## Résumé

Ce rapport présente le travail effectué au sein de l'UMR EMMAH à Avignon sur le code éléments finis Fafemo. Ce code donne une solution numérique des équations non linéaires de Richards traduisant les écoulements dans les milieux poreux non saturés. J'ai d'abord amélioré le code séquentiel en utilisant des méthodes itératives pour résoudre les systèmes linéaires creux (PETSc) et le mailleur Gmsh. Les coûts de calcul ont été significativement réduis. Cependant, l'objectif principal de ce stage a été de concevoir une version parallèle et efficace de Fafemo. J'ai effectué les tests de cette version parallèle sur deux clusters (Migale et Jade, le plus puissant cluster français). Ce travail a permis de faire un premier pas vers la modélisation du comportement de milieux poreux fortement hétérogènes comme les substrats des toitures terrasses végétalisées.

**Mots clés :** écoulements d'eau, milieux non saturés, éléments finis, systèmes linéaires, méthodes iteratives, preconditionnement, calcul parallèle.

## Abstract

This manuscrit presents the work done on Fafemo software based on finite element methods at the EMMAH laboratory in Avignon. This software gives us numerical solutions of Richards non linear equations for water flow in unsaturated porous media. First I improved the software using iterative methods to solve sparse linear systems (PETSc) and Gmsh as mesher. Computational costs have been significantly reduced. But the main goal of this training course was to develop a parallel version of Fafemo. This version was tested on Migale and Jade (the most powerful French cluster). This works leads to a first step in simulating water flow in heterogeneous ground with application to green roofs.

**Keywords :** water flow, unsaturated media, finite element, sparse linear systems, iterative methods, preconditioning techniques, parallel computing.

*Je tiens à remercier mes tuteurs Samuel Buis et Arnaud Mesgouez ainsi que sa femme Gaëlle pour leur disponibilité et leur confiance. Je tiens également à remercier Stéphane Ruy pour le temps qu'il m'a accordé, et pour m'avoir fait découvrir le volet expérimental de la problématique. Merci aussi à tous ceux que j'ai pu croiser que ce soit à l'Université d'Avignon ou à l'INRA pour leur accueil.*

# Table des matières

# Introduction

### Présentation de L'INRA - EMMAH

Le dispositif de recherche de l'Institut National de la Recherche Agronomique est structuré en 14 départements comprenant chacun plusieurs unités réparties dans des centres de recherche sur tout le territoire national. Mon stage s'est déroulé au sein de l'unité mixte de recherche EMMAH (Environnement Méditerranéen et Modélisation des AgroHydrosystèmes) du pôle Adaptation au Changement Global (ACG) d'Avignon. Cette unité mixte de recherche regroupe des enseignants-chercheurs et des chercheurs de l'Université d'Avignon et des Pays de Vaucluse (UAPV) et de l'Institut National de Recherche Agronomique (INRA) Site d'Avignon. Elle s'intéresse à l'impact des changements globaux sur les agro-hydrosystèmes méditerranéens. Les recherches qui y sont menées portent notamment sur les conséquences des évènements climatiques extrêmes, pluie forte ou sècheresse intense par exemple, sur la qualité et la quantité des ressources hydriques disponibles dans le milieu naturel. Dans ce contexte, la description réaliste et la caractérisation des transferts de masse, eau ou solutés, et d'énergie dans le continuum végétation / sol hétérogène / nappe sont cruciales pour optimiser les ressources environnementales, prédire l'impact des pratiques humaines agricoles et industrielles, ou encore quantifier la recharge et la qualité physico-chimique des nappes souterraines.

### Objectifs du stage

Le stage s'inscrit dans la thématique de la modélisation des écoulements d'eau dans les milieux poreux non saturés. Je suis intervenu sur le développement du code Fafemo qui propose une approche par éléments finis en espace et différences finies en temps de cette problématique. L'objectif du stage a été d'améliorer les performances de ce code, et d'en implémenter une version parallèle afin de pouvoir traiter des cas d'étude réels tels que le transfert d'eau du sol vers nappe phréatique. La modélisation de ces phénomènes demande d'effectuer des calculs tellement importants qu'aujourd'hui encore, beaucoup de simulations sont effectuées en 2D, certaines même en 1D, ce qui ne permet pas de simuler des environnements réels complexes. Fafemo possède l'avantage de traiter aussi bien les problèmes en 2D (donc simplifiés) que les problèmes en 3D (réels). Cependant, cela représente un coût de calcul pénalisant, qui l'empêchait de traiter des problèmes de grandes dimensions, ou de simuler de longues périodes de temps. L'ambition de Fafemo étant de devenir l'outil de référence au sein de l'UMR, il fallait pallier à ces problèmes de performance.

Dans ce rapport de stage je vais d'abord présenter la physique du problème, la méthode de résolution choisie et le code tel qu'il était au début du stage. Ensuite je présenterai le travail que j'ai effectué : l'optimisation du code séquentiel, sa parallélisation et l'application à un cas d'étude.

# 1 Modélisation des transferts d'eau par le code Fafemo

## 1.1 La physique du problème

L'eau s'infiltre dans le sol sous l'effet de la capillarité et de la gravité. La propagation du front d'infiltration dans le sol dépend de sa composition et de son état de sécheresse. Deux grandeurs vont nous intéresser : la teneur en eau notée $\theta$ et la pression ou potentiel ou encore hauteur d'eau noté $h$. Ici, les pressions sont négatives ; elles varient typiquement de $h = 0$ pour un sol saturé en eau, à -100m pour un sol extrêmement sec. L'eau se propage donc dans le sol sous la forme d'un front d'infiltration, c'est-à-dire d'une zone où l'on passe brutalement d'une teneur en eau importante à une teneur en eau très faible. Les écoulements en milieux poreux non saturés sont régis par les équations de Richards, ces équations permettent de simuler indifféremment les phénomènes d'infiltration mais également d'évaporation selon les conditions limites fixées.

### 1.1.1 Equations de Richards

Il existe trois formes des équations de Richards. Nous travaillons uniquement avec la forme dite mixte qui fait intervenir à la fois la pression et la teneur en eau car la discrétisation de celle-ci assure la conservation de la masse. Cette équation est donc :

$$\frac{\partial \theta(h)}{\partial t} = \nabla \cdot K(h) \nabla h + \frac{\partial K(h)}{\partial z} \tag{1.1}$$

Où $K(h)$ est la loi de comportement de la conductivité hydraulique. Nous faisons un changement de variable $u(h) = h + z$ ce qui donne :

$$\frac{\partial \theta(h)}{\partial t} - \nabla \cdot (K(h) \nabla u(h)) = \frac{\partial \theta(h)}{\partial t} - \nabla \cdot K(h) \nabla u(h) - K(h) \Delta u(h) = 0 \tag{1.2}$$

Ces équations sont non-linéaires, il suffit de regarder par exemple la courbe de $K$ en fonction de $h$ (Fig.1.3) pour s'en rendre compte.

### 1.1.2 Les lois de comportement

On s'intéresse ici aux lois de Van Genutchen modifiées, qui sont une généralisation des lois de Van Genutchen classiques, les plus utilisées dans ce domaine. Avant tout, il faut introduire les notations suivantes :

- $z$ est la hauteur, qui est négative ici (décroissante depuis la surface z=0)
- $\theta(h)$ est la teneur en eau du sol
- $C(h)$ est la capacité capillaire
- $K(h)$ est la conductivité hydraulique.

#### 1.1.2.1 Teneur en eau du sol $\theta(h)$

Appelée aussi fonction de rétention, elle relie la teneur en eau au potentiel, c'est-à-dire à la pression. Son graphe pour deux matériaux est présenté en Figure 1.1, on y voit que la teneur en eau augmente de manière brutale lorsque la pression approche 0. On définit les

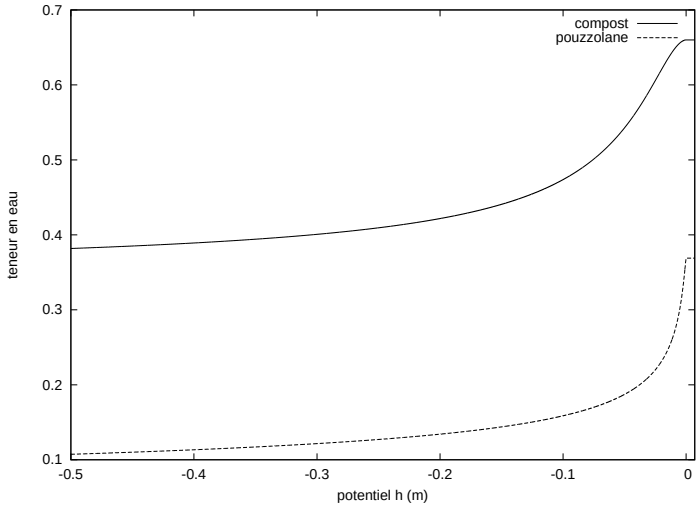


FIG. 1.1 – *Teneur en eau $\theta$ fonction de la pression $h$ en mètres*

paramètres suivants, caractéristiques des matériaux :

- $\theta_s$ : la teneur en eau à saturation, c'est-à-dire la teneur en eau maximale, $\theta_s \simeq 0.9n$
- $\theta_r$ : la teneur en eau résiduelle
- $h_a$ : la pression d'entrée d'air, de l'ordre de quelques centimètres en valeurs négatives
- $S_e^*$ : un paramètre correctif, sans dimension, tenant compte de la pression d'entrée d'air.
- $n$ et $m = 1 - \frac{1}{n}$ des paramètres de forme.

Voici donc l'équation reliant $\theta$ et h :

$$\begin{cases} \frac{\theta(h)-\theta_r}{\theta_s-\theta_r} = \frac{1}{S_e^*}\left[1+\left(\frac{h}{h_e}\right)^n\right]^{-m} \text{ si } h < h_a \\ \frac{\theta(h)-\theta_r}{\theta_s-\theta_r} = 1 \text{ soit } \theta(h) = \theta_s \text{ si } h \geq h_a \end{cases} \tag{1.3}$$

avec

$$S_e^* = \left[1+\left(\frac{h_a}{h_e}\right)^n\right]^{-m} \tag{1.4}$$

#### 1.1.2.2 Capacité capillaire $C(h)$

La capacité capillaire traduit la capacité du sol à retenir l'eau par capillarité. Elle est donc définie par la dérivée de $\theta$ par rapport au potentiel $h$. C'est également une fonction plus ou moins raide et non monotone selon les matériaux, comme le montre la Figure 1.2. Les lois

de Van Genutchen modifiées donnent :

$$\begin{cases} C(h) = \frac{d\theta(h)}{dh} = -\frac{nm(\theta_s - \theta_r))}{h_e S_e^*} \left(\frac{h}{h_e}\right)^{n-1} \left[1 + \left(\frac{h}{h_e}\right)^n\right]^{-m-1} & \text{si } h < h_a \\ C(h) = \text{compressibilité de l'eau} = 5e^{-10} \text{ si } h \geq h_a \end{cases} \tag{1.5}$$

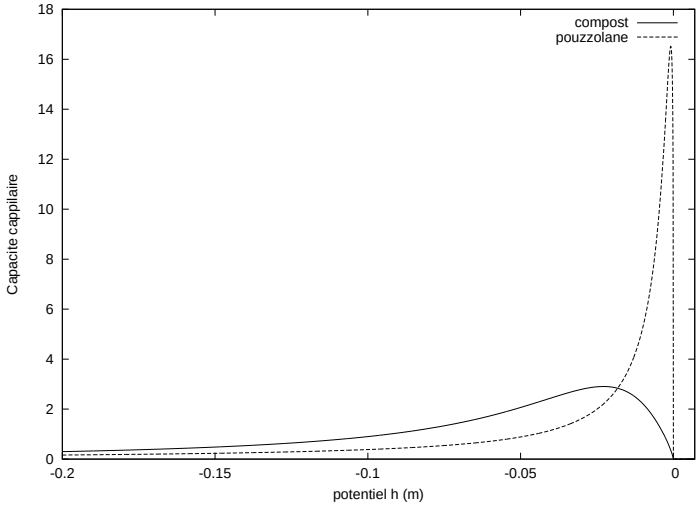


FIG. 1.2 – *Capacité capillaire C fonction de la pression h en mètres, pour différents matériaux.*

#### 1.1.2.3 Coefficient de conductivité hydraulique $K(h)$

Ce coefficient exprime l'aptitude du milieu poreux à laisser circuler l'eau à travers lui. L'évolution de la conductivité hydraulique en fonction de la pression est donnée par la relation suivante :

$$\begin{cases} K(\theta) = K_s.\Theta^{0.5} \frac{\left[1-\left(1-(S_e^*\Theta)^{1/m}\right)^m\right]^2}{\left[1-\left(1-S_e^{*1/m}\right)^m\right]^2}, \text{ avec} \\ \Theta = \frac{\theta-\theta_r}{\theta_s-\theta_r} \text{ , si } h < h_a < 0 \; K(\theta) = K_s \text{ si } \Theta \geq S_e^* \text{ soit si } h \geq h_a \end{cases} \tag{1.6}$$

Un exemple de courbe du coefficient de conductivité hydraulique en fonction de la pression est tracée en Fig.1.3 pour deux matériaux. On y voit que la conductivité varie brutalement lorsque $h$ s'approche de 0 par les valeurs négatives, et pour $h > 0$ la conductivité vaut $K_s$ qui est le coefficient de conductivité à saturation .

