Cette méthode est présentée par Mandel [47], avec un choix de Z tel que $Ker(S_i) \subset Range(Z)$, pour $i = 1 \dots n$. D'un point de vue algébrique, elle s'écrit

$$M_b^{-1} = P^T M^{-1} P + ZS_c^{-1}Z^T$$

Elle correspond donc à une combinaison de la déflation et de la grille grossière additive.

Si l'on choisit x_0 tel que $Z^T r_0 = 0$, on a

$$Z^T r_k = 0, \forall k \geqslant 0$$

Il faut ajouter dans la démonstration du point précédent le terme correspondant à la grille grossière. On a alors

$$\begin{aligned} PSp_k &= PSz_k - \frac{\beta_{k-1}}{\beta_k} PSp_{k-1} \\ &= PS(P^T P r_k + ZS_c^{-1}Z^T r_k) - \frac{\beta_{k-1}}{\beta_k} Sp_{k-1} \\ &= SP^T P r_k + PSZS_c^{-1}Z^T r_k) - \frac{\beta_{k-1}}{\beta_k} Sp_{k-1} \\ &= SP^T P r_k - \frac{\beta_{k-1}}{\beta_k} Sp_{k-1} \\ &= Sp_k \end{aligned}$$

La partie de la preuve concernant $Zr_k = 0$ est inchangée.

Avec un tel choix pour x_0, le balancing est équivalent à la déflation.

3.6.5 Définition de l'espace réduit

Espace réduit associé aux sous-domaines

Nous utilisons la déflation comme préconditionnement global, Nabben et Vuik ayant montré que ce préconditionnement est plus efficace que la grille grossière [56]. Il est nécessaire de définir l'espace associé.

La déflation est souvent utilisée en définissant Z comme un ensemble de vecteurs propres associés à la matrice du système à résoudre. Plus précisément, Z correspond aux vecteurs propres associés aux plus petites valeurs propres de la matrice [66]. Ce choix garantit une diminution du conditionnement en éliminant les petites valeurs propres du spectre de la matrice. Par contre, calculer les valeurs propres est très coûteux.

Frank et Vuick analysent dans [35] un autre choix pour Z, correspondant à une signature de sous-domaines. Ce choix pour Z dans le cadre d'une méthode de sous-domaines a été introduit par Nicolaides[58] et Mansfield [49, 48].

Si le domaine Ω est décomposé en n sous-domaines sans recouvrement Ω_j, alors n vecteurs Z_j sont définis par :

$$Z_{i,j} = \begin{cases} 1, i \in \Omega_j \\ 0, i \notin \Omega_j \end{cases}$$

Chaque sous-domaine est réduit à une inconnue dans le système réduit. Ce choix pour Z, en effectuant un bon choix pour le découpage en sous-domaines, est efficace [35], lorsqu'il est appliqué au système associé au domaine Ω entier.

Application au complément de Schur

Nous voulons faire un choix similaire pour la déflation appliquée au complément de Schur. Nous avons donc défini une signature des sous-domaines dans le cadre du problème réduit aux interfaces. Dans la décomposition de Ω en n sous-domaines Ω_i sans recouvrement.

Nous définissons alors $\bar{Z}$ par

$$\begin{aligned} \bar{Z} &= [\bar{Z}_1, \dots \bar{Z}_n] \\ \bar{Z}_i &\in \mathbb{R}^{n_0} \\ \bar{Z}_{k,i} &= \begin{cases} \frac{1}{n_k}, k \in \Gamma_i \\ 0, i \notin \Gamma_i \end{cases} \end{aligned}$$

avec n_k le nombre de sous-domaines dont la frontière contient l'arête k. Avec cette définition de Z, les inconnues du système associé au complément de Schur et correspondant à un sous-domaine sont réduites à une seule inconnue.

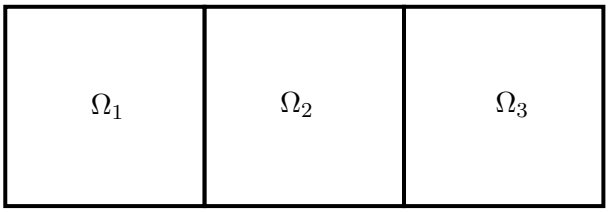


FIGURE 3.3 – Décomposition en trois sous-domaines d'un domaine 2D

En appliquant cette définition, il se peut que la matrice $\bar{Z}$ ne soit pas de rang plein. En effet, pour un découpage en 3 sous-domaines d'un domaine 2D (FIG : 3.3), on obtient

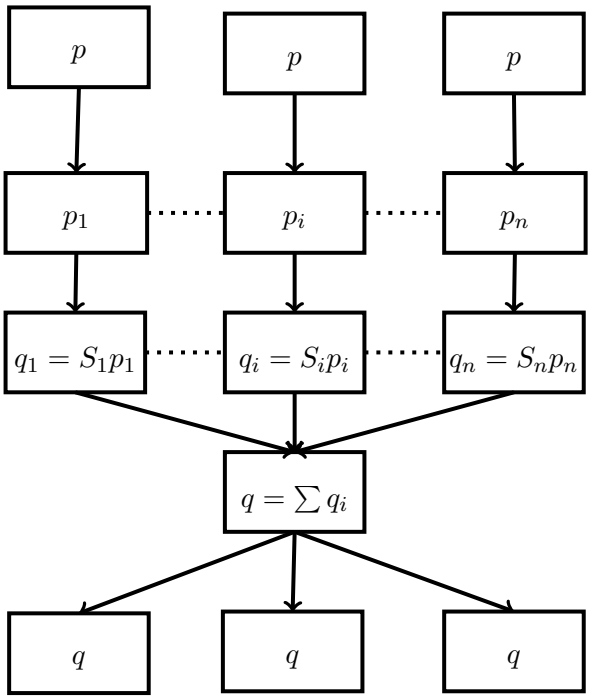


FIGURE 3.4 – Exemple de parallélisme pour le produit Sp

la matrice suivante

$$\bar{Z} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \vdots & \vdots & \vdots \\ \frac{1}{2} & \frac{1}{2} & 0 \\ & & \\ 0 & \frac{1}{2} & \frac{1}{2} \\ \vdots & \vdots & \vdots \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Dans ce cas $\bar{Z}$ est de rang $n-1$.

Pour définir Z en utilisant la signature des sous-domaines, il est nécessaire d'extraire de la matrice $\bar{Z}$ une matrice de rang plein.

Cette extraction peut se faire en utilisant une méthode d'orthogonalisation avec détection du rang. Le coût de ces méthodes augmente rapidement avec la taille de la matrice à orthogonaliser. Leur utilisation est donc pénalisante, et il est préférable de construire $\bar{Z}$ de telle sorte qu'elle soit directement de rang plein, en adaptant sa construction en fonction du découpage en sous-domaine. Par exemple, dans le cas d'un découpage en sous-domaines connectés deux à deux, il suffit de ne pas prendre en compte le vecteur correspondant au dernier sous-domaine.

La méthode du Gradient Conjugué préconditionné par Neumann-Neumann et par la déflation est décrite par l'algorithme 12.

3.7 Parallélisme

Les résolutions des systèmes locaux, lors du produit par S_i ou lors de l'application du préconditionnement de Neumann-Neumann, peuvent être faites indépendamment. Il est alors naturel d'écrire une version parallèle de la méthode.

Algorithm 12 Deflation et complément de Schur

procedure PROJECTION(z)
  $u = C^T z$
  Solve $L_c t = u$
  Solve $L_c^T v = t$
  $z = z - Zt$
  return z
end procedure

for all Ω_i, $i = 1 \dots n$ **do** ▷ Pre-processing
  if non floating **then**
    FACTORIZE(B_i)
  else
    FACTORIZE($\tilde{B}_i$)
  end if
end for
if defl **then** ▷ Coarse system construct
  $C =$LOCALSCHURPRODUCT(Z) ▷ Block operation
  $S_c = Z^T C$
  $L_c =$FACTORIZE(S_c)
end if
define x_{init} ▷ Initialization
$r =$INITIALRESIDUAL(x_{init})
if defl **then** ▷ modification of x_0 to have $Pr = r$
  $x_0 = x_{init} + Z L_c^{-T} L_c^{-1} Z^T r$
  $r =$INITIALRESIDUAL(x_0)
else
  $x_0 = x_{init}$
end if
$p =$NEUMANNNEUMANN(r)
if defl **then**
  $p =$PROJECTION(p)
end if
$\beta = r^T p$

Algorithm 12 Deflation et complément de Schur (Suite)

repeat ▷ Iterations
  $q =$LocalSchurProduct(p)
  $\delta = p^T q$
  $\alpha = \frac{\beta}{\delta}$
  $x_0 = x_0 + \alpha p$
  $r = r - \alpha q$
  $z =$NeumannNeumann(r)
  if defl **then**
    $z =$Projection(z)
  end if
  $\beta_{tmp} = \beta$
  $\beta = r^T z$
  $p = p + \frac{\beta}{\beta_{tmp}} z$
until convergence or nbit $>$ max

$x =$Extension(x_0) ▷ Solution extension

Le corps de l'algorithme 12 correspond à la partie séquentielle. Le parallélisme est contenu dans les procédures décrites dans les algorithmes 9, 10 et 11. Chacune de ces procédures correspond à une série d'opérations effectuées en parallèle, suivie d'une opération de synchronisation au niveau de la somme, permettant d'assembler le vecteur calculé sur l'ensemble des processus (sauf pour l'opération de factorisation qui ne nécessite pas de synchronisation).La figure 3.4 illustre ce parallélisme sur l'exemple du produit $q = Sp$.

Les opérations sur les sous-domaines peuvent être faites en parallèle ou de façon séquentielle, ce qui permet une approche hybride en attribuant plusieurs sous-domaines à chaque processus. Avec p processus et n sous-domaines, on peut avoir $1 \leq p \leq n$. Lorsque p est superieur à 1 et inférieur à n, cela permet une plus grande latitude pour répartir la charge, même si les sous-domaines ne correspondent pas à une partition ou chaque sous-domaine et de taille égale.

Il est nécessaire de centraliser les résultats à plusieurs reprises lors d'une itération de PCG. Ces points de synchronisation imposent une taille minimale sur chaque processus pour avoir des performances intéressantes. Cette taille minimale peut être atteinte en regroupant plusieurs sous-domaines sur un même processus. Ce choix peut s'avérer pertinent, comme nous le verrons dans l'étude de complexité appliquée au problème de l'écoulement en milieu fracturé (section 5.3.3, page 85). En augmentant le nombre de sous-domaines, on diminue le temps nécessaire pour effectuer la factorisation, même si le nombre de processus n'augmente pas. Il est nécessaire de trouver un équilibre entre le nombre d'itérations et la vitesse de factorisation.

Chapitre 4

Bibliothèque logicielle Schur ; analyse de la complexité et du parallélisme

4.1 Introduction

A notre connaissance, une bibliothèque logicielle permettant d'utiliser la méthode du complément de Schur pour résoudre un système d'équations n'existe pas. Le logiciel `MUMPS`[3] permet de calculer le complément de Schur et la factorisation partielle de la matrice qui a permis de le calculer. L'utilisation de `MUMPS` impose de calculer le complément de Schur de façon explicite, or nous avons vu que cela n'était pas souhaitable. Il serait possible de décomposer le problème en sous-matrices, une par sous-domaine, et de les factoriser successivement pour ne calculer que les compléments de Schur locaux. Mais pour pouvoir appliquer le préconditionnement de Neumann-Neumann, il serait nécessaire de factoriser ce complément de Schur local. L'algorithme serait ensuite équivalent à la version avec mutualisation de la factorisation. Cette façon de faire compliquerait la gestion des sous-domaines flottants, puisqu'il faudrait modifier les compléments de Schur locaux correspondant avant de les factoriser pour pouvoir appliquer le préconditionnement.

La bibliothèque `HIPS`[2] s'approche du solveur que nous voulons utiliser, mais utilise une factorisation ILU comme préconditionnement et non un préconditionnement de Neumann-Neumann.

Nous avons donc implémenté l'algorithme 12, décrit au chapitre 3 : le Gradient Conjugué appliqué au complément de Schur, préconditionné par Neumann-Neumann et déflation.

Deux versions ont été développées, une utilisant Matlab, l'autre écrite en `C++` pour pouvoir être utilisée au sein de la plate-forme `H2oLab`. Ces deux versions sont identiques d'un point de vue fonctionnel.

4.1.1 Données en entrée

L'utilisateur doit fournir le problème déjà décomposé en sous-domaines. En effet, comme la bibliothèque a vocation à être utilisée pour résoudre des problèmes liés à des applications physiques, il est utile de pouvoir exploiter les propriétés physiques du problème pour réaliser la partition en sous-domaines. Il est donc préférable de séparer la phase de partitionnement de celle de résolution. Ce travail préliminaire est donc laissé à la charge de l'utilisateur, qui peut utiliser un partitionneur de graphe généraliste, ou découper le problème en utilisant des connaissances sur la physique à la source des équations.

Il faut fournir les matrices A_i associées aux sous-domaines. Ces matrices correspondent aux matrices B_i, définient à l'équation (3.8), page 46, étendue pour obtenir des matrices ayant la taille du système complet. Afin de pouvoir construire les opérateurs de restriction/extension $E_i = \begin{pmatrix} R_{ii} \\ R_i \end{pmatrix}$, permetant d'obtenir les B_i à partir des A_i, il faut fournir la limite entre les éléments internes aux sous-domaines et les intersections. La matrice R_i, présentée précédemment (section 3.2.3, page 43), correspond à l'opérateur de restriction entre l'espace associé à toutes les intersections et celui associé aux intersections du sous-domaine Ω_i. La matrice R_{ii} correspond elle au même opérateur, mais pour les espaces associés aux arêtes internes.

En notant A la matrice du système, b le second membre, A_i et b_i les matrices et seconds membres associés au sous-domaine i, le découpage de la matrice doit respecter ces propriétés :

- $E_i A_i E_i^T = B_i$,
- $\sum_i A_i = A$ et $\sum_i b_i = b$,
- dans A, comme dans chaque A_i les éléments correspondant à la partie interne aux sous-domaines doivent être séparés des éléments intersection, et être numérotés en premier. La matrice obtenue doit être une matrice en forme de flèche, comme celle présentée au chapitre 3 (Eq : 3.6).

4.1.2 Description fonctionnelle

Les systèmes locaux sont chargés, puis factorisés. On a vu dans le chapitre précédent qu'il était nécessaire de calculer des permutations pour réduire le remplissage dans les facteurs de Cholesky. Si l'on utilise la version classique de l'implémentation, sans mutualiser la factorisation pour le produit par S_i et l'application du préconditionnement de Neumann-Neumann, il faut calculer deux permutations par sous-domaine. Ces permutations sont obtenues en utilisant `AMD`[9]. Dans le cas où la version mutualisant la factorisation est privilégée, il faut calculer une permutation par sous-domaine. Cette permutation doit respecter la structure de la matrice B_i. Nous utilisons dans ce cas `CAMD` qui permet de contraindre la permutation en définissant des groupes. Si un système possède un noyau de dimension supérieur à 1, la matrice B_i correspondante ne peut être factorisée en appliquant la modification présentée dans le chapitre précédent, section 3.3.2. Dans le cas où la matrice B_i est séparable, la factorisation peut aussi échouer, la modification locale ne suffisant pas à rendre la matrice inversible. Le processus de résolution s'arrête et le numéro du système correspondant est renvoyé à l'utilisateur, afin qu'il puisse revoir la partition du problème initial.

Les intersections sont analysées pour en déduire la matrice de pondération D, introduite dans la section 3.3.1, qui est utilisée pour le préconditionnement de Neumann-Neumann. Si la déflation doit être utilisée, la signature des sous-domaines est construite au même moment puisqu'elle utilise les mêmes données. Les itérations du gradient conjugué sont appliquées jusqu'à convergence et la solution est retournée à l'utilisateur.

