Simulations massivement parallèles des écoulements turbulents à faible nombre de Mach

Mathias Malandain

Thèse

présentée devant

L'Institut National des Sciences Appliquées de Rouen

En vue de l'obtention du

Grade de Docteur

Discipline : Sciences pour l'Ingénieur
Spécialité : Mécanique des Fluides Numérique
Ecole doctorale : Sciences Physiques, Mathématiques et de l'Information pour l'Ingénieur

par

Mathias MALANDAIN

Simulation massivement parallèle des écoulements turbulents à faible nombre de Mach

Directeur de thèse : Luc Vervisch
Encadrant de thèse : Vincent Moureau

Devant le jury composé de :

L. Vervisch	Professeur, INSA, Rouen	Directeur de thèse
V. Moureau	Chargé de recherches, CORIA, Rouen	Encadrant de thèse
B. Knaepen	Professeur, Université Libre de Bruxelles	Examinateur
J.-D. Mueller	Maître de conférences, Queen Mary, University of London	Examinateur
F. Nicoud	Professeur, I3M, Montpellier	Examinateur
L. Giraud	Directeur de recherches, INRIA/CERFACS, Toulouse	Rapporteur
B. Nkonga	Professeur, JAD/INRIA, Nice	Rapporteur

Simulation massivement parallèle des écoulements turbulents à faible nombre de Mach

L'objectif de cette thèse est l'accélération des solveurs utilisés pour la résolution de l'équation de Poisson pour la pression, dans le cas de la simulation d'écoulements à faible nombre de Mach sur des maillages non structurés. Cet objectif est complété par un besoin de stabilité, en particulier sur des géométries complexes. Plusieurs modifications de la méthode des Gradients Conjugués avec déflation ont été considérées à cet effet. Une méthode de redémarrage basée sur une estimation de l'effet des erreurs numériques a été mise en oeuvre et validée. Par la suite, une méthode consistant à calculer des solutions linéaires ou quadratiques par morceaux sur le maillage grossier s'est avérée instable dans le solveur non structuré YALES2. La nouvelle méthode alors développée consiste à transformer la méthode standard de déflation à deux niveaux de maillage en une méthode à trois niveaux. Cependant, le nombre élevé d'itérations sur le troisième niveau de maillage nouvellement créé ralentit le solveur, ce que nous avons rectifié grâce à deux méthodes développées particulièrement pour réduire le nombre d'itérations sur les niveaux grossiers. La première méthode est la création de solutions initiales grâce à une méthode de projection adaptée. La seconde consiste en une adaptation du critère de convergence sur les niveaux grossiers. Les résultats numériques sur des simulations massivement parallèles, avec le solveur à deux niveaux classique, montrent une réduction considérable du temps de calcul du solveur et une amélioration importante de sa scalabilité faible. L'application de ces techniques à la déflation à trois niveaux induit des gains supplémentaires en termes de temps de calcul. Outre le perfectionnement de ce solveur, des recherches supplémentaires doivent être conduites sur l'équilibrage dynamique de charges de calcul, qui pourrait devenir un développement-clé du solveur.

Massively parallel simulation of low-Mach number turbulent flows

The main objective of this thesis is to accelerate the solvers used for solving the pressure Poisson equation, for the simulation of low-Mach number flows on unstructured meshes. This goal is completed with a need for stability, in particular when dealing with complex geometries. To this effect, several modifications of the deflated Conjugate Gradient method have been assessed. A restart method based on an estimation of the effect of numerical errors has been implemented and validated. Then, a method consisting in computing piecewise-linear or piecewise-quadratic solutions on the coarse grid level has proven unstable in the unstructured solver YALES2. The new method developed then consists in turning the standard two-level deflated Conjugate Gradient solver into a three-level method. Therefore, the high number of iterations on the newly created third level slows down the solver, which we have corrected thanks to two methods developed in order to reduce the number of iterations on the coarse levels. The first method is the creation of initial guesses thanks to a well-suited projection method. The second one consists in adapting the convergence criterion on the coarse grids. Numerical results on massively parallel simulations, with the standard two-level solver, show a drastic reduction of the computational times of the solver and an important improvement of its weak scaling. The implementation of these techniques to the three-level deflation induces additional gains in terms of computational times. Besides perfecting this solver, complementary research has to be conducted regarding dynamic load balancing, which could become a key development of the solver.

Remerciements

Je considère la rédaction des remerciements comme un exercice à la fois périlleux et exaltant, dans le sens où il m'est difficile de n'oublier personne parmi tous ceux qui ont contribué à ce que j'ai fait ces trois dernières années et à qui je suis devenu, mais qu'il m'est très agréable de me souvenir ce que chacun a fait pour moi. Me devant de toute façon de m'atteler à cette tâche délicate, je m'excuse d'avance auprès de ceux que je vais oublier de citer, car je sais d'avance qu'il y en aura.

En premier lieu, je me dois de remercier Luc Vervisch, directeur de cette thèse, pour la confiance qu'il m'a accordé, et Vincent Moureau, encadrant, pour le temps qu'il m'a consacré et la patience avec laquelle il a répondu à mes questions, même les plus stupides. Pour l'accueil au sein du laboratoire et pour m'avoir tenu par la main pour toutes ces démarches administratives qui m'inspirent toujours l'ennui le plus profond et l'incompréhension la plus totale, je salue bien bas Mourad Boukhalfa, directeur du CORIA, ainsi que Valérie Thieury et Laure Paris.

François Pellegrini, qui a accepté de m'aider en profondeur pour l'utilisation correcte de la bibliothèque PT-Scotch dont il est le créateur, mérite largement que je le remercie très chaleureusement.

Obviously, pour la qualité de leur accueil, je salue respectueusement les organisateurs du BIRS Workshop 2010 et, non sans un soupçon d'ironie, du THMT 2012 de Palerme. Je pense également aux participants de ces deux évènements, et particulièrement à Jonathan, Christian et Stefan (the beers were great, guys), Sheddia et David (ensemble ou pas ? le débat perdure), ainsi qu'à la troupe de snowboarders avec qui j'ai partagé un succulent fish'n'chips et des cocktails bizarrement trafiqués à Banff.

Une pensée particulière, parmi tous les collègues avec qui j'ai parlé sciences, déliré, bu des cafés, et j'en oublie, à la team inébranlable du bureau 205, Guimch, Pec et Suresh, qui m'ont fait prendre des fous-rires inoubliables. Je dois également citer, parmi les permanents, Pascale, Ghislain, Alexis, Yves, Guillaume, Gérard (merci encore pour Die Walküre !) et Jérôme, puis, dans le désordre le plus total : Nico et sa maladie californienne, qui m'ont tellement rendu service que je suis redevable à vie, Cindy et sa belle voiture, Bebeu (délire, Terrance !) et Micheline, le viking, Lola, Xavier, Jon et sa haine des accordéonistes, Benjamin et Benjamin (faciles à différencier : l'un des deux est grec affirmé, l'autre refoule), Alexandre et sa touffe, Chloé qui a l'audace de soutenir sa thèse le même jour que moi (aaaah, vile créature démoniaque !), Fabien, Emilien, Florian, Damien, Ni-

colas avec qui ce fut un plaisir de se faire souffrir les mollets en altitude, Yann, Pradip, Dounia, Paul, et tous ceux que j'oublie (et qui m'en voudront... pas vrai, les koupains ?).

S'il est très probable qu'ils ne liront jamais ces lignes, il y a néanmoins tout un tas d'artistes qui m'ont énormément aidé à tenir le coup et que je tiens à citer. Qu'ils m'aient aidé à m'apaiser dans les périodes de tension ou à me réveiller, parfois brutalement, durant les semaines de débuggage et de tests à la chaîne, je salue au nom de l'Art les groupes suivants : Igorrr ("je suis dans la salle de bains..."), Whourkr, Drumcorps, TesseracT, Vildhjarta, Septicflesh, Yog, Trepalium, Twinkle ("Joue avec Moi" est la meilleure track d'indus jamais composée, les mecs), Klone, Hacride, Gojira, A.I.(d), The Algorithm (qui me doit toujours une photo dédicacée sur un tracteur), A Backward Glance on a Travel Road, The Amenta, Stupeflip, Niveau Zero, Synth-Etik, Ruby My Dear, Wormskull, auxquels je dois rajouter pour l'ensemble de leurs catalogues les labels Hands, Ant-Zen et Basick Records.

La liste pourrait être complétée par tous mes anciens élèves de l'INSA, mais pour n'en citer que quelques-uns, je salue et remercie JB (tes carnets de Dikpakparé se lisent comme un bon roman), Faustine, Florian ("Des maths et de la musique... Qu'est-ce qu'on pourrait bien faire d'autre dans la vie ?"), Camille, Gaby, Merlin et Atanley.

Evidemment, mes deux familles se doivent de figurer dans ces pages : la première, constituée de mes donneurs de vie et de leur autre enfant, sait que je continue à les aimer même quand je ne leur donne pas de nouvelles ; la seconde, dont j'ai grappillé les membres ça et là au cours d'une existence parfois tumultueuse et souvent totalement idiote, m'aide à garder la tête haute même quand j'ai envie de l'enfoncer dans le sable.

Claudine, Daniel et Cynthia constituent la première ; Camille (Toukinette), Seb (Nours), Vincent (Plouki), Baptiste (le Raptor), Aurèl (Pépin Bref-Le) et Guimch (Guimch) sont au coeur de la seconde.

A cette partie interminable doit s'ajouter, bien sûr, une personne très particulière.

Mon immigrée préférée, j'ai déjà tant partagé avec toi, et déjà tant appris de toi, que toute tentative de liste exhaustive serait vouée à l'échec. Jouvencelle ensorceleuse, tu as irradié mon existence. (Amateurs d'acrostiches, bonjour.)

Pfiouh, je crois que je suis enfin arrivé au bout. Allez, un dernier calcul et on s'en va.

Table des matières

Nomenclature

Acronymes

3LDPCG	Solveur de Gradient Conjugué Préconditionné avec Déflation sur trois niveaux de maillage
A-DEF2	Deuxième Variante Adaptée de la Déflation
AITEB2	Aerothermal Investigations on Turbine Endwalls and Blades
CERFACS	Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
CFD	Computational Fluid Dynamics
CORIA	COmplexe de Recherche Interprofessionnel en Aérothermochimie
CPU	Central Processing Unit
DCG	Gradient Conjugué avec Déflation
DDD	Double Domain Decomposition
DNS	Direct Numerical Simulation
FLOPS	Floating Point Operation Per Second
HPC	High Performance Computing
INRIA	Institut National de Recherche en Informatique et en Automatique
LDPCG	Gradient Conjugué Préconditionné avec méthode de déflation plane
LES	Large Eddy Simulation
MG	Méthodes Multi-Grilles
MIMD	Multiple Instruction, Multiple Data

MIMD	Single Instruction, Single Data
MISD	Multiple Instruction, Single Data
MPI	Message Passing Interface
MTU	Motoren- und Turbinen-Union GmbH
OpenMP	Open MultiProcessing
PCG	Gradient Conjugué Préconditionné
PRECCINSTA	PREdiction and Control of Combustion INSTAbilities for industrial gas turbines
R3LDPCG	Solveur de Gradient Conjugué Préconditionné avec Déflation sur trois niveaux de maillage, avec application des méthodes de réduction du nombre d'itérations sur les maillages intermédiaire et grossier
RA-DEF2	Solveur A-DEF2 avec application des méthodes de réduction du nombre d'itérations sur le maillage grossier
RANS	Reynolds-Averaged Navier-Stokes
RHS	Membre de droite de l'équation, de l'anglais Right Hand Side
RK4	Runge-Kutta au quatrième ordre
SDD	Single Domain Decomposition
SIMD	Simple Instruction, Multiple Data
SPMD	Single Process/Program, Multiple Data
TGCC	Très Grand Centre de Calcul
TTG4A	Two-step Taylor-Galerkin au quatrième ordre, première variante
YALES2	Yet Another Large Eddy Simulation Solver

Lettres grecques

$\bar{\Phi}$	Champ filtré
ϵ	Dissipation
γ	Critère de convergence sur le maillage fin

γ	Critère de convergence sur le maillage fin
γ_i	Critère de convergence sur le niveau grossier
γ_i	Critère de convergence sur le niveau grossier
λ_0	Echelle de Kolmogorov
μ	Viscosité dynamique
μ_t	Viscosité turbulente
μ_{artif}	Viscosité artificielle
ν	Viscosité cinématique
Ω	Volume de contrôle
Φ	Champ quelconque
ϕ	Champ scalaire potentiel de la décomposition de Helmholtz
Φ'	Fluctuations du champ
ρ	Masse volumique
τ	Tenseur des contraintes visqueuses
τ_C	Temps de coupure

Lettres latines

$\bar{\mathbf{n}}$	Vecteur normal à la surface
$\bar{\mathbf{u}}$	Vitesse moyenne
$\hat{A}$	Opérateur laplacien projeté sur le maillage grossier
$\mathbf{b}$	Second membre du système à résoudre par le PCG généralisé de Tang *et al.*
$\mathbf{g}$	Pesanteur
$\mathbf{n}$	Pas de temps
$\mathbf{q}$	Quantité de mouvement
$\mathbf{u}'$	Fluctuations de la vitesse

$\mathbf{x}$	Vecteur inconnu dans le PCG généralisé de Tang *et al.*
$\mathcal{A}$	Matrice du système à résoudre par le PCG généralisé de Tang *et al.*
$\mathcal{P}$	Préconditionneur du PCG généralisé de Tang *et al.*
A	Matrice de l'opérateur laplacien
b	Vecteur second membre
C	Fonction de coût
c_s	Vitesse du son
E_C^{turb}	Energie cinétique turbulente
e_T	Energie totale par unité de masse
G	Noyau de convolution
k	Dimension du sous-espace de déflation
l_C	Longueur de coupure
L_i	Levelset du graphe associé à un maillage partitionné
L_t	Echelle intégrale
n	Nombre d'éléments d'un maillage, nombre d'inconnues d'un système linéaire
n_b	Nombre de blocs d'un maillage prépartitionné
n_f	Nombre de faces
n_{cv}	Nombre de volumes de contrôle dans un maillage
N_{proc}	Nombre de processeurs
n_{sub}	Nombre de sous-groupes par groupe
P	Pression
$P^{\perp}$	Matrice de projection de la méthode de déflation
Q	Matrice de transfert de la méthode de déflation
Q_i	Ratio d'efficacité pour l'étude de *scaling*

R_i	Efficacité réduite
Re_t	Nombre de Reynolds turbulent
u'	Vitesse caractéristique
V_i	Volume de contrôle d'un noeud
W	Matrice de déflation
w_i	Vecteur de déflation
w_p^i	Facteurs multiplicateurs pour le rééquilibrage de charges
x	Vecteur inconnu

Symboles

∇	Opérateur vectoriel nabla
$\partial\Omega$	Frontière du volume de contrôle

Chapitre 1

Physique des écoulements turbulents

L'étude numérique des écoulements turbulents est réalisée grâce aux équations de Navier-Stokes. Il s'agit d'un système d'équations aux dérivées partielles, établies à partir de simples lois de conservation de la masse, de la quantité de mouvement et de l'énergie totale, qui lie entre elles les variables d'état (vitesse, pression, densité...) du système étudié.

Pour établir ces équations, l'approximation des milieux continus est fondamentale. Elle consiste à considérer que les plus petits volumes de matière que nous observons ou simulons sont très grands par rapport aux dimensions des molécules qui composent le fluide, ce qui fait que la nature granulaire de la matière n'est pas à prendre en compte. A l'échelle où nous étudions les phénomènes physiques, les variables d'état de l'écoulement sont donc des fonctions continues, ce qui permet de recourir aux outils mathématiques différentiels.

En outre, dans le contexte des équations de Navier-Stokes, le fluide étudié est supposé newtonien, c'est-à-dire que la contrainte de cisaillement exercée dans une certaine direction est proportionnelle au gradient de vitesse dans la direction orthogonale. Le coefficient de proportionnalité entre ces deux grandeurs est la viscosité dynamique du fluide. Cette propriété permet d'exprimer simplement le tenseur des contraintes visqueuses, qui intervient dans l'expression de la force induite par la viscosité.

Dans ce cadre de travail, nous allons construire dans une première section une forme commune des équations de Navier-Stokes à densité variable. Nous dériverons ensuite les équations permettant d'étudier des écoulements à densité constante, en précisant dans quels contextes un écoulement peut être considéré incompressible et indilatable. Enfin, nous exhiberons l'équation de Poisson pour la pression qui apparait pour la simulation des écoulements à densité constante et discuterons son rôle et les enjeux numériques qu'elle représente.

1.1 Equations de Navier-Stokes à densité variable

Pour un écoulement fluide à densité variable, les équations de Navier-Stokes lient entre eux le champ de densité, les composantes de la vitesse, le champ de pression et la température (ou l'énergie), en considérant

- le bilan local de la masse, qui lie la densité et la vitesse en considérant la convection de la masse par l'écoulement ;
- le bilan de quantité de mouvement, qui lie ces deux variables au gradient de la pression ;
- le bilan d'énergie, qui lie la vitesse et la pression en faisant également intervenir la température (ou l'énergie interne, que l'on peut assimiler au niveau microscopique à l'agitation thermique des particules).

Pour que le système puisse être résolu, nous devons ajouter aux équations décrites ci-avant une équation d'état (respectivement une équation d'énergie), pour lier la température (respectivement l'énergie interne) aux autres variables d'état. Dans cette section, nous présentons une forme commune des équations de Navier-Stokes à densité variable.

1.1.1 Equation de continuité

L'équation de continuité concerne le bilan de masse de l'écoulement. On le dresse sous sa forme intégrale en considérant un volume de contrôle Ω fixe, dit eulérien, dans lequel la matière peut entrer et sortir[1]. La variation globale de la masse dans ce volume est alors due exclusivement au flux de matière qui traverse sa frontière $\partial\Omega$, ce qui s'écrit

$$\frac{\partial}{\partial t}\int_{\Omega}\rho dV = -\int_{\partial\Omega}(\rho\mathbf{u})\cdot\bar{\mathbf{n}}dS \tag{1.1}$$

où ρ est la masse volumique, $\mathbf{u}$ le vecteur vitesse, et $\bar{\mathbf{n}}$ le vecteur normal à la surface $\partial\Omega$, unitaire et orienté vers l'extérieur du volume de contrôle.

En utilisant le théorème de Green-Ostrogradski pour transformer le second membre de l'équation, on obtient

$$\int_{\Omega}\left(\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\mathbf{u})\right)dV = 0\,. \tag{1.2}$$

Comme cette équation est vraie pour tout volume de contrôle eulérien Ω, on obtient la forme locale conservative de l'équation de continuité :

[1] Cette démarche s'oppose à la démarche lagrangienne, dans laquelle un volume de contrôle suit le fluide en mouvement de façon à toujours contenir les mêmes particules.

$$\underbrace{\frac{\partial \rho}{\partial t}}_{\text{Variation locale}} + \underbrace{\nabla \cdot (\rho \mathbf{u})}_{\text{Divergence des flux}} = 0\,. \tag{1.3}$$

1.1.2 Equation de bilan de quantité de mouvement

La quantité de mouvement $\mathbf{q}$ est une variable conservative d'un système physique, qui vaut sur un volume de contrôle donné

$$\mathbf{q} = \int_V \rho \mathbf{u} dV\,. \tag{1.4}$$

Le principe fondamental de la dynamique, c'est-à-dire la deuxième loi de Newton, nous dit que la variation temporelle de la quantité de mouvement est égale à la somme des forces exercées sur le système. Dans un écoulement turbulent, ces forces sont engendrées par la pression, la gravité et la viscosité. On peut alors écrire, en utilisant encore une fois le théorème de Green-Ostrogradski,

$$\int_\Omega \left(\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) \right) dV = \int_\Omega \left(\rho \mathbf{g} - \nabla P + \nabla \cdot \tau \right) dV\,, \tag{1.5}$$

équation dont la validité pour tout volume de contrôle Ω nous permet d'écrire sa forme locale

$$\underbrace{\frac{\partial \rho \mathbf{u}}{\partial t}}_{\text{Instationnaire}} + \underbrace{\nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u})}_{\text{Convection}} + \underbrace{\nabla P}_{\text{Gradient de pression}} = \underbrace{\rho \mathbf{g}}_{\text{Pesanteur}} + \underbrace{\nabla \cdot \tau}_{\text{Viscosité}}\,, \tag{1.6}$$

où P est la pression, $\mathbf{g}$ le champ de pesanteur, et τ le tenseur des contraintes visqueuses qui permet de décrire l'entrainement visqueux du fluide.

1.1.3 Equation d'énergie

Le bilan d'énergie totale est dressé en considérant que la variation temporelle de l'énergie totale massique e_T sur un volume de contrôle est due aux flux convectfis d'énergie et de diffusion de chaleur à la frontière, et au travail des forces appliquées au système. Les flux s'écrivent

$$-\int_{\partial\Omega} (e_T \mathbf{u} + \mathbf{q}) \cdot \bar{\mathbf{n}} dS \tag{1.7}$$

et les travaux des forces exercées sur le fluide

$$-\int_{\partial\Omega} P\mathbf{u} \cdot \bar{\mathbf{n}} dS + \int_{\partial\Omega} (\tau \mathbf{u}) \cdot \bar{\mathbf{n}} dS + \int_\Omega \mathbf{g} \cdot \mathbf{u} dV\,. \tag{1.8}$$

On obtient alors la forme locale du bilan d'énergie

$$\frac{\partial e_T}{\partial t} + \nabla \cdot (\mathbf{u}\,(e_T + P)) = \nabla \cdot (\tau \mathbf{u}) - \nabla \cdot \mathbf{q} + \mathbf{g} \cdot \mathbf{u}\,. \tag{1.9}$$

Une information supplémentaire étant nécessaire pour la résolution du système, qui compte une équation de moins que d'inconnues, on peut utiliser l'équation d'énergie

$$e_T = \epsilon + \frac{1}{2}\mathbf{u} \cdot \mathbf{u}\,, \tag{1.10}$$

où ϵ est l'énergie interne par unité de masse, ou une équation d'état du fluide. En pratique, certains solveurs, comme celui utilisé dans cette thèse, s'appuient sur des méthodes de tabulation pour ne pas avoir à résoudre de telles équations.

1.2 Equations de Navier-Stokes à densité constante

Dans un écoulement à faible nombre de Mach, traditionnellement pour une vitesse inférieure à $0,3c_s$ où c_s est la vitesse du son dans le milieu considéré, on peut considérer que le fluide, qu'il soit liquide ou gazeux, est incompressible, c'est-à-dire que les effets acoustiques sur la dynamique du fluide sont ignorés. Lorsque, de plus, la température est constante, l'écoulement est alors à densité constante. Seules les composantes de la vitesse et le champ de pression sont à déterminer dans ce cas, ce qui est possible en considérant uniquement l'équation de continuité et l'équation de bilan de la quantité de mouvement.