## 1.2 Résolution numérique choisie

### 1.2.1 Formulation éléments finis

La méthode de résolution choisie [1] utilise les éléments finis de type linéaires par morceaux (P1). Nous multiplions l'équation 1.2 par les fonctions test notées $\psi$, puis nous intégrons sur

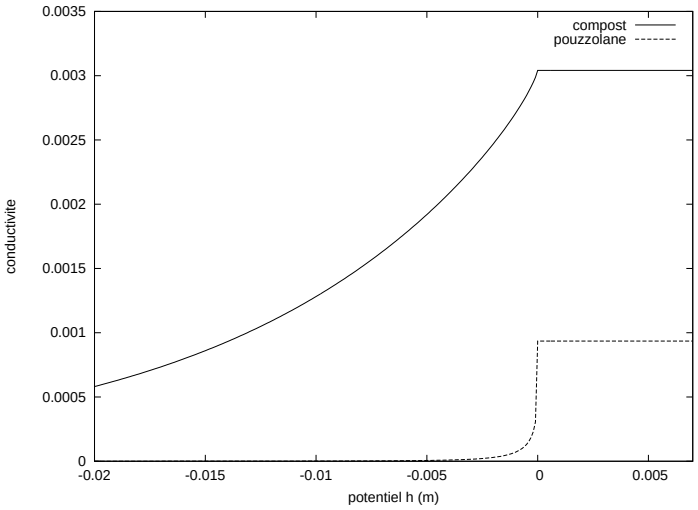


FIG. 1.3 – *Conductivité hydraulique $K$ fonction de la pression $h$ en mètres pour deux matériaux*

le domaine $\Omega$.

$$\int_\Omega \frac{\partial \theta(h)}{\partial t}\psi d\Omega - \int_\Omega [\nabla \cdot K(h)\nabla u(h) - K(h)\Delta u(h)]\psi d\Omega = 0 \tag{1.7}$$

Après avoir abaissé le degré de régularité demandé à $u$ avec une intégration par partie, et en notant $n$ la dimension du problème et $T$ la durée de simulation, la formulation faible s'écrit :

$$\begin{cases} \text{Trouver u} \in V = [0,T] \times \{\frac{\partial u}{\partial t} \in L^2(\Omega), f \in H_1(\Omega)\}^n \text{ Tel que} \\ \int_\Omega \dot{\theta}\delta u d\Omega - \int_\Gamma K(h)(ue_i)_{,i}\delta u d\Gamma + \int_\Omega K(h)u_{,i}\delta u_{,i} d\Omega = 0 \end{cases} \tag{1.8}$$

Si E est le nombre de points de discrétisation, la solution cherchée est de la forme :

$$u(h,t) = \sum_{0<j\leq E} u_j(t) \cdot \psi_j \tag{1.9}$$

## 1.2.2 Schéma temporel

### 1.2.2.1 Discrétisation en temps

La discrétisation en temps s'effectue avec un schéma de type Euler implicite :

$$\frac{\partial \theta^{t+1}}{\partial t} = \frac{\theta^{t+1} - \theta^t}{\Delta t} \tag{1.10}$$

En notant $M$ la matrice de masse, nous avons donc à résoudre le sytème matriciel suivant à chaque pas de temps :

$$\frac{M^{t+1}}{\Delta t}\theta^{t+1} - \frac{M^{t+1}}{\Delta t}\theta^t + K^{t+1}u^{t+1} = F^{t+1} \tag{1.11}$$

#### 1.2.2.2 Linéarisation

Une méthode de Picard (ou de substitution) est utilisée pour linéariser le problème :

$$\frac{M^{t+1,m}}{\Delta t}\theta^{t+1,m+1} - \frac{M^{t+1,m}}{\Delta t}\theta^{t} + K^{t+1,m}u^{t+1,m+1} = F^{t+1,m} \tag{1.12}$$

Le deuxième exposant désigne l'itération interne de la méthode de Picard. Notons $\delta_h = h^{t+1,m+1} - h^{t+1,m}$ et $\delta_\theta = \theta^{t+1,m+1} - \theta^{t+1,m}$. La résolution du système se fait en $u$. Il faut donc approcher le terme $\theta^{t+1,m+1}$ par son développement de Taylor en $h$.

$$\theta^{t+1,m+1} = \theta^{t+1,m} + \left\{\frac{d\theta}{dh}\right\}^{t+1,m}\delta_h + O(\delta_h^2)... \tag{1.13}$$

En utilisant le fait que $C(h)^{t+1,m} \approx \frac{d\theta}{dh}^{t+1,m}$, et $\delta h = h^{t+1,m+1} - h^{t+1,m} = u^{t+1,m+1} - z - u^{t+1,m} + z = u^{t+1,m+1} - u^{t+1,m} = \delta_u$ nous obtenons :

$$\frac{M^{t+1,m}C(h)^{t+1,m}}{\Delta t}\delta_u + K^{t+1,m}\delta_u = F^{t+1,m} - \frac{M^{t+1,m}}{\Delta t}\theta^{t+1,m} + \frac{M^{t+1,m}}{\Delta t}\theta^{t} - K^{t+1,m}u^{t+1,m} \tag{1.14}$$

Nous nous sommes donc enfin ramenés à la résolution d'un système linéaire de la forme $A^{t+1,m}\delta_u = b^{t+1,m}$ à chaque itération interne de la méthode de Picard avec :

$$\begin{cases} A^{t+1,m} = \left[\frac{M^{t+1,m}C(h)^{t+1,m}}{\Delta t} + K(h)^{t+1,m}\right] \\ b^{t+1,m} = F^{t+1,m} - \frac{M^{t+1,m}}{\Delta t}\theta^{t+1,m} + \frac{M^{t+1,m}}{\Delta t}\theta^{t} - K^{t+1,m}u^{t+1,m} \end{cases} \tag{1.15}$$

#### 1.2.2.3 Critère de convergence

Fafemo utilise par défaut un critère de convergence relatif : $\frac{|\delta_u^m|}{|u^m|} \leq \epsilon_e$. Toutefois ce critère est restrictif car l'erreur tolérée sur les sols secs devient très petite. Un critère de convergence absolue pourrait être utilisé pour avoir une convergence plus rapide : $|\delta u^m| = |h^{t+1,m+1} - h^{t+1,m}| \leq \epsilon_a$. L'utilisation d'un critère mixte $|\delta u^m| \leq \epsilon_e|h^{t+1,m+1}| + \epsilon_a$ permettrait d'avoir une tolérance minimale absolue sur les sols secs, car sur les sols secs une petite variation de $h$ entraîne très peu de variations sur $\theta$.

Dans la publication de Huang [2], on comprend bien qu'utiliser une norme sur $\theta$ revient à être plus tolérant lorsque la pression est faible, et de plus en plus restrictif à mesure que l'eau s'infiltre. Cette norme semble donc plus adaptée. Notons $\epsilon_\theta$ la tolérance sur $\theta$ et $\epsilon_h$ la tolérance sur la pression. Une des principales questions est donc de savoir quel est le lien entre les valeurs de ces deux tolérances. En utilisant la norme sur $h$, les itérations de Picard s'arrêtent lorsque :

$$|\delta_h| \leq \epsilon_h \tag{1.16}$$

Et en utilisant la norme sur $\theta$, les itérations de Picard s'arrêtent lorsque :

$$|\delta_\theta| \leq \epsilon_\theta \tag{1.17}$$

En utilisant l'équation 1.13 il vient :

$$\delta_\theta = \left\{\frac{d\theta}{dh}\right\}^{t+1,m}\delta_h + O(\delta_h^2)... \tag{1.18}$$

En utilisant l'approximation $\frac{d\theta}{dh} = C^{t+1,m}$ on obtient :

$$|\delta_\theta| \leq |C^{t+1,m}\delta_h| \tag{1.19}$$

Et donc :

$$|\delta_\theta| \leq |C^{t+1,m}||\delta_h| \tag{1.20}$$

Finalement nous avons la relation :

$$|\delta_h| \leq \epsilon_h \Rightarrow |\delta_\theta| \leq |C^{t+1,m}||\epsilon_h| \tag{1.21}$$

En choisissant $\epsilon_\theta = |C|\epsilon_h$ on est donc sûr d'avoir la même qualité d'approximation de $\theta$ au niveau du front d'infiltration avec les deux critères, les zones plus sèches seront mieux approchées avec le critère sur $h$.

#### 1.2.2.4 Adaptation du pas de temps

Le calcul du pas de temps est régi par un système de pas adaptatif selon l'algorithme suivant :

- On se fixe un nombre d'itérations de Picard maximum $I_{max}$
- Pour chaque pas de temps :
  - Si la méthode a convergé, le pas de temps est accepté.
    - Si la méthode a convergé en moins de $0.3 \times I_{max}$ itérations, le pas de temps est multiplié par 1.3.
    - Si la méthode a convergé en plus de $0.7 \times I_{max}$ itérations, le pas de temps est multiplié par 0.7.
  - Si la méthode n'a pas convergé, le pas de temps est rejeté. Un nouvel essai est effectué avec un pas de temps 3 fois plus petit.

## 1.3 Le code Fafemo

### 1.3.1 Architecture générale

Fafemo est une plateforme permettant de traiter plusieurs problèmes liés au sol, tels que les écoulements d'eau mais aussi la propagation d'ondes par une méthode d'éléments finis 2D ou 3D (Mesgouez et al.[7] et [8]). C'est un code écrit en C++ orienté objet, d'environ 2000 lignes. Pour passer d'un problème physique à un autre il faut compiler les classes correspondantes. Il existe cinq classes en héritage simple. Les principales fonctionnalités de chacune sont représentées dans le diagramme des classes présenté en annexe Fig.D.1 page vii. Il existe plusieurs versions de ces classes, selon le problème traité, le mailleur etc. L'articulation des classes est présenté en annexe Fig.D.2 page viii .

### 1.3.2 Fonctionnement du code

Fafemo utilisait exclusivement le mailleur GID, le maillage était sauvegardé sous la forme d'un fichier texte qui devait être traité par un autre programme avant de pouvoir être lu par Fafemo. La plupart des données devaient être compilées, que ce soit les informations sur les matériaux, les tolérances, les informations sur les sorties à effectuer.

L'algorithme de Fafemo est finalement assez simple. Il faut cependant bien comprendre qu'il s'agit d'un problème instationnaire non-linéaire, c'est-à-dire que pour chaque pas de temps la méthode de Picard demande d'inverser de nombreux systèmes linéaires. Ces inversions se faisaient directement depuis Fafemo par une méthode directe. De plus, la matrice était stockée sous un format dit skyline ou bande. Voici l'algorithme :

```
Lecture du maillage
Lecture des matériaux
Conditions limites, conditions initiales
Tant que t < Tmax
  Tant que non convergence
      Construction de A et de b
      Inversion du système : Ax=b
  Fin tant que
  Gestion du pas de temps
  Sorties
Fin tant que
```

# 2 Optimisations du code séquentiel

L'optimisation du code séquentiel vise à réduire les temps de calcul et à optimiser la gestion de la mémoire afin de pouvoir traiter des problèmes de taille plus importante. Le diagnostic du code a été effectué par une analyse visuelle et par le profileur Valgrind.

## 2.1 Taille des problèmes gérés

### 2.1.1 Allocation dynamique

Jusqu'ici des informations relatives au maillage telles que le nombre de nœuds et le nombre d'éléments étaient compilées, ce qui permettait une allocation statique de presque tous les tableaux. J'ai modifié le code pour utiliser l'allocation dynamique, qui a deux avantages principaux : traiter des problèmes de taille plus grande, car nous allouons des zones non contiguës, et lire les informations du maillage après la compilation, ce qui permet d'avoir un seul exécutable pour traiter plusieurs problèmes.

### 2.1.2 Mailleur GMSH

L'un des premiers travaux que j'ai effectué a été de rendre le code compatible avec le mailleur GMSH. Gid présentait les inconvénients d'être payant, et de ne pas pouvoir générer de gros maillages. Nous avons donc choisi de travailler avec GMSH qui est le plus populaire des mailleurs libres, et qui permet de générer aisément des maillages de quelques milions de nœuds.

La principale difficulté a été de gérer les conditions aux limites. En effet, GMSH ne permet pas de gérer directement les conditions aux limites ; celles-ci se présentent sous la forme d'éléments dans la table de connectivité. Par exemple en 2D, la table de connectivité contient aussi bien des triangles (nos éléments finis) que des segments (conditions de flux et de Dirichlet). À la lecture du fichier *input.msh*, il faut donc séparer ces informations : les éléments finis sont stockés à travers la table de connectivité, les flux sont stockés dans un objet implémenté pour l'occasion. Cette classe *Flux* permet un stockage creux des flux : seules les arêtes effectivement soumises à un flux sont stockées, ce qui n'était pas le cas auparavant. Cette classe stocke également les numéros des nœuds de la facette soumise à un flux ; auparavant on effectuait une recherche pour retrouver ces numéros.