4.1.3 Choix du solveur direct

Nous avons choisi d'utiliser la bibliothèque `CHOLMOD`[18], disponible au sein de la suite d'outils `SUITESPARSE`[8]. Cette bibliothèque est écrite en `C`, ce qui simplifie son utilisation avec `H2oLab`, la plate-forme étant écrite en `C/C++`.

De plus, il est possible, avec `CHOLMOD`, de décomposer les étapes de la résolution du système. Nous pouvons donc calculer la factorisation de la matrice et l'utiliser avec plu-

sieurs seconds membres. Cette fonctionnalité permet de calculer la factorisation dans la phase de préparation de l'algorithme 12, et de s'en servir à chaque itération.

Un argument supplémentaire pour l'utilisation de `CHOLMOD` est la présence, dans `SUITESPARSE`, de deux bibliothèques permettant de calculer les permutations, `AMD` et `CAMD`.

Enfin, `CHOLMOD`, `AMD` et `CAMD` sont disponibles à la fois avec une interface pour un code en `C` et pour un code Matlab.

4.2 Implémentation avec un langage orienté Objet

4.2.1 Choix du langage

Les outils utilisés dans `SuiteSparse` sont écrits en `C`, tandis que les logiciels de la bibliothèque `H2oLab` sont écrits en `C++`.

Pour des raisons évidentes de simplicité, nous avons le choix entre écrire `SolveurSchur` en `C` ou en `C++`. L'utilisation de `C++` permet d'utiliser une programmation orientée objet. La notion d'objet permet de faciliter le développement en définissant des entités aux propriétés et fonctions distinctes. La possibilité d'utiliser l'héritage de classe et la surcharge de méthode permet en outre de faciliter l'intégration efficace de la bibliothèque dans un projet existant, tout en simplifiant la mise en oeuvre. En effet, le chargement des matrices est fait par défaut en utilisant la lecture de fichiers proposée par `CHOLMOD`. La conversion depuis le format interne au projet vers le format demandé par `CHOLMOD` se fait alors en surchargeant les méthodes adéquates. Ceci permet de limiter le nombre de conversions entre les formats, sans imposer une charge de développement trop importante. Et en fonction des données connues lors de la conversion, celle-ci pourra être optimisée par l'utilisateur.

4.2.2 Objet *Subdomain*

Une instance de classe *Subdomain* est créée pour chaque sous-domaine. Sur ces instances sont appelées les méthodes correspondant aux étapes locales de la préparation et de la résolution.

Lors de la phase de préparation, chaque système associé à un sous-domaine doit être converti au format utilisé par `CHOLMOD`. Les matrices locales destinées à être factorisées (B_i et A_{ii} pour la version classique, B_i pour la version avec mutualisation) sont converties au format Compact Sparse Row (CSR). De plus, lors de cette conversion, il faut construire les opérateurs de restriction et d'extension nécessaires pour passer du sytème global au système local. Les outils de lecture de matrice au format matrice market sont fournis dans `CHOLMOD`, ce qui permet de lire facilement des fichiers standards. Dans le cadre d'une inclusion dans un projet plus complexe, il sera préférable de réaliser la conversion et les constructions des opérateurs directement depuis le format utilisé en amont, en surchargeant les méthodes concernées.

Le second membre, le vecteur contenant l'approximation initiale (si nécessaire) et celui qui contient la solution sont stockés en plein.

Les matrices et les facteurs de Cholesky utilisés dans la bibliothèque `CHOLMOD` sont stockés en utilisant des structures `C`. Nous avons pu utiliser directement les données stockées pour effectuer des transformations ou des copies par blocs, afin d'accélérer le code. La documentation de `CHOLMOD` a permis d'exploiter au mieux ces possibilités.

Une fois que les matrices sont converties au bon format, la factorisation peut être faite et les facteurs mémorisés pour permettre les résolutions nécessaires.

Les résolutions des systèmes locaux interviennent dans différentes étapes, correspondant chacune à des méthodes de la classe *Subdomain* :

- la construction de c_0 (EQ : 3.3, page 42), correspondant à la restriction du second membre au système aux interfaces
- le produit par S_i (ALG : 5, page 49)
- l'application de Neumann-Neumann, correspondant suivant la version choisie à une résolution d'un système avec B_i (ALG : 6, page 49) ou avec S_i (ALG : 7, page 50)
- l'extension de la solution x_0 obtenue pour le problème réduit aux interfaces au sous-domaine complet, pour former x_i.

Aucune des méthodes de la classe n'utilise des données externes aux instances, ce qui permet un passage aisé au parallélisme. Cette classe fait fortement appel aux fonctions de la bibliothèque `CHOLMOD` et accède directement à certaines données à l'intérieur des structures `C`. Ainsi, lors de la mutualisation de la factorisation, le facteur correspondant au bloc interne est extrait du facteur global. Plus exactement, seuls les indices des éléments non nuls sont copiés. Le reste de la structure est identique et n'est pas dupliqué en mémoire. Le bloc correspondant aux intersections est aussi extrait, et, si le nombre d'éléments dans ce bloc est supérieur au nombre d'éléments dans le facteur interne, il est aussi converti en matrice creuse afin de pouvoir l'utiliser pour le produit par S_i.

4.2.3 Objet *SolveurSchur*

Le coeur algorithmique, correspondant au corps de l'algorithme 12, est contenu dans la classe *SolveurSchur*. Dans cette classe sont définies les méthodes globales au système, comme celle permettant de réaliser une itération de PCG, ou encore celle permettant de construire le système grossier nécessaire à la déflation. Cette classe est aussi fortement liée à `CHOLMOD`, tant par l'appel de fonction que par l'accès aux données, notamment pour la construction du système grossier.

L'utilisation d'un langage orienté objet permet une plus grande souplesse lors de l'intégration du module dans le logiciel utilisateur. Grâce au mécanisme d'héritage, il est possible d'utiliser la bibliothèque dans un projet, quel que soit le format de matrice utilisé, en limitant les coûts de conversion. Il suffit de surcharger les méthodes qui permettent de construire les opérateurs de restriction tout en convertissant les systèmes au format CSR.

4.3 Analyse de la complexité

Dans cette section, nous effectuons une analyse de la compléxité, afin d'estimer les coûts, en nombre d'opérations flottantes, des différentes opérations. Nous définissons les formules qui permettrons, dans le chapitre suivant, de calculer une compléxité empirique. Les compléxités sont détaillées pour la verison avec mutualisation et pour la version sans.

4.3.1 Notations

Soient
- n_0 le nombre d'arêtes intersections dans le domaine complet ;
- $n_0^{(i)}$ (resp. n_{ii}) le nombre d'arêtes intersections, *ie* appartenant à Γ (resp. internes, *ie* appartenant à $\Omega_i \bigcup \partial\Omega_i \setminus \Gamma$) dans le sous-domaine Ω_i ;
- n_i la taille de la matrice B_i (nombre d'arêtes total dans le sous-domaine $\Omega_i \bigcup \partial\Omega_i$) ;
- n_A la taille de la matrice A ;
- n le nombre de sous-domaines ;
- $nnz(A)$ le nombre d'éléments non nuls dans la matrice A.

4.3.2 Coût de stockage

Pour chaque sous-domaine, il faut stocker la matrice B_i pour en effectuer la factorisation. Si l'on n'utilise pas la mutualisation, il faut également stocker la matrice A_{ii}. Il est aussi nécessaire de mémoriser le vecteur permutation, pour chaque matrice factorisée. Pour calculer le produit par le complément de Schur, il faut la matrice $A_{00}^{(i)}$ et la matrice A_{i0}. De plus, il faut pour chaque sous-domaine avoir les opérateurs de restriction et d'extension, permettant une correspondance entre la numérotation sur le sous-domaine et la numérotation globale. Ces opérateurs sont mémorisés sous la forme de deux vecteurs par sous-domaines. Les tailles des différents éléments sont rassemblées dans le tableau 4.1. Avec la version sans mutualisation, il y a moins de matrices à mémoriser. Mais le remplissage de L_i peut faire perdre cet avantage.

matrice	taille sans mutualisation	taille avec mutualisation
A_{ii}	$O(nnz(A_{ii}))$	-
B_i	$O(nnz(B_i))$	$O(nnz(B_i))$
P_i	n_i	n_i
P_{ii}	n_{ii}	-
$A_{00}^{(i)}$	$O(nnz(A_{00}^{(i)}))$	$O(nnz(A_{00}^{(i)}))$
A_{i0}	$O(nnz(A_{i0}))$	$O(nnz(A_{i0}))$
L_i	$O(nnz(L_i))$	$O(nnz(L_i))$
L_{ii}	$O(nnz(L_{ii}))$	-
loc $\rightarrow$ glob	n_i	n_i
glob $\rightarrow$ loc	n_A	n_A

TABLE 4.1 – Matrices à stocker pour pouvoir appliquer l'algorithme 12, pour un sous-domaine Ω_i

4.3.3 Produit par S : nombre d'opérations

Le produit entre une matrice colonne $V \in \mathbb{R}^{n_0 \times m}$ et le complément de Schur S, sans utiliser sa factorisation explicite a un coût total de :

$$(n_0 + \sum_{i=1}^{n}(nnz(A_{00}^{(i)}) + 2nnz(A_{i0}) + 2nnz(L_{ii}))) \times m \times cste$$

Ce coût est identique, que l'on mutualise la factorisation pour le préconditionnement ou non, au nombre d'éléments non nuls dans le facteur L_{ii} près. Le détail des coûts est formulé en détail dans l'algorithme 13.

Dans la version avec mutualisation de la factorisation, il est possible d'utiliser $L_{00}^{(i)}$ pour calculer le produit par le complément de Schur d'un sous-domaine non flottant. Dans ce cas, le coût du produit est alors de $2nnz(L_{00}^{(i)})$. Pour simplifier l'étude de complexité, cette version du calcul ne sera pas prise en compte dans la suite. Comme la factorisation n'est utilisée que si elle coûte moins cher que la forme développée, les coûts présentés sont des majorations du coût réel.

4.3.4 Neumann-Neumann

L'application du préconditionnement de Neumann-Neumann nécessite de résoudre deux systèmes triangulaires par sous-domaine et d'effectuer deux produits par une ma-

Algorithm 13 Coût du produit $q = SV$

for all Ω_i, $i = 1 \dots n$ **do** ▷ Coûts :
  $V_i = R_{i0} V$
  $u_i = P_{ii} A_{i0} V_i$ ▷ $nnz(A_{i0}) \times m \times cste$
  Solve $L_{ii} t_i = u_i$ ▷ $nnz(L_{ii}) \times m \times cste$
  Solve $L_{ii}^T v_i = t_i$ ▷ $nnz(L_{ii}) \times m \times cste$
  $y_i = A_{i0}^T P_{ii}^T v_i$ ▷ $nnz(A_{i0}) \times m \times cste$
  $q_i = R_{i0}^T (A_{00}^{(i)} V_i - y_i)$ ▷ $nnz(A_{00}^{(i)}) \times m + n_0^{(i)} \times m \times cste$
end for
$q = \sum_{i=1}^{n} q_i$ ▷ $n_0 \times m \times cste$

trice diagonale (Alg : 6, page 49 et 7, page 50). Les deux produits par D coûtent $2n_0$. Le coût des résolutions dépend de la version utilisée.

Dans le cas de la version avec mutualisation, les résolutions correspondant à l'application de $L_{00}^{(i)-T} L_{00}^{(i)-1}$, coûtent $2nnz(L_{00}^{(i)})$.

Dans le cas de la version sans mutualisation, les résolutions, correspondant à l'application de $L_i^{-T} L_i^{-1}$, coûtent $2nnz(L_i)$.

Soit au total

$$2n_0 + 2\sum_{i=1}^{n} nnz(L_{00}^{(i)})$$

pour la version avec mutualisation et

$$2n_0 + 2\sum_{i=1}^{n} nnz(L_i)$$

pour la version sans.

4.3.5 Déflation

L'application de la déflation correspond au produit $z = Pz$. Les étapes nécessaires pour calculer ce produit et les coûts associés sont indiqués dans l'algorithme 14. Le coût par itération de la déflation est de

$$nnz(Z) + nnz(SZ) + n^2$$

Algorithm 14 Coût de la projection Pr

procedure Projection(r)
  $u = C^T r$ ▷ $nnz(C) = nnz(SZ)$
  Solve $L_c t = u$ ▷ $\frac{n^2}{2}$
  Solve $L_c^T v = t$ ▷ $\frac{n^2}{2}$
  $z = z - Zt$ ▷ $nnz(Z)$
  return z
end procedure

4.3.6 Préparation

Lors de la préparation, il faut effectuer n factorisations de Cholesky sur des matrices de tailles n_i. Si l'on ne mutualise pas la factorisation, il faut aussi effectuer n factorisations sur des matrices de tailles n_{ii}. Le coût de la factorisation est proche du nombre final d'éléments non nuls, ce qui conduit à un coût de l'ordre de $nnz(L_i)$ pour la version avec mutualisation et de l'ordre de $nnz(L_i) + nnz(L_{ii})$ pour la version sans. Comme le remplissage peut être très différent d'une version à l'autre, il faudra comparer les temps expérimentaux de factorisation (comparaison faite dans le chapitre suivant pour l'application aux réseaux de fractures).

Il faut aussi calculer le second membre réduit c_0 (Eq : 3.3), ce qui coûte

$$\sum_{i=1}^{n}(n_{ii} \times n_0^{(i)} + 2nnz(L_{ii})) + n_0$$

Si l'on prévoit d'utiliser la déflation, il faut calculer la matrice de restriction Z et le système grossier. Si la matrice Z est de rang plein, son calcul est peu coûteux. S'il faut extraire une matrice de rang plein pour former Z, le coût augmente fortement. Nous utilisons actuellement une méthode de Gram-Schmidt modifiée pour cette extraction, qui a un coût en $O(n_0 n^2)$. Nous supposerons dans la suite, sauf indication contraire, que cette extraction n'est pas nécessaire.

Le calcul du système grossier est alors fait avec un produit entre le complément de Schur et la matrice Z, puis un produit entre le résultat et la matrice Z^T. D'après le paragraphe 4.3.3, si Z est de taille n, le coût de ces produits est de

$$\sum_{i=1}^{n}(nnz(A_{00}^{(i)}) + 2nnz(A_{i0}) + 2nnz(L_{ii})) \times n$$

Ce système grossier est factorisé sous la forme $L_c L_c^T$, opération ayant un coût négligeable. La taille du système grossier est égale au nombre de sous-domaines. La factorisation à alors un coût majoré par n^3 (coût de la factorisation pour une matrice pleine).

La projection est ensuite appliquée, ce qui coûte

$$nnz(Z) + nnz(SZ) + n^2$$

comme indiqué au paragraphe 4.3.5.

4.3.7 Itération

En utilisant les coûts décrits précédemment, on obtient les coûts suivants pour chaque itération du Gradient Conjugué, avec ou sans préconditionnement :

Sans préconditionnement :

$$n_0 + \sum_{i=1}^{n}(nnz(A_{00}^{(i)}) + 2nnz(A_{i0}) + 2nnz(L_{ii}))$$

Avec Neumann-Neumann :

Il faut différencier deux cas, suivant que l'on a mutualisé la factorisation ou non.
Avec mutualisation de la factorisation

$$3n_0 + \sum_{i=1}^{n}(nnz(A_{00}^{(i)}) + 2nnz(A_{i0}) + 2nnz(L_{ii})) + 2nnz(L_{00}^{(i)})$$

Sans

$$3n_0 + \sum_{i=1}^{n}(nnz(A_{00}^{(i)}) + 2nnz(A_{i0}) + 2nnz(L_{ii})) + 2nnz(L_i)$$

La déflation rajoute deux produits par Z et Z^T ainsi que la résolution du système grossier. Ces opérations sont négligeables devant les résolutions nécessaires pour effectuer le produit par le complément de Schur et l'application du préconditionnement de Neumann-Neumann.