Ces deux équations sont simplifiées par l'hypothèse de la densité constante. Avec ρ constante, l'équation de continuité devient l'équation d'incompressibilité

$$\nabla \cdot \mathbf{u} = 0\,. \tag{1.11}$$

Cette incompressibilité modifie l'expression de la force de viscosité qui intervient dans l'équation du bilan de quantité de mouvement. D'une part, des contraintes normales apparaissent dans l'expression du tenseur des contraintes visqueuses, qui sont des contraintes exercées sur l'écoulement dans la direction du gradient de vitesse correspondant ; ces contraintes, directement proportionnelles à $\nabla \cdot \mathbf{u}$, s'annulent si l'écoulement est incompressible. D'autre part, puisque le tenseur des contraintes visqueuses s'exprime alors

$$\tau = \mu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right]\,, \tag{1.12}$$

avec μ la viscosité dynamique du fluide, sa divergence vaut, dans le cas où μ est constante,

$$\nabla \cdot \tau = \mu \left[\nabla^2 \mathbf{u} + \nabla \underbrace{(\nabla \cdot \mathbf{u})}_{=0}\right]\,. \tag{1.13}$$

On peut également développer $\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = (\mathbf{u} \cdot \nabla + \nabla \cdot \mathbf{u})\,\mathbf{u}$ et utiliser la propriété d'incompressibilité pour obtenir

$$\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = (\mathbf{u} \cdot \nabla)\,\mathbf{u}\,. \tag{1.14}$$

En appliquant ces modifications à l'équation (1.6) et en la simplifiant par ρ, on obtient finalement

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} + \frac{1}{\rho}\nabla P = \mathbf{g} + \nu \nabla^2 \mathbf{u}\,, \tag{1.15}$$

où $\nu = \frac{\mu}{\rho}$ est la viscosité cinématique du fluide, ici supposée constante.

1.3 Equation de Poisson pour la pression

Le couplage fort entre la pression et la vitesse ne permet pas, dans le cadre d'un écoulement incompressible voire à densité constante, de faire apparaître explicitement la valeur de la pression. Nous présentons dans cette section la façon la plus commune d'exprimer la pression en fonction de la vitesse pour un écoulement incompressible et les enjeux que cette expression implicite du champ de pression représente pour la simulation numérique.

1.3.1 Emergence d'une équation de Poisson

La façon la plus directe d'obtenir le champ de pression à partir du champ de vitesse est d'appliquer l'opérateur de divergence à tous les termes de l'équation de bilan de quantité de mouvement (1.15), afin d'y supprimer des termes dépendants de $\mathbf{u}$ en utilisant la condition d'incompressibilité (1.11). On obtient

$$\nabla^2 P = \rho \nabla \cdot \left(\mathbf{g} + \nu \nabla^2 \mathbf{u} - \mathbf{u} \cdot \nabla \mathbf{u}\right)\,. \tag{1.16}$$

Cette équation est une équation de Poisson, c'est-à-dire une équation de la forme $\nabla^2 x = f$ où f est une fonction connue. Nous savons donc que, pour des conditions aux limites appropriées, le problème consistant à déterminer le champ de pression en résolvant l'équation de Poisson pour la pression est bien posé, c'est-à-dire qu'il admet une solution unique qui dépend de façon continue du second membre[2]. On y supprime généralement le terme en $\mathbf{g}$, puisque l'influence de la gravité est négligeable dans les écoulements étudiés.

Une autre équation de Poisson, dont le second membre est plus simple, apparaitra lors de la discrétisation temporelle des équations de Navier-Stokes utilisant la méthode de projection, ce qui est détaillé dans la partie 2.2.1.2.

[2]En pratique, l'unicité de la solution de l'équation de Poisson résolue par un solveur de CFD dépend des conditions aux limites. Parfois, elle est en fait unique à une constante près, comme dans le cas du solveur YALES2 utilisé dans le contexte de cette thèse.

1.3.2 Enjeux pour la CFD

La simulation d'un écoulement incompressible non réactif est généralement réalisée grâce à une méthode de pas fractionnaire [12, 38, 77], qui consiste à séparer le gradient de pression des autres termes de l'équation de quantité de mouvement. La version modifiée de la méthode de pas fractionnaire utilisée dans le solveur YALES2 dans le contexte de cette thèse est présentée plus en détail en section 2.2, en même temps que les méthodes de discrétisation spatiale employées ; nous pouvons d'ores et déjà la présenter de façon simplifiée, en exposant les trois étapes fondamentales de calcul à chaque pas de temps.

1. A partir du champ de vitesse $\mathbf{u^n}$, un champ de vitesse intermédiaire $\mathbf{u}^\star$, qui ne respecte pas la contrainte d'incompressibilité, est estimé en résolvant une équation de quantité de mouvement tirée de (1.15).

2. Une équation de Poisson pour la pression est résolue afin de déterminer le champ de pression $P^{\mathbf{n}+1}$.

3. L'influence de $P^{\mathbf{n}+1}$ est prise en compte pour le calcul du champ incompressible $\mathbf{u}^{\mathbf{n}+1}$.

Une équation de Poisson pour la pression y est à résoudre dans l'étape 2, différente de l'équation originale (1.16) qui apparait directement à partir des équations de Navier-Stokes en incompressible. Cette équation est de la forme donnée par l'équation 2.12. La résolution de cette équation requiert de faire appel à un solveur linéaire. En effet, une fois l'opérateur laplacien discrétisé selon le schéma utilisé par le solveur, l'équation de Poisson devient un système d'équations linéaires dont l'inconnue est le vecteur des valeurs de la pression sur les points du domaine discrétisé.

La résolution de cette équation peut représenter une part très importante du temps de calcul nécessaire à la simulation de chaque pas de temps. En effet, les solveurs utilisés pour cette résolution sont obligatoirement des solveurs itératifs, et le nombre d'itérations nécessaire à l'obtention d'une estimation suffisamment précise de la solution exacte dépend fortement de l'algorithme utilisé et des caractéristiques de la matrice de l'opérateur laplacien. De plus, le déroulement de tels solveurs nécessite des communications interprocesseurs à chaque itération, et ces communications peuvent représenter une proportion très importante du temps total de la simulation, jusqu'à 80 % dans certains cas.

L'optimisation des méthodes de résolution de systèmes linéaires est donc un point-clé pour la simulation des écoulements incompressibles. Elle est le sujet principal des travaux de recherche présentés dans le chapitre 6 du présent manuscrit. Auparavant, nous présenterons les méthodes utilisées par le solveur YALES2 pour la discrétisation en temps et en espace des équations régissant les écoulements, ainsi que les outils numériques de résolution des systèmes linéaires et leur mise en oeuvre en parallèle.

Chapitre 2

Aspects numériques de la CFD

En amont de toute simulation numérique liée aux écoulements fluides, autrement dit pour toute simulation de CFD, doivent être définis les paramètres de l'écoulement étudié : domaine physique, conditions aux limites et équations à résoudre. Généralement, un maillage est ensuite défini sur le domaine[1], et les équations aux dérivées partielles régissant le ou les phénomènes étudiés sont discrétisées sur ce maillage, en fonction du choix des schémas numériques. La simulation peut alors avoir lieu, au terme de laquelle les données brutes pourront éventuellement passer par une étape de *postprocessing*, c'est-à-dire être traitées en vue de leur visualisation ou de leur interprétation.

Cependant, cette description très générale du processus de simulation néglige une problématique importante, qui est qu'un écoulement turbulent est un phénomène multi-échelles. Cet aspect est évoqué dans une première section ; différentes approches numériques sont alors décrites, qui simulent tout ou partie du spectre énergétique de l'écoulement, en fonction des besoins liés à la simulation et des moyens de calcul disponibles. Nous aborderons ensuite la conception du solveur YALES2, en précisant l'approche sur laquelle il repose, mais également les schémas numériques qui y ont été adoptés pour la discrétisation des équations qui régissent les phénomènes physiques.

2.1 Méthodes de simulation des écoulements turbulents

Les écoulements turbulents sont des phénomènes multi-échelles, en temps comme en espace. Pour être capables de les simuler intelligemment, nous devons donc avoir une idée précise de l'étendue de ces échelles, et savoir lesquelles nous pourrons simuler et lesquelles, inaccessibles par la résolution de nos simulations, devront être modélisées. Dans cette sec-

[1] Il existe des méthodes s'affranchissant du besoin de créer un maillage, dans lesquelles un nuage de points est défini sur le domaine sans que rien ne les lie entre eux. Une revue de littérature à ce sujet est donnée par A. J. Katz dans sa thèse datant de 2009 [35]. Ces méthodes ne sont cependant pas majoritaires dans les codes CFD actuels, et ne seront pas traitées dans ce manuscrit, qui se focalise sur des méthodes numériques basées sur un maillage du domaine, telles que celle mise en eouvre dans le solveur YALES2.

tion, nous nous intéresserons dans un premier temps aux échelles de la turbulence et aux valeurs permettant de les caractériser, puis nous aborderons les grandes approches qui en découlent pour la simulation des écoulements turbulents.

2.1.1 Echelles de la turbulence

L'observation montre que plusieurs tourbillons de tailles différentes peuvent coexister en un même lieu de l'écoulement, et que les tourbillons de grande taille peuvent se diviser en tourbillons plus petits. En outre, la structure de l'écoulement turbulent est quasi-fractale : le zoom sur une partie d'un écoulement met en évidence cette propriété jusqu'à une certaine échelle, qui est celle des plus petits tourbillons.

Les échelles spatiales de la turbulence vont alors de l'échelle de Kolmogorov, qui correspond à la taille des plus petites structures, à l'échelle intégrale qui correspond à la taille des plus grands tourbillons. Ces deux échelles sont déterminées grâce à une analyse dimensionnelle, qui permet de les exprimer à partir des paramètres qui influent sur eux.

- L'échelle intégrale L_t est liée à deux grandeurs : l'énergie cinétique turbulente E_C^{turb}, dont les grands tourbillons portent la plus grande partie ; et la dissipation ϵ, qui dicte la destruction des grandes structures pour en former des plus petites, et par conséquent qui en limite la taille.

- L'échelle de Kolmogorov λ_0 dépend de la dissipation et de la viscosité du fluide, car en-dessous d'une certaine taille, la force de frottement provoque la destruction immédiate de toute structure en formation et la dissipation sous forme de chaleur de l'énergie cinétique correspondante.

Le rapport entre ces deux échelles s'exprime alors

$$\frac{L_t}{\lambda_0} = \left(\frac{u' L_t}{\nu}\right)^{\frac{3}{4}} = Re_t^{\frac{3}{4}}\,, \tag{2.1}$$

où Re_t est le nombre de Reynolds turbulent, un nombre sans dimension utilisé dès la fin du XIXme siècle pour représenter le rapport entre les forces d'inertie et les forces visqueuses, et qui s'écrit

$$Re_t = \frac{u' L_t}{\nu} \tag{2.2}$$

avec u' la vitesse caractéristique de l'écoulement à l'échelle intégrale. Comme les écoulements turbulents, dans lesquels les forces d'inertie sont considérables par rapport aux forces visqueuses, ont typiquement des Reynolds supérieurs à 2000 et parfois même bien plus élevés, le rapport de taille entre les plus petits et les plus grands tourbillons présents dans l'écoulement est au strict minimum de l'ordre de quelques centaines.

Les échelles de temps caractéristiques de l'écoulement sont bien entendu calquées sur les échelles spatiales ; l'échelle intégrale en temps correspond au temps de parcours moyen de la distance L_t à la vitesse de l'écoulement à cette échelle, qui est proche de la vitesse caractéristique u', tandis que l'échelle de temps de Kolmogorov correspond à un temps de parcours moyen à l'échelle de Kolmogorov. On peut montrer que le rapport entre ces deux échelles de temps est de l'ordre de $Re_t^{1/2}$, c'est-à-dire, au mieux, de l'ordre de quelques dizaines dans un écoulement turbulent.

Le maillage utilisé doit donc être particulièrement raffiné pour parvenir à capturer toutes les échelles spatiales de l'écoulement, et plusieurs centaines de pas de temps au minimum doivent être simulés pour que toutes les échelles temporelles soient analysables dans le contexte de la simulation d'un temps physique de l'ordre du temps de résidence moyen d'une particule fluide dans le domaine. Cet enjeu de la simulation directe des écoulements turbulents s'accentue encore lorsque l'écoulement est réactif, puisque les temps caractéristiques de la chimie peuvent encore accroître le rapport entre les plus grandes et les plus petites échelles temporelles. Des méthodes ont donc été développées pour permettre des simulations ne prenant pas en compte toutes les échelles de temps et/ou d'espace, par l'application de moyennes en temps ou de filtres spatiaux.

2.1.2 Approches DNS - LES - RANS

Les trois principales méthodes pour la simulation numérique sont présentées ci-après, allant de la plus coûteuse à la plus rapide. On peut donner une première idée de ces trois méthodes en mentionnant que la méthode DNS consiste en la simulation de toutes les échelles de la turbulence, tandis que la méthode LES modélise les effets turbulents aux plus petites échelles et que la méthode RANS modélise toutes les échelles de la turbulence ; cependant, il faut également mentionner que la méthode RANS est basée sur une décomposition en quantités moyennes et en quantités fluctuantes, ce qui ne lui permet au final de représenter que le comportement moyen d'un écoulement en lissant complètement les fluctuations liées à la turbulence.

DNS est l'acronyme de *Direct Numerical Simulation*, c'est-à-dire que l'on réalise une simulation sur un maillage assez fin et avec des pas de temps assez réduits pour que toutes les échelles temporelles et spatiales de l'écoulement soient prises en compte. Il est évident que cette méthode est la plus simple à mettre en oeuvre, mais qu'elle est particulièrement coûteuse, car la plus grande taille de maille doit être inférieure à l'échelle de Kolmogorov et le pas de temps inférieur à l'échelle de temps de Kolmogorov. Elle est donc principalement destinée à des calculs de recherche ou académiques, généralement avec des nombres de Reynolds relativement peu élevés.

LES est l'acronyme de *Large Eddy Simulation*, que l'on pourrait traduire par "simulation aux grandes échelles". Dans cette méthode, les équations de Navier-Stokes sont filtrées en espace de façon à ce que seules les structures de taille supérieure à une

certaine longueur soient simulées. Les effets turbulents sont modélisés sur les plus petites échelles. Le filtre utilisé peut également être en temps, de façon à ce que le champ résultant corresponde à une moyenne mobile du champ filtré en espace.

Pour le filtrage, on utilise un filtre passe-bas qui est défini pour tout champ $\Phi(x,t)$ par une convolution de la forme

$$\bar{\Phi} = G \star \Phi \text{ soit } \bar{\Phi}(x,t) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \Phi(r,\tau) G(x-r, t-\tau) d\tau dr \tag{2.3}$$

où G est le noyau de convolution. Pour la LES, ce noyau utilise une longueur de coupure l_C et un temps de coupure τ_C ; le filtre supprime alors les composantes de hautes fréquences temporelles et spatiales du champ de départ, que l'on peut écrire comme la somme du champ filtré et d'une fluctuation en temps et en espace

$$\Phi = \bar{\Phi} + \Phi' \,, \tag{2.4}$$

le terme de fluctuations Φ' vérifiant la condition $\bar{\Phi}' = 0$. Si le filtre utilisé est seulement en espace, la convolution devient

$$\bar{\Phi}(x,t) = \int_{-\infty}^{+\infty} \Phi(r,t) G(x-r) dr \,. \tag{2.5}$$

Le filtre le plus simple est uniquement un filtre en espace de type boîte, de la forme

$$G(x) = \begin{cases} \dfrac{1}{l_C} \text{ si } |x| \leq \dfrac{l_C}{2} \,, \\ 0 \text{ sinon.} \end{cases} \tag{2.6}$$

On peut alors considérer le filtrage comme une forme de moyenne mobile en espace. D'autres filtres, plus complexes mais continus en espace, existent et sont utilisés dans le même but [69]. En effet, l'introduction de l'équation (2.4) dans les équations d'incompressibilité (1.11) et de bilan de la quantité de mouvement (1.15) fait apparaitre une forme filtrée des équations de Navier-Stokes.

Un terme non-linéaire d'advection de la quantité de mouvement apparait alors, dont la valeur est inconnue mais requise pour la résolution. Ce terme est en effet lié aux transferts de quantité de mouvement des plus grandes structures vers les plus petites, et vice-versa. La résolution de ce problème est généralement basée sur l'hypothèse de Boussinesq [9], selon laquelle ces transferts peuvent être modélisés par l'ajout d'une viscosité turbulente μ_t, de façon à ôter aux structures simulées l'énergie qui est transmise aux petites structures [40]. Différents modèles existent pour le calcul

de la viscosité turbulente, que nous ne détaillerons pas dans le contexte de cette thèse.

Cette approche permet l'étude d'écoulements instationnaires, et peut tendre vers la DNS pour un maillage suffisamment fin. Le développement des moyens de calcul permet son utilisation de plus en plus fréquente, aussi bien dans le milieu académique, où elle est déjà très courante, que dans l'industrie où son usage est en pleine expansion.

RANS est l'acronyme de *Reynolds-Averaged Navier-Stokes*, c'est-à-dire que l'on considère dans cette méthode les équations de Navier-Stokes moyennées en temps à partir de la décomposition de Reynolds, qui est

$$\mathbf{u} = \bar{\mathbf{u}} + \mathbf{u}', \tag{2.7}$$

où $\bar{\mathbf{u}}$ est le champ de vitesse moyenné en temps et $\mathbf{u}'$ est la partie fluctuante de la vitesse, dont la moyenne est nulle. Lorsque l'on décompose la vitesse de cette manière dans les équations d'incompressibilité (1.11) et de bilan de la quantité de mouvement (1.15), de nombreuses simplifications apparaissent à cause de termes dont la moyenne est nulle. L'intégration en temps élimine finalement la dérivée temporelle $\frac{\partial \bar{\mathbf{u}}}{\partial t}$, ce qui aboutit à un système d'équations aux dérivées partielles où toutes les dérivées sont en espace. Les effets de la turbulence sont alors modélisés à toutes les échelles pour fermer le système d'équations.

La résolution de ce système donne une approximation des champs moyens, qui peut être insuffisante selon les besoins de la simulation. Le rendement énergétique d'un brûleur, par exemple, peut convenablement être estimé grâce à une simulation RANS, mais les variations plus ou moins brutales de pression et de température jouent un rôle déterminant dans sa longévité. En outre, l'application d'une telle méthode requiert idéalement une connaissance *a priori* de la turbulence, ce qui rend très lourd l'impact du choix du modèle. La méthode RANS est néanmoins largement utilisée dans l'industrie, notamment à cause de ses coûts de calcul très réduits.

Certaines approches hybrides, dont les approches DES [73] et URANS [89], ne sont pas abordées dans cette classification.

2.2 Construction d'un solveur LES non structuré

Le solveur LES utilisé dans le cadre de cette thèse est le code YALES2, dont le développement a été initié en 2007 par Vincent Moureau. Actuellement développé au sein du CORIA, YALES2 est un solveur volumes finis non-structuré conçu pour la simulation massivement parallèle sur des maillages pouvant atteindre plusieurs milliards d'éléments.

Ce code multi-physique, conçu pour l'étude de la combustion turbulente diphasique, depuis l'atomisation du combustible jusqu'à la prédiction des taux de polluants en sortie, contient différents solveurs permettant, entre autres, la simulation de sprays, la prise en compte de phénomènes magnétohydrodynamiques, acoustiques ou radiatifs, ou encore la gestion de domaines mouvants, pour la biomécanique par exemple.

Dans cette section, nous commencerons par établir le cadre de travail numérique, en abordant la modalité de filtrage des équations appliquée dans le code YALES2 et la méthode de projection qui fait apparaître l'équation de Poisson pour la pression résolue à chaque pas de temps du solveur incompressible. Nous présenterons alors la méthode des volumes finis et son application dans YALES2, ainsi que les schémas de discrétisation en espace utilisés par le code. Cette description nous amènera à considérer les opérateurs différentiels sous leur forme discrétisée, et la façon dont ils sont déclarés dans le solveur ; la déclaration de l'opérateur laplacien discrétisé est particulièrement importante dans le cadre du travail mené lors de cette thèse. Dans une dernière sous-section, nous parlerons brièvement des schémas de discrétisation en temps, afin de compléter le panorama simplifié mais global des aspects numériques du solveur incompressible utilisé dans le cadre de cette thèse.

2.2.1 Cadre de travail numérique

Nous nous intéresserons ici tout particulièrement au choix du filtrage des équations utilisé, ainsi qu'à la méthode de projection qui permet d'ordonnancer la résolution des équations de Navier-Stokes en un pas de temps donné dans le code YALES2.

2.2.1.1 Filtrage des équations

Le filtrage des équations utilisé dans YALES2 est totalement implicite, c'est-à-dire que seul l'ajout de la viscosité turbulente μ_t différencie les équations résolues dans le code des équations de Navier-Stokes standard. Le terme de viscosité turbulente est calculé en fonction de la taille de maille, qui est directement utilisée comme longueur de coupure. Cette méthode tire le plus grand avantage de la finesse du maillage, et permet à une simulation LES de devenir une simulation numérique directe (DNS) si le maillage devient assez raffiné, c'est-à-dire si la plus grande taille de maille devient inférieure à l'échelle de Kolmogorov ; dans un tel cas, la viscosité turbulente devient uniformément nulle.

Différents modèles de turbulence ont été implémentés dans le code : Smagorinsky constant [71, 74], Smagorinsky dynamique localisé [39], WALE [85] et SIGMA [25] ; l'utilisateur peut choisir le modèle utilisé dans une simulation grâce à un simple mot-clé à ajouter dans un fichier texte d'entrée, dont l'extension est `.in`, ou laisser le soin au solveur de choisir un modèle par défaut.

2.2.1.2 Méthode de projection

La méthode de projection, fréquemment utilisée pour la simulation d'écoulements incompressibles, est basée sur le théorème de Helmholtz, aussi nommé théorème fondamental du calcul vectoriel. Ce théorème, énoncé lors de la deuxième moitié du XIXme siècle, énonce que tout champ vectoriel suffisamment lisse dans un espace en trois dimensions peut être exprimé comme la somme d'un champ irrotationnel et d'un champ solénoïdal. Appliqué au champ de vitesse, nous obtenons une expression de la forme

$$\mathbf{u} = \mathbf{u}_i + \mathbf{u}_s \,. \tag{2.8}$$

Dans cette équation :

- $\mathbf{u}_i$ est la composante irrotationnelle du champ de vitesse, qui vérifie $\nabla \times \mathbf{u}_i = 0$. Elle représente le flux qui entre ou sort d'un volume infinitésimal autour de chaque point.
- $\mathbf{u}_s$ est la composante solénoïdale du champ de vitesse, qui vérifie $\nabla \cdot \mathbf{u}_s = 0$. Elle représente en quelque sorte l'enroulement local du champ de vitesse autour de chaque point.

On peut écrire $\mathbf{u}_i$ comme le gradient d'un certain champ scalaire ϕ, soit $\mathbf{u}_i = \nabla\phi$. L'opérateur divergence appliqué à l'équation (2.8) donne alors

$$\nabla \cdot \mathbf{u} = \nabla^2\phi \,, \tag{2.9}$$

puisque, par définition, $\nabla \cdot \mathbf{u}_s = 0$. On peut ainsi déterminer le champ scalaire ϕ à partir de $\mathbf{u}$ par la résolution d'une équation de Poisson, afin de déterminer la partie irrotationnelle du champ de vitesse $\mathbf{u}_i = \nabla\phi$.

Cette méthode permet de décomposer la résolution des équations de Navier-Stokes incompressibles en trois étapes, qui correspondent à celles évoquées en sous-section 1.3.2. Voici le détail de ces trois étapes, telles que formulées par Chorin en 1968 [13].