### 2.1.3 Stockage creux

La matrice représente le plus gros coût de stockage. Elle était stockée sous le format *skyline* c'est-à-dire que l'on stockait une bande centrée autour de la diagonale. Cependant, cette bande était toujours constituée à 90% de 0. De plus, le stockage par bande n'est pas compatible avec l'utilisation du Mailleur Gmsh. En effet, GID renumérote les nœuds selon

un algorithme RCM (Reverse Cuthill McKee) qui minimise la largeur de la bande ; ce n'est plus le cas avec Gmsh. Pour résoudre ce problème nous avons choisi de passer en stockage creux, c'est à dire de ne stocker que les éléments non nuls de la matrice. Il fallait donc également changer de méthode de résolution afin qu'elle soit adaptée à ce stockage. Nous avons choisi d'utiliser la bibliothèque PETSc [6] qui permet un large choix de méthodes de préconditionnement et de résolution en séquentiel et en parallèle. De plus, il s'agit d'une bibliothèque libre et largement utilisée. Cela a un fort impact sur le code puisque le stockage creux et la résolution s'effectuent directement à l'aide des formats et fonctions que fournit PETSc.

## 2.2 Coûts de calcul

### 2.2.1 Méthodes de Krylov et préconditionnement

Que ce soit avec une méthode directe ou une méthode itérative, le stockage creux permet de n'effectuer les opérations que sur les éléments non nuls. Le choix de la méthode de préconditionnement et de la méthode de résolution est un vaste problème. J'ai effectué de nombreux tests, notamment pour savoir ce qui impactait sur le nombre de condition de la matrice. Le nombre de conditions des systèmes est très variable d'un problème à un autre. Il dépend en priorité du pas de temps, du maillage (taille et régularité des mailles) et dans une plus faible mesure des matériaux.

En ce qui concerne les problèmes d'infiltration, le conditionnement du système n'est pas un problème puisque le pas de temps est limité par le système adaptatif. Ainsi dans ce genre de problème, les pas de temps sont de l'ordre de la seconde ou de la minute. Les systèmes sont donc suffisament bien conditionnés pour permettre une convergence rapide, et le code passe au maximum 30% du temps CPU à résoudre les systèmes linéaires.

L'évaporation est un phénomène beaucoup plus lent. Les pas de temps sont donc bien plus grands (de l'ordre de l'heure) et les systèmes beaucoup plus mal conditionnés. La résolution nécessite donc beaucoup plus d'itérations de la part de la méthode de Krylov, et finalement plus de 95% du temps CPU est utilisé pour inverser les systèmes linéaires.

Pour les problèmes de très petites tailles, tels que les problèmes de type 1D, l'utilisation d'un ILU(1) avec un gradient conjugué (ici les matrices sont symétriques) s'avère efficace. Pour les autres problèmes, le couple ILU(0)-Gmres permet d'avoir des performances assez bonnes, et surtout stables selon les cas. Il est important de noter que ces conclusions ne sont valables que dans le cas du code séquentiel, car par exemple les préconditionneurs de type ILU pour ne sont pas implémentés par défaut pour le cas paralléle dans PETSc.

### 2.2.2 Calcul des lois de comportement

La présence de puissances réelles dans les lois de comportement est un vrai problème : grâce aux profileurs, j'ai constaté que le code pouvait passer jusqu'à 40% de son temps dans

l'appel à la fonction `pow`. J'ai donc effectué des modifications structurelles du code pour éviter des doubles appels aux fonctions contenant les lois de comportement. Le calcul de certains termes en amont a également permis de limiter le nombre d'appels à la fonction `pow`. En effet les lois sont calculées consécutivement en un même point, nous pouvons donc sauvegarder les termes $(\frac{h}{h_e})^n$ et $[1+(\frac{h}{h_e})^n]^{-m}$ qui sont présents plusieurs fois dans les lois de comportement (équations 1.3, 1.5 et 1.6).

### 2.2.3 Autres améliorations

La gestion des conditions de Dirichlet a été modifiée. Avant, les inconnues soumises à des Dirichlet étaient retirées de la matrice. Lorsqu'une condition limite change au cours du temps, comme par exemple lorsqu'une condition de flux devient une condition de Dirichlet quand la surface sature, la taille des matrices, qui concerne les problèmes d'infiltration, de l'inconnue et du second membre changent. Pour garder une numérotation contiguë des inconnues il fallait donc renuméroter l'ensemble du domaine. Maintenant les conditions de Dirichlet sont laissées dans la matrice, ce qui permet de gagner du temps puisque l'allocation de la matrice se fait une fois pour toutes.

Le code permet de traiter d'autres problèmes physiques, notamment des problèmes où il y a plusieurs degrés de liberté par nœud. Dans la version du code sur laquelle j'ai travaillé, il n'y a qu'un seul degré de liberté par nœud. J'ai donc mis en place un système de balises de précompilation qui permet d'éviter les calculs liés aux multiples degrés de liberté. Par exemple, les boucles sur le nombre de degrés de liberté ne sont plus compilées.

## 2.3 Ergonomie du code

### 2.3.1 Gestion des sorties

Historiquement la solution était écrite tous les $n$ pas de temps dans un fichier. Avec la méthode de pas adaptatif, nous ne pouvions donc pas obtenir la solution à un temps exact. J'ai modifié le code, et notamment la gestion du pas adaptatif pour pouvoir imprimer la solution tous les $\Delta_t sortie$. L'impression de la solution se fait sous un format directement lisible par Gnuplot. En dehors de Fafemo, j'ai également implémenté un script qui crée un fichier *post.msh* contenant les informations sur la solution et sur le maillage. Ce fichier est directement lisible par Gmsh et permet de manipuler aisément la solution graphique. Les sorties à l'écran ont également été revues. Enfin j'ai créé un fichier contenant des informations sur l'aspect numérique du problème (nombre d'itérations par pas de temps, erreurs etc).

### 2.3.2 Utilisation des différentes configurations

La structure du code a été modifiée de telle sorte que les conditions limites et les conditions initiales soient dans un fichier séparé du reste du code. Pour passer d'un problème à un autre, c'est uniquement ce fichier qu'il faut modifier. J'ai également réuni dans un seul code les classes permettant de traiter les problèmes 2D et 3D et des options de précompilation permettant de passer d'une dimension à l'autre.

## 2.4 Résultats

Toutes ces améliorations ont eu un fort impact sur le code, qui est passé de 2000 à 4000 lignes. Cependant, la structure du code a été conservée. Auparavant, la mémoire était un facteur limitant : un ordinateur de bureau ne supportait que quelques dizaines de milliers de nœuds. Aujourd'hui, des cas à 1M de nœuds peuvent s'exécuter sur ces mêmes ordinateurs.

Pour comparer les temps de calcul avant et après optimisation, nous utilisons un domaine représentant une colonne de sable très étroite : 1 cm de largeur et 2 m de profondeur. Ce cas test est appelé cas test de Vanderborght, il est présenté en annexe page i. Il est important de noter qu'en ce qui concerne la version non optimisée, le mailleur est GID, et dans le cas de la version optimisée le mailleur est Gmsh. Un exemple de résultat pour un maillage d'environ 400 éléments est présentés en Fig.2.1.

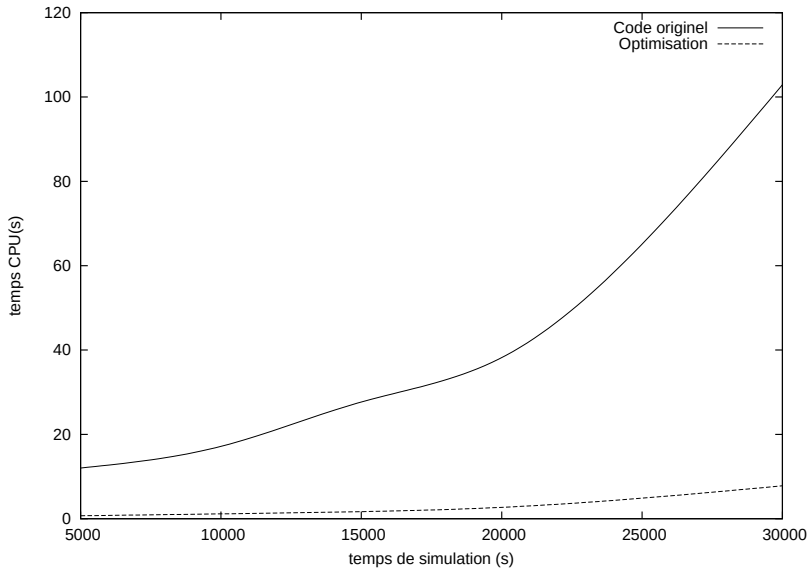


FIG. 2.1 – *Temps CPU en fonction du temps simulé*

On peut voir sur ce graphe qu'il n'y a pas de relation linéaire entre le temps d'exécution du code et le temps simulé, ceci s'explique par la non linéarité du problème. Dans un problème d'infiltration continue comme celui-ci, une partie de plus en plus grande du domaine approche de la saturation, or on a vu que les lois de comportement sont fortement non linéaire lorsqu'on se rapproche de la saturation. Il est important de noter que l'utilisation d'une méthode itérative au lieu d'une méthode directe n'a pas ou très peu d'impact sur la solution obtenue, car la tolérance utilisée pour la méthode itérative est bien plus petite que celle de la méthode de Picard. Globalement, les temps de calcul ont étés divisés par un facteur supérieur à 10 (qui augmente avec la taille des problèmes).

# 3 La parallélisation du code

Malgré les diverses optimisations, la parallélisation est nécessaire pour modéliser des cas réels. Par exemple, avec le code optimisé, pour simuler 2h de pluie sur $1m^2$ discrétisé en 430.000 nœuds, il faut compter 30h de calcul. Avec une parallélisation efficace, on peut envisager d'obtenir les résultats en quelques minutes. On pourra également traiter des problèmes bien plus coûteux que l'on ne pourrait pas traiter avec le code séquentiel.

## 3.1 Principe de la parallélisation

### 3.1.1 Single Process Multiple Data

Le principe de cette technique est que tous les processeurs vont exécuter le même code, mais sur des données différentes. Dans notre cas chaque processeur calculera la solution sur un sous-domaine. Pour cela, il faut procéder à un découpage du maillage. Dans le cas des éléments finis, plusieurs types de découpage sont envisageables. On peut par exemple découper selon une suite d'arêtes du maillage. Dans ce cas, il s'agit d'une décomposition élémentaire : chaque domaine est constitué d'un jeu d'éléments distincts. Ici, nous avons choisi de découper selon les nœuds : chaque processeur fera la résolution sur un jeu de nœuds distinct. Par exemple sur la figure Fig.3.1, le processeur $P_1$ doit connaître toutes les informations (table de connectivité, coordonnées des points, solution aux points) concernant le domaine $\Omega_1$ mais n'effectue la résolution que sur les nœuds symbolisés par des $\bullet$. La zone de recouvrement $\Gamma$ contient donc les éléments qui sont stockés à la fois sur $P_1$ et $P_2$.

### 3.1.2 Dépendances de données

#### 3.1.2.1 Construction de la matrice

Nous notons $\Omega_i$ la partie du maillage sur laquelle le processeur i résout le problème. Le choix d'une décomposition par points plutôt qu'une décomposition par éléments est en réalité imposé par PETSc, qui stocke la matrice du problème à résoudre sous la forme de l'équation 3.1.

$$\begin{pmatrix} D_1 & A_1^2 & \cdots & \cdots & A_1^N \\ A_2^1 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & A_{N-1}^N \\ A_N^1 & \cdots & \cdots & A_N^{N-1} & D_N \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix} \tag{3.1}$$

où $D_i$ représente l'influence des nœuds de $\Omega_i$ sur ses propres nœuds, $A_i^j$ représente l'influence des nœuds de $\Omega_j$ sur ceux de $\Omega_i$, etc.

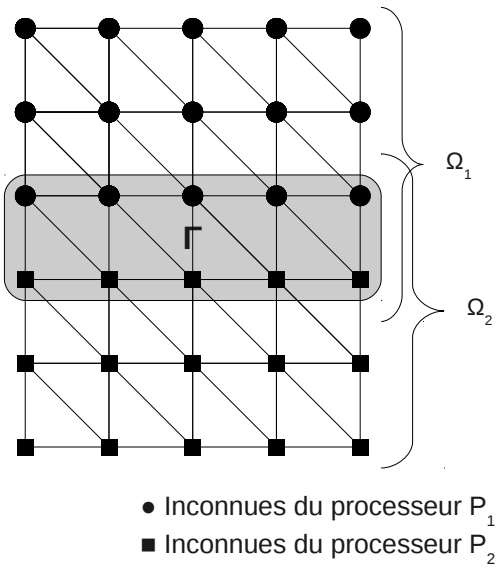


FIG. 3.1 – *Exemple de deux sous-domaines* $\Omega_1$ *et* $\Omega_2$ *et leur zone de recouvrement* $\Gamma$

Les blocs $D_i$ et $A_i^j$ pour $\{i = 1..N, i \neq j\}$ sont stockés sur le processeur i. Bien sûr, les matrices de la forme de l'équation 3.1 s'obtiennent en renumérotant les inconnues, de telle sorte que les inconnues de $\Omega_1$ soient numérotées de 0 à $E_1$, celles de $\Omega_2$ de $E_1 + 1$ à $E_2 + E_1$ etc. Les blocs $A_i^j$ sont non nuls lorsque les sous-domaines $\Omega_i$ et $\Omega_j$ sont frontaliers. Dans ce cas, des points de $\Omega_i$ appartiennent aux mêmes éléments que des points de $\Omega_j$ ce qui explique la dépendance des solutions calculées sur ces points.