4.4 Version Parallèle

Chaque itération du Gradient Conjugué nécessite de résoudre des systèmes locaux aux sous-domaines. Le produit par S et l'application de Neumann-Neumann sont décomposables en opérations indépendantes, que l'on peut faire en parallèle, comme on l'a décrit dans le chapitre précédent, page 59.

4.4.1 Modèle de programmation choisi

Le modèle de programmation est un modèle SPMD (Single Process, Multiple Data). Les mêmes instructions sont exécutées en parallèle par tous les processus, mais avec des données différentes. Ce type de programmation permet d'exploiter les architectures à mémoire distribuée, comme les grilles de calcul ou les super-calculateurs. Les opérations d'échange de données utilisent MPI.

Les opérations réalisables de façon indépendante sur chaque processus, c'est-à-dire le calcul de r_i lors de l'initialisation et celui de q_i et z_i ensuite, sont concentrées dans les méthodes de la classe *Subdomain*. Ces opérations correspondent aux opérations les plus coûteuses effectuées lors d'une itération. La répartition de la charge sur les différents processus est effectuée en amont, par l'utilisateur qui peut ainsi exploiter les informations supplémentaires qu'il possède sur le problème physique. Les opérations séquentielles, correspondant à l'algorithme 12 (page 60) sont contenues dans la classe *SolveurSchur*. De même, c'est dans les méthodes de cette classe que sont effectuées toutes les opérations de communication entre les différents processus.

Deux choix étaient possibles pour les opérations séquentielles. Soit assembler les données nécessaires sur un processus maître qui effectue ensuite les opérations et distribuer les résultats sur l'ensemble des processus. Soit distribuer les données sur l'ensemble des processus puis effectuer en parallèle les mêmes opérations sur chacun. C'est cette deuxième solution que nous avons choisie. Dans une optique de calcul sur une architecture à mémoire distribuée, ce choix est pertinent, puisqu'il évite une phase de synchronisation supplémentaire, à la fin des calculs séquentiels. Il serait nécessaire de le revoir dans le cadre d'une exécution sur une architecture à mémoire partagée.

4.4.2 Distribution des données

Dans le cas d'un préconditionnement par déflation, le système grossier associé (*ie* L_c) est présent sur tous les processus. Ce système est construit sur chaque processeur lors de la phase d'initialisation. Sa taille étant réduite, son stockage local ne pose pas de problème. Pour pouvoir appliquer la projection correspondant à la déflation, il est aussi nécessaire d'avoir l'opérateur de restriction Z, ainsi que le produit SZ. Ces derniers sont donc également présents sur tous les processus.

Les vecteurs et scalaires correspondants aux résultats intermédiaires d'une itération du Gradient Conjugué sont aussi présents sur l'ensemble des processus.

Les sous-domaines sont répartis sur les processus. Chaque processus ne possède alors que les matrices locales correspondant aux sous-domaines qui lui sont attribués. Ces matrices sont :

- A_{ii}, la matrice correspondant aux arêtes internes au sous-domaine Ω_i ;
- $A_{00}^{(i)}$, la matrice correspondant aux arêtes intersections du sous-domaine Ω_i ;
- A_{i0}, la matrice correspondant aux interactions entre les arêtes internes au sous-domaine et les arêtes intersections ;
- L_{ii}, le facteur de Cholesky de la matrice A_{ii} ;
- $L_{00}^{(i)}$ ou L_i, suivant que l'on utilise ou non la mutualisation de la déflation, qui correspond au facteur de Cholesky utilisé pour appliquer le préconditionnement de Neumann-Neumann.

4.4.3 Communications

Une fois les systèmes locaux résolus, une phase de communication est nécessaire afin d'effectuer la somme des vecteurs locaux. Cette opération correspond à l'envoi d'un vecteur de taille n_0 de tous les processus vers tous les processus.

Il y a, dans chaque itération, deux opérations de synchronisation. Ces opérations sont effectuées à la fin du calcul du produit $q = Sp$ et à la fin de l'application du préconditionnement de Neumann-Neumann.

Lors de la phase d'initialisation, il est nécessaire de faire la synthèse de deux vecteurs de taille n_0. Il est, de plus, nécessaire de faire la synthèse de deux autres vecteurs de même taille ainsi que d'une matrice de taille $n_0 \times n$ pour construire le système grossier et modifier les vecteurs initiaux pouvoir utiliser la déflation.

4.5 Application à un problème d'écoulement en milieu poreux

Nous avons utilisé le solveur développé pour résoudre un problème d'écoulement dans un milieu poreux homogène, en 2D. Ce problème simple à mettre en œuvre a permis de valider le code avant l'application au problème fracturé.

4.5.1 Description du problème

Nous modélisons un phénomène d'écoulement dans un domaine rectangulaire Ω. Les écoulements considérés suivent la loi de Darcy, qui s'écrit $K\nabla(h) = v$, où h est la charge (ou pression) et v la vitesse de l'eau. On considère en outre l'hypothèse de conservation de la masse à travers le domaine Ω qui s'exprime par $\nabla.(v) = 0$, ce qui donne finalement l'équation aux dérivées partielles

$$\nabla.(K(x,y)\nabla h(x,y)) = 0, \forall (x,y) \in \Omega \tag{4.1}$$

où $K(x,y)$ représente la perméabilité au point de coordonnées (x,y) et $h(x,y)$ la charge en ce même point.

On se place dans une situation correspondant à la section d'un tube. L'écoulement se fait de la gauche vers la droite, les bords supérieur et inférieur étant imperméables. Ce qui revient à fixer la pression sur les bords latéraux, et à imposer un flux nul sur les autres. Les conditions aux limites s'écrivent alors :

- condition de Dirichlet : $h(x,y) = h_i$ sur $\partial\Omega_{Di}$, $i = 1, 2$ et $h_1 > h_2$;

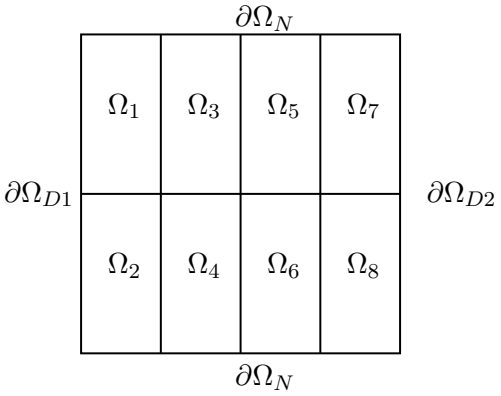


FIGURE 4.1 – Numérotation des bords et découpage en 8 sous-domaines

– condition de Neumann homogène : $\frac{\partial h(x,y)}{\partial n} = 0$, où n est la normale à la frontière au point (x, y) sur les frontières $\partial\Omega_N$.

Le domaine, un carré de côté 1, est discrétisé avec un maillage rectangulaire uniforme. Le système linéaire correspondant est obtenu avec des volumes finis.

4.5.2 Résultats

Nous avons effectué des tests en faisant varier le pas de maillage ($m = 1/256,\ 1/512,\ 1/768$) et le nombre de sous-domaines. Le domaine est découpé en grille. Le nombre de sous-domaines est fixé par le nombre de sous-domaines suivant l'axe horizontal (dom_i) et suivant l'axe vertical (dom_j).

La figure (4.1) représente le domaine Ω et la numérotation de ses frontières. Il est découpé en 8 sous-domaines avec $dom_i = 2$ et $dom_j = 4$.

Nous avons étudié l'impact de l'utilisation de Neumann-Neumann comme préconditionnement, ainsi que l'utilisation de la déflation conjuguée à Neumann-Neumann. Nous pouvons ainsi verifier que le choix que nous avons fait pour définir Z est efficace.

Les résultats présentés utilisent la version avec mutualisation de la factorisation. Les résultats sans mutualisation sont très proches pour ce cas test.

4.5.3 Efficacité des préconditionnements

Nous avons vu (section 3.3.1, page 45) que le préconditionnement de Neumann-Neumann permet de rendre le conditionnement indépendant du pas de maillage. Nous avons, pour les différentes combinaisons de pas de maillage et de partition en sous-domaines, relevé le nombre d'itérations nécessaires au Gradient Conjugué pour converger, avec et sans le préconditionnement de Neumann-Neumann. Les résultats obtenus sont rassemblés dans les tableaux 4.2a et 4.2b.

Sans le préconditionnement de Neumann-Neumann, le nombre d'itérations augmente lorsque l'on utilise un pas de maillage plus fin. Le préconditionnement permet d'éliminer totalement cet effet pour un faible nombre de sous-domaines (moins de 8 sous-domaines). Avec l'augmentation du nombre de sous-domaines, l'efficacité du préconditionnement se dégrade, le nombre d'itérations augmentant lorsque le pas de maillage diminue. Mais cette augmentation est nettement moins importante que celle obtenue sans préconditionnement.

Ces résultats mettent par ailleurs en lumière la dépendance introduite par Neumann-Neumann entre le nombre de sous-domaines et le conditionnement. Le nombre d'itérations

nécessaires pour converger est multiplié par un facteur de l'ordre de 2.5 entre 2 et 64 sous-domaines lorsque l'on n'active pas le préconditionnement. Avec le préconditionnement, ce facteur est de l'ordre de 40.

L'utilisation de la déflation comme préconditionnement, conjuguée à Neumann-Neumann, permet de limiter fortement l'augmentation du nombre d'itérations (Tab : 4.2c). Le facteur du paragraphe précédent est cette fois compris entre 8 et $12,5$.

4.5.4 Temps de calcul

Les préconditionnements de Neumann-Neumann et de déflation induisent un coût supplémentaire. Il faut donc effectuer un relevé des temps nécessaires pour obtenir la solution. Ces temps, pour un maillage avec un pas $m = 1/512$, sont rassemblés dans le tableau 4.3. Ils correspondent à une execution du code `C++` sur une machine sous linux, avec un processeur dual-core cadencé à $3GHz$.

Le préconditionnement de Neumann-Neumann seul est le plus efficace. La déflation, même si elle fait baisser le nombre d'itérations, introduit un coût de préparation (colonnes *prep*) trop important, sans apporter un gain conséquent au niveau du temps nécessaire pour effectuer les itérations (colonnes *iters*). Ce temps de préparation est du à l'orthogonalisation. Cette phase peut donc être améliorée en définissant une matrice Z de rang plein, comme indiqué dans la section 3.6.5.

4.6 Conclusion

Nous avons mis en œuvre la méthode du complément de Schur associée au préconditionnement de Neumann-Neumann et à la déflation. Les préconditionnements donnent les résultats attendus.

Si le nombre de sous-domaines est faible, le préconditionnement de Neumann-Neumann est à privilégier. Le surcoût induit par ce préconditionnement est largement contrebalancé par le gain en itérations. La déflation a un bilan plus mitigé. Si elle permet de réduire de façon importante le nombre d'itérations lorsque le nombre de sous-domaines augmente, elle ne permet pas de gagner du temps par rapport au préconditionnement de Neumann-Neumann seul. La construction de la matrice Z est trop coûteuse dans les cas étudiés.

Le chapitre suivant est consacré à l'étude de l'application de ces méthodes aux réseaux de fractures.

# i	# j	#dom	# it		
			256	512	768
2	1	2	96	135	165
2	2	4	129	176	214
4	1	4	140	189	230
2	4	8	164	225	274
4	2	8	150	207	252
8	1	8	177	244	297
2	8	16	200	280	339
4	4	16	177	250	304
8	2	16	186	259	314
4	8	32	216	297	362
8	4	32	208	294	359
8	8	64	242	338	413

(a) Préconditionnement : Aucun

# i	# j	#dom	# it		
			256	512	768
2	1	2	2	2	2
2	2	4	8	9	9
4	1	4	6	6	6
2	4	8	25	26	26
4	2	8	15	15	15
8	1	8	14	13	13
2	8	16	48	51	53
4	4	16	42	45	47
8	2	16	26	26	28
4	8	32	62	69	73
8	4	32	60	65	68
8	8	64	80	86	90

(b) Préconditionnement : Neumann-Neumann

# i	# j	#dom	# it		
			256	512	768
2	1	2	2	2	1
2	2	4	7	8	9
4	1	4	4	6	4
2	4	8	12	13	14
4	2	8	12	14	18
8	1	8	8	12	8
2	8	16	16	17	18
4	4	16	16	17	21
8	2	16	22	23	28
4	8	32	18	20	25
8	4	32	19	21	27
8	8	64	16	19	25

(c) Préconditionnement : Neumann-Neumann et déflation

TABLE 4.2 – Efficacité des préconditionnements pour un problème d'écoulement en milieu poreux :
Nombre d'itérations pour trois pas de maillage différents ($m = 1/256,\ 1/512,\ 1/768$) pour différentes partitions, sans et avec Neumann-Neumann puis avec déflation et Neumann-Neumann

#dom i	#dom j	#dom	Aucun			NN			NN+Defl		
			prep	iters	total	prep	iters	total	prep	iters	total
2	1	2	4	8	12	4	1	5	4	1	5
2	2	4	3	9	12	3	1	4	3	1	4
4	1	4	5	11	16	5	1	6	5	1	6
2	4	8	3	10	13	3	2	5	4	1	5
4	2	8	4	9	13	3	2	5	4	1	5
8	1	8	6	10	16	6	1	7	5	2	7
2	8	16	3	11	14	4	2	6	3	2	5
4	4	16	2	10	12	2	2	4	3	1	4
8	2	16	3	10	13	3	2	5	4	2	6
4	8	32	2	10	12	2	3	5	3	1	4
8	4	32	2	10	12	2	3	5	2	2	4
8	8	64	1	11	12	2	3	5	4	2	6

TABLE 4.3 – Comparaison des temps de calculs (en secondes) avec les différents préconditionnements

Chapitre 5

Applications de la méthode de Schur aux réseaux de fractures 3D

Nous allons dans ce chapitre étudier l'application de la méthode du complément de Schur, décrite dans les chapitres 3 et 4, au calcul de l'écoulement dans un réseau de fractures discret. Le réseau de fractures est modélisé, comme décrit dans le chapitre 1, par un ensemble d'ellipses. Nous considérons que l'écoulement n'est présent qu'au sein des fractures, la matrice rocheuse étant supposée imperméable. Seules celles-ci sont donc maillées, en utilisant une première discrétisation en 3D des frontières et des intersections, suivie d'une projection dans le plan des fractures.

Les résultats obtenus en utilisant les solveurs décrits au chapitre 2 ont conduit à utiliser la méthode du complément de Schur pour résoudre le système linéaire associé. En effet, nous avons vu que le solveur direct était très efficace pour les petits systèmes et que le Gradient Conjugué avait une complexité linéaire et permettait de résoudre tous les systèmes. La méthode du complément de Schur permet de cumuler les avantages de ces deux solveurs, en utilisant un solveur direct pour résoudre les systèmes locaux aux sous-domaines, et en appliquant le Gradient Conjugué pour résoudre le problème aux interfaces. Nous comparons les résultats obtenus avec cette méthode avec les résultats des solveurs précédents.

Nous réalisons dans un premier temps une étude détaillée sur un nombre limité de réseaux. Nous nous intéressons en particulier à la partition en sous-domaines. Ces tests vont également nous permettre de déterminer quelle est la version de l'algorithme à utiliser. Nous étudions les deux méthodes de factorisation et l'utilisation de la déflation.