1. Pour obtenir une première estimation du champ de vitesse $\mathbf{u^{n+1}}$ à partir de $\mathbf{u^n}$, on réécrit l'équation de bilan de la quantité de mouvement (1.15) sans le gradient de pression, qui ne contribue qu'à la composante irrotationnelle du champ de vitesse. On obtient alors une expression pour le calcul explicite de $\mathbf{u}^\star$, à savoir

$$\frac{\mathbf{u}^\star - \mathbf{u^n}}{\Delta t} = \nu\nabla^2\mathbf{u^n} - (\mathbf{u^n} \cdot \nabla)\,\mathbf{u^n} \,. \tag{2.10}$$

2. La prise en compte du gradient de pression se fait alors tout simplement grâce à l'équation

$$\frac{\mathbf{u^{n+1}} - \mathbf{u^\star}}{\Delta t} = -\frac{1}{\rho}\nabla P^{\mathbf{n}+\frac{1}{2}}\,, \tag{2.11}$$

ce qui est alors impossible à cause de la méconnaissance de $P^{n+\frac{1}{2}}$. Cependant, en appliquant l'opérateur divergence à cette équation et en imposant la contrainte d'incompressibilité à $\mathbf{u^{n+1}}$, on obtient

$$\frac{\rho}{\Delta t}\nabla \cdot \mathbf{u^\star} = \nabla^2 P^{\mathbf{n}+\frac{1}{2}}\,, \tag{2.12}$$

qui est une équation de Poisson pour la pression.

3. Un parallèle direct est à faire avec l'équation (2.9) ci-dessus : l'équation ainsi créée permet d'extraire la composante irrotationnelle de $\mathbf{u^\star}$, qui lui est alors soustraite pour obtenir le nouveau champ de vitesse

$$\mathbf{u^{n+1}} = \mathbf{u^\star} - \frac{\Delta t}{\rho}\nabla P^{\mathbf{n}+\frac{1}{2}}\,. \tag{2.13}$$

L'application qui en est faite dans YALES2 est différente. En effet, l'équation (2.10), qui permet d'obtenir l'estimation $\mathbf{u^\star}$ du champ de vitesse $\mathbf{u^{n+1}}$ dans la méthode de Chorin, peut fausser le comportement de l'écoulement : la disparition du terme lié à la force de pression peut provoquer l'éclatement numérique des grandes structures. La méthode est donc modifiée comme suit pour son implémentation dans le code YALES2.

- Une première estimation du champ de vitesse est calculée à partir de l'équation de bilan de quantité de mouvement complète, en considérant le champ de pression au pas de temps $\mathbf{n}$, soit

$$\frac{\mathbf{\hat{u}} - \mathbf{u^n}}{\Delta t} = \nu\nabla^2\mathbf{u^n} - (\mathbf{u^n}\cdot\nabla)\,\mathbf{u^n} - \frac{1}{\rho}\nabla P^{\mathbf{n}-\frac{1}{2}}\,. \tag{2.14}$$

La similarité entre le champ de pression déjà connu $P^{\mathbf{n}-\frac{1}{2}}$ et celui, encore à déterminer, au pas de temps $\mathbf{n}+1$, permet d'obtenir une très bonne estimation $\mathbf{\hat{u}}$ du champ de vitesse $\mathbf{u^{n+1}}$, qui prend en compte les conditions aux limites imposées sur $\mathbf{u^{n+1}}$.

- On soustrait ensuite à cette estimation l'impact du champ de pression $P^{\mathbf{n}-\frac{1}{2}}$, c'est-à-dire

$$\frac{\mathbf{u^\star} - \mathbf{\hat{u}}}{\Delta t} = \frac{1}{\rho}\nabla P^{\mathbf{n}-\frac{1}{2}}\,. \tag{2.15}$$

- Il ne reste plus qu'à passer de $\mathbf{u}^\star$ à $\mathbf{u}^{\mathbf{n}+1}$ comme dans l'équation (2.13), en prenant en compte le gradient de pression au pas de temps $\mathbf{n}+1$, lequel est obtenu comme dans l'équation (2.12).

On obtient bien entendu l'équation originelle de bilan de quantité de mouvement en sommant les équations (2.14), (2.11) et (2.15), tout comme nous l'obtenons en sommant (2.10), (2.11) et (2.13) ; cependant, l'intégration de l'ancien gradient de pression à l'équation d'estimation du champ de vitesse au pas de temps suivant permet de réduire les erreurs liées au fractionnement de l'avancement temporel.

2.2.2 Discrétisation spatiale : méthode des volumes finis

La méthode des volumes finis est basée sur l'intégration des équations à résoudre sur des volumes de contrôle polyédriques. Intégrons à titre d'exemple l'équation de bilan de la quantité de mouvement d'un écoulement incompressible (1.15), à laquelle nous enlevons le terme de force de pesanteur, sur un volume de contrôle polyédrique Ω à n_f faces. Le théorème de Green-Ostrogradski permet de se ramener à un second membre sous la forme d'une intégrale surfacique, exprimant directement le fait que la variation de la quantité de mouvement dans le volume Ω est due au transport de cette quantité de mouvement vers ou depuis le voisinage de ce volume, donc au travers de sa frontière. On obtient

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} dV = \int_\Omega \left(\nu \nabla^2 \mathbf{u} - (\mathbf{u} \cdot \nabla)\, \mathbf{u} - \frac{1}{\rho} \nabla P \right) dV = \int_{\partial\Omega} \left(\nu \nabla \mathbf{u} - \mathbf{u} \otimes \mathbf{u} - \frac{P}{\rho} \bar{\mathbf{n}} \right) \cdot d\bar{S} \,. \tag{2.16}$$

L'intégrale sur $\partial\Omega$, la frontière du domaine, qui apparait dans le second membre peut alors être séparée sur les différentes faces planes du volume polyédrique considéré, soit

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} dV = \sum_{i=1}^{n_f} \int_{S_i} \left(\nu \nabla \mathbf{u} - \mathbf{u} \otimes \mathbf{u} - \frac{P}{\rho} \bar{\mathbf{n}} \right) \cdot d\bar{S} \,. \tag{2.17}$$

L'approche de la méthode des volumes finis consiste à considérer l'équation (2.17) en chaque polyèdre qui constitue le maillage du domaine physique, en d'autres termes une partition du domaine (ou d'une approximation de ce domaine si certaines de ses frontières sont courbes) en un ensemble de polyèdres, et à estimer la valeur des intégrales sur les éléments de surface à partir des valeurs des variables en certains points.

Dans la partie 2.2.2.1 ci-après, nous présentons l'utilisation qui est faite du maillage dans la méthode de volumes finis du solveur YALES2. Ensuite, nous donnons en partie 2.2.2.2 les schémas numériques pouvant y être utilisés pour l'estimation des intégrales sur les faces, montrons de quelle façon cette modalité de discrétisation peut être exploitée pour la déclaration des opérateurs différentiels, et détaillons la déclaration de l'opérateur laplacien discrétisé, c'est-à-dire la matrice du système linéaire correspondant à l'équation de Poisson pour la pression.

2.2.2.1 Maillage

Avant la simulation proprement dite, un maillage doit être créé, c'est-à-dire un partitionnement du domaine physique en polyèdres, ou en polygones pour un domaine en deux dimensions. L'usage de maillages non-structurés, c'est-à-dire dont les mailles ne sont pas uniquement des hexaèdres (ou des parallélogrammes, en deux dimensions), est facilement justifiable par la complexité de certaines géométries étudiées par la simulation numérique : injecteurs et chambres de combustion de moteurs aéronautiques ou automobiles, profils d'ailes d'avions ou de pales de turbines, et autres. Il présente d'autres avantages, notamment concernant la possibilité de raffinement local du maillage, qui ne seront pas détaillés ici. En contrepartie, la discrétisation spatiale nécessitée par la méthode des volumes finis est bien plus délicate sur un maillage non-structuré.

Plusieurs approches sont possibles concernant la localisation des variables d'état sur un maillage et les volumes de contrôle considérés. Dans YALES2, les choix suivants sont faits :

- Les variables sont colocalisées (*collocated*), c'est-à-dire que toutes les variables d'état sont déclarées sur les mêmes points, que l'on nomme les **noeuds** du maillage, et qui correspondent à l'ensemble des noeuds des polyèdres du maillage. Ces polyèdres sont nommés les **éléments** du maillage.

  Deux avantages conséquents motivent la colocation des variables : d'une part, elle simplifie considérablement la gestion des données ; d'autre part, le fait de conserver toutes les données aux noeuds nous permet de bénéficier de schémas isotropes. Le bénéfice tiré de ces avantages est particulièrement grand pour les calculs des opérateurs différentiels, pour lesquels des données sont à collecter dans un voisinage plus ou moins étendu de chaque noeud (voir partie 2.2.2.2).

  La colocation des variables peut provoquer des oscillations numériques de la solution, évoquées dans la littérature sous le nom de *checkerboard problem* ou problème de l'échiquier [60]. Pour appréhender ce problème, considérons un maillage cartésien en une dimension, c'est-à-dire une partition d'un axe en segments de même longueur Δx. Dans ce cas, la dérivée en espace d'un champ P est estimée au second ordre en un point i par

$$\left.\frac{\partial P}{\partial x}\right|_i = \frac{P_{i+1} - P_{i-1}}{2\Delta x}\,, \tag{2.18}$$

  ce qui provoque un découplage entre les valeurs du champ P sur les éléments d'indice pair et celles sur les éléments d'indice impair. On peut alors obtenir deux sous-champs, l'un en ne considérant que les valeurs sur les éléments pairs et l'autre en ne prenant en compte que les valeurs sur les éléments impairs, qui correspondront l'un à l'autre à une constante près. Le champ complet, en revanche, présentera des

oscillations de période $2\Delta x$. Ce problème peut apparaitre sur tout maillage structuré, mais également, globalement ou localement, sur un maillage non-structuré, et c'est à l'alternance possible des valeurs des champs sur le maillage qu'il doit son nom.

Une autre approche existante consiste alors à décaler les variables (on parle en anglais de *staggering* [30]), en déclarant les vitesses au centre des faces des éléments tandis que les autres variables d'état sont sur les noeuds : cette approche stabilise la résolution numérique, mais sa mise en oeuvre sur des maillages non-structurés peut conduire à certaines difficultés numériques liées à la possible déformation des mailles qui constituent le maillage.

- Les volumes de contrôle utilisés dans le cadre de la méthode des volumes finis ne sont pas directement les éléments dans YALES2, mais des volumes de contrôle créés autour des noeuds du maillage.

  En deux dimensions, chaque polygone est séparé en zones triangulaires. Chacun de ces triangles a pour sommets un noeud, le centre d'un des deux côtés contenant ce noeud, et le barycentre de l'élément. La figure 2.1 montre la décomposition des éléments utilisés par YALES2 en deux dimensions, à savoir les triangles et les quadrilatères, en sous-triangles. Le volume de contrôle associé à un noeud du maillage est alors l'ensemble des sous-triangles qui contiennent ce noeud, comme illustré par la figure 2.2 sur un exemple d'un maillage hybride constitué des deux types d'éléments.

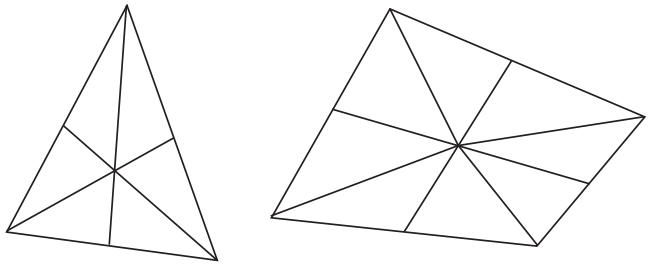

FIGURE 2.1 – *Décomposition des deux types d'élements de maillage en deux dimensions en sous-triangles.*

En trois dimensions, une scission similaire a lieu, qui se base sur la séparation des faces d'un élément telle que décrite ci-dessus. Nous développons ici l'exemple de la partition d'un tétraèdre. En liant les milieux des trois arêtes contenant un noeud de l'élément, au noeud en question d'une part, et au centre de masse de l'élément d'autre part, on obtient le sous-polygone associé au noeud dans cet élément, comme montré en figure 2.3. Les sous-polygones associés aux quatre noeuds forment naturellement une partition de l'élément, et le volume de contrôle d'un noeud est obtenu par réunion des sous-polygones associés à ce noeud dans tous les polyèdres contenant ce noeud.

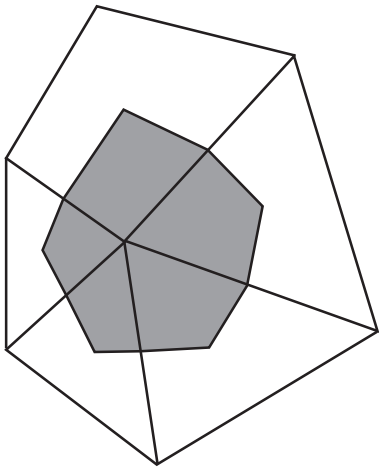

FIGURE 2.2 – *Création du volume de contrôle autour d'un noeud d'un maillage non-structuré hybride en deux dimensions.*

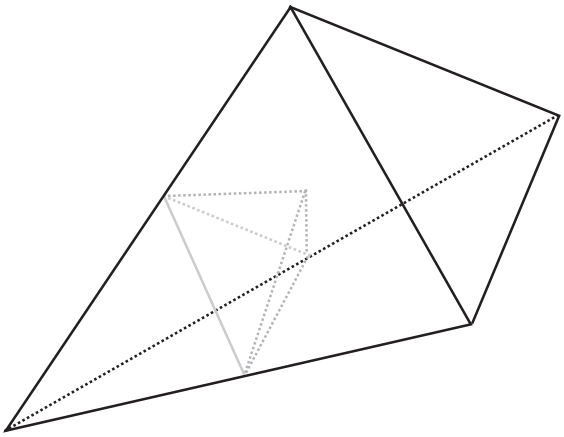

FIGURE 2.3 – *Sous-polygone associé au noeud d'un élément tétraédrique, délimité à l'intérieur de l'élément par les traits pointillés.*

> Les modalités d'estimation des intégrales surfaciques qui apparaissent dans l'équation (2.17) ne sont pas détaillées dans ce manuscrit. Le lecteur peut se référer à la thèse de Stijn Vantieghem [80] pour plus de détails sur ce sujet.

L'ensemble des volumes de contrôle forment une nouvelle partition du domaine physique, nommée **maillage dual** [7].

L'utilisation d'une méthode de volumes finis permet la conservation des quantités physiques à l'intérieur du domaine, puisque chaque quantité sortant d'un volume de contrôle par une de ses faces entrera exactement dans le volume de contrôle voisin. Des efforts particuliers doivent cependant être faits, notamment au niveau des conditions aux limites,

pour assurer la conservativité globale de la masse et de l'énergie ; ces efforts ne seront pas détaillés dans le cadre de cette thèse.

La création de volumes de contrôle "centrés" sur les noeuds permet une estimation plus juste des contributions locales à la quantité de mouvement, mais peut être biaisée par le fait que, si le maillage est trop distordu, le centre de gravité d'un volume de contrôle peut être décalé par rapport au noeud correspondant. En outre, comme cela a déjà été évoqué ci-avant, la colocalisation des variables sur le maillage peut entraîner des oscillations, notamment du champ de pression, liées au problème de l'échiquier. Une vigilance accrue doit donc être portée sur les schémas de discrétisation en espace.

2.2.2.2 Schémas en espace

Un schéma de discrétisation du second ordre et un schéma du quatrième ordre sont actuellement utilisés par le code YALES2.

- Dans le schéma du second ordre centré, l'opérateur de divergence en un noeud du maillage s'écrit comme une somme pondérée de contributions des noeuds voisins, sous la forme

$$\nabla \cdot \mathbf{u}|_i = \frac{1}{V_i} \sum_{j \leftrightarrow i} \frac{\mathbf{u}|_i + \mathbf{u}|_j}{2} \cdot \bar{\mathbf{n}}_{ij} \,, \tag{2.19}$$

  où V_i est le volume de contrôle du noeud i, et $\bar{\mathbf{n}}_{ij}$ est le vecteur normal à la portion du volume de contrôle du noeud i associée à l'arête entre les noeuds i et j. Un développement de Taylor à l'ordre 2 au voisinage du noeud i est à l'origine de cette expression, qui crée donc un schéma du second ordre. D'une façon similaire, le gradient d'un champ scalaire en un noeud du maillage s'écrit

$$\nabla P|_i = \frac{1}{V_i} \sum_{j \leftrightarrow i} \frac{P|_i + P|_j}{2} \bar{\mathbf{n}}_{ij} \,. \tag{2.20}$$

  L'application successive de ces opérateurs pour la création de l'opérateur laplacien fait ressurgir le problème dit de l'échiquier, ou *checkerboard problem*, évoqué en partie 2.2.2.1. De plus, un tel opérateur n'est pas compact, c'est-à-dire demande de relier chaque noeud, non plus seulement aux noeuds voisins, mais aux voisins de ses voisins, ce qui ajoute de nouvelles difficultés à la mise en oeuvre.

  La résolution de ce problème n'est pas abordée en détail dans ce manuscrit ; le lecteur peut, encore une fois, se référer à la thèse de Stijn Vantieghem [80] pour de plus amples détails. Nous n'en donnons ici que les grands principes pour le calcul du laplacien sur un volume de contrôle.

- En utilisant le théorème de flux-divergence, on peut écrire l'intégrale de $\nabla^2 g dV$ sur un volume de contrôle comme la somme des intégrales de $\nabla g \cdot dS$ sur les faces de ce volume.
- L'approximation de ∇g sur la normale d'une face est compliquée par le fait que, dans un maillage non-structuré, cette normale n'est en général pas parallèle à la droite qui relie les deux noeuds adjacents, à cause de la déformation des mailles. Il faut donc parvenir à approximer toutes les composantes du gradient pour apporter une correction à l'estimation du laplacien.
- L'approche de Zwart *et al.* [90] est considérée, qui consiste à décomposer le gradient en une composante parallèle à la paire de noeuds et une composante orthogonale. En revanche, là où l'estimation de la composante orthogonale par Zwart ne respecte pas la condition de compacité exigée, une deuxième utilisation du théorème de flux-divergence permet de calculer le gradient via l'utilisation des voisinages immédiats de chaque point uniquement.

Ces étapes successives permettent le calcul de coefficients w_{ij} grâce auxquels le calcul du laplacien d'un champ scalaire en un noeud du maillage s'écrit finalement sous la forme

$$\Delta P|_i = \frac{1}{V_i} \sum_{j \leftrightarrow i} w_{ij} \left(P|_j - P|_i \right) . \tag{2.21}$$

- Un schéma de discrétisation du quatrième ordre a également été développé. Pour cela, une déconvolution est nécessaire. En effet, on ne connait pas directement les valeurs des champs scalaires et vectoriels dans le code, mais leurs intégrales sur les volumes de contrôle ; pour développer un schéma numérique d'ordre élevé, il faut parvenir à retrouver les valeurs ponctuelles sur les noeuds, avec une certaine précision. Un développement en série de Taylor permet d'exprimer l'intégrale Φ_{Ω_i} d'une quantité Φ sur le volume de contrôle Ω_i associé au noeud i. On obtient alors une expression liant la quantité ponctuelle $\Phi(i)$ à la quantité Φ_{Ω_i} et aux dérivées successives de Φ.

  L'expression des flux sur les frontières du volume de contrôle Ω_i, obtenue par développement limité, fait alors intervenir les valeurs ponctuelles de Φ. Après moult simplifications, on obtient un schéma pour la divergence d'un champ vectoriel de la forme

$$\nabla \cdot \mathbf{u} = \frac{1}{V_i} \sum_{j \leftrightarrow i} \left(\frac{\mathbf{u}|_i + \mathbf{u}|_j}{2} + \Delta_{ik} \frac{\nabla \cdot \mathbf{u}|_i - \nabla \cdot \mathbf{u}|_j}{6} \right) \bar{\mathbf{n}}_{ij} , \tag{2.22}$$

  où Δ_{ik} est la longueur de l'arête reliant les noeuds i et k. La discrétisation au quatrième ordre des opérateurs gradient et laplacien s'obtient d'une façon similaire.

Le lecteur est invité à consulter le manuscrit de thèse de Stijn Vantieghem [80] pour le développement de ces constructions d'opérateurs discrétisés.

Pour le calcul de la divergence d'un champ vectoriel sur le maillage, la première étape consiste en un précalcul dans lequel sont utilisés les opérateurs au second ordre décrits ci-avant. Les valeurs ainsi déterminées sont utilisées pour le calcul de l'estimation au quatrième ordre.

Ce schéma de discrétisation est utilisé pour toutes les simulations décrites dans cette thèse.

Dans les deux cas, l'application d'un opérateur différentiel en un noeud du maillage se fait par une (ou plusieurs) boucle(s) effectuée(s) sur les paires de noeuds voisins le contenant, c'est-à-dire que le calcul global est fait en considérant toutes les paires de noeuds. Le choix de discrétisation des opérateurs différentiels dans YALES2 suit cette logique en considérant les contributions de chaque paire de noeuds à l'application d'un opérateur : dans le cas du laplacien, celui qui nous intéresse tout particulièrement dans le cadre de cette thèse, seuls les coefficients w_{ij} sont stockés. La matrice A correspondant à l'opérateur laplacien discrétisé n'est donc pas déclarée explicitement, et une boucle sur les paires de noeuds est effectuée pour calculer le laplacien d'un champ scalaire, c'est-à-dire pour multiplier un vecteur sur le maillage fin par A.

Cette conception incite naturellement à l'utilisation de méthodes de Krylov pour la résolution de tout système linéaire de matrice A, puisque cette famille de méthodes ne requiert aucune connaissance explicite de la matrice A, mais seulement la possibilité d'obtenir le produit Ax à partir d'un vecteur x quelconque.[2]

2.2.3 Schémas en temps

Bien que le choix des schémas de discrétisation temporels ne soit pas en lien direct avec la problématique principale traitée dans cette thèse, qui concerne l'équation de Poisson pour la pression, il est fondamental dans la conception d'un solveur LES. En effet, les méthodes de discrétisation employées se doivent d'avoir un ordre de précision élevé et de limiter autant que possible la diffusion numérique du champ de vitesse, laquelle est déjà induite par l'ajout de la viscosité turbulente μ_t introduite en sous-section 2.1.2.

Trois schémas numériques sont utilisés dans YALES2 : le schéma implicite classique de Crank-Nicolson, qui est du second ordre, le schéma de Runge-Kutta explicite *low-storage* au quatrième ordre, et le schéma TFV4A développé par Matthias Kraushaar *et al.* dans le cadre du développement de YALES2.

[2]Une implication directe dans le cadre de notre travail concerne l'application de la méthode de déflation, qui a requis la création de l'opérateur $\hat{A}$, à savoir l'opérateur laplacien projeté sur un sous-espace de l'espace de travail imposé par le maillage fin, selon le même paradigme. Cet opérateur a donc été créé de façon à être appliqué par une boucle sur les paires de groupes de volumes de contrôle, dans le cas de la déflation géométrique standard d'une part (voir sous-section 5.2.3), dans celui de nouvelles méthodes de déflation dites "planes" et "quadriques" d'autre part (voir sous-section 6.2.3).