Pour calculer le bloc $A_i^j$ et le vecteur $b_i$, le processeur $i$ a donc besoin d'informations calculées sur le domaine $j$ : les informations géométriques (comme les coordonnées des nœuds de la zone de recouvrement) peuvent être données une fois pour toutes au processeur i, mais la solution sur les nœuds de la zone de recouvrement doit être récupérée à chaque itération de la méthode de Picard.

#### 3.1.2.2 Résolution des systèmes linéaires

Les méthodes itératives utilisées pour la résolution des systèmes linéaires sont constituées essentiellement de produits matrice-vecteur que l'on peut décomposer par bloc comme dans l'équation 3.2.

$$\begin{pmatrix} D_1 & A_1^2 & \cdots & \cdots & A_1^N \\ A_2^1 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ A_N^1 & \cdots & \cdots & A_N^{N-1} & D_N \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_N \end{pmatrix} = \begin{pmatrix} D_1 v_1 + \sum_{j=2}^{N} A_1^j v_j \\ D_2 v_2 + \sum_{j=1;j\neq 2}^{N} A_2^j v_j \\ \vdots \\ \vdots \\ D_N v_N + \sum_{j=1}^{N-1} A_N^j v_j \end{pmatrix} \tag{3.2}$$

Ainsi, à chaque itération interne de la méthode itérative, le processeur $j$ doit communiquer le vecteur $v_j$ au processeur $i$ pour que ce dernier puisse calculer $A_i^j \times v_j$. Ces communications doivent être faites uniquement lorsque c'est nécessaire, c'est-à-dire lorsque $A_i^j \neq 0$.

#### 3.1.2.3 Autres dépendances

Chaque processeur a besoin de connaître l'erreur globale commise à chaque itération pour pouvoir tester le critère d'arrêt de la méthode de Picard. Cette erreur est calculée à partir de la solution sur l'ensemble du domaine.

Si une condition limite bascule sur un sous-domaine, et que le pas de temps est invalidé, tous les autres processeurs doivent en être informés pour qu'eux aussi n'incrémentent pas le temps.

### 3.1.3 Implémentation de la parallélisation

#### 3.1.3.1 Découpage du maillage : le code splitmesh

Les algorithmes de découpage de maillage proposés par Gmsh (Chaco et Metis) font un découpage par éléments. Pour obtenir un découpage par nœuds, j'ai implémenté un programme en C++, appelé *splitmesh*, qui effectue le découpage par point des maillages générés par Gmsh en créant une zone de recouvrement d'une maille. Ce programme, utilisé tel un pré-process afin de ne pas dégrader la lisibilité du code Fafemo, gère également les dépendances de données. À partir du fichier sorti par Gmsh, ce pré-process écrit un fichier destiné à chaque processeur. Ces fichiers contiennent les points et la partie de la table de connectivité nécessaire à chaque processeur. Il faut bien comprendre que pour passer d'une décomposition par éléments à une décomposition par nœuds, il suffit d'affecter les nœuds de la frontière à l'un des domaines qui le partage. Voici son pseudo-code :

```
Lecture du maillage
Pour tous les domaines
   Pour les points à la frontière
     Si aucun autre domaine ne gère ce point
       Alors le point est géré ici
       Récupération des éléments contenant ce point
     Fin si
    Fin pour
Fin pour
Renumérotation des points
Ecriture des fichiers
```

#### 3.1.3.2 Modification du code Fafemo

Même si j'ai cherché au maximum à limiter l'impact de la parallélisation dans le code pour des raisons de lisibilité, des modifications ont dues être faites. Il y a deux types de modifications, celles qui sont liées à la numérotation et celles liées aux communications.

Lorsque Fafemo est exécuté, il n'a plus connaissance de la numérotation des nœuds de Gmsh, une renumérotation a été effectuée sur tous les fichiers d'entrée (comme expliqué section 3.1.2.1). Fafemo doit cependant faire le lien entre deux types de numérotations :

- Numérotation globale : Elle numérote tous les nœuds du problème, elle est commune à tous les domaines, le numéro global i correspond à l'inconnue numéro $i$.
- Numérotation locale : Chaque domaine numérote tous les nœuds dont il a connaissance, que la solution à ce nœud soit calculée par le processeur en charge du domaine ou non.

Dans les fichiers d'entrée, la table de connectivité suit la numérotation globale. À la fin de la lecture des fichiers, il faut donc numéroter localement la table de connectivité. Par la suite, tous les calculs internes à chaque processeur seront faits suivant la numérotation locale. Le lien est fait avec la numérotation globale au moment d'insérer les valeurs dans la matrice et le second membre, et au moment de l'écriture des résultats.

Les communications entre les processeurs sont gérées soit par PETSc, soit directement dans le code avec des fonctions MPI :

- Résolution itérative : les méthodes itératives sont essentiellement constituées de produit matrice - vecteur. Ici, PETSc gère entièrement les communications. Il faut tout de même garder à l'esprit qu'un préconditionnement efficace permettra de limiter les communications. L'utilisation d'un préconditionneur de type Schwartz additif permet d'avoir des résultats stables, même si parfois un Jacobi par bloc s'avère plus efficace.
- Echange de la zone de recouvrement : l'échange se fait directement à l'aide de fonctions MPI dans le code, une fois la résolution effectuée. Seuls les messages nécessaires sont échangés à l'aide de communications non blocantes, cependant le recouvrement du temps d'attente par des calculs est limité par la structure du code. On notera toutefois que la durée des échanges ne représente qu'un faible pourcentage du temps total d'exécution du code (voir section 3.2.2.2 page 18 et annexe C page v).
- Calcul de la norme de l'erreur : elle est effectué avec la fonction `MPI_AllReduce` de MPI. Chaque processeur calcule sa contribution à la norme, puis `MPI_AllReduce` est utilisé pour sommer ces contributions et communiquer le résultat à l'ensemble des processeurs.
- Information sur les bascules : avec `MPI_AllReduce`, on effectue une somme sur les processeurs du nombre de solutions incohérentes. Si la solution est cohérente partout le pas de temps est validé, si au moins un processeur a une solution incohérente, tous les processeurs rejettent le pas de temps.

## 3.2 Les tests effectués

### 3.2.1 Test du cluster Migale (INRA)

#### 3.2.1.1 Présentation du Cluster

La plateforme MIGALE est hébergée au sein de l'unité Mathématiques, Informatique et Génome de Jouy-en-Josas. Sa mission est avant tout de proposer une puissance de calcul pour les activités en Biologie et en Génétique. Or les logiciels utilisés dans ces domaines sont en général séquentiels, et ne nécessitent pas de bibliothèques d'algèbre linéaire. L'environnement parallèle est LAM (prédécesseur d'Open MPI), et PETSc a été installé par nos soins. MIGALE

compte 376 cœurs, répartis en 4 catégories de nœuds présentées en Tab. 3.1. Le code Fafemo est naturellement destiné à être utilisé sur ce cluster INRA.

| Type | Processeurs | cœurs par nœud | Fréquence | Mémoire | Cache |
|---|---|---|---|---|---|
| 1 | Intel Quad Core | 16 | 2.77 Ghz | 32 Go | 8Mo partagés |
| 2 | Intel Dual Core | 4 | 2.33 Ghz | 8 Go | 2Mo / cœur |
| 3 | Intel Quad Core | 8 | 2.33 Ghz | 32 Go | 4Mo / 2 cœurs |
| 4 | AMD Quad Core | 16 | 2.2 Ghz | 96 Go | 512k / cœur |

TAB. 3.1 – *Les différents nœuds de MIGALE*

La répartition des tâches sur le cluster est organisée en deux types de queues (une queue courte pour les exécutions durant moins de 4h, et une queue longue). Chaque nœud permet l'accès à tous ses cœurs aux processus de la queue longue, et à la moitié à ceux en courte. Par exemple un nœud à 16 cœurs pourra accueillir simultanément 16 processus d'un ou plusieurs utilisateurs en queue longue et 8 d'autres utilisateurs en queue courte. Il peut donc y avoir jusqu'à 24 processus s'exécutant sur 16 cœurs. Les tests que j'ai effectués sur ce cluster ont eu pour but principal de trouver les meilleures configurations d'utilisation possibles du code Fafemo. Ces tests vont également permettre de donner une première idée des performances du code.

#### 3.2.1.2 Tests

Premièrement pour tester si l'hétérogénéité des nœuds a un impact sur les temps de calcul, j'ai exécuté le code séquentiel sur les deux types de nœuds intéressants pour le parallèle, c'est-à-dire sur les nœuds de type 3 et 4 dans le tableau Tab.3.1. Les nœuds de type 1 n'ont pas été opérationnels durant mon stage. Le cas test utilisé est celui de Belfort 3D à 56334 points (ce cas est présenté en annexe A page ii). Les durées des calculs avec le code séquentiel sont de 4h21 pour les nœuds de type 3, et de 6h16 pour ceux de type 4. Lors des tests parallèles, il sera donc important que le code s'exécute sur des nœuds de même type, sous peine de ne pas être efficace : les processus lancés sur les nœuds de type 3 passeraient énormément de temps à attendre les messages des nœuds de type 4. Le gestionnaire de batch ne nous permet pas de choisir un type de nœud. Nous pouvons cependant demander un nœud spécifique, ce qui limite les essais à 8 ou 16 cœurs. Les speedup présentés dans le tableau Tab.3.2 sont les speedups maximums que l'on peut espérer sur ce cas, dans la mesure où ils ont été obtenus en faisant attention à ce que les nœuds ne soient pas partagés avec d'autres utilisateurs. Pour cela, il a fallu faire de nombreux tests en surveillant l'utilisation du nœud.

| Nombre de cœurs | 2 | 4 | 8 | 16 |
|---|---|---|---|---|
| Speedup | 1.98 | 3.78 | 8.27 | 17.48 |

TAB. 3.2 – *Speedup du cas test de Belfort sous MIGALE*

#### 3.2.1.3 Conclusion

Il s'avère très hasardeux d'exécuter le code sur plusieurs nœuds : lorsqu'on ne spécifie pas que l'on souhaite exécuter le code sur un nœud en particulier, le système de gestion des tâches répartit les processus sur différents nœuds, selon l'occupation de la machine. Le code peut donc s'exécuter sur des nœuds de puissances différentes plus ou moins occupés par d'autres utilisateurs. Bien sûr, plus on demande de processus pour l'exécution du code parallèle et plus la probabilité d'utiliser un nœud surchargé augmente. L'occupation des nœuds par d'autres utilisateurs est d'autant plus problématique que cela implique de devoir partager non seulement le CPU mais aussi une partie de la mémoire cache ce qui dégrade très fortement les performances du code. Dans ces conditions, la meilleure configuration possible sur cette machine est donc d'exécuter le code sur un nœud à 16 cœurs qui n'est pas occupé par des processus venant de la queue *short*. Dans ce cas, ce premier test de performance du code parallèle montre des résultats satisfaisant avec un speedup parfois superlinéaire.

### 3.2.2 Tests de performance

#### 3.2.2.1 Présentation du Cluster JADE

Au cours du stage nous avons effectué une demande d'heures de calcul auprès du CINES. Ainsi 10.000h de calcul nous ont été allouées pour finaliser le développement et les tests du code Fafemo. Le cluster jade est, à cette date, le plus puissant des clusters français (voir Tab.3.3) et le 18ème au niveau mondial.

| | |
|---|---|
| Nombre de cœurs | 23040 |
| Puissance Maximale de calcul | 237.80 Tflops |
| Mémoire physique | 172.800 Go |

TAB. 3.3 – *Principales propriétés du cluster JADE*

Ce cluster comprend donc 23040 cœurs répartis sur 2880 nœuds, eux-mêmes répartis sur 46 armoires (appelées racks). Chaque nœud est constitué de deux Intel Quad-Core (E5472 ou X5560) 3.00Ghz et dispose de 30G de mémoire physique. Les tests sous MIGALE nous ont montré que les performances du code sont intimement liées à la gestion de la mémoire cache ; il est donc bon de savoir que chaque cœur dispose de 64k de cache L1, et que deux cœurs se partagent 6Mo de cache L2. En termes de communication, la bande passante entre les nœuds est de 25.6GB/s. JADE est donc tout à fait adapté au calcul parallèle. Il dispose d'ailleurs de plusieurs implémentations de MPI ; les tests ont été effectués ici avec l'environnement proposé par le constructeur de la machine (SGI) et non celui proposé par INTEL.