5.1 Méthode de Schur appliquée à un réseau de fractures

5.1.1 Partition du réseau de fractures en sous-domaines

Chaque fracture peut être vue comme un sous-domaine. Les arêtes du maillage sont numérotées au niveau global, en commençant par les arêtes internes aux fractures et en finissant par les arêtes intersections. Les numéros des arêtes sont attribués fracture après fracture, ce qui conduit à une matrice flèche, comme la matrice représentée figure 5.1. Chaque bloc diagonal correspond à une fracture, les montants de la flèche correspondent aux intersections entre fractures. La similitude entre cette matrice et la matrice (3.6), page 43, est évidente. Nous pouvons donc appliquer la méthode du complément de Schur directement aux réseaux de fractures.

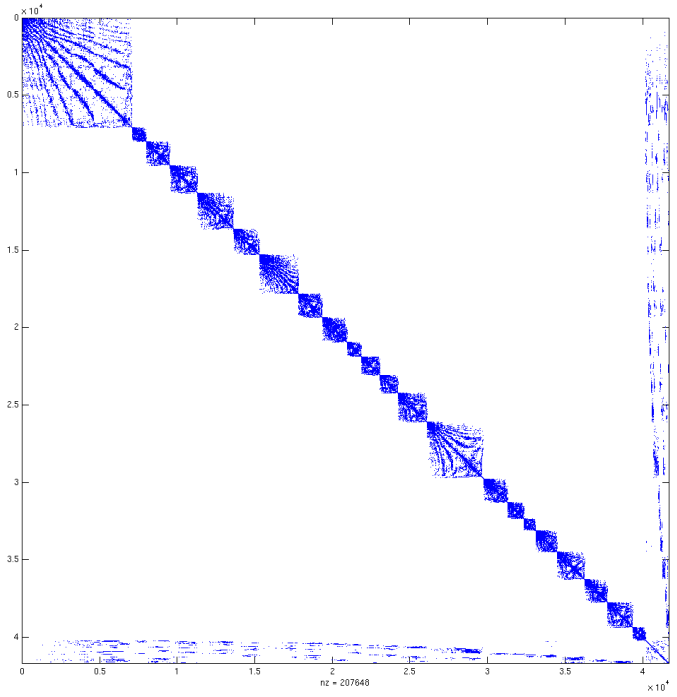


FIGURE 5.1 – Matrice issue d'un réseau de fractures

Cette partition a l'avantage de ne pas nécessiter de traitement supplémentaire. La méthode utilisée pour réaliser le maillage des fractures, et particulièrement la phase de correction après la projection, permet de garantir la connexité de chaque sous-domaine. Il est donc possible de traiter simplement les sous-domaines flottants comme proposé au chapitre 3, page 47. Les sous-domaines flottants correspondent dans ce cas aux fractures qui ne sont connectées à aucun bord du cube imposant une condition de type Dirichlet. Ces fractures sont appelées fractures flottantes.

Il n'est pas possible de maîtriser le nombre de sous-domaines avec ce choix de partition. Ceci peut devenir problématique lorsque le nombre de fractures augmente, puisque le nombre d'itérations du Gradient Conjugué augmente aussi. C'est pourquoi nous avons étudié comment regrouper plusieurs fractures au sein d'un même sous-domaine. Nous cherchons en effet à ne pas séparer une fracture dans plusieurs sous-domaines.

5.1.2 Plusieurs fractures par sous-domaine

Lorsque l'on effectue le regroupement de plusieurs fractures pour définir un sous-domaine, le traitement des fractures flottantes nécessite un soin particulier.

Le graphe d'un réseau est définit à partir des liens entre fractures. Chaque fracture du réseau correspond à un noeud du graphe, et il y a un arc entre deux noeuds si les fractures correspondantes sont connectées.

Lorsque le réseau est composé d'un seul amas de fractures, le graphe correspondant est connexe. Nous nous plaçons dans ce cas dans la suite.

Regrouper les fractures pour définir les sous-domaines revient alors à partitionner le graphe du réseau. Si l'on suppose que cette partition définit des parties connexes, alors les matrices B_i sont non séparables et leur noyaux sont de dimension inferieure ou égale à 1.

5.1.3 Base utilisée pour la déflation

La déflation est basée sur la définition d'un système libre Z utilisant la signature des sous-domaines, comme décrit dans la section 3.6.5, page 57. Le système Z est construit à partir de $\bar{Z}$ avec

$$\bar{Z}_{k,i} = \begin{cases} \frac{1}{n_k}, k \in \Gamma_i \\ 0, i \notin \Gamma_i \end{cases}$$

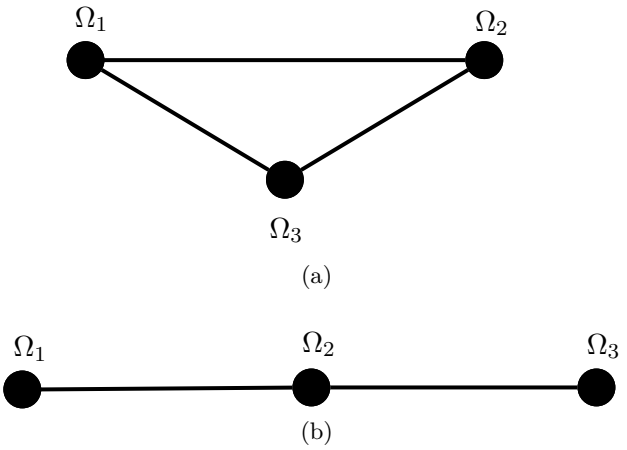


FIGURE 5.2 – Graphes de réseaux de fractures

Dans la plupart des cas, à cause de la présence de circuit dans le graphe de connexion des fractures, la matrice $\bar{Z}$ est de rang plein et $Z = \bar{Z}$. C'est le cas, par exemple, lorsque l'on a trois fractures se coupant deux à deux (FIG : 5.2a). La matrice $\bar{Z}$ obtenue est alors

$$\bar{Z} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \vdots & \vdots & \vdots \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ \vdots & \vdots & \vdots \\ 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \vdots & \vdots & \vdots \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

qui est une matrice de rang plein.

Pour certains réseaux $\bar{Z}$ peut être de rang $n-1$. En effet, trois fractures connectées entre elles sans former de boucle (FIG : 5.2b) conduisent à une matrice $\bar{Z}$ similaire à celle obtenue dans la section 3.6.5.

Mais $\bar{Z}$ peut aussi être de rang $n-2$ ou moins. La phase de projection peut rassembler en une seule intersection les intersections entre plusieurs fractures. Ainsi une intersection peut être commune à trois fractures.

Considérons un réseau de trois fractures se coupant au même endroit après projection,

on obtient la matrice $\bar{Z}$ suivante :

$$\bar{Z} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \vdots & \vdots & \vdots \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

Cette matrice est de rang $1 = n - 2$.

Lorsque la matrice $\bar{Z}$ est de rang plein, elle peut être utilisée directement. Mais lorsqu'elle n'est pas de rang plein, il est nécessaire d'en extraire une matrice de rang plein. Ce calcul peut être coûteux et risque de dégrader les performances du solveur. Il faut donc le réserver uniquement au cas où $\bar{Z}$ n'est pas de rang plein.

Afin de réduire les coûts, nous adoptons la stratégie suivante. Dans un premier temps, nous choisissons $Z = \bar{Z}$. Si la factorisation du système grossier $Z^T S Z$ échoue, c'est que la matrice $\bar{Z}$ n'était pas de rang plein. Dans ce cas, nous effectuons une factorisation QR avec détection de rang de $\bar{Z}$. La méthode utilisée doit permettre de trouver expérimentalement le rang de $\bar{Z}$, la matrice Q obtenue étant alors de rang plein. Nous posons ensuite $Z = Q$.

5.2 De la génération à la résolution

5.2.1 Présentation des étapes

La simulation de l'écoulement dans un réseau de fractures se fait en plusieurs étapes, décrites dans la suite de cette section. Certaines étapes sont spécifiques à la méthode de Schur.

- Le réseau est généré et le système linéaire associé est construit ;
- une méthode de résolution est choisie ;
- les fractures sont éventuellement regroupées en sous-domaines, le système linéaire est modifié en conséquence (renumérotation) ;
- le système est résolu.

5.2.2 Génération des réseaux et des systèmes linéaires

L'exécutable `MP_Frac_D3_Flow` de la plate-forme `H2oLab`, décrit dans la section 1.4, est utilisé pour générer les réseaux de fractures et les systèmes associés. Une fois le domaine et les fractures créés, celles-ci sont maillées et une matrice est construite pour chaque fracture. Ces matrices correspondent aux matrices $A_i = E_i^T B_i E_i$ présentées dans la section 4.1 page 63, avec $\sum A_i = A$. Ces matrices peuvent être utilisées directement, ou écrites sur le disque pour une utilisation ultérieure.

La fonctionnalité d'export de matrices présente dans la plate-forme `H2oLab` a été étendue afin de pouvoir exporter les matrices A_i, locales aux fractures. Un fichier par fracture est créé, dans lequel est enregistrée la matrice associée. Le graphe de connexion des fractures est aussi sauvegardé, ainsi que la limite entre arêtes internes et arêtes intersections.

Comme la génération des réseaux, la projection et le maillage sont effectués par du code séquentiel, la possibilité d'exporter les systèmes permet de générer les matrices sur une machine possédant beaucoup de mémoire vive, puis de réaliser la suite des calculs sur des machines parallèles. Cela permet aussi d'utiliser un cluster de Power6 [5], même si la bibliothèque `Cgal`, nécessaire à la génération des réseaux, n'est pas disponible sur cette plate-forme, pour des raisons techniques de compilation.

5.2.3 Partitions des fractures en sous-domaines

Comme on l'a vu, lorsque l'on regroupe les fractures, il est suffisant de partitionner le graphe de connexion des fractures en parties connexes afin de pouvoir traiter simplement les sous-domaines flottants. Cette partition est obtenue en utilisant le partitionneur de graphe `SCOTCH` [6]. Un post traitement sur les partitions, qui minimalise le nombre de liens, génère dans la majorité des cas des partitions connexes. Il est toujours théoriquement possible d'obtenir des partitions non connexes, mais nous n'avons pas trouvé de partitionneur permettant de garantir la connexité. Lors de nos tests, les partitions que nous avons obtenues étaient toutes connexes.

Une fois la partition calculée, la matrice correspondant au sous-domaine est construite en sommant les matrices de chaque fracture appartenant au sous-domaine. Il faut ensuite renuméroter la matrice afin de réintégrer dans le bloc interne de chaque sous-domaine les éléments qui sont des intersections entre deux fractures regroupées dans le même sous-domaine. Il est en effet nécessaire d'avoir d'abord les éléments internes au sous-domaine puis les éléments intersections. Ce traitement est actuellement fait en utilisant Matlab. Les matrices sont lues depuis les fichiers écrits par H2oLab, et les matrices construites sont enregistrées dans de nouveaux fichiers.

5.2.4 Résolution

La résolution peut être faite en utilisant le code Matlab, ou la bibliothèque en C++. Dans ce dernier cas, nous utilisons un utilitaire de la plate-forme, `D3_Flow_Solver`, qui permet de charger un système linéaire correspondant à un réseau de fractures, et d'effectuer la résolution dans les mêmes conditions qu'avec `MP_Frac_D3_Flow`. Nous pouvons ainsi tester l'utilisation de `SolveurSchur` dans la plate-forme.

Lorsque les matrices des sous-domaines sont analysées au début du processus de résolution, les matrices $\bar{Z}$ sont construites pour pouvoir utiliser la déflation.

5.3 Analyse détaillée sur quelques systèmes

5.3.1 Description des domaines utilisés

Nous avons effectué des tests en utilisant 3 réseaux de 300 fractures, avec des valeurs du paramètre α de la loi puissance régissant la distribution des longueurs de fractures valant 2.5, 3.5, et 4.5. Le réseau avec $\alpha = 3.5$ a été maillé avec deux pas m différents, $m = 0.05$ et $m = 0.08$. Les tailles de système générés sont indiquées dans le tableau 5.1.

α	$size(A)$	$nnz(A)$
2.5	393714	2366999
3.5 ($m = 0.05$)	832099	4440803
3.5 ($m = 0.08$)	296633	1646171
4.5	246479	1323379

TABLE 5.1 – Taille et nombre d'éléments non nuls des systèmes testés, pour différents paramètres α et plusieurs pas de maillage

Pour chaque réseau de fractures, nous avons effectué des regroupements pour faire varier le nombre de sous-domaines, allant de 2 à 300 sous-domaines. Les résultats obtenus sont détaillés et analysés dans la section suivante.

5.3.2 Partitionnement et taille des systèmes

Pour chacune des décompositions du réseau en sous-réseaux, nous avons relevé le nombre d'éléments non nuls dans chaque sous-domaine, ainsi que la taille des blocs importants. Les blocs pris en compte sont :

- le bloc A_{ii}, correspondant aux arêtes internes du sous-domaine Ω_i ;
- le bloc $A_{00}^{(i)}$, correspondant aux arêtes intersections du sous-domaine Ω_i ;
- le bloc A_{0i} (nombre d'éléments non nuls uniquement, la taille de ces blocs est donnée par celle des blocs $A_{00}^{(i)}$ et A_{ii}), correspondant aux interactions entre les arêtes internes et les arêtes intersections du sous-domaine Ω_i ;
- le bloc A_{00}, rassemblant les intersections entre tous les sous-domaines.

Pour chacun de ces blocs (sauf le bloc A_{00} qui est global), les résultats sont sommés sur l'ensemble des sous-domaines. Les résultats sont synthétisés dans les tableaux 5.2, 5.3, 5.4 et 5.5.

On peut voir dans ces tableaux que le regroupement en sous-domaines a un impact direct sur la taille des blocs intersections A_{00} et $A_{00}^{(i)}$. La taille du complément de Schur est égale à la taille du bloc A_{00} et la taille des compléments de Schur locaux est égale à celle des blocs $A_{00}^{(i)}$. Augmenter le nombre de sous-domaines augmente donc, logiquement, la taille du système à l'interface associé.

Le nombre total d'éléments non nuls dans les matrices locales B_i ($nnz(B_i) = nnz(A_{ii}) + 2nnz(A_{i0}) + nnz(A_{00}^{(i)})$) augmente aussi avec le nombre de sous-domaines. En effet, lorsque l'on regroupe des fractures au sein d'un même sous-domaine, les éléments diagonaux correspondants aux intersections entre ces fractures ne sont plus dédoublés dans deux sous-matrices, mais sommés dans la même, ce qui fait baisser le nombre total d'éléments non nuls.