2.2.3.1 Méthode de Crank-Nicholson

Le schéma numérique utilisé de façon implicite en sous-section précédente, dans l'équation de bilan du quantité de mouvement (2.14), est un schéma explicite en temps, c'est-à-dire que l'avancement en temps est fait grâce à un second terme dont les valeurs sont toutes déjà calculées sur le pas de temps précédent. On y calcule $\hat{\mathbf{u}}$ et $\mathbf{u}^\star$ en fonction de $\mathbf{u}^\mathbf{n}$ et $P^\mathbf{n}$: plus précisément, c'est le schéma numérique d'Euler qui y est utilisé, dans lequel l'avancement en temps de la variable $\mathbf{u}$ telle que

$$\frac{\partial \mathbf{u}}{\partial t} = f\left(\mathbf{u}, P\right) \tag{2.23}$$

est effectué grâce à l'approximation

$$\frac{\mathbf{u}^{\mathbf{n}+1} - \mathbf{u}^\mathbf{n}}{\Delta t} = f\left(\mathbf{u}^\mathbf{n}, P^\mathbf{n}\right) . \tag{2.24}$$

Un autre choix consiste à utiliser des méthodes implicites, dans lesquelles l'avancement en temps est fait à la fois grâce à des quantités connues, c'est-à-dire calculées en $t_\mathbf{n}$, et inconnues, c'est-à-dire en $t_{\mathbf{n}+1}$. De telles méthodes sont plus coûteuses à chaque pas de temps, à cause du besoin d'estimer certaines quantités, mais permettent de s'affranchir de certaines conditions de stabilité. L'augmentation du coût de calcul à chaque pas de temps peut alors être compensé par l'augmentation du pas de temps Δt.

Le schéma implicite intégré dans YALES2 est la méthode de Crank-Nicolson [32], un cas particulier des schémas Θ, dont une forme générale est donnée par

$$\frac{\mathbf{u}^{\mathbf{n}+1} - \mathbf{u}^\mathbf{n}}{\Delta t} = f\left(\Theta \mathbf{u}^{\mathbf{n}+1} + (1-\Theta)\,\mathbf{u}^\mathbf{n}, P^\mathbf{n}\right) . \tag{2.25}$$

Le choix de la méthode de Crank-Nicolson est $\Theta = \frac{1}{2}$, ce qui rend la méthode du second ordre en temps. De plus, cette méthode est inconditionnellement stable. (Notons que cette stabilité est également garantie par le choix de $\Theta = 1$, qui donne la méthode d'Euler implicite.)

2.2.3.2 Méthode de Runge-Kutta

La méthode de Runge-Kutta [44, 65] au quatrième ordre, désignée par l'acronyme RK4, consiste en une estimation itérative du champ $\mathbf{u}^{\mathbf{n}+1}$ à partir $\mathbf{u}^\mathbf{n}$. On utilise ici sa version explicite *low-storage*, c'est-à-dire nécessitant un espace mémoire limité [88]. Pour l'équation (2.23), quatre calculs sont effectués successivement :

$$\begin{cases} \mathbf{u}^{(1)} = \mathbf{u}^{\mathbf{n}} + \dfrac{1}{4}\Delta t f\left(\mathbf{u}^{\mathbf{n}}, P^{\mathbf{n}}\right) , \\ \mathbf{u}^{(2)} = \mathbf{u}^{\mathbf{n}} + \dfrac{1}{3}\Delta t f\left(\mathbf{u}^{(1)}, P^{\mathbf{n}}\right) , \\ \mathbf{u}^{(3)} = \mathbf{u}^{\mathbf{n}} + \dfrac{1}{2}\Delta t f\left(\mathbf{u}^{(2)}, P^{\mathbf{n}}\right) , \\ \mathbf{u}^{\mathbf{n+1}} = \mathbf{u}^{\mathbf{n}} + \Delta t f\left(\mathbf{u}^{(3)}, P^{\mathbf{n}}\right) . \end{cases} \tag{2.26}$$

2.2.3.3 Méthode TFV4A

Les schémas numériques présentés ci-avant comportent des inconvénients. En particulier, le schéma de Crank-Nicolson est particulièrement dispersif et limité au second ordre. De même, le schéma RK4 est couramment utilisé en CFD à cause de sa faible diffusion, mais une viscosité artificielle μ_{artif} est souvent ajoutée, en plus de la viscosité turbulente μ_t, pour diffuser les artefacts numériques, c'est-à-dire les signaux de très haute fréquence qui apparaissent à cause de la dispersion numérique.

Des recherches ont donc été menées par Matthias Kraushaar *et al.* [42], dans le cadre d'une collaboration entre le CORIA et le CERFACS, pour développer un schéma numérique basé sur les méthodes de Runge-Kutta, mais capable de réduire autant que possible la dispersion sur les hautes fréquences tout en conservant une diffusion presque nulle sur le reste du spectre. Ces développements se basent sur l'introduction d'une diffusion numérique, ajustable en fonction des besoins, pour compenser les oscillations à haute fréquence qui apparaissent à cause du choix de discrétisation spatiale.

La première étape du développement de cette méthode consiste à combiner deux à deux les étapes de la méthode RK4 classique de l'équation (2.26) pour obtenir

$$\begin{cases} \mathbf{u}^{(2)} = \mathbf{u}^{\mathbf{n}} + \dfrac{1}{3}\Delta t f\left(\mathbf{u}^{\mathbf{n}} + \dfrac{1}{4}\Delta t f\left(\mathbf{u}^{\mathbf{n}}, P^{\mathbf{n}}\right), P^{\mathbf{n}}\right) , \\ \mathbf{u}^{\mathbf{n+1}} = \mathbf{u}^{\mathbf{n}} + \Delta t f\left(\mathbf{u}^{\mathbf{n}} + \dfrac{1}{2}\Delta t f\left(\mathbf{u}^{(2)}, P^{\mathbf{n}}\right), P^{\mathbf{n}}\right) . \end{cases} \tag{2.27}$$

La connaissance de f nous permet de réécrire ces équations de manière détaillée, ce qui fait apparaître une expression similaire à celle du schéma stable et dissipatif TTG4A[3] présenté entre autres dans [46]. La seule différence se situe au niveau de la modalité de calcul des termes de second ordre en temps. Une combinaison convexe des résultats de ces deux calculs permet de créer une méthode hybride entre RK4 et TTG4A qui, pour un bon choix du coefficient de la combinaison convexe, améliore significativement le solveur RK4. Un article concernant cette famille de méthodes "hybrides" et les résultats obtenus avec ces méthodes est en préparation.

[3] Two-step Taylor-Galerkin, quatrième ordre, utilisé entre autres dans le code AVBP du CERFACS.

Ce schéma numérique a été utilisé pour toutes les simulations présentées dans cette thèse.

Chapitre 3

Algèbre linéaire appliquée à la simulation des écoulements turbulents

Comme évoqué dans les parties 1.3.2 et 2.2.2.1, la discrétisation d'équations aux dérivées partielles, notamment celles liées aux écoulements incompressibles qui sont plus particulièrement l'objet de la présente thèse, par le solveur YALES2 donne naissance à des systèmes de n équations linéaires, où n est le nombre de noeuds du maillage considéré, l'ordre de grandeur de n pouvant atteindre plusieurs dizaines de milliards pour les applications actuelles.

La résolution de ces systèmes de grande taille se fait grâce à des algorithmes itératifs dont Saad, en 2000, fournit un précieux panorama global dans son livre "Iterative Methods for Sparse Linear Systems" [66], ainsi que Richard Barrett *et al.* avant lui dans "Templates for the Solution of Linear Systems : Building Blocks for Iterative Methods" [6]. Dans la partie qui suit, après quelques rappels d'algèbre linéaire et une sommaire introduction aux principales familles de méthodes de résolution de systèmes linéaires, nous nous intéresserons à la méthode des Gradients Conjugués (CG) classiquement utilisée pour la simulation de phénomènes physiques [3, 56].

3.1 Quelques rappels

Dans cette section, nous rappelons les bases de l'algèbre linéaire et présentons les deux grandes familles de méthodes de résolution de systèmes linéaires, à savoir les méthodes directes et les méthodes itératives. Nous introduisons également la notion de préconditionnement d'un système linéaire.

3.1.1 Bases de l'algèbre linéaire : produits scalaires, normes, conditionnement

Dans ce qui suit, on souhaite résoudre le système linéaire $Ax = b$ avec $A \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$. La matrice A est semi-définie positive, c'est-à-dire $\forall x \in \mathbb{R}^n, x^T Ax \geq 0$.

La résolution du système linéaire $Ax = b$ est alors équivalente à la minimisation de la fonction de coût associée

$$\forall x \in \mathbb{R}^n,\ C(x) = \frac{1}{2}x^T Ax - b^T x\,. \tag{3.1}$$

On appelle **produit scalaire** l'opération $\langle \cdot, \cdot \rangle$ définie par

$$\forall x, y \in \mathbb{R}^n, \langle x, y\rangle = x^T y\,. \tag{3.2}$$

Avec A semi-définie positive, on peut également considérer le **A-produit scalaire**, qui est l'opération $\langle \cdot, \cdot \rangle_A$ définie par

$$\forall x, y \in \mathbb{R}^n, \langle x, y\rangle_A = x^T Ay\,. \tag{3.3}$$

Deux vecteurs x et y sont alors **A-orthogonaux** si et seulement si $\langle x, y\rangle_A = 0$.

La norme $\| \cdot \|_A$ induite par ce produit scalaire est définie par

$$\forall x \in \mathbb{R}^n,\ \|x\|_A = \sqrt{x^T Ax} \tag{3.4}$$

et appelée **A-norme** ou encore **norme d'énergie.** Les normes usuelles sont, quant à elles, définies comme suit :

$$\forall x \in \mathbb{R}^n,\ \|x\|_1 = \sum_{k=1}^{n} |x_k|\,; \tag{3.5}$$

$$\forall x \in \mathbb{R}^n,\ \|x\|_2 = \sqrt{\langle x, x\rangle} = \left(\sum_{k=1}^{n} x_k^2\right)^{\frac{1}{2}}\ ; \tag{3.6}$$

$$\forall x \in \mathbb{R}^n,\ \|x\|_\infty = \max_{k\in\{1;..;n\}} |x_k|\,, \tag{3.7}$$

et la norme 2 est nommée **norme euclidienne.** Ce sont en fait des cas particuliers de la p-norme vectorielle usuelle

$$\forall x \in \mathbb{R}^n,\ \|x\|_p = \left(\sum_{k=1}^{n} x_k^p\right)^{\frac{1}{p}}\,. \tag{3.8}$$

On démontre facilement l'équivalence de deux p-normes quelconques ; en particulier, l'équivalence entre les normes 1, 2 et ∞ s'écrit

$$\forall x \in \mathbb{R}^n,\ \|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty \text{ et } \|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty\,. \tag{3.9}$$

A partir de toute norme vectorielle $\|\cdot\|$, on construit une **norme matricielle induite**, définie par

$$\forall A \in M_{n\times n}, \|A\| = \max_{\|x\|\neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\| \text{ avec } x \in \mathbb{R}^n\,. \tag{3.10}$$

La norme d'une matrice A subordonnée à une norme vectorielle donnée est donc, au final, la plus grande valeur par laquelle la norme d'un vecteur peut être multipliée lorsque ce vecteur est multiplié par A. Non seulement il est logique qu'elle dépende de la norme considérée, mais on peut déjà pressentir un lien avec les valeurs propres de A. Précisons néanmoins qu'il existe également des normes matricielles non subordonnées à une quelconque norme vectorielle : c'est notamment le cas de la **norme de Frobenius** définie par

$$\forall A \in M_{n\times n}, \|A\|_F = \sqrt{\mathrm{tr}(A^T A)} \tag{3.11}$$

où $\mathrm{tr}(M)$ est la trace de la matrice M, c'est-à-dire la somme de ses éléments diagonaux. Les normes matricielles usuelles sont elles aussi équivalentes deux à deux, et on a en particulier pour toute matrice A carrée

$$\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{n}\|A\|_\infty \text{ et } \frac{1}{\sqrt{n}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1\,. \tag{3.12}$$

Le **nombre de conditionnement** d'une matrice, que nous pourrons également relier au spectre de la matrice A, est alors défini par la formule

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|\,. \tag{3.13}$$

L'importance du conditionnement d'une matrice réside dans le fait qu'il représente l'influence de légères perturbations des données du système $Ax = b$ sur sa solution. Plus précisément, lorsque l'on ajoute au vecteur b une erreur Δb, ou si une modification ΔA est apportée à la matrice A, on obtient respectivement les majorations suivantes sur l'erreur relative de x :

$$\begin{aligned} \frac{\|\Delta x\|}{\|x\|} &= \kappa(A)\frac{\|\Delta b\|}{\|b\|}\,; \\ \frac{\|\Delta x\|}{\|x + \Delta x\|} &= \kappa(A)\frac{\|\Delta A\|}{\|A\|}\,. \end{aligned} \tag{3.14}$$

La démonstration en est aisée. En cas de perturbation de b, on a

$$\begin{cases} Ax = b \\ \Delta x = A^{-1}\Delta b \end{cases} \tag{3.15}$$

d'où

$$\begin{cases} \dfrac{1}{\|x\|} \leq \dfrac{\|A\|}{\|b\|} \\ \|\Delta x\| \leq \|A^{-1}\| \cdot \|\Delta b\| \end{cases} . \tag{3.16}$$

Le produit de ces deux inéquations donne le résultat souhaité. La preuve du deuxième point se fait d'une façon similaire.

On considère souvent le conditionnement associé à la 2-norme matricielle, elle-même induite par la norme euclidienne sur $\mathbb{R}^n$, et on note ce conditionnement $\kappa_2(A)$. En pratique, lors de la résolution d'un système linéaire par un algorithme itératif (voir sous-section 3.1.4), plus le conditionnement de la matrice est important, plus l'erreur sur la solution calculée peut être importante pour une même condition d'arrêt sur le résidu. La valeur du conditionnement de A aura alors son importance, et nous chercherons à la réduire par des méthodes dites de préconditionnement (voir sous-section 3.1.5).

3.1.2 Valeurs et vecteurs propres

La matrice A est celle d'un endomorphisme de $\mathbb{R}^n$, noté ci-après f_A. Comme le comportement de cette fonction peut sembler erratique lorsque nous l'étudions dans la base canonique, nous pouvons être tentés de chercher une base de $\mathbb{R}^n$ dans laquelle la i-ième composante de $f_A(x)$, pour tout x dans $\mathbb{R}^n$, ne dépendrait que de la i-ième composante de x ; ce changement de base conservant la linéarité de la fonction, nous obtiendrons dans cette nouvelle base

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{pmatrix} \Rightarrow f_A(x) = \begin{pmatrix} \lambda_1 x_1 \\ \lambda_2 x_2 \\ \dots \\ \lambda_{n-1} x_{n-1} \\ \lambda_n x_n \end{pmatrix} .$$

Autrement dit, en appelant P la matrice des vecteurs de la nouvelle base exprimés dans la base canonique, la matrice $A' = P^{-1}AP$ de la fonction exprimée dans cette nouvelle base sera diagonale. Les vecteurs de cette base sont précisément les **vecteurs propres** de la matrice A, que l'on définit ainsi : le vecteur non nul u est un vecteur propre de A si et seulement si

$$\exists \lambda \in \mathbb{R}, Au = \lambda u . \tag{3.17}$$

La valeur λ_i telle que $Au_i = \lambda_i u_i$ pour un vecteur propre u_i donné est la **valeur propre** associée à ce vecteur propre. En pratique, la recherche des valeurs et vecteurs propres se fait dans l'autre sens : puisque toute valeur propre λ est telle que, pour "au moins" un vecteur u non nul donné, $Au = \lambda u$ (bien entendu, tout vecteur de la forme αu

avec $\alpha \in \mathbb{R}^*$ sera également solution de cette équation), on obtient $(A - \lambda I)u = 0_{\mathbb{R}^n}$. Le fait que le système homogène $(A - \lambda I)x = 0_{\mathbb{R}^n}$ admette une autre solution que la solution triviale $x = 0_{\mathbb{R}^n}$ implique que $A - \lambda I$ est non inversible, et donc que son déterminant est nul.

L'équation à une inconnue λ, et de degré exactement égal à n, $\det(A - \lambda I) = 0$, admet comme solutions les valeurs propres de la matrice A, qui seront au plus au nombre de n. Le polynôme $\det(A - \lambda I)$ est nommé **polynôme caractéristique** de l'endomorphisme f_A. L'ordre de multiplicité d'une racine du polynôme caractéristique est nommé **ordre de multiplicité algébrique** de la valeur propre ; l'ensemble des vecteurs propres associés à une valeur propre est un sous-espace vectoriel de $\mathbb{R}^n$, dont la dimension est l'ordre de multiplicité de la valeur propre, nommé le **sous-espace propre** associé à cette valeur propre.

La plus grande valeur propre en valeur absolue d'une matrice est son **rayon spectral**. On peut facilement montrer que le rayon spectral d'une matrice est un minorant de la norme de cette matrice, et ce quelle que soit la norme matricielle choisie, qu'elle soit subordonnée à une norme vectorielle ou non.

On peut également relier le conditionnement usuel κ_2 aux valeurs propres de la matrice pour une matrice symétrique à coefficients réels (comme celles que nous obtenons par discrétisation sur un maillage quelconque de l'équation de Poisson pour la pression pour un écoulement incompressible, voir sous-section 2.2.2.2), car sous cette hypothèse, on a

$$\kappa_2(A) = \left| \frac{\lambda_M(A)}{\lambda_m(A)} \right| , \tag{3.18}$$

où $\lambda_M(A)$ et $\lambda_m(A)$ sont respectivement la plus grande valeur propre et la plus petite valeur propre de A.

3.1.3 Méthodes directes

Les méthodes directes utilisées pour la résolution d'un système linéaire $Ax = b$, avec A matrice carrée non singulière (autrement dit, l'équation $Ax = b$ admet une et une seule solution), se basent sur le fait qu'un système matriciel est simple à résoudre directement si sa matrice est diagonale, triangulaire ou orthogonale ; si l'on peut écrire A comme un produit de telles matrices, la résolution du système par étapes deviendra simple.

C'est le principe de la décomposition LU, probablement la méthode directe la plus utilisée. Elle consiste à écrire $A = LU$, avec L matrice triangulaire inférieure à diagonale unitaire et U matrice triangulaire supérieure. Pour une matrice non singulière, cette décomposition est unique. La résolution de $Ax = b$, soit $LUx = b$, se fait ensuite en deux étapes : déterminer y tel que $Ly = b$, puis x tel que $Ux = y$. Cette méthode est basée sur l'élimination de Gauss-Jordan, aussi nommée pivot de Gauss, et qui tient son nom du mathématicien Carl Friedrich Gauss et du géodésien Wilhelm Jordan, tous deux ayant vécu au XIXme siècle.

La décomposition LU, tout comme son équivalent de la forme LL^T pour les matrices symétriques développé par André-Louis Cholesky à l'aube du XXme siècle, a un coût de calcul en n^3, c'est-à-dire que le nombre d'opérations requises pour résoudre $Ax = b$ est à peu de choses près proportionnel au cube de la taille du problème. Les méthodes directes sont donc théoriquement mal adaptées à la résolution de systèmes linéaires de grande taille. A cet inconvénient vient s'ajouter une contrainte pratique, qui est celle de la mémoire : l'application d'une décomposition LU nécessite de stocker n^2 données, à savoir les composantes non-nulles de L et U (à l'exclusion de la diagonale de L) ; en outre, même si la matrice A est creuse, ce qui est typiquement le cas de la matrice d'un opérateur différentiel discrétisé, les matrices L et U ne le seront très probablement pas. Bien que cette sous-section ne prétende à aucune exhaustivité, nous mentionnerons quand même que les problèmes de mémoire et de coûts élevés sont communs à toutes les méthodes directes standard.

Les méthodes directes peuvent cependant être adaptées à des problèmes de grande taille grâce aux méthodes dites frontales. En 1970, Irons [31] crée un programme qui, par sélections successives de sous-ensembles de l'ensemble des inconnues à déterminer, résout une suite de sous-problèmes linéaires permettant de connaitre la solution globale du système. La matrice du système original n'est jamais créée explicitement ; seules des matrices de taille modérée sont créées et détruites au cours de l'exécution, sans jamais coexister en mémoire. Cette méthode est étendue à la résolution en parallèle [17, 49] de façon à créer une méthode multifrontale, dans laquelle un ensemble de sous-problèmes seront résolus en parallèle. Des bibliothèques comme MUMPS ou UMFPACK proposent des méthodes multifrontales de résolution de problèmes à matrices creuses.

Cependant, l'utilisation de méthodes itératives reste plus simple dans le contexte de la CFD, grâce à la décomposition du domaine de simulation en sous-domaines affectés aux différents processeurs, et ces méthodes doivent se baser sur des multiplications successives par A ou des quelconques sous-matrices de A facilement isolables. Les besoins en mémoire de telles méthodes resteront limités, puisqu'aucune matrice pleine de dimension n ne devra être déclarée, et leur développement devra se faire de façon à ce que le coût de chaque itération soit négligeable devant n^3 pour de grandes valeurs de n ; cette condition est respectée par les méthodes itératives courantes dont les coûts sont ordinairement en n^2 ou en $n \log n$.

3.1.4 Méthodes itératives

L'idée de base de toute méthode itérative est de trouver un moyen, en partant d'un vecteur x_0 si possible quelconque, de le modifier itérativement, chaque itération transformant le vecteur x_k en un vecteur x_{k+1} de façon à ce que la suite $\{x_i\}_{i\geq 0}$ converge vers la solution x du système. En pratique, plusieurs questions se posent : d'une part, y a-t-il des matrices A pour lesquelles la méthode ne converge pas ? D'autre part, pour une matrice A *ad hoc*, la méthode converge-t-elle à partir de toute solution initiale x_0 ? Enfin, quelle est la vitesse

de convergence de la méthode ? La dernière de ces questions présente un intérêt pratique évident, puisqu'un solveur numérique considèrera avoir atteint une solution satisfaisante en fonction d'une certaine condition d'arrêt, que l'on espère remplir après le plus petit nombre possible d'itérations.

Pour toute méthode itérative, on nomme **erreur** à l'étape k le vecteur $e_k = x_k - x$. Comme l'erreur est inconnue au moment de la résolution, on se sert en pratique du **résidu** $r_k = b - Ax_k$, que l'on peut également écrire $r_k = -Ae_k$, pour établir une condition d'arrêt.

Un critère fréquemment utilisé consiste à exiger qu'une certaine norme du résidu soit diminuée d'un certain nombre d'ordres de grandeur ; si $x_0 = 0$, alors $r_0 = b$, et on considère que x_k est une approximation suffisamment précise de x si et seulement si

$$\|r_k\| < \gamma \|b\| \,. \tag{3.19}$$

La norme du résidu est alors diminuée de $-\log_{10}(\gamma)$ ordres de grandeur par rapport au résidu initial, ce pourquoi γ se voit communément attribuer une valeur qui est une puissance entière négative de 10. La norme utilisée a relativement peu d'importance tant qu'il s'agit d'une p-norme usuelle, dans le sens où toutes ces normes sont équivalentes ; pour des raisons de commodité, on préférera utiliser la norme infinie, qui est la plus grande composante en valeur absolue du vecteur.