#### 3.2.2.2 Tests 1

Le but de ce premier test est de tester la machine, plus particulièrement l'influence du nombre de nœuds utilisés sur les temps de calcul. Pour cela, nous effectuons le test de Belfort 3D sur un maillage à 56334 nœuds, découpé en 16 sous-domaines. Seule la répartition des processus sur les nœuds diffère d'un test à l'autre.

| Noeuds | Cœurs/nœud | Assemblage | Inversion | Echange | Total |
|---|---|---|---|---|---|
| 2 | 8 | 216,43 | 79,34 | 0,53 | 302,27 |
| 4 | 4 | 211,6 | 67,43 | 0,31 | 282,06 |
| 16 | 1 | 211,14 | 68,11 | 0,31 | 282,44 |

TAB. 3.4 – *Temps elapse en secondes de chaque phase du code.3.1.3.2*

J'ai donc effectué 3 tests, en diminuant le nombre de cœurs par nœud utilisés. Les résultats sont présentés en Tab.3.4, Assemblage correspond à la phase de calcul et de stockage de la matrice et du second membre, Inversion à la phase d'inversion du système linéaire avec PETSc, et Echange à la phase d'échange des messages avec MPI décrit section. Dans le premier cas, sur deux nœuds, tous les cœurs sont occupés, c'est-à-dire que les mémoires caches L2 sont bel et bien partagées entre les cœurs, alors que sur 4 et 16 nœuds, le gestionnaire de batch distribue les processus de façon à ce que chaque cœur soit seul à accéder à la mémoire L2. C'est cette utilisation de la cache qui explique les différences que l'on observe en termes de temps d'assemblage entre le premier et les deux suivant. Les petites différences entre le cas à 4 nœuds et celui à 16 montrent que les temps de communications dépendent peu du nombre d'armoires utilisées : en effet, sur les tests à 2 et 4 nœuds, tous les cœurs sont sur la même armoire, et le cas à 16 nœuds a été réparti inéquitablement sur 4 armoires. Ces résultats nous garantissent une bonne stabilité des temps de calcul quelle que soit la répartition sur les nœuds du cluster. Lors des tests suivants, nous ne nous soucierons plus de la répartition des calculs sur la machine. Nous pouvons également constater sur ces résultats que le temps consacré à la phase d'échange est négligeable par rapport au temps total. Cela se vérifiera sur l'ensemble des tests effectués, même pour un très grand nombre de cœur.

Toujours sur ce même test, nous nous intéressons cette fois aux performances en termes de Speedup Tab.3.5. La super-linéarité du speedup s'explique ici aussi par la multiplication

| Cœurs | 8 | 16 | 32 |
|---|---|---|---|
| Speedup | 9,32 | 22,43 | 43,52 |

TAB. 3.5 – *Speedup pour le cas de Belfort à 56334 nœuds*

de la mémoire cache. Les speedups sont ici bien meilleurs que ceux obtenus sur le cluster MIGALE, ce qui s'explique par une mémoire cache disponible plus importante, mais aussi par des communications plus rapides : Petsc est compilé avec l'environnement parallèle fourni par le constructeur de la machine, et nous avons ici la garantie de bénéficier de la totalité de la bande passante du nœud.

#### 3.2.2.3 Tests 2

Il s'agit ici d'un test de speedup sur un cas d'infiltration continue durant 0.1 jour sur un volume d'environ $1m^3$ de sable ou de limon. Ce test est comparable à celui effectué pour le sable dans la publication de Herbst et al.[9]. Avant de nous intéresser aux performances de

la parallélisation, regardons le comportement numérique de ces deux simulations dans le cas séquentiel. En séquentiel, le temps d'exécution est de 14h53 pour le sable et 30h02 pour le limon, pourtant le nombre de résolutions de systèmes linéaires est plus grand dans le cas du sable comme le montre le graphe Fig.3.2.

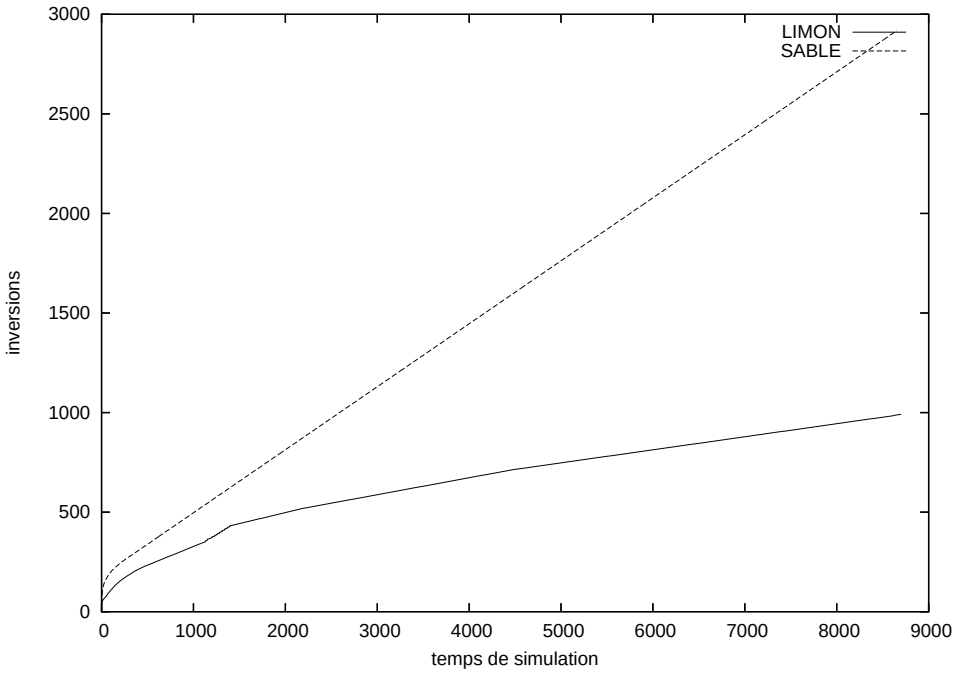


FIG. 3.2 – *Nombre d'inversions en fonction du temps simulé*

L'eau s'infiltrant plus rapidement dans le cas du sable que dans celui du limon, le pas de temps est plus grand dans le cas du limon. Or, un pas de temps plus grand et des lois de comportement plus raides font que les matrices associées sont moins bien conditionnées et donc les systèmes linéaires sont plus difficiles à inverser. C'est pourquoi le temps de calcul est plus important pour le limon, alors que beaucoup moins de systèmes ont été inversés. En réalité, le système de pas de temps adaptatif, basé uniquement sur le nombre d'itérations de la méthode de Picard, n'est pas très performant dans le cas du sable. En effet, l'erreur relative observée à la fin de la linéarisation est de l'ordre de $1e^{-5}$ alors que la tolérance est de $1e^{-4}$. Un système de calcul du pas de temps basé sur l'erreur serait plus adapté ici : on pourrait imaginer que le pas temps augmente lorsque l'erreur est faible et qu'elle est obtenue avec peu d'itérations internes de la méthode de résolution, et diminue lorsque l'erreur se rapproche de la tolérance.

Les performances du code parallèle pour une simulation de 0.1 jours de pluie sont présentés en Tab.3.6. Le speedup est mesuré relativement au cas ayant tourné sur 1 cœur, ce qui est pénalisant. En effet, lorsque le programme s'exécute sur un seul coeur, ce coeur dispose de la totalité du cache L2. Dans les autres cas, deux processus doivent se partager le cache L2. Faire des tests de speedup revient donc à comparer le cas à 1 processus disposant de 6Mo de

| nombre de cœurs : | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| speedup sable : | 6 | 13.91 | 35.94 | 92.22 | 184.59 |
| speedup limon : | 11.83 | 21.41 | 62.3 | 305.86 | 890.36 |

TAB. 3.6 – *Speedup en fonction du nombre de cœurs*

cache L2, à des cas où les processus disposent de 3Mo. Cela se voit en particulier lorsque peu de cœurs sont utilisés, mais globalement les speedups obtenus sont excellents.

On constate qu'il y a de grandes différences de speedup selon le matériau, si bien que sur 16 nœuds (128 cœurs) il faut 120s pour effectuer les calculs dans le cas du limon, et 285s dans le cas du sable. Le speedup est bien meilleur dans le cas du limon. Cela s'explique en grande partie par l'efficacité de l'assemblage. Le temps d'assemblage de la matrice, constitué essentiellement d'appels aux lois de comportement, représente entre 70 et 90% du temps d'exécution du code.

| nombre de cœurs : | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| speedup sable : | 6.14 | 14.17 | 37.35 | 100.79 | 209.29 |
| speedup limon : | 12.78 | 22.52 | 67.41 | 389.03 | 1207.68 |

TAB. 3.7 – *Speedup de la phase d'assemblage en fonction du nombre de cœurs*

Les speedups super linéaires de la phase d'assemblage, présentés en Fig.3.7 s'expliquent facilement : lorsqu'un sous-domaine est dans un état homogène, les lois de comportement sont calculées en un point uniquement (on sauvegarde les derniers appels aux lois de comportement, si les lois sont appelées deux fois consécutivement avec les mêmes paramètres, le calcul n'est pas refait). On comprend que plus le domaine est partitionné, plus il y aura de sous-domaines dont l'état est homogène. L'état homogène correspond en fait à la condition initiale, c'est-à-dire que les appels aux lois de comportement sont économisés lorsque le front d'infiltration n'a pas encore atteint le sous-domaine. L'eau s'infiltre plus rapidement dans le sable que dans le limon, c'est pourquoi le speedup de la phase d'assemblage est largement meilleur dans le cas du limon. Les speedups des phases de résolution (préconditionnement et inversion) sont quant à eux quasi linéaires.

# 4 Le projet Agrobat

## 4.1 Présentation du projet

### 4.1.1 Présentation générale

Le but de ce projet financé par l'ANR est de mieux comprendre le comportement global des toitures terrasses végétalisées et leur impact sur la thermique des bâtiments. L'objectif de ce projet et de fournir un modèle validé expérimentalement qui puisse notamment prendre en compte le caractère vivant des végétaux et les transferts de chaleur et d'humidité à l'echelle de la toiture et du bâtiment.

### 4.1.2 Travaux effectués par l'UMR EMMAH

Les travaux devront permettre une caractérisation des paramètres hydrodynamiques du substrat [10], un mélange de compost et de graviers de pouzzolane, par l'imagerie du fonctionnement hydrique. Ces travaux comprennent un aspect expérimental et un aspect modélisation numérique. Dans le cadre de mon stage, j'ai créé la version numérique d'une expérience en laboratoire qui consiste à laisser s'assécher un échantillon de substrat hétérogène au contact de l'atmosphère. En laboratoire des sondes permettent de connaître le potentiel matriciel, et l'évolution de la masse de l'échantillon nous donne le flux d'évaporation.

## 4.2 Modélisation du problème

### 4.2.1 Données expérimentales

#### 4.2.1.1 Propriétés des matériaux

Les expérimentations ont permis d'obtenir les propriétés des matériaux présentées dans le tableau Tab.4.1.

| Grandeur | Compost | Pouzzolane |
|---|---|---|
| $\theta_r$ | 0.334 | 0 |
| $\theta_s$ | 0.660 | 0.369 |
| $n$ | 1.8037 | 1.2444 |
| $K_s$ | $3.041e^{-3}$ | $9.34e^{-4}$ |
| $h_e$ | $-0.0358$ | $-0.0032$ |

TAB. 4.1 – *Propriétés du compost et de la pouzzolane*

Les matériaux ont des comportements différents, comme le montrent les graphiques Fig.1.1 p.3 et Fig.1.3 p.5. La pouzzolane contient moins d'eau et conduit moins l'eau que le compost.

La pouzzolane a donc une inertie plus forte que le compost : l'eau s'infiltre et s'évapore plus lentement dans la pouzzolane.

#### 4.2.1.2 Conditions limites

Le récipient est étanche, ce qui correspond à une condition de flux nul. L'évaporation est créée par un flux sortant imposé sur la surface de l'échantillon. Les valeurs de ce flux sont calculées à partir des expérimentations en laboratoire en suivant l'évolution de la masse de l'échantillon. Le graphique Fig.4.1 présente les valeurs expérimentales et les différentes fonctions de flux utilisées dans les simulations. Ces deux fonctions sont obtenues par l'interpolation linéaire d'une régression non paramètrique, les fonctions utilisées différent donc seulement avant 300.000 secondes, pourtant on verra que cette différence a un fort impact sur la solution. Le choix de la fonction flux est très important, une simple régression linéaire ne permet d'avoir des résultats proches de ceux obtenus expérimentalement : la régression linéaire surestime la fonction de flux lors des premières 500.000 secondes, et le milieu s'assèche donc très rapidement. Lors des phases d'infiltrations, une condition de flux peut basculer en

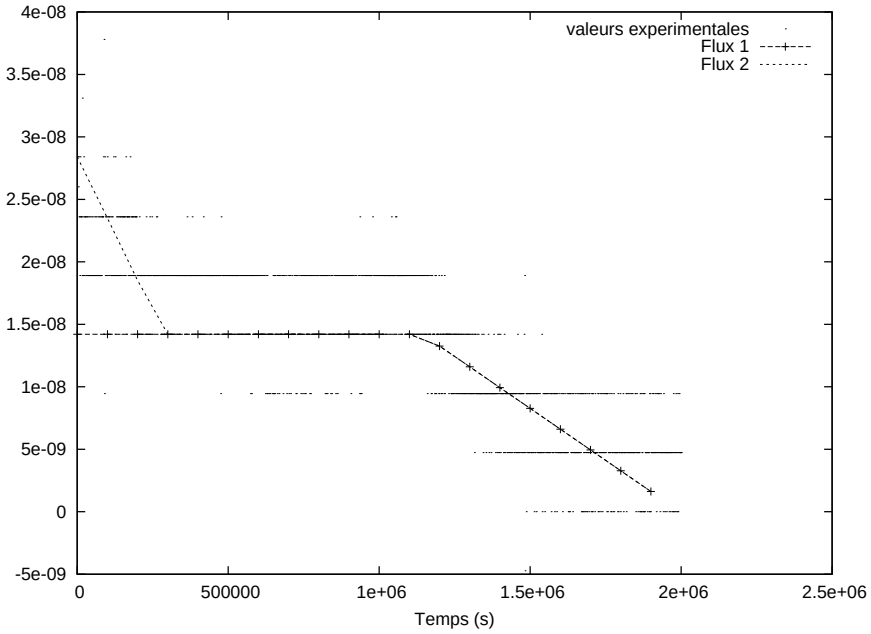


FIG. 4.1 – *Les fonctions de flux utilisées*

condition de Dirichlet $h = 0$ lorsque le milieu est saturé. Dans le cas de l'évaporation il faut se fixer une valeurs minimale de la pression $h_{min}$ telle que si un point est soumis a une condition de flux, et que $h < h_{min}$ alors la condition de flux est changée en condition de Dirichlet $h = h_{min}$. Sans ces bascules, nous aurions $h \to -\infty$ en certains points de la surface, en plus d'être incohérent cela empêche de terminer les simulations, car cela entraine $\Delta t \to 0$. Typiquement on fixe $h_{min} = -10$ ou $h_{min} = -1000$, en gardant à l'esprit que plus la valeur

de $h$ sera négative, plus le pas de temps sera réduit, et donc plus les temps de calcul seront longs.