# dom	$\sum size(A_{ii})$	$\sum size(A_{00}^{(i)})$	$size(A_{00})$
2	365862	55706	27853
4	352718	91295	40997
8	343460	120827	50255
16	337547	144692	56168
32	333804	162970	59911
64	331746	175700	61969
110	330681	183527	63034
300	329559	193573	64156

(a) Taille des blocs pour différentes partitions

# dom	$\sum nnz(A_{ii})$	$\sum nnz(A_{i0})$	$\sum nnz(A_{00}^{(i)})$	$nnz(A_{00})$
2	1752744	260257	135870	93741
4	1491480	365970	217813	143579
8	1316576	435237	283503	179949
16	1214351	474462	332524	203724
32	1153422	497325	368334	218927
64	1122940	508314	390942	227431
110	1107849	513766	404477	231618
300	1092545	519157	420571	236140

(b) Nombre d'éléments non nuls pour différentes partitions

TABLE 5.2 – Réseau de fractures avec $\alpha = 2.5$ et pas de maillage $m = 0.08$

# dom	$\sum size(A_{ii})$	$\sum size(A_{00}^{(i)})$	$size(A_{00})$
2	287185	18895	9448
4	281113	32396	15520
8	275539	46232	21094
16	271059	58225	25574
32	266900	70793	29733
64	264126	80068	32507
112	263001	85142	33632
300	261231	93350	35402

(a) Taille des blocs pour différentes partitions

# dom	$\sum nnz(A_{ii})$	$\sum nnz(A_{i0})$	$\sum nnz(A_{00}^{(i)})$	$nnz(A_{00})$
2	1457019	81979	37547	25194
4	1336709	133110	65706	43242
8	1235975	175323	92544	59550
16	1156221	208431	115771	73088
32	1085680	236773	140343	86945
64	1041384	254463	157496	95861
112	1024163	261102	166518	99804
300	998571	271170	179878	105260

(b) Nombre d'éléments non nuls pour différentes partitions

TABLE 5.3 – Réseau de fractures avec $\alpha = 3.5$ et pas de maillage $m = 0.08$

# dom	$\sum size(A_{ii})$	$\sum size(A_{00}^{(i)})$	$size(A_{00})$
2	815226	33745	16873
4	803618	58448	28481
8	792780	83232	39319
16	783789	104612	48310
32	774742	127510	57357
64	768896	143560	63203
112	766033	152422	66066
300	761905	166007	70194

(a) Taille des blocs pour différentes partitions

# dom	$\sum nnz(A_{ii})$	$\sum nnz(A_{i0})$	$\sum nnz(A_{00}^{(i)})$	$nnz(A_{00})$
2	4126562	138358	57613	37523
4	3911898	231712	101908	65479
8	3722102	313573	144250	91553
16	3567567	379745	181196	113744
32	3415158	443882	221848	137879
64	3320478	483842	248686	152639
112	3276319	501730	264306	161022
300	3213703	527745	285965	171608

(b) Nombre d'éléments non nuls pour différentes partitions

TABLE 5.4 – Réseau de fractures avec $\alpha = 3.5$ et pas de maillage $m = 0.05$

# dom	$\sum size(A_{ii})$	$\sum size(A_{00}^{(i)})$	$size(A_{00})$
2	241966	9026	4513
4	238934	15460	7545
8	235665	22779	10814
16	231532	32128	14947
32	228047	41124	18432
64	225468	48513	21011
116	223777	53940	22702
300	221946	61030	24533

(a) Taille des blocs pour différentes partitions

# dom	$\sum nnz(A_{ii})$	$\sum nnz(A_{i0})$	$\sum nnz(A_{00}^{(i)})$	$nnz(A_{00})$
2	1238340	37147	16314	10745
4	1182068	61521	27956	18269
8	1124111	86473	40829	26322
16	1046872	119457	58772	37593
32	986883	144567	75324	47362
64	944216	162185	88503	54793
116	917545	173123	97962	59588
300	890586	183906	109708	64981

(b) Nombre d'éléments non nuls pour différentes partitions

TABLE 5.5 – Réseau de fractures avec $\alpha = 4.5$ et pas de maillage $m = 0.08$

5.3.3 Partitionnement, factorisation et remplissage

Au chapitre 3, nous avons présenté deux variantes de la méthode du complément de Schur. Il est en effet possible de faire deux factorisations de Cholesky, pour accélérer le produit par le complément de Schur et l'application du préconditionnement de Neumann-Neumann, ou de ne faire qu'une seule factorisation et d'extraire les informations nécessaires pour chaque étape. Nous parlerons dans la suite de version sans mutualisation pour la première possibilité, classiquement utilisée, et de version avec pour la deuxième, que nous avons introduit. En fonction de la version que l'on choisit, la permutation appliquée pour réduire le remplissage n'est pas la même. Il faut donc comparer les remplissages dans chaque version.

Dans la version sans mutualisation, il faut factoriser les blocs A_{ii} et les blocs B_i. Ces blocs donnent respectivement les facteurs L_{ii} et L_i.

Dans la version avec mutualisation, les seuls blocs à factoriser sont les blocs B_i. Le facteur correspondant est le bloc L_i. De ce facteur sont extraits les facteurs L_{ii} et $L_{00}^{(i)}$, correspondant respectivement aux blocs A_{ii} et $A_{00}^{(i)}$ et utilisés pour l'application du produit par S_i et le préconditionnement de Neumann-Neumann.

Les tableaux 5.6, 5.7, 5.8 et 5.9 indiquent les différents remplissages pour les deux versions, en fonction des différentes partitions.

Lorsque l'on augmente le nombre de sous-domaines, le nombre d'éléments non nuls total dans les facteurs diminue, quelle que soit la version choisie.

Pour comparer le remplissage entre les deux versions, il faut comparer le nombre d'éléments non nuls dans le facteur L_i pour la version avec mutualisation avec la somme du nombre d'éléments non nuls dans L_i et dans L_{ii} pour la version sans mutualisation. On peut voir que la contrainte sur la permutation, imposée dans la version avec mutualisation, augmente le remplissage lorsque le nombre de sous-domaines est faible. Ce remplissage supplémentaire diminue lorsque le nombre de sous-domaines augmente. Pour tous les réseaux, le remplissage de la version avec mutualisation est équivalent à celui de la version sans mutualisation pour 300 sous-domaines.

	mutu			no mutu	
# dom	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$	$\sum nnz(L_{00}^{(i)})$	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$
2	147835757	5938619	124007207	41400217	4588603
4	61764391	2397941	53626606	25797334	2018193
8	29335657	1503666	25499695	16084417	1312466
16	16187104	1220095	13466583	10393913	1073197
32	10562203	1114067	8141977	7552720	980128
64	8241394	1066778	5926286	6335186	939799
110	7294666	1043726	5027872	5814678	920222
300	6605083	1022910	4374467	5394541	903343

TABLE 5.6 – Remplissage avec différentes partitions, avec et sans mutualisation
Réseau de fracture avec $\alpha = 2.5$ et pas de maillage $m = 0.08$

	mutu			no mutu	
# dom	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$	$\sum nnz(L_{00}^{(i)})$	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$
2	45303633	10268663	21180946	19988705	8602381
4	41495920	3970376	29333553	12817172	3165858
8	28265920	2267968	21709728	8297564	1905314
16	14377159	1551452	10677351	6019562	1344168
32	8326865	1257946	5828770	4473256	1101368
64	5907125	1125259	3772340	3785862	982128
112	5223754	1089529	3178351	3550544	949384
300	4249990	1037597	2306548	3307522	903911

TABLE 5.7 – Remplissage avec différentes partitions, avec et sans mutualisation
Réseau de fracture avec $\alpha = 3.5$ et pas de maillage $m = 0.08$

	mutu			no mutu	
# dom	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$	$\sum nnz(L_{00}^{(i)})$	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$
2	169717327	37808639	87438182	65926217	30719984
4	155175329	15275458	108001139	43329770	12113277
8	99593595	8512441	76581504	27456730	7202258
16	48462861	5936586	34946026	18915789	5105145
32	27511930	4719866	18747730	13964553	4159944
64	19477881	4232905	12073872	11967735	3706808
112	16491093	4087215	9462750	11209658	3572896
300	13421859	3891871	6770491	10536265	3395124

TABLE 5.8 – Remplissage avec différentes partitions, avec et sans mutualisation
Réseau de fracture avec $\alpha = 3.5$ et pas de maillage $m = 0.05$

	mutu			no mutu	
# dom	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$	$\sum nnz(L_{00}^{(i)})$	$\sum nnz(L_i)$	$\sum nnz(L_{ii})$
2	26757621	7659781	10395974	10793359	6341556
4	21856421	4107217	11939083	7337433	3570383
8	16690544	2799097	9557123	5497124	2399576
16	12040466	1645824	8140731	4113462	1432297
32	6978190	1275526	4474893	3211286	1125232
64	4939856	1106475	2929381	2814665	970080
116	3864134	1031729	2042837	2617090	899988
300	3127612	968783	1425793	2451777	844009

TABLE 5.9 – Remplissage avec différentes partitions, avec et sans mutualisation
Réseau de fracture avec $\alpha = 4.5$ et pas de maillage $m = 0.08$

5.3.4 Partitionnement et coût des opérations

A partir des tailles des systèmes et du nombre d'éléments non nuls dans les différents blocs, et en utilisant les formules donnant la complexité des différentes opérations (section 4.3, page 66), nous avons construit les tableaux 5.10, 5.11, 5.12 et 5.13. Ces tableaux donnent, pour chaque réseau et pour les différentes partitions, les complexités pour les deux versions de la méthode. La colonne *produit* comporte le nombre d'opérations flottantes nécessaire pour calculer le produit par S, celle *NN* le nombre d'opérations pour appliquer le préconditionnement de Neumann-Neumann et celle *déflation* le nombre d'opérations pour appliquer la déflation. Ces opérations sont utilisées dans chaque itération. Le coût d'une itération utilisant les deux préconditionnements est indiqué dans la colonne *iter*. La colonne *diff*, calculée en faisant la différence entre les colonnes *iter* pour les deux versions permet de voir rapidement quelle version est optimale. On peut ainsi remarquer qu'au delà de 100 sous-domaines, la mutualisation de la factorisation induit un coût d'itération plus faible que lors de l'utilisation de deux factorisations distinctes. Ce gain augmente lorsque le paramètre α augmente. Pour les réseaux avec beaucoup de petites fractures ($\alpha = 4.5$), le gain atteint 23%, pour 300 sous-domaines.

Lorsque le nombre de sous-domaines augmente, la déflation devient de plus en plus couteûse. Comme la taille de la matrice Z augmente, la taille du système grossier augmente aussi. De plus, comme la taille du complément de Schur augmente aussi, cela a un impact sur le coût de la déflation. Le coût de préparation augmente aussi. Il n'est pas présent dans les tableaux de cette section, mais est pris en compte dans ceux de la section suivante.

	mutu			no mutu		
# dom	produit	NN	déflation	produit	NN	déflation
2	12561475	248070120	83563	9861443	82856140	83563
4	5786632	107335206	247083	5027136	51676662	247083
8	4211564	51099900	424978	3829164	32269344	424978
16	3777806	27045502	576156	3484010	20900162	576156
32	3651029	16403776	710461	3383151	15225262	710461
64	3603095	11976510	838279	3349137	12794310	838279
110	3582495	10181812	944453	3335487	11755424	944453
300	3568861	8877246	1205118	3329727	10917394	1205118

(a) Coûts des opérations pour différentes partitions, avec et sans mutualisation

# dom	iter mutu	iter no mutu	diff
2	260715158	92801146	167914012
4	113368921	56950881	56418040
8	55736442	36523486	19212956
16	31399464	24960328	6439136
32	20765266	19318874	1446392
64	16417884	16981726	-563842
110	14708760	16035364	-1326604
300	13651225	15452239	-1801014

(b) Coût d'une itération pour différentes partitions, avec et sans mutualisation, différence entre les deux versions

TABLE 5.10 – Nombre d'opérations flotantes pour un réseau de fractures avec $\alpha = 2.5$ et $m = 0.08$

	mutu			no mutu		
# dom	produit	NN	déflation	produit	NN	déflation
2	20748279	42380788	37795	17415715	39996306	37795
4	8288198	58698146	93354	6679162	25665384	93354
8	5000220	43461644	190070	4274912	16637316	190070
16	3661111	21405850	282351	3246543	12090272	282351
32	3159514	11717006	376328	2846358	9005978	376328
64	2949447	7609694	447929	2663185	7636738	447929
112	2901412	6423966	521423	2621122	7168352	521423
300	2832814	4683900	719335	2565442	6685848	719335

(a) Coûts des opérations pour différentes partitions, avec et sans mutualisation

# dom	iter mutu	iter no mutu	diff
2	63166862	57449816	5717046
4	67079698	32437900	34641798
8	48651934	21102298	27549636
16	25349312	15619166	9730146
32	15252848	12228664	3024184
64	11007070	10747852	259218
112	9846801	10310897	-464096
300	8236049	9970625	-1734576

(b) Coût d'une itération pour différentes partitions, avec et sans mutualisation, différence entre les versions

TABLE 5.11 – Nombre d'opérations flotantes pour un réseau de fractures avec $\alpha = 3.5$ et $m = 0.08$

	mutu			no mutu		
# dom	produit	NN	déflation	produit	NN	déflation
2	75968480	174910110	67495	61791170	131886180	67495
4	31144729	216059240	170389	24820367	86716502	170389
8	17835597	153241646	350452	15215231	54992098	350452
16	12862168	69988672	530036	11199286	37928198	530036
32	10606701	37610174	693975	9486857	28043820	693975
64	9745383	24274150	826711	8693189	24061876	826711
112	9508262	19057632	927945	8479624	22551448	927945
300	9195391	13681370	1208195	8201897	21212918	1208195

(a) Coûts des opérations pour différentes partitions, avec et sans mutualisation

# dom	iter mutu	iter no mutu	diff
2	250946085	193744845	57201240
4	247374358	111707258	135667100
8	171427695	70557781	100869914
16	83380876	49657520	33723356
32	48910850	38224652	10686198
64	34846244	33581776	1264468
112	29493839	31959017	-2465178
300	24084956	30623010	-6538054

(b) Coût d'une itération pour différentes partitions, avec et sans mutualisation, différence entre les versions

TABLE 5.12 – Nombre d'opérations flotantes pour un réseau de fractures avec $\alpha = 3.5$ et $m = 0.05$

	mutu			no mutu		
# dom	produit	NN	déflation	produit	NN	déflation
2	15414683	20800974	13543	12778233	21595744	13543
4	8372977	23893256	45314	7299309	14689956	45314
8	5822783	19135874	101208	5023741	11015876	101208
16	3604281	16311356	178509	3177227	8256818	178509
32	2933942	8986650	237343	2633354	6459436	237343
64	2646834	5900784	301281	2374044	5671352	301281
116	2530368	4131078	343392	2266886	5279584	343392
300	2439619	2900652	501067	2190071	4952620	501067

(a) Coûts des opérations pour différentes partitions, avec et sans mutualisation

# dom	iter mutu	iter no mutu	diff
2	36229200	34387520	1841680
4	32311547	22034579	10276968
8	25059865	16140825	8919040
16	20094146	11612554	8481592
32	12157935	9330133	2827802
64	8848899	8346677	502222
116	7004838	7889862	-885024
300	5841338	7643758	-1802420

(b) Coût d'une itération pour différentes partitions, avec et sans mutualisation, différence entre les deux versions

TABLE 5.13 – Nombre d'opérations flotantes pour un réseau de fractures avec $\alpha = 4.5$ et $m = 0.08$

5.3.5 Convergence et temps de calcul

Les tableaux de cette section sont construits à partir des temps mesurés lors de la résolution des systèmes associés aux différents réseaux, avec les même partitions que dans les sections précédentes. Nous avons utilisé deux préconditionnements : Neumann-Neumann seul et Neumann-Neumann conjugué à la déflation. Les tableaux 5.14a, 5.15a, 5.16a et 5.17a contiennent les temps mesurés avec le préconditionnement de Neumann-Neumann. Les tableaux 5.14b, 5.15b, 5.16b et 5.17b contiennent ceux correspondants au préconditionnement conjuguant Neumann-Neumann et déflation. Les temps sont mesurés en utilisant la bibliothèque `SolveurSchur`, sur une machine sous linux, avec un processeur dual-core cadencé à $3GHz$.

Ces tableaux sont analysés plus en détail dans la suite.