Les méthodes itératives peuvent se séparer en deux catégories : les méthodes stationnaires et les méthodes instationnaires[1]. Dans ce qui suit, nous dresserons un panorama partiel de ces deux familles de méthodes. Le lecteur peut se référer au livre de Saad "Iterative Methods for Sparse Linear Systems" [66] pour des compléments d'information.

3.1.4.1 Méthodes itératives stationnaires

Dans une méthode dite stationnaire, l'itération peut s'écrire sous la forme

$$x_{k+1} = Bx_k + c \,, \tag{3.20}$$

avec B et c indépendants de k. Nous allons présenter ici les méthodes de Jacobi, de Gauss-Siedel, de sur-relaxation et de sur-relaxation successive, qui sont les principales méthodes itératives stationnaires.

La décomposition de la matrice A sous la forme d'une somme de matrices (on parle de "splitting" en anglais, pour différencier de "decomposition" qui se rapporte à la décomposition en un produit) permet de créer les méthodes de Jacobi et de Gauss-Seidel. En écrivant $A = M - N$, on transforme $Ax = b$ en $Mx = Nx + b$, équation qui nous donne l'itération $Mx_{k+1} = Nx_k + b$ soit

[1] L'utilisation des termes "stationnaire" et "instationnaire" est ici à différencier clairement de leurs significations dans le cadre physique. Une méthode itérative est "stationnaire" si l'expression de x_{k+1} en fonction de x_k est la même quelle que soit la valeur de k ; dans le cas contraire, elle est "instationnaire".

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b\,. \tag{3.21}$$

On notera dans ce qui suit $B = M^{-1}N$. On souhaite alors que M soit inversible, si possible "facilement", c'est-à-dire que la résolution numérique de tout système linéaire ayant M pour matrice puisse être effectuée directement et sans encombre majeur. Les méthodes de Jacobi et de Gauss-Seidel choisissent ainsi respectivement $M = D$ et $M = D + L$, où $D = \mathrm{diag}(A)$ et L est la partie triangulaire inférieure de A définie par

$$l_i^j = \begin{cases} a_i^j \text{ si } i > j, \\ 0 \text{ sinon.} \end{cases} \tag{3.22}$$

Pour la méthode de Jacobi, le calcul de x_{k+1} est direct ; pour la méthode de Gauss-Seidel, on calcule tout d'abord le second membre $b_k = Nx_k + b$, puis l'équation $Mx_{k+1} = b_k$ est résolue par descente.

On sait que, si une méthode de ce type converge, ce sera vers un point fixe de la suite $\{x_i\}$, c'est-à-dire vers une solution de $Ax = b$. Dans les deux cas, et même pour tout autre choix de matrices M et N telles que $A = M - N$, la méthode converge si et seulement si $\lim_{k\to\infty} B^k = 0_n$, où 0_n est la matrice carrée de rang n nulle.

Cette condition est équivalente à $\rho(B) < 1$, c'est-à-dire que toutes les valeurs propres de B sont dans $]-1;1[$. Cette condition assure que, d'une itération à la suivante, une telle méthode diminuera toutes les composantes de l'erreur exprimée dans la base des vecteurs propres de A ; cette propriété vaut à ces méthodes la dénomination de *smoothers*, ou méthodes de lissage.

La convergence est donc assurée pour toute solution initiale x_0 si A est à diagonale strictement dominante ou à diagonale dominante irréductible[2]. Dans le cas de la méthode de Gauss-Seidel, la convergence est également assurée pour toute matrice A symétrique définie positive.

Mentionnons également que la vitesse de convergence de ces méthodes est directement liée au rayon spectral de B. En effet, en partant de l'équation (3.21), et en considérant que la solution x du système $Ax = b$ vérifie

$$x = Bx + M^{-1}b\,, \tag{3.23}$$

alors

$$x_{k+1} - x = B\,(x_k - x) \tag{3.24}$$

d'où

$$x_k - x = B^k\,(x_0 - x)\ . \tag{3.25}$$

[2]Une matrice est à diagonale dominante irréductible si elle est à diagonale dominante et si elle est irréductible, c'est-à-dire non similaire par permutation à une matrice triangulaire par blocs.

On peut donc estimer le nombre d'itérations nécessaires pour réduire l'erreur d'un certain facteur, en écrivant

$$-\ln \frac{\|x_k - x\|_2}{\|x_0 - x\|_2} \leq -k \ln(\rho(B)) \ . \tag{3.26}$$

On peut relaxer ces méthodes en remplaçant l'itération $x_{k+1} = M^{-1}Nx_k + M^{-1}b$ par

$$x_{k+1} = \omega \left(M^{-1}Nx_k + M^{-1}b\right) + (1-\omega)x_k \tag{3.27}$$

où ω est un paramètre non nul. On parle alors de méthode de sur-relaxation (JOR) et de sur-relaxation successive (SOR), pour les versions relaxées des méthodes de Jacobi et Gauss-Seidel successivement. Pour $\omega = 1$, on retombe bien entendu sur les méthodes de splitting, mais l'intérêt ici vient du fait que la matrice de l'itération, c'est-à-dire la matrice par laquelle x_k est multipliée pour l'obtention de x_{k+1}, n'est plus $M^{-1}N$ mais $\omega M^{-1}N + (1-\omega)I_n$, avec I_n la matrice identité sur $\mathbb{R}^n$. Alors, comme $\rho\left(\omega M^{-1}N + (1-\omega)I_n\right)$ est une fonction continue de ω, on peut trouver la valeur du paramètre ω qui rend ce rayon spectral minimal, et forcément inférieur à celui de $M^{-1}N$; la méthode relaxée convergera donc plus vite que la méthode originale.

3.1.4.2 Méthodes itératives instationnaires

Une méthode itérative est dite instationnaire lorsque la récurrence qu'elle utilise diffère d'une itération à la suivante. De manière générale, l'itération peut s'écrire sous la forme

$$x_{k+1} = B_k x_k + c_k \, , \tag{3.28}$$

c'est-à-dire une forme similaire à l'équation 3.20, mais avec la matrice B et le vecteur c qui varient au cours du temps. En pratique, on applique généralement une correction à x_k en lui ajoutant un c_k tel que, selon un certain critère, x_{k+1} est "plus proche" de x que ne l'était x_k, ce qui ne revient pas toujours à choisir $B_k = I_n$ dans l'équation (3.28) ; en effet, c_k peut lui-même dépendre de x_k, comme dans les méthodes de descente de gradient.

Les méthodes de descente suivent à chaque itération une direction de descente p_k qui permettra de se rapprocher de la solution. Pour cela, elles s'intéressent à une fonctionnelle $\mathcal{F}$, fonction de $\mathbb{R}^n$ vers $\mathbb{R}$ qui atteint son minimum global en x, solution du système $Ax = b$. Le choix de la direction de descente peut alors se faire selon le gradient de $\mathcal{F}$, qui est la direction de la pente la plus raide autour d'un point, afin de descendre aussi vite que possible vers la solution ; une fois cette direction p_k choisie, on veut déterminer un α_k particulier, de façon à ce que $\mathcal{F}\left(x_k + \alpha_k p_k\right)$ soit le plus petit possible.

En pratique, on utilise fréquemment la fonctionnelle quadratique de coût définie par l'équation (3.1). Le gradient de cette fonctionnelle en un point x_k est

$$\vec{\Delta} C\left(x_k\right) = Ax_k - b = -r_k \, . \tag{3.29}$$

La descente se fait ainsi dans la direction du résidu, ce qui revient à une itération du type

$$x_{k+1} = x_k + \alpha_k r_k \,. \tag{3.30}$$

Notons que, si on choisit un pas fixe, c'est-à-dire si $\forall k \in \mathbb{N}, \alpha_k = \alpha$, on obtient

$$x_{k+1} = (I - \alpha A)\, x_k + \alpha b \,. \tag{3.31}$$

Ce choix, qui est celui de l'itération de Richardson, fait de cette méthode une méthode itérative stationnaire. Dans le cas où toutes les valeurs propres de A sont positives, la valeur du paramètre α est alors choisie pour minimiser la norme de $I - \alpha A$, ce qui donne

$$\alpha = \frac{2}{\lambda_M(A) + \lambda_m(A)} \,, \tag{3.32}$$

avec $\lambda_M(A)$ et $\lambda_m(A)$ la plus grande valeur propre et la plus petite valeur propre de A, respectivement. Dans cette partie, on considèrera donc un pas α_k variable selon une contrainte déterminée. Par exemple, choisir α_k de façon à ce que $C\,(x_k + \alpha_k r_k)$ soit minimal mène à chercher

$$\min_{\alpha_k \in \mathbb{R}} \left[\frac{1}{2} \left(x_k + \alpha_k r_k\right)^T A \left(x_k + \alpha_k r_k\right) - b^T \left(x_k + \alpha_k r_k\right) \right] \,, \tag{3.33}$$

ce qui mène à

$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k} \,. \tag{3.34}$$

L'itération ainsi décrite est celle de la méthode de descente de gradients la plus simple. Le choix de paramètres qui y est fait assure à chaque itération la minimisation locale de la fonctionnelle C dans la direction du résidu, mais rien n'empêche que cette direction doive être empruntée de nouveau lors de l'algorithme. La méthode des Gradients Conjugués, présentée en section 3.2, peut être vue comme une amélioration de cette méthode, qui modifie les directions de descente successives de façon à ce que la convergence en n itérations soit assurée.

3.1.5 Préconditionnement

Le préconditionnement d'un système linéaire est une opération visant à améliorer le conditionnement du système. On utilise pour cela le fait que, pour toute matrice M de rang n inversible, le système linéaire

$$M^{-1} A x = M^{-1} b \tag{3.35}$$

admet la même solution que le système $Ax = b$. Préconditionner un système revient donc à chercher une matrice M qui soit "la plus proche possible de A". Cette définition est bien entendu ambiguë, mais elle s'appuie sur le fait que le choix $M = A$ serait de toute évidence celui qui rendrait le système (3.35) le plus simple à résoudre. On souhaite donc déterminer une matrice M "proche" de A, mais facile à inverser ou, tout du moins, pour laquelle la résolution d'une équation du type $My = c$ serait relativement aisée. Cette notion de proximité peut être affinée, en exigeant par exemple que le spectre de M soit proche de celui de A, ou qu'une certaine norme de $M^{-1}A - I$ soit aussi petite que possible.

Les méthodes itératives stationnaires présentées en partie 3.1.4.1 fournissent des exemples de matrices de préconditionnement, dans le sens où le splitting $A = M - N$, avec M inversible, fait que le système linéaire $Ax = b$ peut se réécrire sous la forme

$$\left(I - M^{-1}N\right) x = M^{-1}b\,, \tag{3.36}$$

ce qui revient exactement à l'équation (3.35), car $M^{-1}A = M^{-1}(M-N) = I - M^{-1}N$. Si l'on s'appuie sur le splitting de la méthode de Jacobi, on obtient $M = D$, ce qui revient à multiplier à gauche le système $Ax = b$ par l'inverse de la diagonale de A. Ce préconditionnement est loin d'être optimal, mais le surcoût de calcul qu'il induit est négligeable et son impact en termes de nombre d'itérations est relativement important, particulièrement lorsque A est à diagonale dominante.

Parmi les méthodes plus élaborées, on peut citer la factorisation de Cholesky incomplète. Cette méthode est une variante de la factorisation de Cholesky d'une matrice symétrique, qui détermine une matrice L telle que $A = LL^T$. Cette factorisation est unique si on impose que les coefficients diagonaux de L soient positifs. Si A est en plus une matrice creuse, ce qui est typiquement le cas pour des systèmes linéaires issus de la discrétisation d'équations aux dérivées partielles, on peut créer une approximation Q de la matrice L en imposant la condition $(a_{i,j} = 0 \Rightarrow q_{i,j} = 0)$, et utiliser le préconditionneur $M = QQ^T$. Cette méthode a été introduite par Meijerink et Van Der Vorst [53] en 1977, et généralisée pour l'application à la méthode des gradients conjugués par Kershaw l'année suivante [37]. Lin et Moré ont montré en 1999 que cette méthode est performante même lorsque la mémoire disponible sur chaque processeur est limitée [48].

3.2 L'algorithme des Gradients Conjugués

Comme toute méthode itérative pour la résolution de systèmes linéaires, la méthode des Gradients Conjugués est un algorithme créant une série de valeurs $\{x_k\}_{k\in\mathbb{N}}$ qui converge vers $x = A^{-1}b$. Cependant, elle obéit à une logique particulièrement innovante lors de sa création, dans les années 1950, et toujours digne d'intérêt. L'explication qui suit est fortement inspirée par la joliment nommée "Introduction to the Conjugate Gradient Method without the Agonizing Pain" de Richard Shewchuk [70].

3.2.1 Principe et algorithme

Lors de chaque itération k, l'algorithme calcule une direction de descente, c'est-à-dire un vecteur p_k qui vise à réduire le résidu r_k, et un coefficient α_{k+1} pour p_k ; alors :

$$x_{k+1} = x_k + \alpha_{k+1} p_k \tag{3.37}$$

Pour éviter que le nombre d'itérations nécessaires ne puisse devenir prohibitif, comme c'est typiquement le cas avec des méthodes itératives antérieures, on souhaite que la direction de descente, usuellement construite à partir du vecteur résidu, soit orthogonale à toutes les directions précédentes pour la norme choisie. L'avantage est que l'on limite le nombre d'itérations nécessaires pour la convergence en arithmétique exacte à la valeur de la taille de la matrice ; cela nécessite en revanche que l'on choisisse le paramètre α_{k+1} de façon à s'assurer que la direction p_k n'aura plus à être empruntée. En d'autres termes, les p_k sont choisis de façon à ce que $\{p_1; p_2; \ldots; p_n\}$ soit une base de $\mathbb{R}^n$, et les α_{k+1} sont déterminés de façon à ce que l'erreur e_k soit orthogonale, pour un certain produit vectoriel, aux directions de descente $\{p_1; p_2; \ldots; p_{k-1}\}$.

Si nous désirions que cette orthogonalité soit au sens d'un p-produit scalaire usuel $\langle \cdot, \cdot \rangle_p$ sur $\mathbb{R}^n$, le calcul de α_{k+1} requierrait de connaître la valeur de e_k, donc celle de x. En effet, l'orthogonalité entre e_{k+1} et p_k se traduit par $\langle e_{k+1}, p_k \rangle_p = 0$ soit $\langle e_k - \alpha_{k+1} p_k, p_k \rangle_p = 0$, d'où l'on déduit aisément

$$\alpha_{k+1} = \frac{\langle e_k, p_k \rangle_p}{\langle p_k, p_k \rangle_p} . \tag{3.38}$$

L'adoption du A-produit scalaire $\langle \cdot, \cdot \rangle_A$, rendue possible si et seulement si A est symétrique définie positive (car un produit scalaire doit également être symétrique et défini positif), permet de résoudre ce problème, car $\langle e_k, p_k \rangle_A = p_k^T A e_k = p_k^T r_k$. Le coefficient α_{k+1} ainsi associé à la direction p_k choisie sera

$$\alpha_{k+1} = \frac{r_k^T r_k}{p_k^T A p_k} . \tag{3.39}$$

L'algorithme construira alors une base de vecteurs A-orthogonaux $\{p_1; p_2; \ldots; p_n\}$ de $\mathbb{R}^n$ et les coeffients α_k permettront d'écrire x dans cette base.

Le choix de p_k peut être fait d'une façon très simple, en A-orthogonalisant le résidu r_k par rapport aux directions $p_1, p_2, \ldots, p_{k-1}$ précédentes. Cette construction de la base $p_1, p_2, \ldots, p_n$ a un énorme avantage : r_k est orthogonal à tous les p_i pour $i < k-1$, ce qui se montre par une simple récurrence. Il suffit donc d'A-orthogonaliser r_k par rapport au vecteur p_{k-1}, c'est-à-dire trouver le réel β_k tel que

$$p_k = r_k + \beta_k p_{k-1} \tag{3.40}$$

est A-orthogonal à p_{k-1}. Cette condition s'exprime $(r_k + \beta_k p_{k-1})^T A p_{k-1} = 0$ soit

$$\beta_k = -\frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}}\,. \tag{3.41}$$

On peut simplifier cette expression en se servant de propriétés qui découlent directement du lien entre les r_k et les p_k ; on trouve ainsi

$$\beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}\,. \tag{3.42}$$

Pour l'implémentation, on déclare $\sigma_k = p_k^T A p_k$ et $\rho_k = r_k^T r_k$. Une première forme de l'algorithme CG est alors donnée par l'algorithme 1.

Algorithme 1 Algorithme de la méthode des Gradients Conjugués

Entrées : A, b, x_0

$\boxed{\rho_0 = r_0^T r_0}$

$p_0 = r_0 = b - Ax_0$

$\boxed{\sigma_0 = p_0^T A p_0}$

Pour $k = 0, 1, 2 \ldots$ jusqu'à obtenir la convergence souhaitée **Faire**

$\alpha_{k+1} = \frac{\rho_k}{\sigma_k}$

$x_{k+1} = x_k + \alpha_{k+1} p_k$

$r_{k+1} = r_k - \alpha_{k+1} A p_k$

$\boxed{\rho_{k+1} = r_{k+1}^T r_{k+1}}$

$\beta_{k+1} = \frac{\rho_{k+1}}{\rho_k}$

$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$

$\boxed{\sigma_{k+1} = p_{k+1}^T A p_{k+1}}$

Fin Pour

En préconditionnant par une matrice M^{-1}, la résolution de $Ax = b$ devient équivalente à celle de $M^{-1}Ax = M^{-1}b$, ce qui crée la méthode du Gradient Conjugué Préconditionné ou PCG [82] dont une forme basique est donnée par l'algorithme 2. On remarquera que, cette fois-ci, on a $\rho_k = r_k^T w_k$; en revanche, la valeur de σ_k est inchangée.

Les étapes encadrées dans ces algorithmes correspondent à des étapes pour lesquelles des communications interprocesseurs globales sont requises lors de l'exécution en parallèle. Nous expliquons en sous-section 4.1.3 l'utilité de ces communications et les outils qui permettent de les effectuer, et présentons en section 4.3 une façon d'optimiser les algorithmes CG et PCG pour l'exécution parallèle.

3.2.2 Méthodes de Krylov

Alexeï Krylov, ingénieur naval russe dont les travaux sur le roulis et le tangage dans les années 1890 ont mené à l'adoption massive du stabilisateur gyroscopique, est également

Algorithme 2 Algorithme de la méthode PCG

ENTRÉES : A, b, x_0

$p_0 = r_0 = b - Ax_0$

$\boxed{\sigma_0 = p_0^T A p_0}$

$w_0 = M^{-1} r_0$

$\boxed{\rho_0 = r_0^T w_0}$

Pour $k = 0, 1, 2 \ldots$ jusqu'à obtenir la convergence souhaitée **Faire**

$\alpha_{k+1} = \frac{\rho_k}{\sigma_k}$

$x_{k+1} = x_k + \alpha_{k+1} p_k$

$r_{k+1} = r_k - \alpha_{k+1} A p_k$

$w_{k+1} = M^{-1} r_{k+1}$

$\boxed{\rho_{k+1} = r_{k+1}^T w_{k+1}}$

$\beta_{k+1} = \frac{\rho_{k+1}}{\rho_k}$

$p_{k+1} = w_{k+1} + \beta_{k+1} p_k$

$\boxed{\sigma_{k+1} = p_{k+1}^T A p_{k+1}}$

Fin Pour

connu des mathématiciens et des numériciens pour son article de 1931 relatif aux problèmes de valeurs propres [43]. Il y propose une méthode basée sur des multiplications successives par la matrice A du problème à résoudre, afin de créer les images itérées d'un vecteur par cette matrice. Cette méthode ne nécessite pas la décomposition préalable de la matrice, contrairement aux méthodes de splitting qui sont à l'époque les seules méthodes itératives de résolution de systèmes linéaires.

Ses travaux précurseurs ont donné leur nom aux **méthodes de Krylov**, méthodes itératives basées sur des multiplications successives par la matrice du système à résoudre. L'autre point commun de ces méthodes, plus significatif du point de vue numérique, est la création itérative d'un sous-espace de $\mathbb{R}^n$ nommé **espace de Krylov**, défini par

$$\mathcal{K}_k(A, h) = \text{Span}\left\{h, Ah, A^2h, \ldots, A^{k-1}h\right\} . \tag{3.43}$$

Une méthode de Krylov donnée cherche à chaque itération k la "meilleure" solution approchée du système $Ax = b$ dans le sous-espace $\mathcal{K}_k(A, b)$ ou, plus généralement, dans $\mathcal{K}_k(A, r_0)$. Le critère de qualité des solutions est variable. La méthode GMRES (Generalized Minimal Residual Method, ou méthode du résidu minimal généralisée) développée par Saad et Schultz en 1986 [67] résout par exemple à chaque itération le problème de minimisation

$$\min_{x_k \in \mathcal{K}_k(A,b)} \|b - Ax_k\|_2 \tag{3.44}$$

par la construction itérative d'une base orthonormée de $\mathcal{K}_k(A, b)$.

Les avantages des méthodes de Krylov sont nombreux. On peut citer un coût de calcul potentiellement très réduit, d'une part parce qu'une méthode de Krylov savamment construite ne nécessite qu'un nombre très limité de multiplications matrice-vecteur par itération, d'autre part parce que le nombre d'itérations requis pour atteindre la solution exacte est au plus égal à n et que, dans la pratique, un nombre d'itérations très inférieur à n suffit pour atteindre une précision convenable.

Un autre avantage, quant à lui totalement appliqué, est qu'il n'y a aucun besoin d'une déclaration explicite de la matrice A. La déclaration de l'opérateur A, version discrétisée de l'opérateur laplacien sur un maillage non structuré, qui est effectuée dans YALES2 comme décrit dans la partie 2.2.2.2 ne permettrait pas d'implémenter simplement une méthode nécessitant la connaissance explicite de la matrice A ; pour une méthode de splitting par exemple, le problème de la création des matrices M et N irait de pair avec celui de leur stockage sur chaque processeur. Le développement de méthodes de Krylov a donc été une étape-clé dans le développement de solveurs linéaires parallèles.

Nous avons avancé à la fin de la sous-section précédente que la méthode des Gradients Conjugués est une méthode de Krylov ; cette assertion se prouve rapidement. Rappelons que, dans l'algorithme CG, l'itération (3.37) permettant de passer de x_k à x_{k+1} est construite de façon à ce que l'erreur e_{k+1} soit A-orthogonale à toutes les directions de recherche $\{p_0, p_1, \ldots, p_k\}$ déjà utilisées, ce qui donne à la fois une condition "d'optimalité locale" d'une solution et l'appartenance de x_k à $\text{Span}\{p_0, p_1, \ldots, p_k\}$.