### 4.2.2 Création du maillage

#### 4.2.2.1 Répartition des inclusions

Dans le cadre d'une première approche on modélise les inclusions de pouzzolane par des sphères (de deux tailles différentes) pour s'affranchir des problèmes liés à la création et au rangement de formes complexes. Lors des expériences, la pouzzolane et le compost sont mélangés puis compressés dans le récipient cylindrique. Pour la modélisation, on a choisi de répartir les inclusions de façon aléatoire. Cette répartition se fait selon un algorithme de rejet puisqu'il ne doit pas y avoir d'intersection entre les inclusions. Deux algorithmes ont étés implémentées : dans la première version on tire aléatoirement la position des petites et des grosses billes de pouzzolane, dans la seconde on tire aléatoirement la position des grosses billes puis on tire des positions sur le fond du cylindre et on incrémente la hauteur jusqu'à trouver une position correcte pour les petites billes. Le premier algorithme permet d'avoir une répartition aléatoire lorsque les inclusions occupent moins de 30% du volume total, le second permet d'inclure une plus grande proportion de pouzzolane. Les rayons des billes de pouzzolane ne sont pas tirées aléatoirement car l'algorithme de rejet privilégierait les petites inclusions en rejetant plus fréquemment les grosses inclusions.

Nous sommes cependant confronté à un problème qui ne nous permet pas d'inclure de fortes proportions de pouzzolane. Pour que le maillage se fasse correctement, il faut que les inclusions ne s'intersectent pas, et en plus il faut laisser suffisamment d'espace entre les inclusions pour que le mailleur Gmsh puisse y insérer des mailles non-dégénérées, sans quoi des erreurs se produiront lors de la création du maillage ou de l'exécution du code Fafemo.

Le graphique présenté en Fig.4.2 montre la proportion maximale de pouzzolane possible en fonction de la taille des mailles. Les inclusions sont constituées d'$\frac{1}{3}$ de sphères de 5mm de rayon et de $\frac{2}{3}$ de sphères de 2mm de rayon. Un exemple d'interprétation du graphique est que lorsqu'on inclu 54% de pouzzolane et que l'on maille avec 1mm, le volume occupé par la pouzzolane ajouté au volume de sécurité d'une maille autour des sphères de pouzzolane représente 100% du volume du cylindre. Ainsi, même avec des mailles d'1mm on ne peut pas inclure 60% de pouzzolane, la proportion utilisée lors de l'expérimentation.

#### 4.2.2.2 Maillage avec Gmsh

Un domaine aussi complexe est très difficile à mailler correctement. On atteint les limites de ce que peuvent faire les algorithmes de maillage traditionnels. Même si l'on respecte la contrainte sur la taille des mailles il arrive que Gmsh soit incapable de générer le maillage. Même lorsque c'est le cas il peut y avoir des erreurs dans la numérotation locale des éléments, ce qui se traduit par des matrices jacobiennes singulières. Pour détecter ces problèmes avant d'exécuter le code $Fafemo$, j'ai implémenté un code $Fafemo\_test$ qui reprend uniquement le calcul des matrices jacobiennes du code $Fafemo$, ce qui permet de détecter rapidement

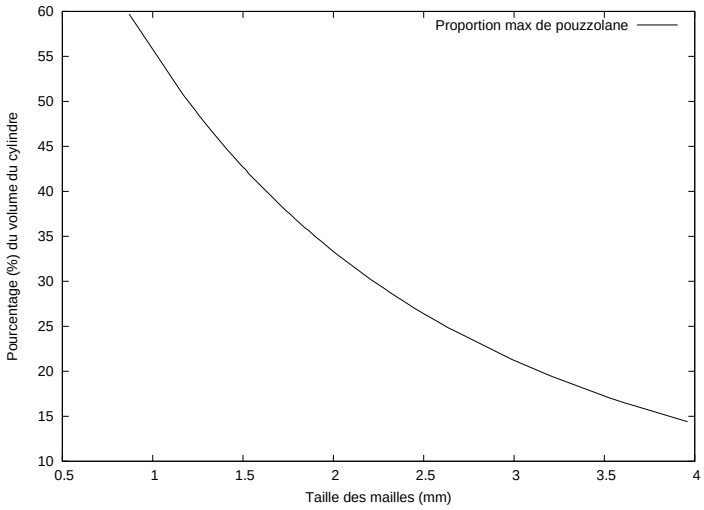


FIG. 4.2 – *Proportion maximale de pouzzolane en fonction de la taille des mailles*

si le maillage sera accepté par $Fafemo$. Pour remédier à ces problèmes on peut changer d'algorithme de maillage ou diminuer un peu le pas d'espace.

## 4.3 Résultats

### 4.3.1 Résultats physiques

#### 4.3.1.1 Evolution du potentiel $h$

On peut identifier trois phases dans l'évolution du potentiel matriciel $h$, qui correspondrons également à des problématiques différentes d'un point de vue numérique :

- Phase 1 : le potentiel matriciel diminue lentement (jusqu'à 900.000 secondes sur le graphe Fig.4.3).
- Phase 2 : on observe une diminution brutale, les bascules des conditions limites à la surface s'effectuent dans cette phase (de 900.000 à 1.000.000 secondes sur le graphe Fig.4.3).
- Phase 3 : après une inflection, la diminution du potentiel devient de plus en plus lente (après 1.000.000 secondes sur le graphe sur le graphe Fig.4.3).

La valeur du potentiel en un point et la durée des phases 1 et 2 dépendent de la condition de flux imposée et de la proportion de pouzzolane. Les deux graphiques 4.7 et 4.4 montrent l'évolution du potentiel matriciel en un point situé à 1cm sous la surface.

Le graphique Fig.4.7 montre que l'évolution du potentiel est très sensible à la condition de flux. Pour le flux 1 la phase 1 dure environ 800.000 secondes, la phase 2 dure environ 100.000 secondes, ici la phase 3 débute à peine. Sur le graphique présenté en Fig.4.4 on peut voir que plus la proportion de pouzzolane est importante, plus la phase 1 dure longtemps.

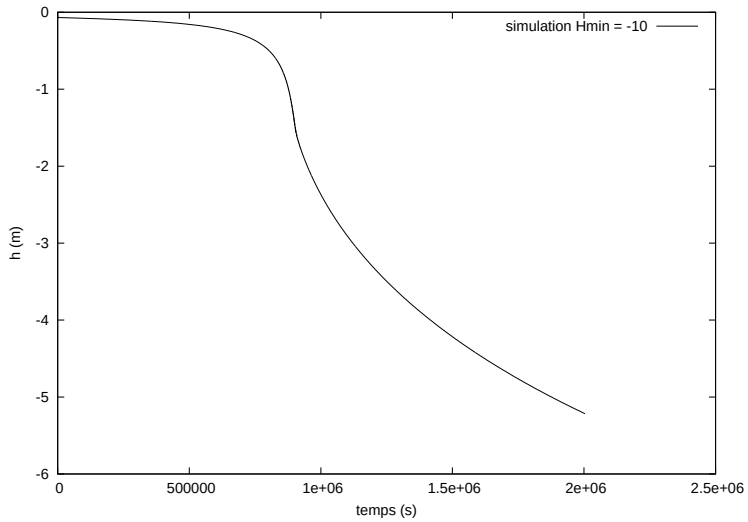


FIG. 4.3 – *Les trois phases de l'évolution du potentiel matriciel*

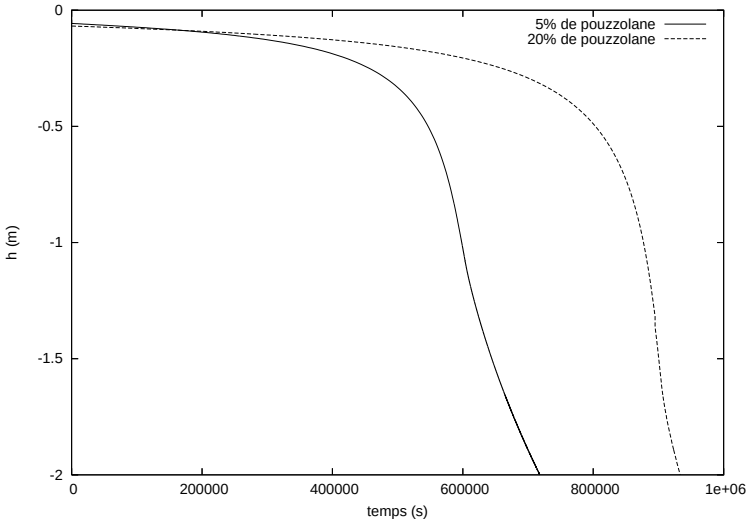


FIG. 4.4 – *Potentiel matriciel en fonction du temps pour différentes proportions de pouzzolane*

#### 4.3.1.2 Variations spatiales du potentiel $h$

Lors de la phase 1, le potentiel matriciel est relativement homogène quelque soit la position et le matériau. Les variations apparaîssent lors de la phase 2. Le graphique Fig.4.5 montre les variations du potentiel $h$ à la surface à 900.000 secondes pour un flux de type 1 avec 20% de pouzzolane. 900.000 secondes correspond ici au début de la phase 2, peu avant les premières bascules des conditions de flux en conditions de Dirichlet.

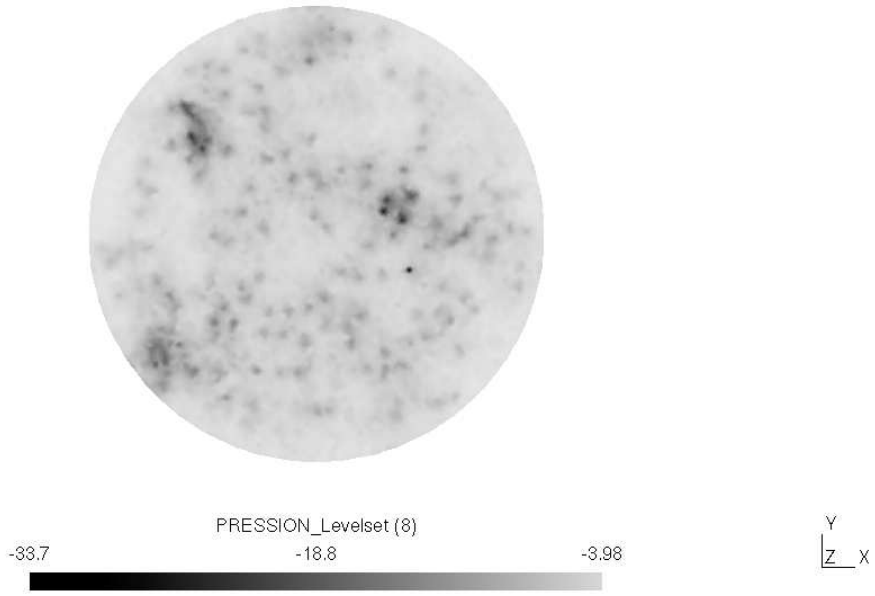


FIG. 4.5 – *Potentiel matriciel sur la surface à 900.000 s*

### 4.3.2 Comportement numérique

#### 4.3.2.1 Inversion des systèmes linéaires

Tous les éléments sont réunis pour avoir des systèmes mal conditionnés : Pas de temps important, forte hétérogénéité des matériaux et du maillage. Dans ce contexte on peut être soumis à plusieurs problèmes lors de la résolution des systèmes linéaires, ainsi la méthode itérative peut ne pas converger ou s'arrêter sur un *breakdown*. En général un preconditionneur de type Jaccobi par bloc et la méthode BiCGSTAB offrent les meilleurs résultats en termes de temps de calcul. Cependant il arrive que ces méthodes divergent, surtout à partir de 25% du volume occupé par la pouzzolane. Dans ce cas on peut préconditionner le système avec Schwarz additif, et le résoudre avec un GMRES bien paramétré : la dimension maximale des espaces de Krylov doit être importante et il faut privilégier les algorithmes d'orthogonalisation stables.