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	40	1227	23	1250	96	9	105
4	91	185	25	210	33	17	50
8	124	32	20	52	10	19	29
16	168	7	19	26	3	20	23
32	224	3	19	22	1	22	23
64	296	2	21	23	2	25	27
110	370	1	26	27	1	32	33
300	400	1	29	30	1	35	36

(a) Préconditionnement : Neumann-Neumann

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	40	1225	22	1247	95	10	105
4	85	186	23	209	33	16	49
8	109	33	18	51	11	16	27
16	125	8	13	21	4	14	18
32	136	4	11	15	2	14	16
64	138	3	12	15	3	13	16
110	130	5	10	15	4	13	17
300	111	10	10	20	9	12	21

(b) Préconditionnement : Neumann-Neumann et déflation

TABLE 5.14 – Comparaison des temps avec et sans mutualisation de la factorisation
Réseau de fractures avec $\alpha = 2.5$ et pas de maillage $m = 0.08$

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	40	168	6	174	42	6	48
4	62	130	11	141	12	7	19
8	85	55	10	65	4	7	11
16	132	11	10	21	1	10	11
32	246	2	14	16	1	15	16
64	275	2	12	14	1	15	16
112	313	1	14	15	1	16	17
300	327	1	14	15	0	20	20

(a) Préconditionnement : Neumann-Neumann

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	40	168	6	174	41	7	48
4	62	131	10	141	13	6	19
8	74	55	9	64	4	6	10
16	85	11	7	18	2	6	8
32	141	3	9	12	1	10	11
64	97	2	5	7	2	6	8
112	102	3	6	9	3	6	9
300	80	6	5	11	6	6	12

(b) Préconditionnement : Neumann-Neumann et déflation

TABLE 5.15 – Comparaison des temps avec et sans mutualisation de la factorisation
Réseau de fractures avec $\alpha = 3.5$ et pas de maillage $m = 0.08$

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	31	1206	19	1225	235	18	253
4	67	959	38	997	75	23	98
8	88	367	38	405	23	26	49
16	142	62	34	96	8	32	40
32	325	13	54	67	3	60	63
64	303	5	40	45	3	48	51
112	374	3	45	48	2	57	59

(a) Préconditionnement : Neumann-Neumann

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	32	1202	20	1222	236	17	253
4	63	960	36	996	74	25	99
8	77	367	34	401	23	22	45
16	91	62	24	86	9	20	29
32	190	15	32	47	5	33	38
64	110	8	16	24	5	19	24
112	115	8	17	25	8	20	28

(b) Préconditionnement : Neumann-Neumann et déflation

TABLE 5.16 – Comparaison des temps avec et sans mutualisation de la factorisation
Réseau de fractures avec $\alpha = 3.5$ et pas de maillage $m = 0.05$

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	32	75	3	78	14	4	18
4	46	43	4	47	5	3	8
8	65	21	6	27	3	4	7
16	94	10	5	15	1	5	6
32	157	2	7	9	1	6	7
64	247	1	9	10	1	10	11
116	267	1	8	9	0	12	12
300	285	1	10	11	1	12	13

(a) Préconditionnement : Neumann-Neumann

		mutu			no mutu		
#dom	#it	prep	iters	total	prep	iters	total
2	33	75	4	79	14	4	18
4	44	43	4	47	5	3	8
8	53	22	3	25	2	5	7
16	60	9	5	14	1	3	4
32	72	3	3	6	1	4	5
64	73	2	3	5	1	4	5
116	73	3	2	5	2	4	6
300	61	4	3	7	4	4	8

(b) Préconditionnement : Neumann-Neumann et déflation

TABLE 5.17 – Comparaison des temps avec et sans mutualisation de la factorisation
Réseau de fractures avec $\alpha = 4.5$ et pas de maillage $m = 0.08$

Analyse de la convergence

Dans la série de tableaux 5.14a–5.17b, la colonne *#it* contient le nombre d'itérations nécessaire, dans chaque cas, pour que le Gradient Conjugué converge. Le nombre d'itération est le même quelle que soit la version de l'algorithme choisie (avec ou sans mutualisation de la factorisation).

Lorsque l'on observe les résultats obtenus pour le préconditionnement de Neumann-Neumann (Tab : 5.14a, 5.15a, 5.16a et 5.17a), on remarque un fort ralentissement de la convergence lorsque le nombre de sous-domaines augmente. Ce résultat, conforme à la théorie et aux résultats du chapitre 4, confirme que le préconditionnement de Neumann-Neumann seul n'est pas adapté pour un grand nombre de sous-domaines.

Lorsqu'on ajoute un préconditionnement par déflation, le nombre d'itérations augmente nettement moins vite. De plus, comme nous avons défini le sous-espace associé au problème grossier en fonction des sous-domaines, sa taille augmente quand le nombre de sous-domaines augmente. Ceci explique que le nombre d'itérations nécessaire pour converger diminue pour les découpages avec le plus grand nombre de sous-domaines. L'utilisation de la déflation permet donc de faire diminuer de façon suffisamment importante (d'un facteur 3 à 5 suivant les cas) le nombre d'itérations pour qu'un découpage en sous-domaines où chaque sous-domaine est défini par une fracture puisse être considéré comme une alternative. Dans l'ensemble des cas testés, la matrice $\bar{Z}$ est de rang plein est peu être utilisé directement. la construction de cette matrice est donc peu couteuse.

Ce choix pour la partition en sous-domaines présente l'avantage de ne pas nécessiter de préparation particulière. Les sous-systèmes associés aux fractures sont directement construits lors de la génération du réseau par `H2oLab`.

Avant de valider ce choix, il est important de vérifier que l'utilisation de la déflation ne conduit pas à un surcoût trop important, et que le nombre d'itérations plus important que pour une partition en 2 ou 4 sous-domaines n'est pas pénalisant. C'est l'objet des points suivants de notre analyse.

Analyse du temps de préparation

Pour chaque partition, nous avons mesuré le temps nécessaire pour factoriser les systèmes locaux et construire le système grossier utilisé par la déflation. Les temps mesurés sont indiqués dans les colonnes *prep* des tableaux. Ces temps étant différents pour les versions avec et sans mutualisation, les temps des deux versions sont indiqués (colonnes *mutu* et *no mutu* respectivement).

Lorsque seul Neumann-Neumann est utilisé, le temps de préparation correspond uniquement au calcul des facteurs L_{ii} et L_i pour la version sans mutualisation, et au calcul de L_i suivi de l'extraction de L_{ii} et de $L_{00}^{(i)}$ pour la version avec.

Dans les deux versions, le temps requis par cette étape diminue avec l'augmentation du nombre de sous-domaines. On retrouve ici l'impact de la diminution du nombre d'éléments non nuls dans les facteurs (Tab : 5.6–5.9).

Le surcoût engendré par le remplissage supplémentaire induit par la contrainte sur la permutation est aussi nettement visible. Le temps nécessaire pour que cette opération se réalise étant, dans le cas où $\alpha = 2.5$, jusqu'à 12 fois supérieur à celui mesuré en l'absence de mutualisation, pour deux sous-domaines. Ce surcoût diminue, comme l'indiquaient les mesures de remplissage, lorsque le nombre de sous-domaines augmente.

Le coût de la mise en place de la déflation correspond à la différence entre les tableaux avec Neumann-Neumann seul et les tableaux avec Neunmann-Neumann et la déflation. Lorsque le nombre de sous-domaines augmente, ce coût augmente de façon relativement

importante. Mais le coût total de la préparation lorsque les sous-domaines sont définis par les fractures (300 sous-domaines) reste inférieur au coût lié à la factorisation lorsque l'on n'utilise que 2 sous-domaines.

Analyse du temps pour les itérations

Les colonnes *iters* contiennent le temps correspondant à l'ensemble des itérations, pour chaque partition.

Que l'on utilise ou non la mutualisation de la factorisation, le même nombre d'itérations est nécessaire pour que le Gradient Conjugué converge.

Ici aussi, pour le préconditionnement de Neumann-Neumann, l'impact du remplissage est important. Avec un petit nombre de sous-domaines, le temps nécessaire est plus important lorsque l'on mutualise la factorisation. A l'opposé, lorsque le nombre de sous-domaines augmente, comme chaque itération coûte moins cher avec la version mutualisée, les itérations sont plus rapides. Ceci est particulièrement visible avec ce préconditionnement puisque le nombre d'itérations augmente fortement avec le nombre de sous-domaines. Cette augmentation du nombre d'itérations pénalise les résolutions avec beaucoup de sous-domaines, qui sont moins performantes que celles avec peu de sous-domaines.

Lorsque l'on utilise la déflation, l'écart entre les deux versions est moins important pour un grand nombre de sous-domaines. En effet, comme le nombre d'itérations est plus faible, l'écart final est moins important. La différence de remplissage en faveur de la mutualisation est pour le bloc correspondant au préconditionnement de Neumann-Neumann. Or ce bloc n'intervient pas dans la déflation. Ce coût identique écrase l'écart entre les deux versions.

Comme le nombre d'itérations est moins important lorsque l'on utilise la déflation, l'utilisation d'un grand nombre de sous-domaines devient envisageable. Dans tous les cas étudiés, pour 300 sous-domaines, la version avec mutualisation de la factorisation effectue le même nombre d'itérations en moins de temps que la version sans. Ce temps n'est pas toujours le temps optimal obtenu. Il est en revanche toujours proche de cet optimal.

Analyse du temps total

La colonne *total* correspond à la somme des deux colonnes précédentes. Lorsque le nombre de sous-domaines est faible, le coût de préparation est très élevé, comparativement au temps nécessaire pour converger. Il n'est donc pas pertinant, malgré un temps pour les itérations optimal, d'utiliser peu de sous-domaines. Comme le coût de préparation diminue rapidement, l'utilisation de 16 ou 32 sous-domaines permet, sans mutualisation de la factorisation d'obtenir un temps total optimal.

Lorsque l'on utilise la version mutualisation, le remplissage plus important pénalise cette version tant que le nombre de sous-domaines n'augmente pas suffisemment. Cet effet est nettement visible dans le cas test avec un maillage plus fin (Tab : 5.16).

Malgré un coût de préparation croissant, la déflation permet d'obtenir un temps total quasi optimal pour 300 sous-domaines lorsque l'on utilise la mutualisation, qui permet de diminuer encore le remplissage. De plus, la préparation de la déflation consiste principalement à effectuer le produit SZ, qui est fortement parallélisable. Le coût de cette préparation devrait donc diminuer fortement lors de l'utilisation de machine parallèle pour effectuer la résolution, devenant de ce fait encore plus compétitive.

Conclusion pour les trois réseaux testés

L'utilisation conjuguée du préconditionnement de Neumann-Neumann, de la déflation et de la mutualisation de la factorisation permet d'obtenir un résultat quasi optimal lorsque l'on utilise les fractures comme définition des sous-domaines. Le temps de résolution avec cette méthode n'est pas le plus bas (le minimum est atteint pour une partition avec moins de sous-domaines, 64 ou une centaine, suivant les cas). Mais l'utilisation de cette partition permet de ne pas avoir à rassembler les fractures au sein d'un même sous-domaine, qui est une opération relativement coûteuse lorsqu'on la réalise *a posteriori* de la génération des sous-systèmes, et dont le coût n'apparaît pas dans les tableaux. De plus, ce regroupement impose l'utilisation d'un partitionneur permettant de garantir la connexité des partitions. `SCOTCH` nous a permis d'obtenir de telles partitions, mais ne garantit pas qu'elles soient connexes dans tous les cas. Il reste donc possible de construire une partition non connexe qui interdirait l'utilisation de l'approximation utilisée pour résoudre le problème des sous-domaines flottants.

C'est donc cette combinaison qu'il faut privilégier lorsque l'on utilise la méthode du complément de Schur pour résoudre le système linéaire associé à l'écoulement dans un réseau de fractures.

5.3.6 Comparaison avec les solveurs du chapitre 2

Solver	#it	tps
umf	-	48
amg	69	28
pcg	3	46
schur	111	20

(a) Réseau de fracture avec $\alpha = 2.5$ et $m = 0.08$

Solver	#it	tps
umf	-	22
amg	108	23
pcg	3	27
schur	80	11

(b) Réseau de fracture avec $\alpha = 3.5$ et $m = 0.08$

Solver	#it	tps
umf	-	72
amg	85	52
pcg	3	78
schur	115	25

(c) Réseau de fracture avec $\alpha = 3.5$ et $m = 0.05$

Solver	#it	tps
umf	-	13
amg	500	68
pcg	4	24
schur	61	7

(d) Réseau de fracture avec $\alpha = 4.5$ et $m = 0.08$

TABLE 5.18 – Performances des solveurs du chapitre 2

Les tableaux 5.18 comparent, pour les systèmes utilisés dans cette analyse, les performances des trois solveurs utilisés au chapitre 2 (*umf* `:UMFPACK`, *amg* `:BoomerAMG` et *pcg* : `PCG` préconditionné par `BoomerAMG`).

La ligne *schur* correspond à l'utilisation du préconditionnement de Neumann-Neumann, de la déflation et de la mutualisation de la factorisation pour résoudre le système.

Comme on peut le voir, dans tous les cas, cette méthode est la plus rapide.

5.4 Analyse avec les systèmes du chapitre 2

5.4.1 Rappels des systèmes

Dans cette partie, nous prolongeons l'analyse du chapitre 2 en utilisant la méthode du complément de Schur avec Neumann-Neumann, la déflation et la mutualisation de la factorisation pour résoudre les systèmes générés avec les paramètres des tableaux 2.1 et 2.2, page 31. Chaque fracture définit un sous-domaine.

L'ensemble des systèmes correspond à 180 réseaux et 420 systèmes linéaires différents. Deux séries de simulations ont été réalisées, en faisant varier la densité de fractures et le pas de maillage. Une présentation plus détaillée des systèmes est donnée dans le chapitre 2, page 31.

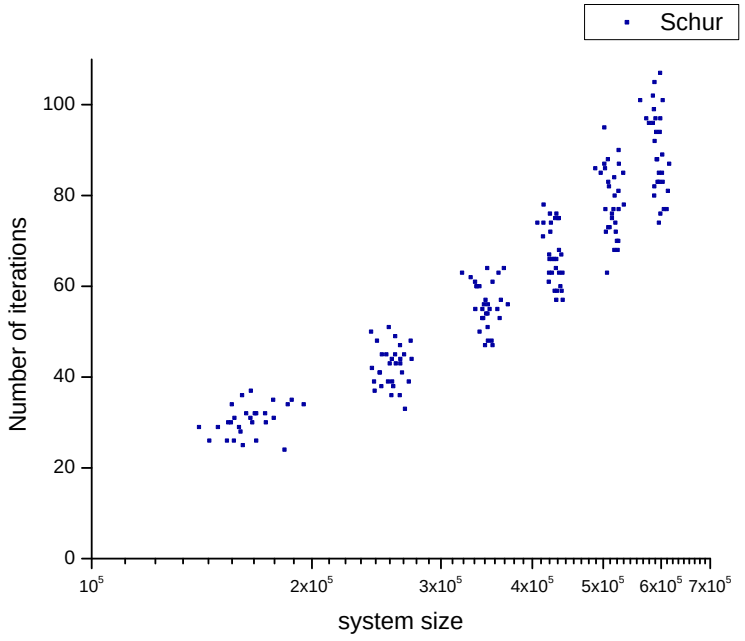
5.4.2 Analyse de la convergence

Les figures 5.3a et 5.3b représentent le nombre d'itérations en fonction de la taille des systèmes. Tous les systèmes ont été résolus en moins de 110 itérations, mais le nombre d'itérations varie fortement, malgré l'utilisation des préconditionnements. La variabilité est plus forte dans le cas de la figure 5.3a. Les réseaux correspondants sont générés en faisant varier la densité de fractures, et donc le nombre de ces dernières. Comme le nombre de sous-domaines est égal au nombre de fractures, la variation importante du nombre d'itérations s'explique par la variation du nombre de sous-domaines. L'utilisation de la déflation permet de limiter cette variation, mais ne la supprime pas totalement.