En outre, les directions de descente sont définies par le fait que p_0 coïncide avec le premier résidu r_0, et les directions p_k suivantes sont les résidus r_k A-orthogonalisés par rapport aux directions $\{p_0, p_1, \ldots, p_{k-1}\}$ précédentes. En ajoutant à cela l'itération $r_{k+1} = r_k - \alpha_{k+1} A p_k$ sur le résidu, on conclut facilement

$$\text{Span}\{p_0, p_1, \ldots, p_k\} = \text{Span}\left\{r_0, Ar_0, \ldots, A^{k-1} r_0\right\} = \mathcal{K}_k(A, r_0) \ . \tag{3.45}$$

Par conséquent, l'algorithme des Gradients Conjugués résout à chaque itération k le problème défini par

$$x_k \in \mathcal{K}_k(A, b) \ ; \ \forall i \in \{0; 1; \ldots; k-1\}, e_k^T A p_i = 0 \, . \tag{3.46}$$

Ce problème est mathématiquement équivalent à celui de la minimisation de l' A-norme de l'erreur, soit

$$\min_{x_k \in \mathcal{K}_k(A,b)} e_k^T A e_k \, . \tag{3.47}$$

Chapitre 4

Simulation massivement parallèle

La simulation massivement parallèle est un outil indispensable, entre autres, au cycle de conception des moteurs, autant dans l'automobile que dans l'aéronautique. Son importance est reconnue dans tous les domaines industriels, à tel point que plus de la moitié des ressources de calcul parallèle sont désormais dédiées à l'industrie[1] (contre à peine 20 % en 1995). Depuis le début des années 1990, les architectures parallèles se développent sans discontinuer et leurs vitesses de calcul augmentent de façon exponentielle, comme le montre la figure 4.1 ; au mois de juin 2012, les 500 plus puissants calculateurs mondiaux fournissaient une vitesse de calcul totale de 126 pétaFLOPS, soit environ 10 fois plus qu'il y a quatre ans.

Dans ce chapitre, nous commencerons par présenter le calcul parallèle et le HPC[2] de façon générale, puis nous nous intéresserons plus particulièrement aux outils et principes de développement de codes CFD pour l'exécution massivement parallèle, en particulier concernant la décomposition de domaine et la résolution parallèle des systèmes linéaires obtenus par discrétisation des équations aux dérivées partielles décrivant les écoulements. Enfin, nous illustrerons nos propos en développant un exemple lié au brûleur PRECCINSTA, que nous présenterons par la même occasion.

4.1 Généralités sur le calcul parallèle

Cette section a pour but de présenter brièvement les principes de base et les enjeux du calcul parallèle. Après avoir rappelé les fondamentaux du calcul parallèle, nous introduirons les deux types d'architectures matérielles les plus courants. Nous nous intéresserons ensuite à la librairie MPI, utilisée pour la communication interprocesseurs. Nous expliquerons enfin comment les performances parallèles d'un programme peuvent être évaluées.

[1] Voir le site du projet TOP500 : http://www.top500.org.
[2] *High Performance Computing* ou Calcul Haute Performance.

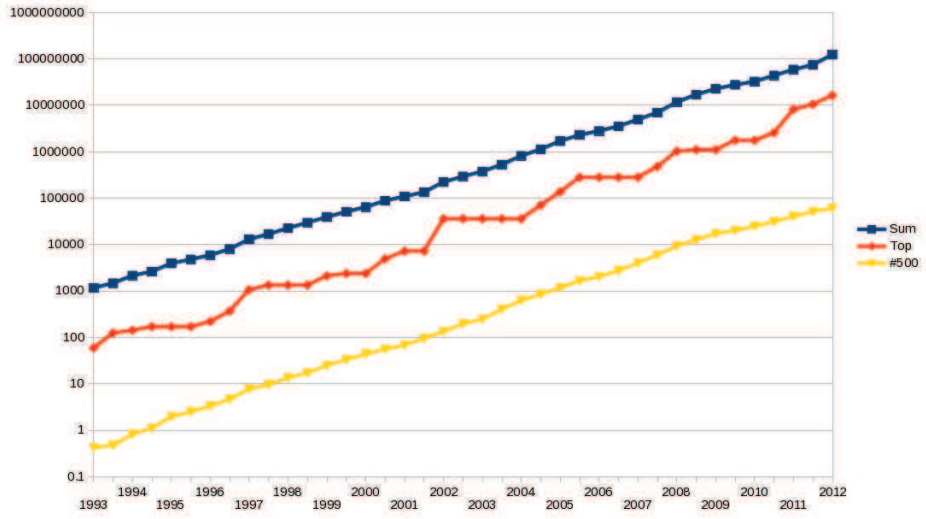

FIGURE 4.1 – *Evolution de la somme des vitesses de calcul des 500 supercalculateurs les plus rapides du monde, ainsi que des vitesses du supercalculateur le plus rapide et du cinq-centième supercalculateur le plus rapide du monde, entre 1993 et 2012. Source : Wikipedia Commons, d'après les valeurs du projet TOP500 (http://www.top500.org).*

4.1.1 Principes de base du calcul parallèle

L'exécution d'un programme en parallèle vise tout simplement à la réduction du temps d'attente de ses résultats, soit en exécutant plusieurs instructions indépendantes en même temps sur les différents processeurs, soit en exécutant la même instruction sur des données différentes sur chaque processeur. En 1966, l'américain Michael Flynn a proposé une taxinomie, ou classification, des ordinateurs selon leur mode de fonctionnement, dans laquelle ces deux fonctionnements parallèles sont affublés des acronymes MIMD[3] et SIMD[4] respectivement. Mentionnons que dans cette classification volontairement simpliste, et donc pas tout à fait exhaustive puisque certains architectures peuvent adapter plusieurs comportements différents (correspondre à différents taxons), existent également deux types à donnée unique, d'une part le désormais rare MISD[5], d'autre part le beaucoup plus courant SISD[6] qui correspond à l'exécution séquentielle des données effectuée par les premiers ordinateurs, avant la création d'ordinateurs vectoriels correspondant au taxon SIMD.

La méthode de calcul parallèle la plus courante, particulièrement pour la CFD, est désignée par l'acronyme SPMD qui signifie *Single Process/Program, Multiple Data*, c'est-à-dire Processus/Programme Unique, Données Multiples. Elle consiste à fournir le même

[3] *Multiple Instruction, Multiple Data*, c'est-à-dire Instructions Multiples, Données Multiples.
[4] *Single Instruction, Multiple Data*, c'est-à-dire Instruction Unique, Données Multiples.
[5] *Multiple Instruction, Single Data*, c'est-à-dire Instructions Multiples, Donnée Unique.
[6] *Single Instruction, Single Data*, c'est-à-dire Instruction Unique, Donnée Unique.

programme à différents processeurs devant traiter différentes parties des données globales. En CFD, on utilise une partition du domaine, chaque processeur étant assigné à une partie différente de ce domaine. Le programme est le même sur chaque processeur, puisque les lois régissant l'interdépendance des variables physiques sont les mêmes sur tout le domaine physique, mais chaque processeur les applique à un sous-domaine différent. Ce paradigme de conception est illustré par la figure 4.2. Pour prendre en compte l'interdépendance de tous ces sous-domaines, les processeurs doivent communiquer les uns avec les autres, ce qui est fait grâce à des instructions MPI (*Message Passing Interface*, ou Interface de Transfert de Messages) ; cet aspect sera traité en sous-section 4.1.3.

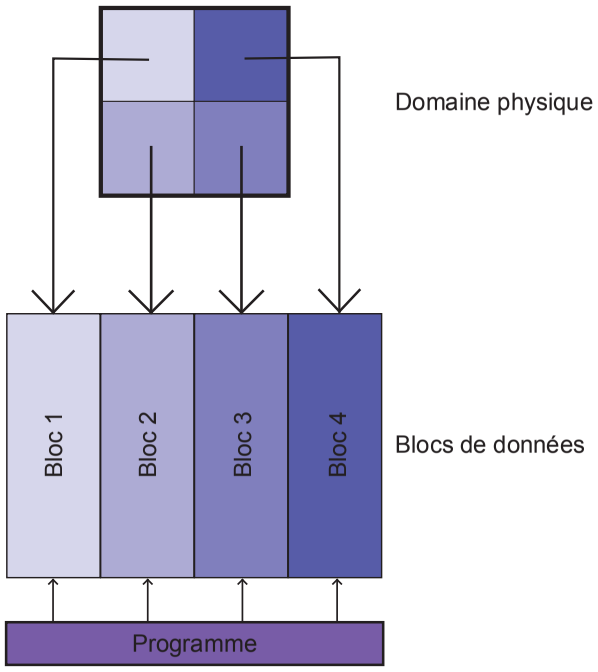

FIGURE 4.2 – *Illustration schématique du fonctionnement en parallèle des logiciels de CFD. Le même programme est utilisé pour traiter, sur chaque processeur, les données associées à un sous-domaine particulier. Chaque processeur est ainsi associé à une partie du domaine physique simulé, et l'ensemble des parties affectées aux différents processeurs forme une partition du domaine physique.*

On peut d'ores et déjà noter que le SPMD est en fait une forme particulière du taxon MIMD, et non du SIMD, dans le sens où, contrairement aux ordinateurs vectoriels par exemple, un supercalculateur pourra exécuter le programme de façon non synchronisée. En d'autres termes, un processeur qui a passé une instruction pourra passer à la suivante sans attendre que les autres processeurs aient eux-mêmes fini d'exécuter cette instruction. Une conséquence importante de cet état de fait est que le programmeur devra imposer lui-même des barrières de synchronisation si besoin est, par exemple, dans le cas de la

CFD, lorsque les processeurs doivent échanger des informations sur leurs sous-domaines de façon à rendre cohérentes les données physiques sur le domaine global. Comme nous le verrons plus loin, certaines des instructions MPI les plus couramment utilisées dans les codes de CFD forcent implicitement cette barrière de synchronisation.

4.1.2 Architectures parallèles

Au mois de juin 2012, les 500 supercalculateurs les plus puissants du monde ne se répartissent qu'entre deux familles d'architectures matérielles : plus de 80 % d'entre eux sont organisés en clusters, tandis que les 18,6 % restants sont des machines massivement parallèles. On peut écrire, au prix d'une légère vulgarisation qu'une machine massivement parallèle est un ensemble de CPUs indépendants, chacun doté de sa mémoire propre, tandis qu'une machine clusterisée est organisée en noeuds de calcul ou clusters, chaque noeud étant constitué de plusieurs processeurs possédant leurs mémoires propres et reliés à une mémoire partagée.

Selon qu'un programme doit être optimisé pour l'une ou l'autre de ces architectures, les outils à déployer ne sont pas les mêmes. En effet, des bibliothèques différentes peuvent être utilisées selon que la mémoire est partagée ou non. Deux bibliothèques de fonctions sont couramment utilisées :

MPI est une bibliothèque portable et standardisée qui établit un protocole de communications interprocesseurs. Elle est devenue au fil des années un standard pour la communication entre processeurs dans les architectures à mémoire distribuée. Son utilisation exclusive est également possible pour un programme destiné à être exécuté sur une machine clusterisée, mais ne garantit pas des performances optimales à cause des surcoûts de transferts de données ; en effet, un programme en MPI pur n'exploite pas le partage des données entre les processeurs d'un même cluster, puisqu'une communication entre les processeurs d'un même cluster suivra le même type de canal qu'une communication entre deux clusters différents. Cette bibliothèque est présentée plus en détail en sous-section suivante.

OpenMP est une interface de programmation permettant la gestion du multithreading, c'est-à-dire de l'exécution de plusieurs *threads*, ou *fils d'exécution*, partageant la mémoire du processus qui les contient. L'intérêt réside en ce que la communication de données est bien plus efficace entre plusieurs fils d'exécution d'un même processus qu'entre plusieurs processus séparés. L'utilisation du multithreading pour exécuter plusieurs tâches différentes en parallèle est extrêmement rare en CFD, où l'utilisation la plus courante reste dans le contexte du SPMD.

Une mise en oeuvre idéale pour une machine clusterisée utiliserait OpenMP pour la répartition des tâches et la communication des processeurs au sein de chaque cluster, tandis que des fonctions MPI s'occuperaient de la répartition des données et de leur

communication entre les différents clusters. Cependant, nous ne nous intéresserons ici qu'à de la programmation parallèle purement MPI, la fusion MPI/OpenMP n'ayant pas été étudiée dans le contexte de cette thèse.

4.1.3 Librairie MPI

Devenue de fait un standard pour la gestion des communications interprocesseurs, la librairie MPI repose sur des principes d'utilisation simples que nous présentons dans ce qui suit, avant de lister les fonctions MPI les plus utiles dans le cadre de la CFD. Nous nous intéresserons également aux temps de communication et à certaines façons de réduire leur importance dans le temps de simulation, ce qui sera mis en pratique en sous-section 4.3 pour optimiser les algorithmes CG et PCG pour l'exécution parallèle. Le lecteur intéressé est invité à se référer au guide de Snir *et al.* [72] pour tout renseignement complémentaire.

4.1.3.1 Principes d'utilisation et concepts-clé

Lors de l'exécution parallèle d'un programme, toutes les données sont privées, c'est-à-dire que chaque processeur a un accès exclusif aux données stockées dans sa mémoire. Pour qu'un processeur A ait accès à une partie des données d'un processeur B, les deux processeurs doivent établir un protocole de communication, d'une part pour assurer le transit correct du message, d'autre part afin que les données envoyées par A puissent être correctement interprétées par B. Il est également possible que des données doivent être échangées entre plusieurs processeurs, ce qui peut être décomposé en une série de communications point-par-point, c'est-à-dire ne concernant que deux processeurs ; en outre, tout ou partie de ces communications point-par-point peuvent souvent être effectuées en même temps. Par exemple, la diffusion à tous les processeurs d'une valeur par un processeur donné revient à l'envoi simultané de cette valeur à chacun des processeurs.

La librairie MPI fournit des outils simples pour effectuer ces communications, sous la forme de fonctions à utiliser pour les communications point-par-point ou pour les communications globales. Ces fonctions ne peuvent être utilisées qu'une fois l'environnement MPI initialisé, grâce à l'instruction **MPI_INIT()**. L'appel de cette instruction, forcément par un seul processeur, c'est-à-dire un seul fil d'exécution, a deux effets :

- les processeurs utilisés par le programme sont numérotés à partir de 0, et ces identifiants uniques des processeurs sont nommés rangs ; le processeur de rang 0 ou processeur maître est celui qui a appelé la procédure **MPI_INIT** ;
- un communicateur nommé **MPI_COMM_WORLD** est créé, qui regroupe tous les processeurs utilisés (on peut parler de communicateur universel).

Eclaircissons ce dernier point : un communicateur est un ensemble de processeurs pouvant s'engager dans une communication globale. En d'autres termes, pour qu'une communication entre plusieurs processeurs soit possible (diffusion d'une valeur, rapatriement de

valeurs vers un processeur maître, réductions, etc.), il faut définir un communicateur qui regroupe ces processeurs. Le nom de ce communicateur servira de paramètre à la fonction MPI qui s'occupe de la communication globale sur ce communicateur. Le communicateur universel **MPI_COMM_WORLD** rend donc possible les communications concernant tous les processeurs utilisés par le programme, ce qui lui donne un rôle clé.

En effet, la communication la plus courante lors de la résolution de l'équation de Poisson est ce que l'on appelle, dans le vocabulaire MPI, une réduction, et cette réduction est effectuée sur **MPI_COMM_WORLD**, c'est-à-dire qu'elle concerne tous les processeurs à la fois. Il s'agit de collecter une valeur sur chaque processeur et de transformer toutes ces valeurs en une seule par une opération simple. Deux réductions différentes sont utilisées dans la version originale d'un algorithme de type Gradient Conjugué.

- Lorsqu'un produit scalaire de deux vecteurs doit être calculé, chaque processeur ne peut calculer que la somme des produits des composantes de ces vecteurs qui sont dans le sous-domaine qui lui est attribué. Une réduction doit donc être faite pour récolter les contributions de chaque processeur, leur somme donnant la valeur effective du produit scalaire, qui doit ensuite être distribuée à tous les processeurs pour qu'ils puissent l'utiliser dans les instructions suivantes de l'algorithme.

- Pour vérifier si le critère de convergence a été atteint ou non, il faut récolter des informations sur tous les processeurs. Ce critère de convergence concernant la norme infinie du résidu, c'est-à-dire sa plus grande composante en valeur absolue, deux réductions simples peuvent être appliquées. La première consiste à isoler sur chaque processeur la plus grande composante en valeur absolue du "résidu local" ; la réduction consistera alors à calculer le maximum de ces valeurs locales. La seconde consiste à affecter sur chaque processeur une valeur booléenne valant 1 si le critère de convergence est localement respecté et 0 sinon ; le ET logique de ces valeurs permettra de savoir si le critère de convergence est globalement respecté ou non.

D'autres communications, point-par-point cette fois-ci, sont utilisées principalement pour remplir ce qu'on appelle les cellules fantômes de chaque processeur. Nous allons immédiatement éclaircir cette notion.

Certaines opérations, comme par exemple l'application de l'opérateur laplacien à un champ sur le domaine physique, correspond numériquement à un calcul de somme pondérée en chaque point du maillage, après la discrétisation de l'opérateur telle que présentée dans la sous-section 2.2.2. La valeur résultante en un point donné est une combinaison linéaire des valeurs du vecteur en plusieurs autres points du maillage, lesquels se situent tous dans un certain voisinage du point pour lequel le calcul est effectué. La taille de ce voisinage est déterminée par l'ordre de discrétisation des opérateurs : le calcul en un point peut ne nécessiter que les valeurs des cellules voisines, ou également les valeurs de leurs propres voisines, etc.

Lors de l'exécution du programme sur un seul processeur, toutes les données sont directement accessibles ; pour l'exécution en parallèle, en revanche, l'application de certains opérateurs, et particulièrement leur calcul à proximité de la frontière du sous-domaine affecté à un processeur, nécessite que ce processeur récupère des valeurs sur des cellules extérieures. En plus des cellules de son sous-domaine, des cellules fantômes sont donc déclarées sur le processeur, c'est-à-dire des clones de cellules affectées à un autre processeur. Chacune de ces cellules devra recevoir la valeur correspondante d'un vecteur donné avant que l'opérateur puisse être appliqué localement à ce vecteur, ce qui nécessite une réception de données depuis les processeurs auxquels ces cellules sont affectées.

Ces communications en point-par-point impliquent que chaque processeur connaisse les rangs de ses voisins ainsi que le nombre et l'ordre des cellules fantômes correspondantes. Le programme YALES2 déclare pour cela des communicateurs externes, avec dans ce cas un usage du vocable "communicateur" différent de celui effectué dans le cadre de la terminologie MPI, comme expliqué en sous-section 4.2.2 ci-après.

4.1.3.2 Fonctions essentielles pour la CFD

Comme expliqué ci-avant, un logiciel de CFD conçu pour l'exécution en parallèle doit se servir de fonctions de communication en point-par-point et de fonctions de communications globales, ainsi que de fonctions permettant la synchronisation des fils d'exécution. Cette courte partie a pour objectif de lister les fonctions MPI les plus couramment utilisées dans le solveur YALES2 et leurs arguments. Nous ne citerons pas systématiquement les arguments récurrents des fonctions MPI, à savoir le communicateur de référence qui sera **MPI_COMM_WORLD** par défaut pour toutes les communications, ainsi que les variables entières destinées à recevoir les codes de statut et d'erreur des communications.

Communications point-par-point : On utilise principalement deux types de communications ici, des communications bloquantes et non-bloquantes. Une communication bloquante est telle que son exécution doit absolument être terminée, c'est-à-dire que le processeur destinataire doit avoir reçu les données et que le processeur expéditeur doit avoir reçu l'accusé de réception retourné par le destinataire, avant que les instructions suivantes soient traitées. En revanche, pour un envoi non-bloquant, l'expéditeur peut passer à l'instruction suivante dès que les données ont été reçues par une mémoire tampon d'envoi ; de même, le destinataire initialise l'opération de réception des données, mais n'attend pas leur réception effective pour continuer l'exécution du programme. Les communications non-bloquantes permettent de lancer en avance une communication de données qui seront utilisées plus tard dans le programme ; une instruction supplémentaire permet de forcer les processeurs à vérifier que les données ont été correctement reçues par le destinataire avant de passer à la suite du programme.

- **MPI_SEND** effectue l'envoi bloquant d'un bloc de données vers un proces-

seur, désigné par son rang. Sont aussi précisés le format des données envoyées et leur nombre, ainsi que l'adresse de début du bloc de données à envoyer, qui est tout simplement le nom de la variable concernée pour l'envoi d'une valeur seule, ou le nom de la première valeur d'un tableau dans le cas de l'envoi de plusieurs valeurs concaténées en mémoire.

- **MPI_RECV** se charge de la réception d'un bloc de données, et prend comme arguments le rang du processeur expéditeur, le format des données et leur nombre. Cette procédure est bloquante, donc à utiliser lorsque les données reçues sont utilisées immédiatement après la communication.
- **MPI_ISEND** s'utilise de la même façon que **MPI_SEND** mais est non-bloquante. Un identifiant doit être ajouté aux arguments, qui permettra de reconnaitre cet appel de la procédure non-bloquante d'envoi parmi toutes les communications non-bloquantes qui pourraient être effectuées en même temps.
- **MPI_IRECV** s'utilise de la même façon que **MPI_RECV** mais est non-bloquante. A la liste des arguments de **MPI_RECV** s'ajoute l'identifiant de la communication, qui doit être le même que celui utilisé par la fonction **MPI_ISEND** sur le processeur expéditeur.
- **MPI_WAIT** reçoit comme argument l'identifiant d'une communication non-bloquante, et force le processeur à interrompre l'exécution du programme tant que cette communication n'est pas achevée. Elle permet donc de s'assurer que des valeurs particulières ont été reçues par un processeur, et elle doit être autant que possible appelée par le processeur destinataire juste avant sa première utilisation, fût-elle éventuelle, de ces valeurs.

Communications globales : La librairie MPI est capable d'organiser des envois et réceptions globaux de données, en synchronisant ou en désynchronisant des communications point-par-point à la place de l'utilisateur. Chacune des fonctions ci-dessous peut être réécrite en n'utilisant que des fonctions point-par-point et en utilisant des instructions conditionnelles, pour donner un rôle différent au processeur maître et aux autres processeurs. Il faudrait alors veiller à ce que les bonnes communications soient bloquantes ou non-bloquantes, simultanées ou non, et caetera ; une lourde tâche dont ces fonctions nous exemptent.

- **MPI_REDUCE** permet la réduction d'une ou plusieurs valeurs de même type pour tous les processeurs d'un communicateur. En plus du nombre et du type des valeurs concernées, ainsi que de leur nom (le même sur chaque processeur) et du nom de la variable réduite, la fonction admet pour arguments une valeur correspondant au type d'opération à effectuer sur les valeurs, ainsi que le rang du processeur sur lequel sera stockée la variable réduite. Parmi les opérations de réduction possibles, on peut citer la somme, le produit, le calcul du minimum ou du maximum, ou encore le OU et le ET logiques.

- **MPI_BCAST** est une fonction permettant l'envoi d'une valeur ou d'un tableau de valeurs, dont on donne le nom de variable et le rang du processeur source, à tous les autres processeurs du communicateur donné en argument.
- **MPI_ALLREDUCE** est une succession des deux précédents, qui admet les mêmes arguments que **MPI_REDUCE** à l'exception du rang du processeur qui reçoit les valeurs, effectue leur réduction et envoie le résultat à tous les processeurs du communicateur. Ce rôle est attribué par défaut au processeur maître, c'est-à-dire de rang 0.