#### 4.3.2.2 Evolution du pas de temps

Alors qu'en phase 1 les pas de temps peuvent être de l'ordre de l'heure, lors de la phase 2 ils sont de l'ordre de la seconde. Le système de pas adaptatif tel qu'il était implémenté n'était pas adapté à des changements aussi brutaux, ainsi le pas de temps reste important jusqu'à ce que la méthode de Picard diverge ce qui dégrade la qualité de la solution (on peut observer des discontinuités des valeurs de $h$). Ce système peut produire des pas de temps qui tendent vers 0, la simulation ne peut donc pas se terminer. Pour pallier à ce problème j'ai mis en place un système de pas adaptatif (equation 4.1) basé sur l'erreur $e^n$ à l'itération $n$ et sur la tolérance $\epsilon$ :

$$\Delta_t^{n+1} = \Delta_t^n \times min\left(s \times \sqrt{\frac{\epsilon}{e^n}}, 4.0\right) \tag{4.1}$$

Avec $s$ un paramètre pris entre 0 et 1 (0.8 ou 0.9 ici). Ce système de pas adaptatif est inspiré de la publication de Kavetski [5] dans laquelle le pas de temps est calculé en utilisant le rapport entre l'erreur commise et l'erreur de troncature du schéma numérique. Le calcul de cette erreur de troncature demandant des modifications du code que je n'ai pas eu le temps d'implémenter, j'ai utilisé le rapport entre la tolérance et l'erreur commise.

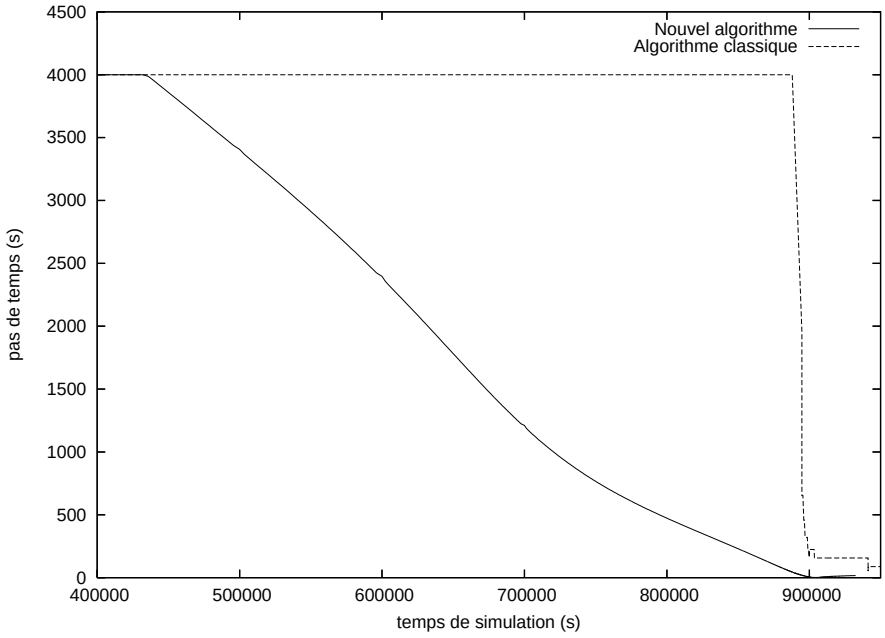


FIG. 4.6 – *Evolution du pas de temps en fonction du temps*

Ce système de pas adaptatif a l'avantage d'éviter les divergences de la méthode de Picard. En pratique la convergence est atteinte au bout de 2 ou 3 itérations. Le graphique 4.6 montre que l'évolution du pas de temps est beaucoup moins raide avec ce nouveau système. Cet algorithme semble moins performant en termes de temps de calcul lors de la phase 2 (le pas de temps atteint un minimum de 1.83 s), mais il permet de garantir la qualité de la solution.

Le graphique Fig.4.6 a été obtenu avec un cas test à 20% de pouzzolane maillé avec 250.000 nœuds environ. Les calculs ont été effectués sur JADE sur 32 cœurs, Il a fallu seulement quelques minutes pour simuler la phase 1, et plusieurs heures pour simuler la phase 2.

## 4.4 Conclusion et perspectives

Ces tests sont une première étape vers la modélisation des transferts d'eau dans les milieux très hétérogènes qui a permis de mettre en avant les problématiques et limitations de ces simulations. Pour la première fois, le choix du preconditioneur et de la méthode de résolution est capital pour le bon déroulement, et la rapidité des calculs. La sensibilité des résultats à

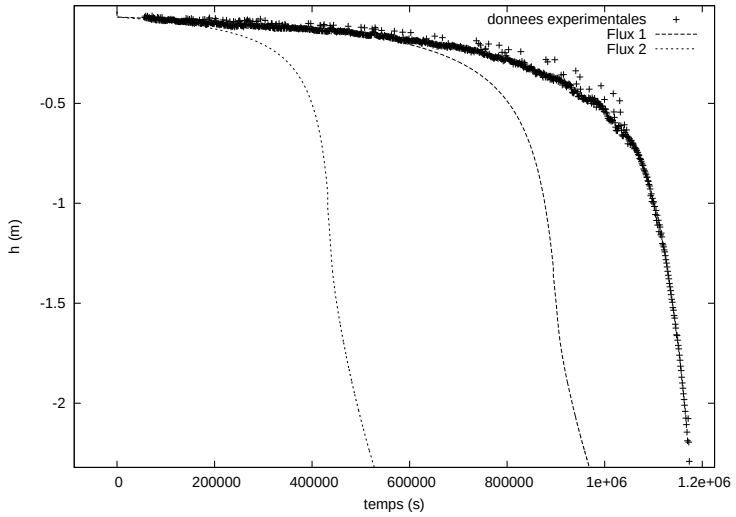


FIG. 4.7 – *Potentiel matriciel en fonction du temps pour les flux de type 1 et 2 et pour les valeurs mesurées*

la fonction de flux est également un point important, par exemple le graphique présenté en Fig.4.7 montre les résultats expérimentaux obtenus avec 60% de pouzzolane, et les résultats numériques obtenus avec 20% de pouzzolane pour les deux type de fonctions de flux utilisées. Dans ce contexte l'étude de l'impact réel de la proportion de pouzzolane est difficile puisque c'est bien la fonction de flux que nous avons du mal à estimer qui est le paramètre le plus important permettant de se rapprocher des valeurs expérimentales. Nous pouvons également nous poser la question des bascules de conditions limites, de leur sens physique, et de la valeur à donner à $h_{min}$. Les résultats expérimentaux ne nous fournissent pas encore d'informations sur la phases 3 qui est observée sur les simulations. Finalement, au jour d'aujourd'hui nous avons réussi à simuler l'évaporation sur un substrat composé au maximum de 30% de pouzzolane, il reste donc encore beaucoup à faire pour obtenir un modèle numérique fidèle aux expérimentations.

# Conclusion

Le but de ce stage était d'obtenir un code parallèle à la fois fonctionnel et efficace pour l'utilisateur, en essayant de faire attention à la clarté des modifications faites sur le code (et sur la documentation associée), et compréhensible pour des développeurs qui ont d'autres problématiques que la parallélisation. De nombreuses modifications ont été effectuées sur le code original pour le rendre plus efficace en temps de calcul, moins consommateur de mémoire et plus ergonomique, puis pour le paralléliser. Au final une seule version du code permet maintenant de traiter des cas 2D ou 3D, en séquentiel ou en parallèle. Les performances obtenues sont très bonnes, tant en termes de réduction du coût de calcul et d'augmentation de la taille des problèmes que l'on peut simuler et permettent d'envisager de nouveaux cas d'application plus ambitieux. Le travail effectué dans le cadre du projet Agrobat va en ce sens, j'ai résolu les principaux problèmes rencontrés, ce qui montre la faisabilité de la simulation de ce cas d'étude et constitue une première phase importante de ce projet. D'un point de vue plus personnel j'ai eu la chance de pouvoir faire des tests sur des clusters très performants.

Il reste toutefois de nombreuses pistes à exploiter pour améliorer le code. De meilleurs algorithmes de gestion du pas de temps, peuvent être implémentés, par exemple en tenant compte de l'erreur du schéma numérique et du coût de calcul de chaque itération. A moyen terme on peut également penser à rassembler dans un même code les différents problèmes physiques que peut traiter Fafemo afin que la simulation de ces problèmes puissent bénéficier des améliorations que j'ai pu apporter au code.

# A Présentation des cas tests

## Cas test de Vanderborght

Il s'agit d'un cas test assez classique dans la littérature : c'est une colonne de sable sec, tout à fait homogène, soumise à une pluie continue. Ce cas est traité en 1D dans la publication de Vanderborght [3]. Une façon assez classique de visualiser les résultats est donc de présenter la teneur en eau en fonction de la profondeur (Fig.A.1).

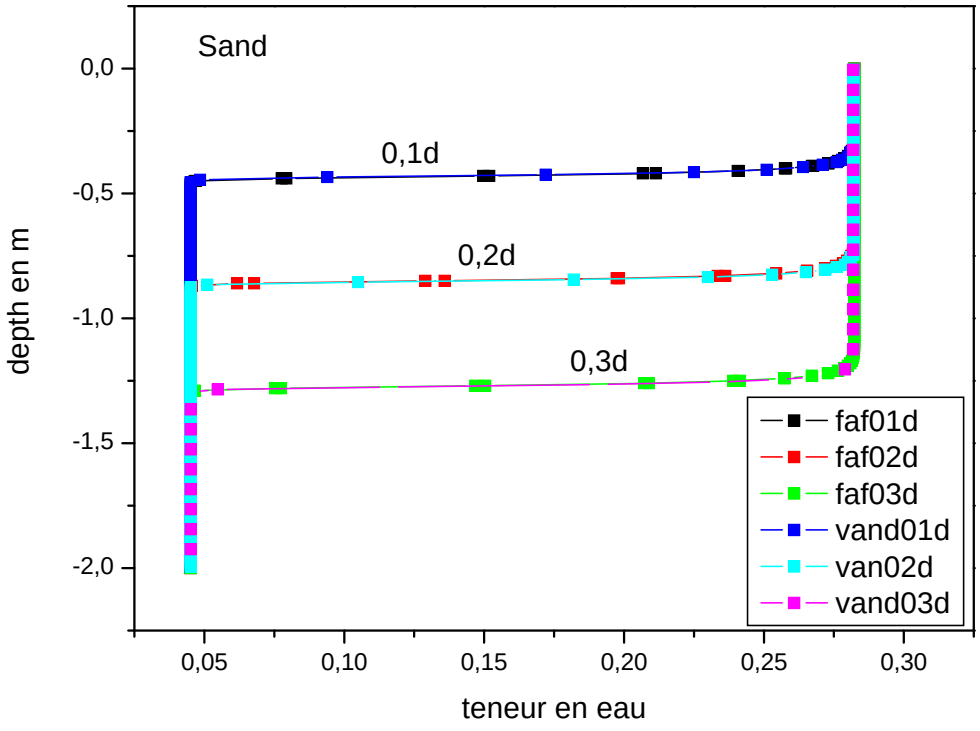


FIG. A.1 – *Teneur en eau à 0.1 jour, 0.2 jour et 0.3 jour pour les différents codes qui existent*

# Cas test de Belfort

Il s'agit d'un cas test pris de la thèse de B.Belfort [4] d'une infiltration décentrée sur un domaine stratifié. En deux dimensions, le domaine est de 1.3m de large pour 1.25m de profondeur, l'infiltration se produit sur 17cm sur le côté gauche de la surface, un exemple de résultats est donné en figure Fig.A.2.