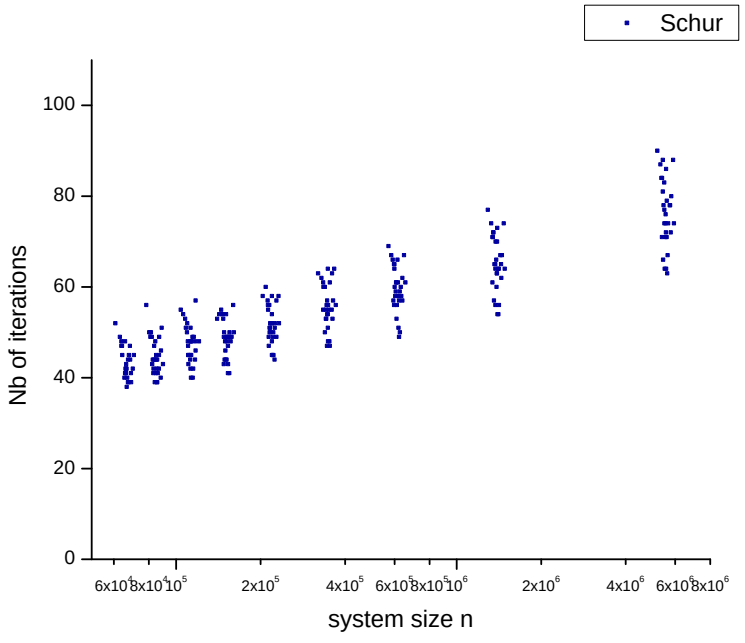
La figure 5.3b correspond à un raffinement du maillage. Le nombre de sous-domaines est donc stable, pour un même réseau.

5.4.3 Analyse du temps de calcul

Les figures 5.4a et 5.4b représentent le temps de calcul nécessaire pour obtenir la convergence en fonction de la taille des systèmes. Alors que le nombre d'itérations variait fortement, le temps de calcul dépend à peu près linéairement de la taille des systèmes. Il sera nécéssaire d'effectuer des simulations avec des systèmes plus grand pour conclure.

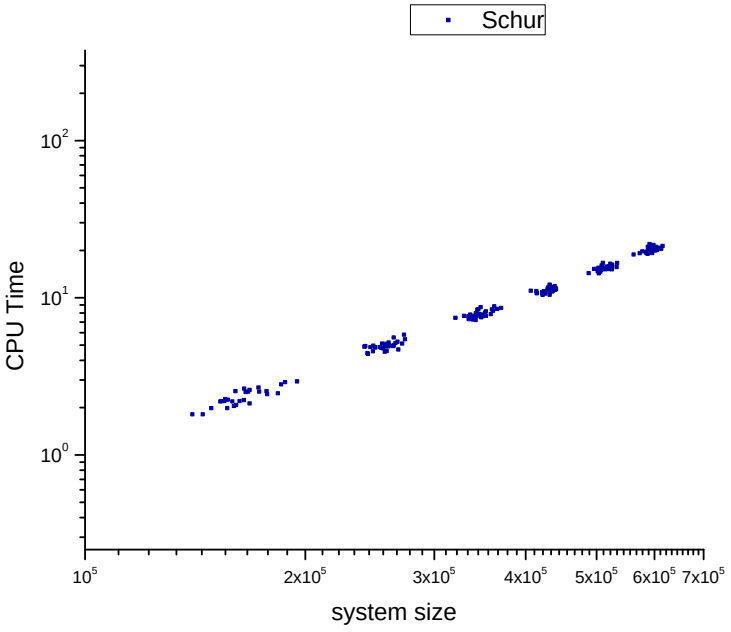


(a) Variation de la densité

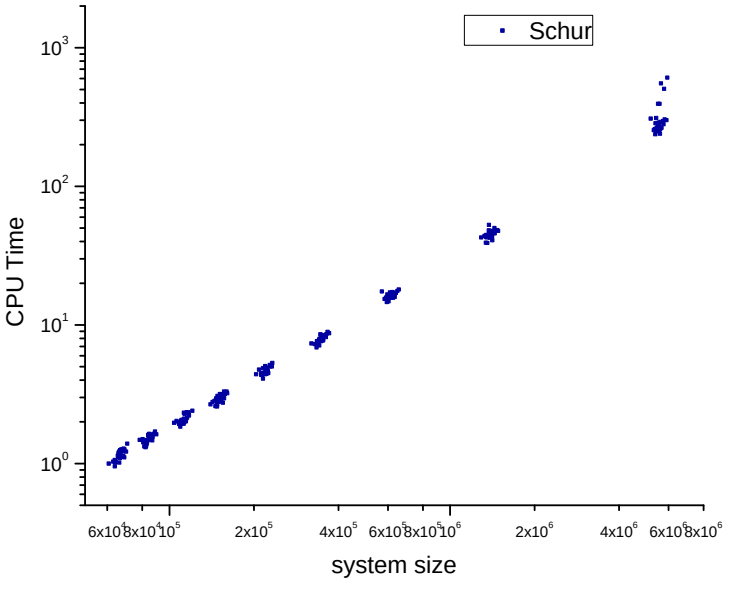


(b) Variation du pas de maillage

FIGURE 5.3 – Nombre d'itérations en fonction de la taille des systèmes avec `SolverSchur`.



(a) Variation de la densité



(b) Variation du pas de maillage

FIGURE 5.4 – Temps de résolution en fonction de la taille des systèmes avec `SolverSchur`.

5.4.4 Comparaison avec les autres solveurs

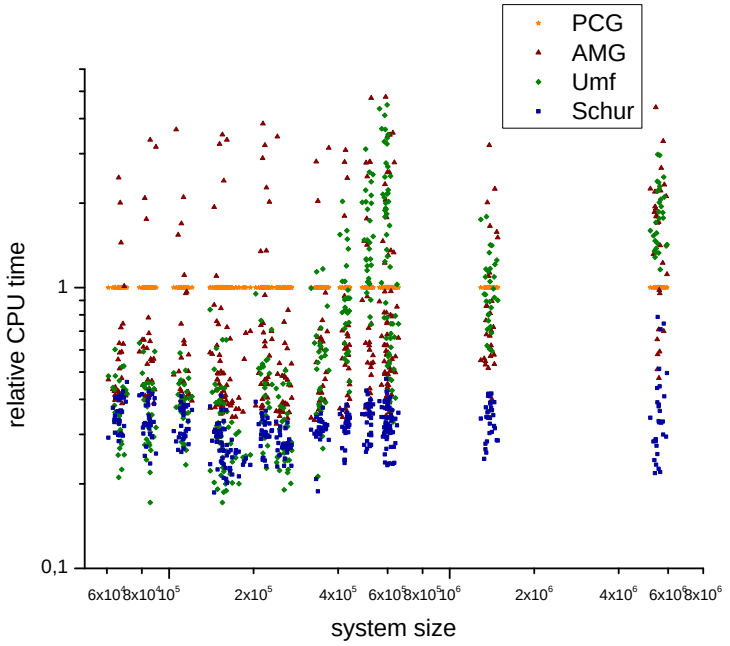


FIGURE 5.5 – Temps de résolution relatif à `PCG` pour les différents solveurs

taille	#total	umf	amg	pcg	schur
<100 000	60	25	0	0	35
<155 000	61	20	0	0	41
<230 000	55	17	1	0	37
<360 000	58	18	1	0	39
<550 000	64	0	6	0	58
<800 000	60	0	5	0	55
<6 500 000	60	0	3	0	57

TABLE 5.19 – Répartition des solveurs (meilleur temps) en fonction de la taille des systèmes

Le graphique 5.5 présente, pour toutes les simulations, le temps relatif des différentes méthodes, la résolution faite avec `PCG` servant de référence. Nous avons choisi PCG comme référence puisqu'il s'agit, sur les trois solveurs testés précédemment, du seul qui permette de résoudre tous les systèmes. On peut voir sur ce graphique que la résolution par la méthode de Schur fait toujours partie des plus rapides.

Nous avons cherché à quantifier le taux de réussite de chaque solveur. Nous avons donc regardé quel solveur était le plus rapide pour chaque test. Le tableau 5.19 présente une synthèse de ces observations, regroupées par tranches. Ces résultats sont repris dans l'histogramme 5.6. Il est visible que la méthode de Schur présente un fort taux de réussite. Elle est sans conteste la plus adaptée pour les systèmes de grande taille.

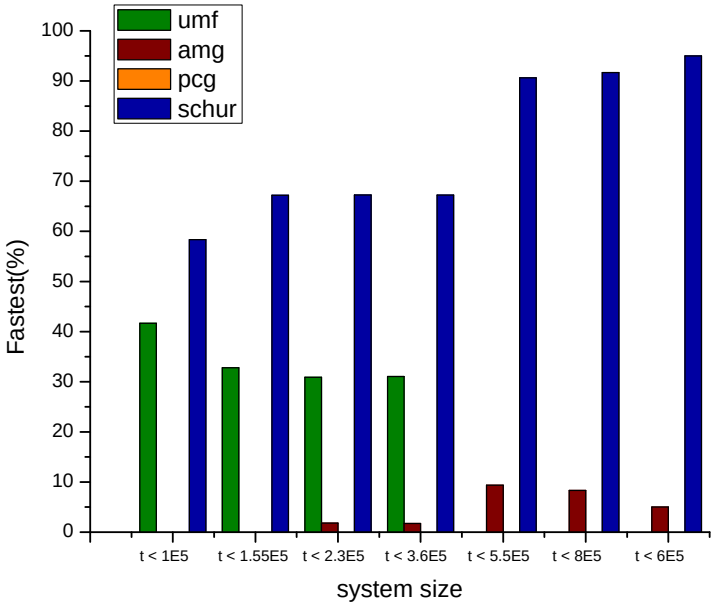


FIGURE 5.6 – Répartition, en pourcentage, du solveur optimal en fonction de la taille des systèmes

5.4.5 Conclusion

Lors de l'étude des solveurs du chapitre 2, nous avions vu que le solveur direct permettait de résoudre rapidement les petits systèmes, et que le Gradient Conjugué avait une complexité linéaire.

L'utilisation du Gradient Conjugué pour résoudre le système associé au complément de Schur permet de combiner les avantages des deux solveurs. Grâce à l'utilisation des préconditionnements de Neumann-Neumann et de la déflation, et grâce à la mutualisation de la factorisation que nous avons introduite, il est possible de décomposer le réseau de fractures en petits sous-domaines. Les systèmes locaux aux sous-domaines sont alors résolus efficacement avec un solveur direct et la solution du problème aux interfaces est obtenue grâce au Gradient Conjugué, ce qui conduit à une complexité linéaire.

Cette méthode permet de résoudre les problèmes rapidement et efficacement, comme l'a montré la comparaison avec les autres solveurs.

Conclusion et perspectives

Bilan

Afin de pouvoir réaliser des études stochastiques, il est nécessaire de pouvoir effectuer des simulations de façon systématique, sans avoir à gérer un grand nombre d'échecs. La procédure de maillage décrite dans le chapitre 1 permet d'atteindre cet objectif. La phase de projection des limites de fractures et des arêtes intersections permet de rendre le schéma numérique stable. La phase de correction que nous avons mise en place permet, quand à elle, de supprimer les causes d'échecs créées par la projection. Sans cette étape de correction, un grand nombre de simulations échouent et rendent impossible une étude à grande échelle.

Nous avons ensuite étudié quelques solveurs, présents dans la plate-forme `H2oLab`. Cette étude, présentée dans le chapitre 2, a permis de mettre en évidence les avantages et les inconvénients de chacun de ces solveurs, lorsqu'on les utilise pour résoudre les problèmes d'écoulements dans les milieux fracturés. Nous avons ainsi vu que le solveur direct, `UMFPACK`, est très efficace pour les systèmes de petite taille. *A contrario*, lorsque la taille augmente, son efficacité diminue fortement. Avec les plus grands systèmes que nous avons générés, le remplissage engendré par la factorisation est trop important et la résolution échoue. Le solveur utilisant le procédé de multigrille algébrique, `BoomerAMG`, est moins efficace que le solveur direct sur les petits systèmes, mais résiste mieux à la charge. L'augmentation de la taille du système ne le pénalise pas, sa complexité étant linéaire. Mais il présente un fort taux d'échec, environ 10% des résolutions n'aboutissant pas. Le Gradient Conjugué préconditionné par BoomerAMG permet de résoudre tous les systèmes. Mais c'est aussi le plus lent des trois solveurs testés.

Suite à ces résultats nous avons étudier une méthode de sous-domaines, présentée au chapitre 3. Elle consiste à résoudre le problème aux interfaces en utilisant le Gradient Conjugué et à résoudre les problèmes internes à chaque sous-domaine avec un solveur direct. Les itérations du Gradient Conjugué préconditionné nécessitent la résolutions de deux systèmes associés aux sous-domaines. Nous avons proposé une mutualisation de la factorisation nécessaire à ces résolutions. Nous avons aussi défini un sous-espace lié aux sous-domaines afin de pouvoir utiliser un préconditionnement basé sur la déflation.

Cette méthode et les préconditionnement associés ont été mis en œuvre au sein d'une bibliothèque en `C/C++`. Il est possible d'utiliser la version classique, utilisant deux factorisations, ou la version avec mutualisation de la factorisation. Cette bibliothèque est décrite dans le chapitre 4.

Les résultats du chapitre 5, obtenus en utilisant la bibliothèque `SolveurSchur` pour résoudre les systèmes linéaires liés aux réseaux de fractures, montrent l'efficacité de la

méthode. Alors que la mutualisation entraîne un remplissage supplémentaire pénalisant lorsque le nombre de sous-domaines est faible, l'effet s'inverse pour un grand nombre de sous-domaines. Ceci permet d'utiliser une définition naturelle pour les sous-domaines, en basant cette définition sur les fractures. Dans ce cas, chaque sous-domaine correspond à une fracture. La déflation permet en outre de réduire efficacement le nombre d'itérations. Notre solveur est plus rapide que les solveurs testés précédemment sur la majorité des systèmes, et permet de résoudre tous ces systèmes sans échec.

Perspectives

La bibliothèque et le code source associé vont être distribués sous une license libre.
Par ailleurs, les résultats obtenus ouvrent la voie à d'autres études :

- tous les résultats exploités ont été obtenus par des exécutions séquentielles. La bibliothèque `SolveurSchur` est utilisable en parallèle. Il sera donc intéressant de valider le parallélisme de la méthode. Comme les systèmes associés aux sous-domaines sont indépendants, les résultats devraient être bons. Les résultats préliminaires que nous avons obtenus sont encourageants.
- le parallélisme actuel est un parallélisme simple. Un parallélisme à deux niveaux peut être envisager, avec une résolution parallèle des systèmes locaux. La bibliothèque utilisée pour résoudre les systèmes locaux, `CHOLMOD`, est une bibliothèque uniquement séquentielle. Pour obtenir un parallélisme à deux niveaux, une autre bibliothèque devra être choisie.
- nous nous sommes limité à des tests sur des systèmes ne dépassant pas 6 millions d'inconnues. Le nombre maximal de fractures atteint était de 300. La méthode devra être testée face à une augmentation de la taille des systèmes et face à une augmentation du nombre de sous-domaines. Pour réaliser cette montée en charge, il sera nécéssaire de paralléliser la génération des systèmes. De plus, si la méthode n'est pas robuste face à une augmentation importante du nombre de sous-domaines, il faudra alors regrouper plusieurs fractures au sein d'un même sous-domaine. Pour que ce regroupement ne soit pas trop pénalisant, il devra être fait en amont de la génération des systèmes locaux.
- la déflation utilise actuellement un sous-espace associé à la signature des sous-domaines. Une définition plus fine de ce sous-espace, en se basant notamment sur les caractéristiques physiques du problème, devrait améliorer les performances de ce préconditionnement.
- les simulations que nous avons réalisées utilisent un champ de perméabilité homogène dans les fractures. Il sera important de tester la robustesse de la méthode face à un champ hétérogène. De plus, les fractures sont toutes maillées avec un pas de maillage identique. L'utilisation d'une méthode de joint (mortar) [60, 61] doit permettre de différencier le maillage en fonction de l'importance des fractures pour l'écoulement dans le domaine. Il sera nécessaire d'adapter la définition des sous-domaines, notamment en terme de contribution propre, pour pouvoir utiliser cette méthode.