Barrière de synchronisation : **MPI_BARRIER** ne reçoit comme argument que le communicateur concerné, et ne permet aux processeurs de ce communicateur de passer à l'instruction suivante qu'une fois qu'ils sont tous entrés dans cette fonction.

Les communications point-par-point sont bien entendu utilisées pour les échanges des données situées près des frontières des sous-domaines assignés aux différents processeurs, notamment pour l'application d'opérateurs différentiels, tandis que les communications globales sur le communicateur universel permettent entre autres de calculer des valeurs de produits scalaires et de vérifier des critères de convergence. La procédure **MPI_ALLREDUCE** est largement utilisée dans le code, notamment pour tous les calculs de produits scalaires.

4.1.3.3 Temps de communication et stratégies de réduction

Le temps que met une communication interprocesseurs à se réaliser peut être décomposé en plusieurs parties, comme montré en figure 4.3. Outre le temps effectif de transfert du message, le temps effectif de la communication comprend un temps de préparation, durant lequel les paramètres réseaux sont initialisés et le message est préparé.

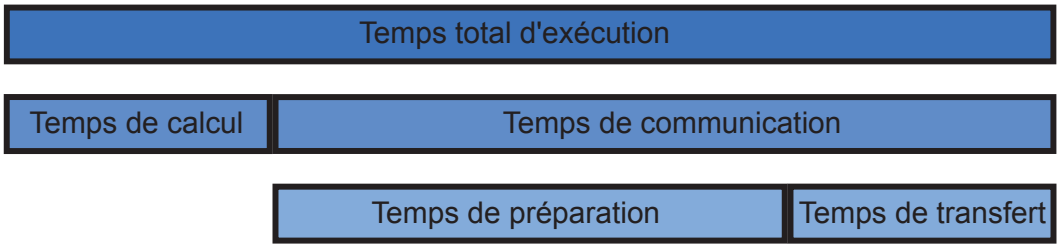

FIGURE 4.3 – *Décomposition schématique du temps d'exécution d'un programme parallèle mettant en évidence les composantes du temps de communication.*

Tandis que le temps de transfert est à peu près proportionnel au volume de données envoyées, le temps de préparation comprend une partie incompressible appelée latence, ce qui fait que le temps de communication global n'est pas proportionnel au volume de

données envoyé. A titre d'exemple, les deux bus[7] les plus représentés parmi les 500 calculateurs les plus puissants au monde (au mois de juin 2012), InfiniBand et Gigabit Ethernet, ont des temps de latence compris entre 200 ns et 1 μs. En conséquence, l'optimisation d'un programme parallèle passe par le regroupement des communications, puisque la communication d'un tableau de valeurs est notablement plus rapide que les communications séparées de chacune de ses valeurs. Nous verrons en sous-section 4.3 comment un algorithme de type Gradients Conjugués peut être réorganisé de façon à ne plus nécessiter qu'une communication globale par itération, au lieu de trois dans une implémentation naïve.

Le temps de préparation comprend également l'écriture du message qui sera envoyé, qui comprend un en-tête décrivant entre autres la structure du message, et son éventuelle copie sur une mémoire tampon d'envoi. On peut y ajouter le temps que prend l'éventuelle copie du message depuis la mémoire tampon de réception vers la mémoire du processeur. La librairie MPI donne la possibilité de forcer la transmission directe du message, sans aucune recopie vers ou depuis une mémoire tampon : il s'agit alors d'envois synchrones. Ces envois permettent de réduire le surcoût mais peuvent engendrer des blocages ; en effet, l'expéditeur est forcé d'attendre que le destinataire soit prêt avant d'envoyer les données, et *a fortiori* avant de passer à l'instruction suivante. Dans de nombreux cas, le recours aux buffers est donc indispensable ; c'est pourquoi les procédures de communications synchrones n'ont pas été présentées dans la partie précédente.

Les gains possibles sur les temps de transfert eux-mêmes sont presque négligeables, tant les vitesses de transfert sont élevées sur les architectures parallèles modernes. Nous nous intéressons donc principalement à deux techniques de réduction des coûts de communication, à savoir l'évitement et le recouvrement.

L'évitement, déjà évoqué ci-dessus, et développé plus en détail ci-après à propos de la parallélisation de l'algorithme CG, consiste à recourir au plus petit nombre de communications possibles, par exemple en regroupant plusieurs communications. Par recouvrement, nous entendons le recouvrement de temps de communication par des calculs : un envoi non-bloquant permet à l'expéditeur du message de continuer l'exécution du programme sans devoir attendre que le message ait effectivement été reçu, de même que le récepteur peut, grâce à une fonction de réception non-bloquante, passer aux instructions suivantes sans être forcé d'attendre un message. Dans les cas où cela est possible, il est donc recommandé d'avoir recours à des communications non-bloquantes, pour déclencher l'envoi d'une donnée dès que sa valeur est calculée et pour ne forcer sa réception que le plus tard possible, c'est-à-dire juste avant sa première utilisation. La figure 4.3 est alors rendue caduque, puisque le temps de transfert peut alors être partiellement ou totalement superposé au temps de calcul.

Notons que les fonctions d'envoi et de réception point-par-point non bloquantes permettent également d'assurer le transit de certaines données sur un réseau complexe de

[7] Le bus est le système de communication mis en place entre les différents composants d'une machine informatique.

processeurs. Ce cas survient par exemple, dans un programme de CFD, pour préparer l'application d'un opérateur différentiel discrétisé sur un champ vectoriel ; chaque processeur devra alors recevoir des données de tous les processeurs voisins, pour remplir ses cellules fantômes, et envoyer des données à tous ces mêmes processeurs. Tandis que l'utilisation de communications bloquantes forcerait le programmeur à imposer manuellement un ordre à ces communications, l'utilisation de communications non-bloquantes simplifie considérablement la tâche : il suffit de vérifier que les identifiants des communications sont uniques, et que les couples de processeurs devant communiquer s'accordent sur la nature des messages communiqués et sur les identifiants de leurs communications entrantes et sortantes. (Pour ce dernier point, on peut adopter une simple convention, en choisissant par exemple de donner l'identifiant $aN_{proc}+b$ à une communication de données du processeur de rang a vers le processeur de rang b, avec N_{proc} le nombre de processeurs.) L'ordre dans lequel les envois et les réceptions sont déclarés est alors bien moins problématique.

4.1.4 Mesures de performance

Dans un programme de simulation numérique idéal, et en ignorant les possibles surcoûts dûs aux recopies de données en mémoire, le temps d'exécution d'une simulation donnée serait inversement proportionnel au nombre de processeurs utilisé, c'est-à-dire que, quel que soit le nombre de processeurs, les communications prendraient un temps nul ou négligeable devant le temps de calcul. En pratique, les latences de communication font qu'aucun code de calcul parallèle n'est capable d'atteindre une telle accélération. La comparaison directe entre deux programmes conçus pour résoudre les mêmes types de problèmes ne permet pas d'évaluer la scalabilité de chacun de ces programmes, c'est-à-dire leur capacité à rester performants sur des grands nombres de processeurs. C'est pourquoi les mesures de performance parallèle d'un programme nécessitent de considérer d'autres valeurs que le temps brut d'une ou plusieurs simulations.

Nous nous intéressons, pour évaluer la rapidité d'une simulation i, à son efficacité réduite, qui est le temps réel passé dans le programme multiplié par le nombre de processeurs puis divisé par le nombre de volumes de contrôle dans le maillage :

$$R_i = \frac{t \times N_{procs}}{n_{cv}} \,. \tag{4.1}$$

Il s'agit du temps de simulation effectif moyen pour chaque volume de contrôle du maillage. Une partie de ce qui nous intéresse dans cette valeur est que, pour un même cas-test, une simulation effectuée sur un certain maillage avec un certain nombre de processeurs, et une simulation effectuée sur un maillage comptant p fois plus de volumes de contrôle avec p fois plus de processeurs, devraient avoir des efficacités réduites très proches.

C'est sur ce principe que se base la mesure de **weak scaling**, ou scalabilité faible. Il s'agit de considérer une même simulation sur des maillages plus ou moins raffinés, et de

consacrer aux cas-tests ainsi créés des nombres de processeurs proportionnels aux nombres de volumes de contrôle des maillages.

Dans un cas idéal, c'est-à-dire si les temps de communication étaient négligeables devant les temps de calcul, les efficacités réduites seraient sensiblement identiques pour toutes les simulations. On peut donc comparer chaque cas-test i au cas-test de référence, c'est-à-dire au cas-test sur le plus petit maillage et le plus petit nombre de processeurs, auquel on donne ici le numéro 1. On calcule dans ce but un ratio d'efficacité

$$Q_i = \frac{R_1}{R_i}\,, \tag{4.2}$$

où R_j est l'efficacité réduite pour le cas-test j. Plus l'efficacité réduite du cas-test i est grande, plus Q_i s'éloigne de la valeur idéale 1. En pratique, pour la commodité de visualisation, on peut placer sur un graphique un point par cas-test i, dont l'abscisse est le nombre de processeurs utilisés pour ce cas-test et l'ordonnée le produit du nombre de processeurs par le ratio Q_i. La courbe reliant ces points peut alors être comparée à la courbe de *weak scaling* idéale, qui est la droite passant par l'origine du repère et le point correspondant au cas-test de référence. Le weak scaling du solveur YALES2, mesuré en 2010, est par exemple donné par la figure 4.4.

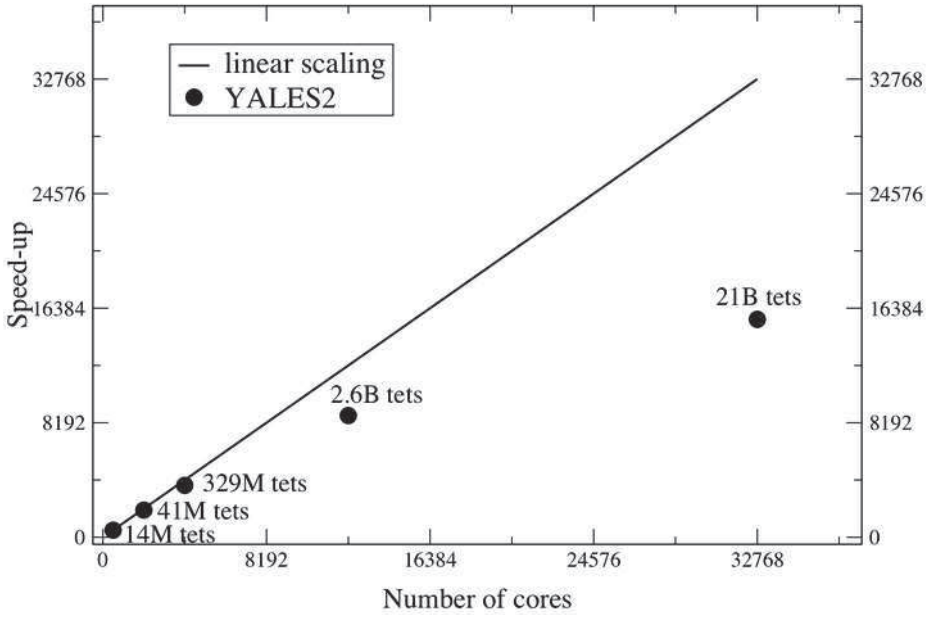

FIGURE 4.4 – *Courbe de weak scaling du solveur YALES2, dressée en 2010.*

Le *weak scaling* est utilisé dans cette thèse pour comparer la scalabilité de deux solveurs linéaires, en ne considérant donc pas l'efficacité réduite globale de la simulation mais seulement celle du solveur linéaire.

Un autre type de mesure de scalabilité existe, nommé **strong scaling** par opposition au précédent, qui est dit "faible". Ce contraste entre "faiblesse" et "force" semble être moins indicatif de la qualité des mesures de scalabilité fournies que de la radicalité de l'approche. En effet, là où le *weak scaling* se base sur l'idée d'une taille de problème fixe par processeur, et donc d'une augmentation des ressources utilisées qui suit l'augmentation de la taille des problèmes, le *strong scaling* s'intéresse aux performances d'un programme effectuant la même simulation sur des nombres de processeurs de plus en plus grands.

Cette fois-ci, c'est donc directement le ratio Q_i qui doit être représenté en fonction du nombre de processeurs pour obtenir une courbe similaire à la courbe de *weak scaling* décrite ci-dessus. Cependant, cette approche atteint rapidement ses limites, puisqu'à partir d'un certain nombre de processeurs, le volume de données sur chaque processeur devient très réduit, ce qui détériore les vitesses de calcul indépendamment de la qualité du programme utilisé. Plus précisément, les temps de recopie de données entre la mémoire vive et la mémoire morte d'un processeur deviennent prépondérants sur les temps effectifs de calcul lorsque le volume de données affecté à chaque processeur est trop faible.

L'approche du *weak scaling* reste donc la plus utilisée, en particulier dans le domaine des simulations physiques. On peut même estimer une fourchette de valeurs "idéale" pour le nombre d'éléments à affecter à chaque processeur d'un supercalculateur donné, qui permet de choisir le nombre de processeurs à utiliser en fonction de la taille du problème.

4.2 Méthodes de décomposition de domaine

Pour optimiser le fonctionnement d'un programme destiné à la CFD en parallèle, il faut entre autres veiller à ce que la répartition du domaine de calcul sur les différents processeurs soit équitable. En effet, à chaque communication globale, les processeurs devront s'attendre les uns les autres, ce qui fait que la simulation se fera globalement à la vitesse du processeur le plus lent. Une attention toute particulière doit donc être portée à la partition initiale du maillage en vue de sa distribution sur les processeurs, ce que nous abordons dans un premier temps. Cependant, cette décomposition simple n'est pas optimale, pour des raisons qui sont développées ensuite. Ce constat a mené, dans le cas du solveur YALES2, à l'adoption dès le départ d'une méthodologie de double décomposition du domaine, que nous présentons plus loin dans cette section. Les explications données dans ces parties ne s'appliquent qu'au cas où un maillage non préalablement partitionné est fourni au solveur ; nous traitons brièvement en fin de section le cas où le maillage fourni au code est prépartitionné.

4.2.1 Single Domain Decomposition

La décomposition de domaine simple, désignée par l'acronyme SDD[8], consiste en une simple partition du domaine en un nombre de sous-domaines égal au nombre de processeurs utilisés pour le calcul, comme illustré par la figure 4.5.

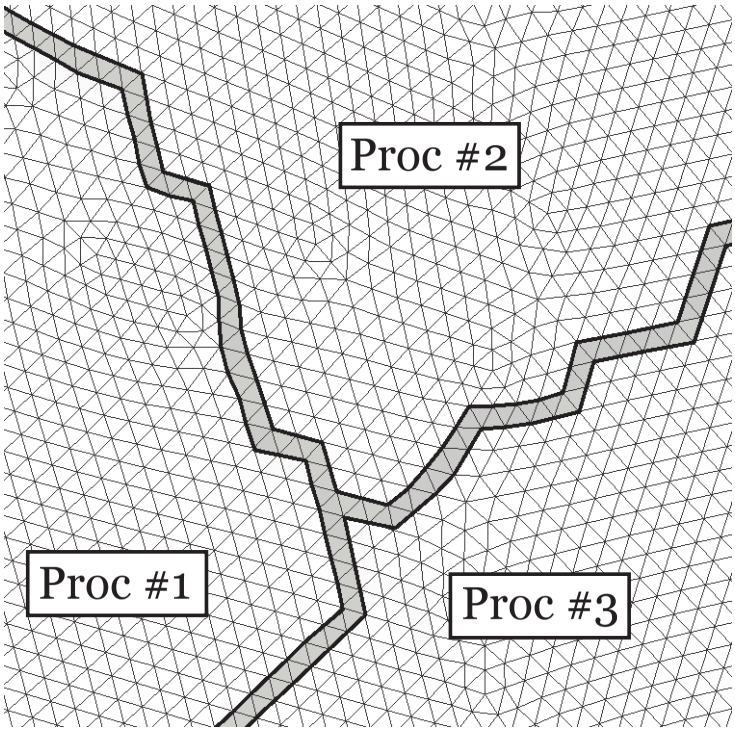

FIGURE 4.5 – *Décomposition de domaine simple d'un maillage non structuré en deux dimensions. Les cellules entre traits épais sont celles contenant des portions de volumes de contrôle prises en charge par différents processeurs.*

Pour cette décomposition, on peut avoir recours à des bibliothèques de partitionnement comme METIS ou Scotch. Puisque ces bibliothèques prennent en charge le partitionnement de graphe, il faut leur fournir le graphe du maillage à partitionner, c'est-à-dire un graphe dans lequel chaque noeud représente un noeud du maillage, et où deux noeuds sont reliés par une arête si et seulement si les deux noeuds du maillage appartiennent à un même élément (voir partie 2.2.2.1), comme représenté en figure 4.6.

Les poids des noeuds du graphe peuvent ici être tous identiques, puisqu'on peut considérer dans un premier temps que la charge de calcul correspondant à chaque cellule est identique. Les poids des arêtes du graphe imposeront une contrainte au partitionneur, qui devra répartir équitablement la charge[9] entre les différentes partitions du graphe tout en

[8] De l'anglais *Single Domain Decomposition*.

[9] La charge désigne le poids total d'un graphe ou d'un sous-graphe, c'est-à-dire la somme des poids de

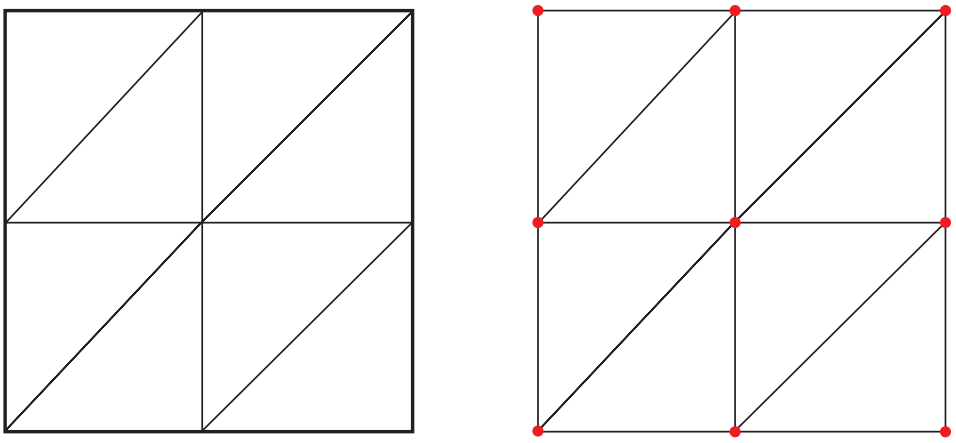

FIGURE 4.6 – *Représentation d'un maillage non structuré en deux dimensions (à gauche) et du graphe correspondant (à droite).*

minimisant le poids de coupe du graphe, qui est la somme des poids des arêtes qui relient des noeuds appartenant à des parties différentes du graphe. Cette dernière notion et son incidence sur la partition d'un graphe sont illustrées par la figure 4.7. Le poids d'une arête entre deux noeuds est donc défini en fonction du volume de données à communiquer dans le cas où les deux noeuds correspondants seraient affectés à deux processeurs différents.

Le choix des stratégies de partitionnement de graphe adoptées par les partitionneurs actuels représente un sujet très vaste, qui sera couvert plus en détail dans la section 7.2.1. Le fonctionnement des partitionneurs utilisés ne sera donc pas abordé dans cette partie.

Une fois qu'une partition du graphe a été créée par la bibliothèque utilisée, le domaine de calcul est séparé et la mémoire de chaque processeur reçoit les données correspondant au sous-domaine qui lui est affecté. Des communicateurs[10] externes sont également créés pour la communication de données entre deux processeurs "voisins", notamment le remplissage des cellules fantômes de chacun.

Cette façon simple de séparer un domaine de calcul entre plusieurs processeurs représente néanmoins une entrave à l'adaptabilité du programme. En effet, toute opération qui modifie, en cours de simulation, le maillage sur le domaine ou sa répartition sur les processeurs, requiert alors un important temps de calcul.

- L'équilibrage dynamique des charges de calcul, qui est abordé dans les perspectives de cette thèse, est essentiel pour certaines simulations lors desquelles les charges

ses noeuds. Le poids des arêtes n'est pas considéré à ce niveau.

[10]L'usage de cette terminologie dans ce qui suit est différent de celui qui en est fait dans la terminologie MPI. Un communicateur est ici une structure de données qui facilite la communication entre deux sous-domaines voisins, notamment en imposant un ordre commun aux cellules situées à la frontière entre ces deux sous-domaines. Cet accord sur l'ordre des cellules entre le processeur auquel elles sont affectées et le processeur qui les a déclaré en tant que fantômes est déjà évoqué à la fin de la partie 4.1.3.1.

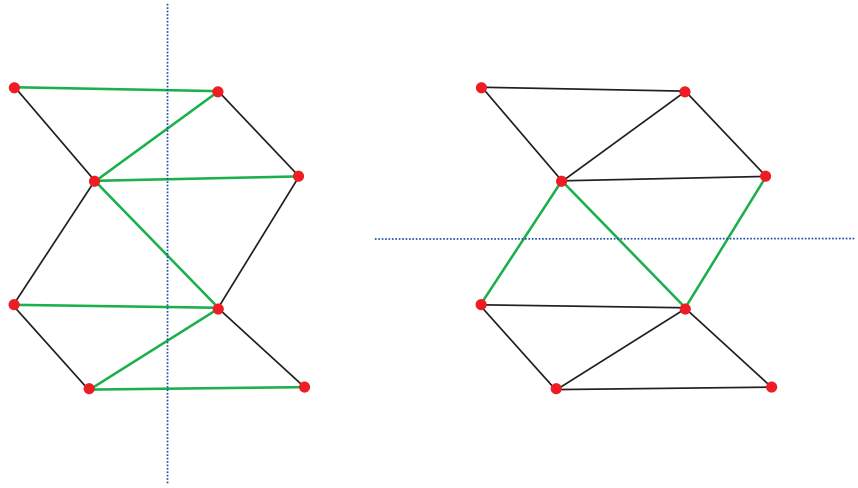

FIGURE 4.7 – *Deux bissections possibles, représentées par des traits pointillés bleus, d'un même graphe. Dans chaque cas, le poids de la coupe est la somme des poids des arêtes représentées en vert. En considérant que les poids de tous les noeuds sont égaux, et que les poids de toutes les arêtes sont égaux, chacune des deux coupes répartit idéalement la charge ; en revanche, le poids de coupe est clairement inférieur pour la coupe représentée à droite, qui est la bissection idéale pour ce graphe.*

de calcul correspondant aux différentes cellules du maillage changent au cours du temps. On peut citer, par exemple, les simulations dans lesquelles des particules lagrangiennes, injectées en un endroit précis du domaine, se dispersent ensuite, portées par l'écoulement ; ou la simulation d'une ignition, lors de laquelle la résolution des équations liées aux processus chimiques est particulièrement coûteuse autour du front de flamme. Pour rééquilibrer les charges de calcul des différents processeurs au sein d'un programme utilisant une stratégie de partitionnement en SDD, il faut de nouveau appeler le partitionneur utilisé pour un graphe très vaste, puis être particulièrement vigilant lors de la migration de données, et enfin reconstruire *ex nihilo* tous les communicateurs externes.