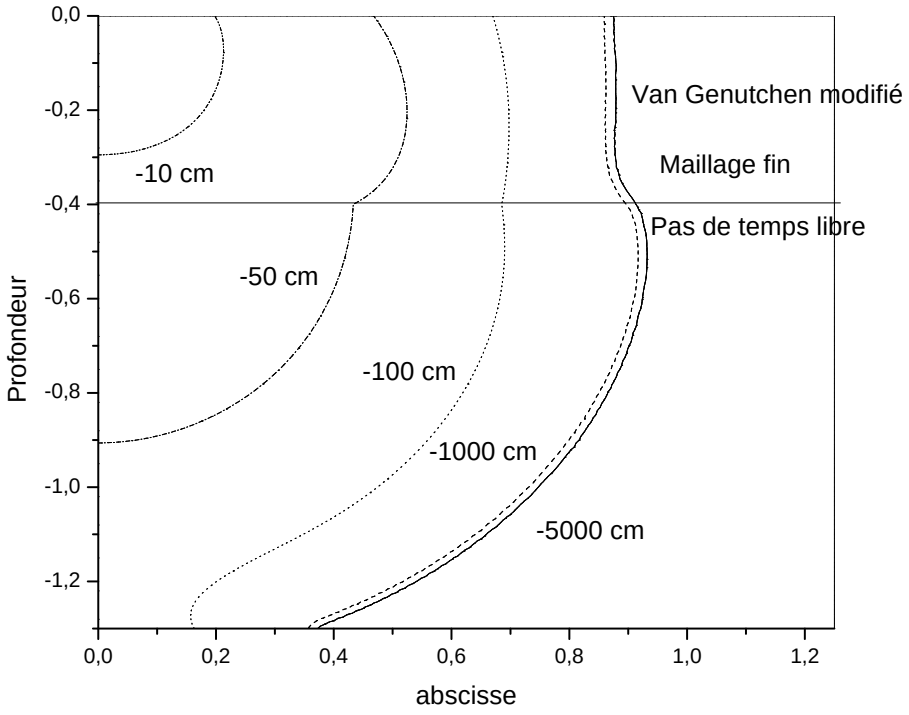


FIG. A.2 – *Isocontours de pressions après 3 jours de simulations*

# B Scripts et programmes annexes developpés

## Agrobat : inclusions aléatoires

Pour le projet Agrobat, j'ai créé plusieurs scripts pour générer les inclusions $d2.m$, $distri.m$ etc. L'appel de ces scripts écrit les coordonnées sur les inclusions dans un fichier nommé *coord*, le script shell *generation_geo* fabrique le fichier *input.geo* à partir des fichiers $partie1$, $partie2$ et *coord*.

```
laurent@ubuntu:~/agrobat>ls
distri.m  generation_geo.sh  partie1  partie2
laurent@ubuntu:~/agrobat>octave distri.m
...
laurent@ubuntu:~/agrobat>ls
coord  distri.m  generation_geo.sh  partie1  partie2
laurent@ubuntu:~/agrobat>./generation_geo.sh
laurent@ubuntu:~/agrobat>ls
coord  distri.m  generation_geo.sh  input.geo  partie1  partie2
laurent@ubuntu:~/Bureau/GENER_ALEAT/ESSAI$
```

## Découpage du maillage

Le découpage des maillages se fait par le programme *splitmesh*, celui-ci s'exécute par l'intermédiaire du script *decoupage.sh*. Par exemple, pour découper un maillage 3D en deux sous-domaines, il faut procéder comme suit :

```
 laurent@ubuntu:~/SPLIT> ls
decoupage.sh  input.geo  splitmesh.cc
laurent@ubuntu:~/SPLIT> gmsh input.geo -3 -part 2
...
laurent@ubuntu:~/SPLIT> ls
decoupage.sh  input.geo  input.msh  splitmesh.cc
laurent@ubuntu:~/SPLIT> ./decoupage.sh
Erreur : Il manque un parametre
Erreur : utilisation : ./decoupage NBDOMAINE DIMENSION
laurent@ubuntu:~/SPLIT> ./decoupage.sh 2 3
 Decoupage en 2 sous-domaines, probleme 3D
Compilation du code splitmesh
Execution du code, cela peut prendre du temps
```

```
fin
laurent@ubuntu:~/Bureau/DV/necessaire$ ll -h
total 14M
-rwxr-x--x 1 laurent laurent 1,3K 2010-07-30 16:17 decoupage.sh
-rw-r--r-- 1 laurent laurent  12K 2010-07-30 16:18 info_dd.txt
-rw-r--r-- 1 laurent laurent 1,2M 2010-07-30 16:16 info_decoupage.txt
-rw-r--r-- 1 laurent laurent 3,2M 2010-07-30 16:16 input0.msh
-rw-r--r-- 1 laurent laurent 3,2M 2010-07-30 16:16 input1.msh
-rw-r--r-- 1 laurent laurent  12K 2010-07-30 16:07 input.geo
-rw-r--r-- 1 laurent laurent 6,0M 2010-07-30 16:13 input.msh
-rw-r--r-- 1 laurent laurent 227K 2010-07-30 16:16 numerotation.txt
-rwxr-xr-x 1 laurent laurent  52K 2010-07-30 16:16 splitin2
-rw-r--r-- 1 laurent laurent  15K 2010-07-29 19:33 splitmesh.cc
```

Plusieurs fichiers sont créés :

- *input*0.*msh* et *input*1.*msh* : les deux parties du maillage
- *numerotation.txt* : Fait le lien entre les anciens et les nouveaux numéros de nœuds (*splitmesh* renumérote les inconnues)
- *splitin*2 : l'exécutable qui effectue le découpage demandé, peut être réutilisé.
- $info\_decoupage.txt$ contient la sortie standard de *splitmesh*.
- $info\_dd.txt$ : Contient le premier et le dernier indice des nœuds de chaque domaine

Les fichiers *numerotation.txt* et $info\_dd.txt$ sont nécessaires au script *split_inputnum* qui effectue le découpage des fichiers *inputnum.dim*, en particulier il traite et disperse les informations concernant les nœuds à visualiser. *split_inputnum* a bien sûr besoin d'un fichier *inputnum.dim* dans lequel les nœuds à visualiser sont donnés dans la numérotation de Gmsh.

# Visualisation des résultats

Le script shell *post_process.sh* génère un fichier *post.msh* lisible par gmsh contenant les informations sur le maillage et sur la solution. (Nécessite la présence des fichiers *input.msh* et des solutions sous la forme *outk.dat*)

# C Résultats détaillés sous JADE

Lors des tests, nous regardons précisément le comportement des trois phases les plus importantes : la phase d'assemblage qui correspond à la création de la matrice et du second membre, la phase de résolution qui correspond au préconditionnement et à la méthode itérative, et la phase d'échange qui correspond à l'échange des frontières. Les résultats présentés ici correspondent au cas test de Vanderborght à 430.000 points.

Les tableaux Tab.C.1 et Tab.C.2 présentent les résultats bruts des durées de calcul. On remarque une différence de comportement selon les matériaux : bien que la durée du calcul sur un seul cœur soit plus longue pour le limon, sur 128 cœurs il s'agit du calcul avec le limon qui est le plus court.

| Sable | | | | |
|---|---|---|---|---|
| Cœurs | Assemblage | Résolution | Echange | Autre |
| 1 | 44567,5 | 9034 | 7,3 | 128.5 |
| 8 | 7254,28 | 1629,39 | 3,21 | 54.56 |
| 16 | 3145,31 | 678,39 | 2 | 28.4 |
| 32 | 1193,39 | 284,2 | 1,25 | 12.71 |
| 64 | 442,18 | 131,56 | 0,89 | 6.71 |
| 128 | 212,95 | 71,66 | 0,68 | 5.13 |

TAB. C.1 – *Temps moyen d'exécution en seconde pour le sable*

| Limon | | | | |
|---|---|---|---|---|
| Cœurs | Assemblage | Résolution | Echange | Reste |
| 1 | 103749 | 4957 | 2,48 | 48.52 |
| 8 | 8115,39 | 1052,51 | 1,2 | 22.01 |
| 16 | 4607,7 | 457,87 | 0,8 | 11.35 |
| 32 | 1539,06 | 199,99 | 0,51 | 5.27 |
| 64 | 266,68 | 85,8 | 0,3 | 2.64 |
| 128 | 85,91 | 34,1 | 0,24 | 1.85 |

TAB. C.2 – *Temps moyen d'exécution en seconde pour le limon*

| | Sable | | Limon | |
|---|---|---|---|---|
| Cœurs | Assemblage | Résolution | Assemblage | Résolution |
| 8 | 6,14 | 5,54 | 12,78 | 4,71 |
| 16 | 14,17 | 13,32 | 22,52 | 10,83 |
| 32 | 37,35 | 31,79 | 67,41 | 24,79 |
| 64 | 100,79 | 68,67 | 389,03 | 57,78 |
| 128 | 209,29 | 126,06 | 1207,68 | 145,37 |

TAB. C.3 – *Speedup de chaque phase*

# D Diagrammes des classes de Fafemo

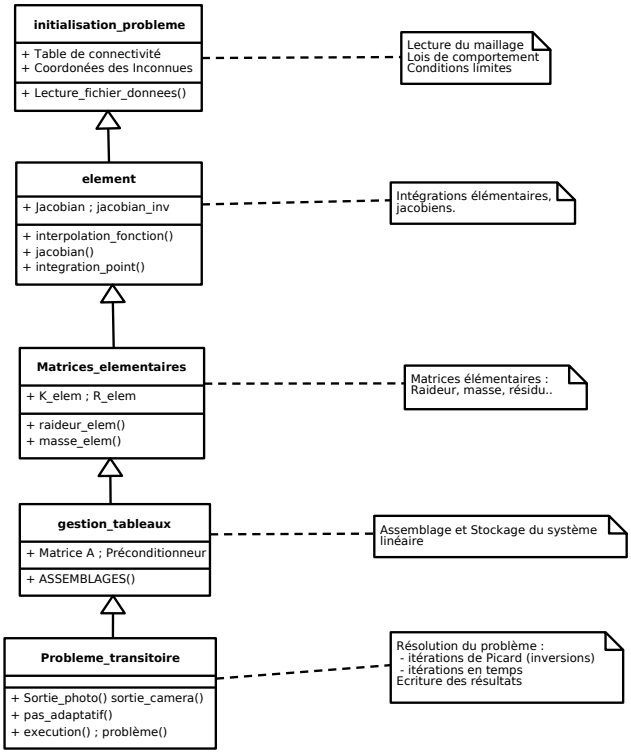


FIG. D.1 – *Diagramme des classes*

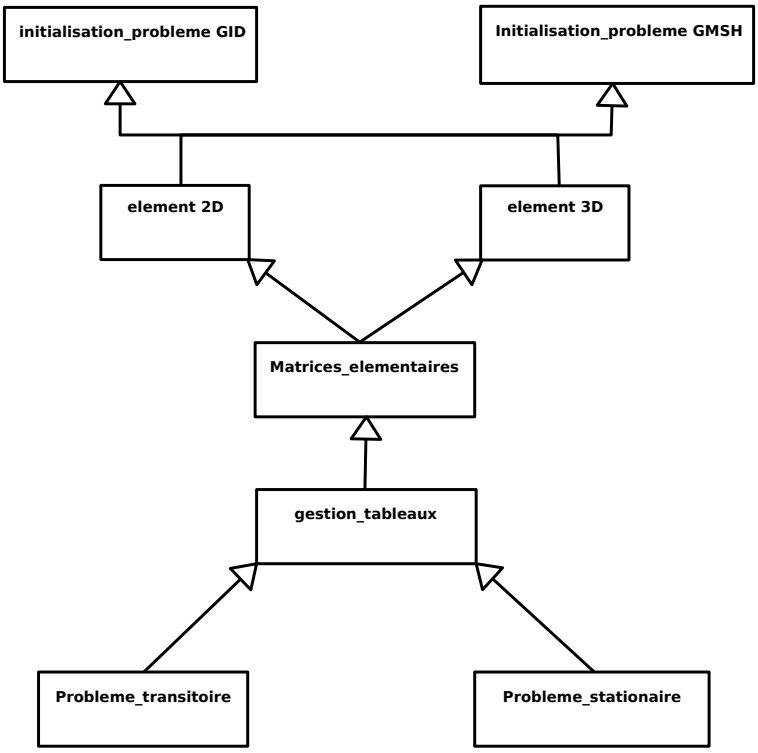


FIG. D.2 – *Diagramme des classes complet*

# Table des figures

# Liste des tableaux

# Bibliographie


[1] Celia M.A., Bouloutas E.T., Zarba R.L. A general mass-conservative numerical solution for the unsaturated flow equation. *Water Resources Research* 1990 ; **26(7)** :1483–1496.

[2] Huang K., Mohanty B.P., van Genuchten M.Th. A new convergence criterion for the modified Picard iteration method to solve the variably saturated flow equation. *Journal of Hydrology* 1996 ; **178** :69–91.

[3] Vanderborght J., Kasteel R., Herbst M., Javaux M., Thiéry D., Vanclooster M., Mouvet C., Vereecken H. A set of analytical benchmarks to test numerical models of flow and transport in soils. *Vadose Zone Journal* 2005 ; **4** :206–221.

[4] Belfort B. Modélisation des écoulements en milieux poreux non saturés par la méthode des éléments finis mixtes hybrides *Thèse de Doctorat* (Novembre 2006).

[5] Kavetski, P.Binning, S.W.Sloan. Adaptive time stepping and error control in a mass conservative numerical solution of the mixed form of Richards equation. *Advances in Water Ressources* 2001 ; **24** : 595-605.

[6] Satish Balay, Kris Bushelman, William D.Gropp, Dinesh Kaushik, Matthew G Knepley, Lois Curfman McInnes, Barry F.Smith, Hong Zhang. ***PETSc*** *Web page : www.mcs.anl.gov/petsc ;* ***PETSc*** *User Manual*. Argonome National Laboratory

[7] Mesgouez, A., Lefeuve-Mesgouez, G., Chambarel, A., "Transient mechanical wave propagation in semi-infinite porous media using a finite element approach", Soil Dynamics and Earthquake Engineering, 2005, 25 (6), 421-430.

[8] Mesgouez A., Lefeuve-Mesgouez G. "Study of transient poroviscoelastic soil motions by semi-analytical and numerical approaches" Soil Dynamics and Earthquake Engineering. 2009. Volume 29, Issue 2, Pages 245-248.

[9] Herbst M, Gottshalk S, Reisel M, Hardelauf H, Kasteel R, Javaux M, Vanderborght J, Vereecken H. On preconditionong for parallel solution of the Richards equation. *Computers and Geosciences*, 2008, 34, 1958-1963.

[10] Bougoul S., Ruy., de Groot F., T.Boulard. Hydraulic and physical properties of stonewool substrate in horticulture. Scienta Horticulturae, 2005, 104, 391-405.