Bibliographie

[1] Domain decomposition methods. http ://www.ddm.org/.

[2] HIPS - hierarchical iterative parallel solver. http ://hips.gforge.inria.fr/index.html.

[3] MUMPS : a parallel sparse direct solver. http ://graal.ens-lyon.fr/MUMPS/.

[4] PETSc : home page. http ://www.mcs.anl.gov/petsc/petsc-as/.

[5] Ressources scalaires de l'IDRIS, IBM SP6. http ://www.idris.fr/su/Scalaire/vargas/hw-vargas.html.

[6] SCOTCH : Distribution logicielle pour le placement statique, le partitionnement de graphes, de maillages et d'hypergraphes, et la renumérotation parallèle et séquentielle de matrices creuses. http ://www.labri.fr/perso/pelegrin/scotch/scotch_fr.html.

[7] Software. https ://computation.llnl.gov/casc/hypre/software.html.

[8] SuiteSparse : a suite of sparse matrix packages. http ://www.cise.ufl.edu/research/sparse/SuiteSparse/.

[9] Patrick R. Amestoy, Enseeiht-Irit, Timothy A. Davis, and Iain S. Duff. Algorithm 837. *ACM Transactions on Mathematical Software*, 30 :381–388, September 2004.

[10] G.-A. Atenekeng-Kahou, E. Kamgnia, and B. Philippe. An explicit formulation of the multiplicative schwarz preconditionner. *Applied Numerical Mathematics (also INRIA research report RR-5685)*, 57(11-12) :1197–1213, 2007.

[11] A. Beaudoin, J-R. de Dreuzy, J. Erhel, and H. Mustapha. Parallel simulations of underground flow in porous and fractured media. In G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, *Parallel Computing : Current and Future Issues of High-End Computing*, volume 33 of *NIC Series*, pages 391–398, Jülich, Germany, 2006. NIC.

[12] A. Beaudoin, J. Erhel, and J.-R. de Dreuzy. A comparison between a direct and a multigrid sparse linear solvers for highly heterogeneous flux computations. In *Eccomas CFD 2006*, volume CD, 2006.

[13] E. Bonnet, O. Bour, N. Odling, P. Davy, I. Main, P. Cowie, and B. Berkowitz. Scaling of fracture systems in geological media. *Reviews of Geophysics*, 39(3) :347–383, 2001.

[14] O. Bour and P. Davy. Connectivity of random fault networks following a power law fault length distribution. *Water Resources Research*, 33(7) :1567–1583, 1997.

[15] O. Bour and P. Davy. On the connectivity of three dimensional fault networks. *Water Resources Research*, 34(10) :2611–2622, 1998.

[16] M. C. Cacas, E. Ledoux, G. de Marsily, A. Barbeau, P. Calmels, B. Gaillard, and R. Magritta. Modeling fracture flow with a stochastic discrete fracture network :

calibration and validation. 1. the flow model. *Water Resources Research*, 26(3) :479–489, 1990. lire.

[17] Xiao-chuan Cai and Marcus Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM J. SCI. COMPUT*, 21 :792—797, 1999.

[18] Y. Chen, T. A Davis, W. W Hager, and S. Rajamanickam. Algorithm 8xx : CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw., submitted in*, 2006.

[19] National Research Council. *Rock Fractures and Fluid Flow.* National Academy Press, Washington, D.C., 1996.

[20] T. Davis. Algorithm 832 : Umfpack, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30 :196–199, 2004.

[21] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM Book Series on the Fundamentals of Algorithms. SIAM, Philadelphia, 2006.

[22] J.-R. de Dreuzy, P. Davy, and O. Bour. Percolation parameter and percolation-threshold estimates for three-dimensional random ellipses with widely scattered distributions of eccentricityand size. *Physical review E*, 62 :5948–5952, 2000.

[23] J-R. de Dreuzy, P. Davy, and O. Bour. Hydraulic properties of two-dimensional random fracture networks following a power law length distribution : 1-effecive connectivity. *Water Resources Research*, 37 :2065–2078, 2001.

[24] J-R. de Dreuzy, P. Davy, and O. Bour. Hydraulic properties of two-dimensional random fracture networks following a power law length distribution : 2-permeability of networks based on log-normal distribution of apertures. *Water Resources Research*, 37 :2079–2095, 2001.

[25] W. S. Dershowitz and C. Fidelibus. Derivation of equivalent pipe networks analogues for three-dimensional discrete fracture networks by the boundary element method. *Water Resources Research*, 35(9) :2685–2691, 1999.

[26] Vitorita Dolean, Frédéric Nataf, and Gerd Rapin. A domain decomposition preconditioner of Neumann-Neumann type for the stokes equations. In *Domain Decomposition Methods in Science and Engineering XVIII*, pages 161–168. 2009.

[27] J. Douglas and J. Roberts. Global estimates for mixed methods for second order elliptic methods. *Mathematics of computation*, 44(169) :39–52, 1985.

[28] J. Erhel, A. Beaudoin, and J.-R. de Dreuzy. Direct and iterative sparse linear solvers applied to groundwater flow simulations. In *Conference on Matrix Analysis and Applications (M2A07)*, 2007. invited talk.

[29] J. Erhel, J.-R. de Dreuzy, A. Beaudoin, E. Bresciani, and D. Tromeur-Dervout. A parallel scientific software for heterogeneous hydrogeology. In Ismail H. Tuncer, Ulgen Gulcat, David R. Emerson, and Kenichi Matsuno, editors, *Parallel Computational Fluid Dynamics 2007*, volume 67 of *Lecture Notes in Computational Science and Engineering*, pages 39–48. Springer, 2009. invited plenary talk.

[30] J. Erhel, J.-R. de Dreuzy, and B. Poirriez. flow simulations in three-dimensional discrete fracture networks. *SIAM Journal on Scientific Computing*, 31(4) :2688–2705, 2009.

[31] J. Erhel, N. Nassif, and B. Philippe. Calcul matriciel et systemes lineaires. *Cours de DEA Informatique et Modélisation*, 2004.

[32] R.D. Falgout, J.E. Jones, and U.M. Yang. *Numerical Solution of Partial Differential Equations on Parallel Computers*, chapter The Design and Implementation of Hypre, a Library of Parallel High Performance Preconditioners, pages 267–294. Springer-Verlag, 2006.

[33] C. Farhat and F. X Roux. Implicit parallel processing in structural mechanics. *Computational Mechanics Advances*, 2(1) :1–124, 1994.

[34] Charbel Farhat and Francois-Xavier Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Int. J. Numer. Meth. Engng.*, 32(6) :1205–1227, 1991.

[35] J. Frank and C. Vuik. On the construction of Deflation-Based preconditioners. 2002.

[36] M. Garbey and D. Tromeur-Dervout. On some Aitken-like acceleration of the Schwarz method. *Internat. J. Numer. Methods Fluids*, 40(12) :1493–1513, 2002.

[37] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10 :345, 1973.

[38] Laura Grigori, Pawan Kumar, Frederic Nataf, and Ke Wang. A class of multilevel parallel preconditioning strategies. Rapport de recherche RR-7410, INRIA, October 2010.

[39] A. Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Transactions on Mathematical Software*, 28(3) :301–324, 2002.

[40] F. Hecht, O. Pironneau, A. Le Hyaric, and K. Ohtsuka. Freefem++, second edition. Technical report, Laboratory Jacques-Louis Lions, University Pierre et Marie Curie, 2007.

[41] M. R Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6) :409–436, 1952.

[42] H. Hoteit, J. Erhel, R. Mosé, B. Philippe, and P. Ackerer. Numerical reliability for mixed methods applied to flow problems in porous media. *Computational Geosciences*, 6 :161–194, 2002.

[43] H. Hoteit, R. Mosé, B. Philippe, P. Ackerer, and J. Erhel. The maximum principle violations of the mixed-hybrid finite-element method applied to diffusion equations. *International Journal for Numerical Methods in Engineering*, 55(12) :1373–1390, 2002.

[44] T. Kalbacher, R. Mettier, C. McDermott, W. Wang, G. Kosakowski, T. Taniguchi, and O. Kolditz. Geometric modelling and object-oriented software concepts applied to a heterogeneous fractured network from the Grimsel rock laboratory. *Computational Geosciences*, 11(1) :9–26, Mar 2007.

[45] J. C. S. Long, J. S. Remer, C. R. Wilson, and P. A. Witherspoon. Porous media equivalents for networks of discontinuous fractures. *Water Resources Research*, 18(3) :645–658, 1982.

[46] Jane C. S. Long, Peggy Gilmour, and Paul A. Witherspoon. A model for steady fluid flow in random three-dimensional networks of disc-shaped fractures. *Water Resources Research*, 21(8) :1105–1115, 1985.

[47] Jan Mandel and Marian Brezina. *Balancing domain decomposition.* University of Colorado at Denver, 1993.

[48] Lois Mansfield. On the conjugate gradient solution of the schur complement system obtained from domain decomposition. *SIAM J. Numer. Anal.*, 27(6) :1612–1620, 1990.

[49] Lois Mansfield. Damped jacobi preconditioning and coarse grid deflation for conjugate gradient iteration on parallel computers. *SIAM J. Sci. Stat. Comput.*, 12(6) :1314–1323, 1991.

[50] Vincent Martin, Jérôme Jaffré, and Jean E. Roberts. Modeling fractures and barriers as interfaces for flow in porous media. *SIAM Journal on Scientific Computing*, 26 :1667–1691, 2005.

[51] J. Maryska, O. Severýn, and M. Vohralík. Numerical simulation of fracture flow with a mixed-hybrid fem stochastic discrete fracture network model. *Computational Geosciences*, 8 :217–234, 2004.

[52] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of Computation*, 31(137) :148–162, January 1977. ArticleType : research-article / Full publication date : Jan., 1977 / Copyright © 1977 American Mathematical Society.

[53] Gérard Meurant. *Computer solution of large linear systems*. North-Holland, June 1999.

[54] H. Mustapha. *Simulation numérique de l'écoulement dans des milieux fracturés tridimensionnels*. thèse de doctorat, University of Rennes 1, october 2005.

[55] H. Mustapha and K. Mustapha. A new approach to simulating flow in discrete fracture networks with an optimized mesh. *SIAM Journal on Scientific Computing (SISC)*, 29(4) :1439–1459, 2007.

[56] R. Nabben and C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM Journal on Numerical Analysis*, 42(4) :1631–1647, 2005.

[57] Shlomo P. Neuman. Trends, prospects and challenges in quantifying flow and transport through fractured rocks. *Hydrogeology Journal*, 13(1) :124–147, 2005.

[58] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2) :355–365, 1987.

[59] A. Wille Nordqvist, Y. W. Tsang, C. F. Tsang, Björn Dverstop, and Johan Andersson. A variable aperture fracture network model for flow and transport in fractured rocks. *Water Resources Research*, 28(6) :1703–1713, 1992.

[60] G. Pichot, J. Erhel, and J-R. de Dreuzy. A mixed hybrid mortar method for solving flow in discrete fracture networks. *Applicable Analysis*, 89(10) :1629–1643, 2010.

[61] G. Pichot, J. Erhel, and J.R. de Dreuzy. A generalized mixed hybrid mortar method for solving flow in stochastic discrete fracture networks. *SIAM Journal on scientific computations*, submitted.

[62] J. S. Przemieniecki. Matrix structural analysis of substructures. *AIAA Journal*, 1 :138–147, January 1963.

[63] Alfio Quarteroni and Alberto Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, 1999.

[64] JW Ruge, K Stuben, and SF Mccormick. Algebraic multigrid. In *Multigrid Methods*, pages 73–130. SIAM, 1987.

[65] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial Mathematics, 2003.

[66] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM J. SCI. COMPUT*, 21 :1909—1926, 2000.

[67] Yousef Saad and Brian Suchomel. ARMS : an algebraic recursive multilevel solver for general sparse linear systems. *NUMER. LINEAR ALG. APPL*, 9 :2002, 1999.

[68] Barry F. Smith, Petter E. Bjørstad, and William Gropp. *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, March 2004.

[69] P.Le Tallec, Y.H.De Roeck, and M. Vidrascu. Domain decomposition methods for large linearly elliptic three-dimensional problems. *Journal of Computational and Applied Mathematics*, 34(1) :93–117, 1991.

[70] A. Toselli and O. B Widlund. *Domain decomposition methods–algorithms and theory*. Springer Verlag, 2005.

[71] J. Verkaik. Deflated Krylov-Schwarz domain decomposition methods for the CFD package x-stream. 2003.

[72] M. Vohralík, J. Maryska, and O. Severýn. Mixed and nonconforming finite element methods on a system of polygons. *Applied Numerical Mathematics*, 57 :176–193, 2007.

Résumé

Les ressources souterraines fournissent une part importante de l'eau douce de notre planète. Notre travail s'inscrit dans une démarche de protection de cette ressource vitale par la modélisation et la simulation numérique. Couplée aux études de terrains, la simulation numérique est en effet un outil indispensable, du fait de l'incertitude sur le milieu géologique. Cette incertitude conduit à une approche stochastique de la modélisation. Nous nous sommes concentrés sur les écoulements dans les réseaux de fractures générés aléatoirement.

Pour permettre la résolution de ces écoulements par une méthode d'éléments finis mixte hybride, nous avons élaboré un algorithme de maillage spécifique aux fractures. Cette technique permet de construire le système linéaire quelle que soit la géométrie du réseau généré stochastiquement. Nous avons ensuite effectué une étude comparative de trois solveurs linéaires : un solveur direct, un multigrille algébrique et un Gradient Conjugué Préconditionné. Cette étude nous a conduit à proposer une méthode de résolution plus efficace pour ce problème.

Nous avons alors étudié une méthode de décomposition de domaine de type Schur, qui permet d'allier les avantages du solveur direct et du Gradient Conjugué Préconditionné. Cette méthode consiste à réduire le problème à un problème aux interfaces, par une définition naturelle des fractures, ou paquets de fractures, comme sous-domaines. Nous avons proposé une approche originale d'optimisation de l'algorithme et un préconditionnement global de type déflation. Notre implémentation de cette méthode est compétitive. Elle permet en effet de résoudre tous les cas tests étudiés. De plus, elle est plus rapide que les trois autres solveurs considérés dans la majorité des cas.

Abstract

The major part of freshwater comes from sub-surface resources. Our work fits in a process to protect this vital resource, based on modelisation and numerical simulation. In conjunction with field studies, numerical simulations have to be used, due to the geological uncertainties. These uncertainties lead to stochastic modelisation. We concentrate on flow in fracture networks, randomly generated.

To be able to solve those flow by a mixed hybrid finite elements method, we develop a meshing algorithm for the fractures. Then we are able to construct the linear systems, whatever the network randomly generated geometry. Then we compare three linear solvers : a direct solver, an algebraic multigrid and a Preconditioned Conjugate Gradient. This study leads us to propose a more efficient method, better suited to our problem solving.

We study a domain decomposition method, which takes advantages from both the direct method and the Preconditioned Conjugate Gradient. This Schur method reduces the global problem to an interface problem, with a natural domain decomposition based on fractures or fracture packs. We propose an original approach for optimizing the algorithm and a global preconditioning of deflation type. Our implementation is efficient, since it succeeds to solve all the systems tested and is, in most cases, faster than the other linear solvers.