- La possibilité d'un raffinement local de maillage peut représenter à la fois un avantage pratique et un gain de temps considérable sur certaines simulations, par exemple pour l'étude d'écoulements autour de profils d'aile d'avion ou de pale de turbine ; un manque de précision de la simulation en des lieux où les phénomènes turbulents sont particulièrement intenses, notamment à proximité du point d'impact du profil, peut être compensé par un raffinement local qui n'augmentera que très raisonnablement le nombre de noeuds du maillage, au lieu de nécessiter un raffinement homogène du maillage ou la création d'un nouveau maillage. Pour que ce raffinement local présente un réel avantage au niveau du temps de calcul, le maillage doit être redistribué, sans quoi les processeurs concernés par ce raffinement dicteront leur vitesse de cal-

cul à tous les autres, ce qui fera que la simulation ne sera probablement pas plus rapide que si le raffinement avait été global. Le rééquilibrage des charges de calcul est donc indispensable à la mise en oeuvre du raffinement local, avec les difficultés déjà mentionnées vis-à-vis de la stratégie de décomposition simple du domaine.

La gestion de la mémoire est, quant à elle, un inconvénient immédiat de la SDD. En effet, la transmission de données entre la mémoire vive et la mémoire cache d'un processeur peut facilement devenir erratique lors de certains calculs, en particulier lors de l'application d'opérateurs différentiels. Il serait plus souhaitable, de ce point de vue, que les opérations sur les données soient appliquées par blocs contigus, de façon à ce que toutes les données nécessaires à un calcul sur un bloc de cellules puissent être chargées en cache simultanément. C'est en appliquant ce principe de conception que la double décomposition de domaine répond aux besoins mis en évidence par la SDD.

4.2.2 Double Domain Decomposition

Utilisée dans le code YALES2 pour échapper aux inconvénients de la SDD, la double décomposition de domaine ou DDD est illustrée par la figure 4.8. Une partition du maillage est effectuée dans un premier temps comme dans le cas de la SDD, et la migration des données permet la création d'un groupe de cellules sur chaque processeur, qui correspond au sous-domaine affecté à ce processeur, et des communicateurs externes correspondants. Dans un second temps, chaque processeur crée le graphe correspondant à son sous-domaine et appelle une procédure de partitionnement pour scinder ce graphe, en imposant comme contrainte le nombre de cellules par groupe souhaité. Le macro-groupe de cellules de chaque processeur peut alors être partitionné en plusieurs groupes dont la taille est la plus proche possible de la taille souhaitée. Entre ces groupes sont mis en place des communicateurs internes, dont le rôle est similaire à celui des communicateurs externes, mais qui ne seront pas concernés par la mise en place de communications MPI. Cette logique de conception est à envisager sous l'angle de la gestion de la mémoire cache, qui est évoqué dans le paragraphe précédent.

Le surcoût de calcul que représente cette double décomposition, par rapport à une SDD, est d'une part justifié par les avantages pratiques de cette décomposition, et d'autre part compensé par le gain de temps d'exécution sur certaines opérations coûteuses. Concernant les avantages pratiques, en plus de la possibilité d'adapter la taille des groupes de cellules à la capacité de la mémoire cache des processeurs utilisés, on peut citer le fait que ces groupes fournissent un maillage grossier du domaine pour des solveurs linéaires bi-niveaux tels que le solveur CG avec déflation présenté dans le chapitre suivant. Les structures de données du programme, adaptées à la méthodologie DDD, permettent de traiter facilement les opérations à effectuer sur le niveau grossier. D'autre part, deux éléments voisins de ce maillage grossier sont reliés entre eux par toutes les paires considérées par le communicateur interne entre ces deux groupes, et seulement par elles, ce qui rend plus directes certaines opérations sur le maillage grossier.

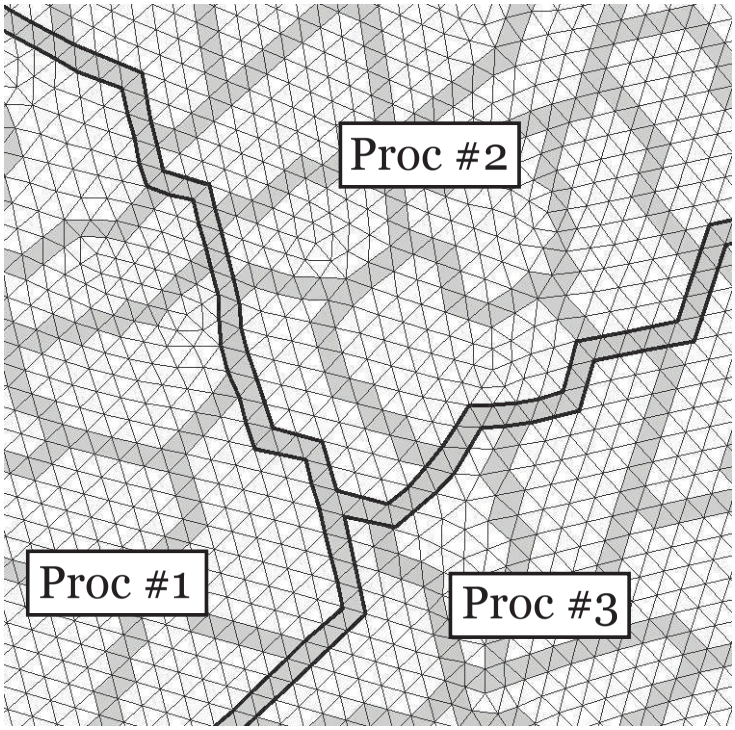

FIGURE 4.8 – *Double décomposition de domaine d'un maillage non structuré en deux dimensions. Les cellules entre traits épais sont celles contenant des portions de volumes de contrôle prises en charge par différents processeurs, et sont impliquées dans les communicateurs externes. Les cellules grisées se trouvent à la frontière de groupes différents, et sont impliquées dans des communicateurs internes.*

Au sujet du gain de temps sur certaines opérations, on peut reprendre les exemples du rééquilibrage de charges et du raffinement local de maillage, déjà cités au sujet de la SDD.

- Pour rééquilibrer les charges de calcul entre les différents processeurs en cours d'exécution, le graphe à envoyer au partitionneur sera de taille bien plus réduite que le graphe du maillage. Dans ce nouveau graphe, chaque noeud représentera un groupe de cellules, et non plus une seule cellule. Une fois la nouvelle partition déterminée, il ne faudra plus faire migrer des cellules isolées mais des groupes. Lors de cette opération, la construction de la nouvelle connectivité entre les processeurs sera plus aisée, tout du moins au niveau des faces, où des communicateurs internes deviendront des communicateurs externes, et vice-versa. La figure 4.9 illustre ce phénomène dans le cas simplifié de la migration d'un groupe de cellules dans une bipartition. La DDD supprime donc le besoin d'une reconstruction intégrale de la connectivité lors du rééquilibrage des charges de calcul.

- Le raffinement local du maillage peut être effectué directement sur des groupes. Un tel raffinement ne nécessitera que la reconstruction des communicateurs internes et externes associés à ce groupe de cellules. En outre, la procédure permettant de scinder un groupe qui a été raffiné, puis de reconstruire les connectivités correspondantes, est en plus facile à coder à ce stade, puisqu'une procédure similaire a permis de séparer chaque macro-groupe lors du partitionnement initial du maillage. Enfin, le repartitionnement du maillage peut être effectué comme indiqué ci-dessus.

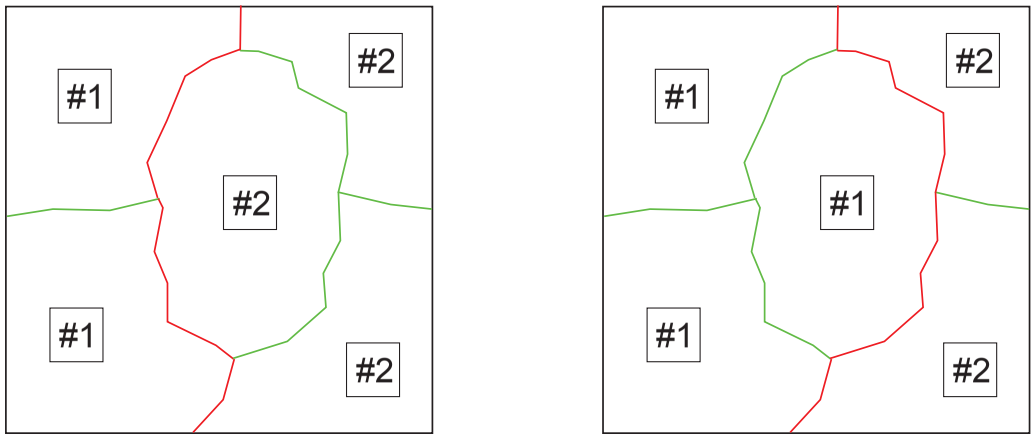

FIGURE 4.9 – *Migration d'un groupe de cellules du deuxième au premier processeur dans le cas d'une bipartition. Les traits verts représentent les faces impliquées dans des communicateurs internes de part et d'autre de la frontière, et les traits rouges les faces impliquées dans des communicateurs externes sur chaque processeur. La migration du groupe central revient à déplacer les données de ce groupe d'un processeur à l'autre et à recréer partiellement la connectivité. Les communicateurs permettant de lier les faces aux éléments des groupes de cellules voisins sont relativement aisés à reconstruire.*

Précisons également, pour la clarté de ce qui suit, que le partitionnement se fait par groupes de cellules tandis que la méthode de déflation, entre autres opérations effectuées sur les données, considère des groupes de volumes de contrôle. Cette distinction sera faite de manière explicite dans le reste de ce manuscrit. Ajoutons également qu'à l'heure actuelle, le partitionnement global pour la répartition des cellules sur les processeurs est effectué grâce à la bibliothèque ParMETIS, tandis que le partitionnement local en groupes de cellules sur chaque processeur est réalisé via la bibliothèque METIS [34].

4.2.3 Partition d'un maillage prépartitionné

Un maillage prépartitionné est enregistré sous la forme de plusieurs fichiers séparés, contenant chacun les informations nécessaires à la construction du maillage d'un sous-domaine

du domaine de calcul et à la prise en compte de la connectivité avec les autres parties du maillage.

YALES2 est capable de prendre en charge un maillage prépartitionné à partir du moment où le nombre de blocs n_b du maillage est un diviseur du nombre de processeurs n_{procs} utilisés pour la simulation. En considérant $r_b = n_{procs}/n_b$ le ratio entre le nombre de processeurs et le nombre de blocs du maillage, le premier bloc du maillage est stocké temporairement sur le processeur de rang 0, le second bloc sur le processeur de rang r_b, le troisième bloc sur le processeur de rang $2r_b$, et ainsi de suite. Chaque processeur de rang multiple de r_b appelle alors une procédure de partitionnement pour scinder le maillage qui lui est attribué en r_b parties et redistribue ces parties aux processeurs de rangs immédiatement supérieurs.

Par exemple, si une simulation sur 8 processeurs doit être faite à partir d'un maillage enregistré en deux blocs, le processeur de rang 0 lira le premier bloc, le partitionnera en quatre parties et distribuera trois de ces parties aux processeurs de rangs 1, 2 et 3 ; le processeur de rang 4, quant à lui, partitionnera le deuxième bloc et en distribuera des parties aux processeurs de rangs 5 à 7.

4.3 Parallélisation de l'algorithme des Gradients Conjugués

Le problème de l'utilisation d'algorithmes itératifs en parallèle est évoquée depuis de nombreuses années, notamment par Van Der Vorst depuis les années 1990 [82]. Dans le cas des algorithmes CG et PCG, les étapes encadrées dans leurs algorithmes respectifs en pages 53 et 54 du présent manuscrit représentent des calculs pour lesquels, lors de l'exécution parallèle du solveur, des réductions MPI sont requises. Pour connaitre la véritable valeur du produit scalaire, il faudra sommer les contributions locales et renvoyer la valeur globale à tous les processeurs, c'est-à-dire établir un protocole de communication entre tous les processeurs.

Chacun des algorithmes 1 et 2 nécessite en théorie l'établissement de trois réductions MPI par itération, puisqu'une troisième réduction est requise pour vérifier si le critère de convergence est atteint ou non. Une astuce très simple permet d'inclure cette dernière réduction avec une des deux précédentes ; en effet, chaque processeur peut créer une valeur c_i égale à 1 si le critère est respecté sur le sous-domaine dont il a la charge, et 0 sinon. Si l'on communique cette valeur en même temps que la composante locale d'un produit scalaire, la nouvelle réduction retourne le produit scalaire global d'une part, et la somme des c_i d'autre part. Si cette somme est égale au nombre de processeurs, c'est que la convergence globale est atteinte.

Pour ne plus nécessiter qu'une seule réduction par itération, une des méthodes proposées par D'Azevedo *et al.* [15] consiste à réécrire σ_k en utilisant la propriété de récurrence $p_k = r_k + \beta_k p_{k-1}$. On obtient

$$\begin{aligned}\sigma_k &= p_k^T A p_k \\ &= (r_k + \beta_k p_{k-1})^T A (r_k + \beta_k p_{k-1}) \\ \sigma_k &= r_k^T A r_k + 2\beta_k r_k^T A p_{k-1} + \beta_k^2 \sigma_{k-1} .\end{aligned} \tag{4.3}$$

On peut développer d'une façon similaire le produit scalaire ρ_k pour obtenir

$$\begin{aligned}\rho_k &= r_k^T r_k \\ &= r_k^T (r_{k-1} - \alpha_k A p_{k-1}) \\ \rho_k &= -\alpha_k r_k^T A p_{k-1} .\end{aligned} \tag{4.4}$$

On obtient donc $r_k^T A p_{k-1} = -\rho_k/\alpha_k$ d'où

$$\begin{aligned}\sigma_k &= r_k^T A r_k - 2\frac{\beta_k \rho_k}{\alpha_k} + \beta_k^2 \sigma_{k-1} \\ &= r_k^T A r_k - 2\frac{\beta_k \rho_k \sigma_{k-1}}{\rho_{k-1}} + \beta_k^2 \sigma_{k-1} \\ \sigma_k &= r_k^T A r_k - 2\beta_k^2 \sigma_{k-1} + \beta_k^2 \sigma_{k-1} .\end{aligned} \tag{4.5}$$

Le réarrangement proposé introduit donc la valeur

$$\lambda_k = r_k^T A r_k , \tag{4.6}$$

que l'on peut calculer en même temps que ρ_k, et à partir de laquelle on exprime σ_k sous la forme

$$\sigma_k = \lambda_k - \beta_k^2 \sigma_{k-1} . \tag{4.7}$$

La méthode des Gradients Conjugués est alors réarrangée sous la forme donnée par l'algorithme 3, qui ne nécessite plus qu'une communication globale par itération.

Algorithme 3 Algorithme réarrangé de la méthode des Gradients Conjugués

ENTRÉES : A, b, x_0

$p_0 = r_0 = b - Ax_0$

$\boxed{\{\rho_0, \sigma_0\} = \{r_0^T r_0, p_0^T A p_0\}}$

Pour $k = 0, 1, 2 \ldots$ jusqu'à obtenir la convergence souhaitée **Faire**

$\alpha_{k+1} = \frac{\rho_k}{\sigma_k}$

$x_{k+1} = x_k + \alpha_{k+1} p_k$

$r_{k+1} = r_k - \alpha_{k+1} A p_k$

$\boxed{\{\rho_{k+1}, \lambda_{k+1}\} = \{r_{k+1}^T r_{k+1}, r_{k+1}^T A r_{k+1}\}}$

$\beta_{k+1} = \frac{\rho_{k+1}}{\rho_k}$

$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$

$\sigma_{k+1} = \lambda_{k+1} - \beta_{k+1}^2 \sigma_k$

Fin Pour

Chapitre 5

L'algorithme DPCG

Lors de certaines simulations d'écoulements incompressibles, entre autres celles d'écoulements dans des géométries complexes, la résolution à chaque pas de temps de l'équation de Poisson pour la pression peut représenter la majeure partie du temps de simulation global. Le développement de solveurs linéaires de plus en plus performants devient, dans ce contexte, une étape indispensable de l'amélioration des outils de simulation. Bien que l'algorithme des Gradients Conjugués préconditionné soit performant pour des simulations de taille raisonnable, les résultats de simulation sur des maillages de taille importante montrent clairement le besoin de développer un solveur plus efficace, ce que nous verrons dans une première section via l'étude de résultats obtenus sur des simulations du brûleur PRECCINSTA.

L'approche de la méthode de déflation consiste à considérer séparément certaines composantes de la solution, de façon à accélérer la convergence numérique d'un algorithme quelconque de résolution itérative de systèmes linéaires. Son principe, ses différences avec les méthodes multigrilles (elles aussi couramment utilisées en simulation numérique) et sa mise en oeuvre au sein d'un algorithme de type Gradients Conjugués sont présentés dans une deuxième section. L'application du nouveau solveur ainsi obtenu au cas-test étudié en début de chapitre met en évidence à la fois l'avantage de cette méthode sur la méthode PCG classique et les nouveaux goulots d'étranglement numériques qu'elle fait apparaitre, ce qui nous orientera vers les nouvelles pistes de développement explorées dans le contexte de cette thèse.

5.1 Limitations du Gradient Conjugué

Nous nous intéressons dans cette section à la simulation d'un écoulement incompressible dans une géométrie complexe. Il s'agit du brûleur étudié dans le cadre du projet PRECCINSTA, dont nous détaillons les caractéristiques dans un premier temps. Les simulations sur des maillages de différentes tailles sont réalisées par le code YALES2 grâce à l'utilisation du solveur linéaire PCG, présenté en section 3.2 et réarrangé selon les principes

introduits dans le chapitre précédent, pour la résolution de l'équation de Poisson pour la pression à chaque pas de temps. L'étude des temps de calcul pour ces différentes simulations met en évidence l'importance du choix du solveur utilisé et l'intérêt que nous avons à vouloir l'améliorer.

5.1.1 Configuration

Le cas-test considéré ici, que nous nommerons tout simplement **Preccinsta** dans les sections de résultats de la partie 6, correspond à la simulation d'écoulements à température constante dans le brûleur PRECCINSTA. Le nom de ce brûleur est l'acronyme imparfait de *PREdiction and Control of Combustion INSTAbilities for industrial gas turbines* ; cette configuration a été étudiée en profondeur, aussi bien expérimentalement [52] que numériquement [47], et régulièrement utilisée comme une base de données de validation de modèles de combustion [24, 64, 55] ou de méthodes numériques appliquées à la résolution des équations de Navier-Stokes [54].

La géométrie considérée, entièrement maillée pour les simulations, comporte un plénum, un dispositif d'injection tourbillonnant ou *swirler*, et une chambre de combustion rectangulaire avec, en sortie, un cône convergent vers un tube de sortie cylindrique. Cette géométrie est représentée en figure 5.1. L'air à température ambiante est pompé à une vitesse de 24,5 $m.s^{-1}$ vers le plénum, de largeur 79 mm, et le méthane est injecté à la même vitesse. par douze tubes d'un millimètre de diamètre. Ils traversent le *swirler*, qui est constitué de douze fentes inclinées à 40 degrés, avant d'entrer dans la chambre de combustion de dimensions 85 mm, 85 mm et 114 mm via un court tube cylindrique de diamètre 27,85 mm. Les gas brûlés sortent de la chambre de combustion par un cône convergent qui les dirige vers un court tube de sortie de diamètre 40 mm. Le nombre de Reynolds turbulent est d'environ 40000. Dans les simulations décrites ci-après, nous considérons uniquement un écoulement non réactif à température constante dans cette géométrie.

5.1.2 Simulations et résultats

Cette sous-section se base sur des simulations effectuées sur deux maillages : le premier, nommé **MP1** dans ce qui suit, compte environ 13,7 millions d'éléments tétraédriques, et le second, nommé **MP2**, compte environ 110 millions d'éléments tétraédriques. Les simulations sur ces deux maillages sont effectuées sur respectivement 64 et 512 processeurs. On ne considère ici que le premier pas de temps de la simulation. Deux critères de convergence différents sont adoptés pour le solveur linéaire, à savoir 10^{-8} et 10^{-10}. La table 5.1 concerne le nombre d'itérations du solveur linéaire PCG pour la résolution de l'équation de Poisson à ce pas de temps.

Il apparait clairement sur ces exemples que le nombre d'itérations du solveur peut devenir prohibitif, particulièrement pour des maillages de taille importante. D'autre part,

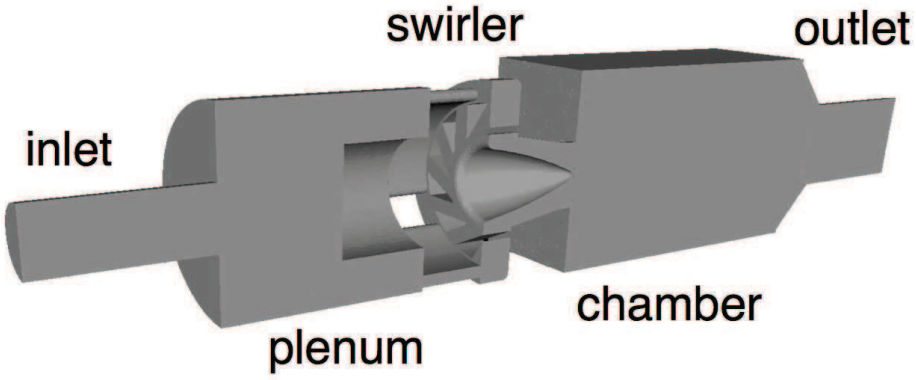

FIGURE 5.1 – *Section longitudinale du brûleur PRECCINSTA.*

Maillage	MP1		MP2	
Critère de convergence	10^{-8}	10^{-10}	10^{-8}	10^{-10}
Itérations	983	1121	12374	16717

TABLE 5.1 – *Nombres d'itérations sur la simulation du premier pas de temps d'un écoulement incompressible dans le brûleur PRECCINSTA sur deux maillages différents et avec deux critères de convergence différents, pour le solveur linéaire PCG.*

le nombre d'itérations sur le maillage fin est multiplié par environ 13 à 15 lorsque le maillage est raffiné, c'est-à-dire lorsque le nombre d'éléments est multiplié par 8.

Comme le temps de calcul requis pour la résolution de l'équation de Poisson pour la pression dépend fortement du nombre d'itérations du solveur, le coût de cette résolution par l'algorithme de Gradients Conjugués préconditionné classique n'augmente pas proportionnellement à la taille du maillage, mais bien plus rapidement. L'algorithme n'est donc pas directement adapté à de tels calculs.

Pour que le nombre d'itérations requises soit moins dépendant de la taille du maillage, on peut utiliser la méthode présentée en section suivante, à savoir l'application "géométrique" commune de la méthode de déflation. Cette méthode établit un W-cycle, comparable dans une certaine mesure à celui utilisé par les méthodes multigrilles mais différent sous plusieurs aspects également évoqués dans ce qui suit, entre le maillage fin et un maillage grossier.

5.2 La méthode de déflation

Tandis que l'idée d'utiliser le préconditionnement sur la méthode CG est apparue quelques années à peine après la méthode elle-même, avec par exemple les travaux de Engell *et al.* en 1959 [18], l'idée de la méthode de déflation n'apparait qu'en 1987 avec les travaux de

Nicolaides [57].

Bien que visant le même but initial, à savoir la réduction du nombre d'itérations pour la résolution d'un système linéaire mal conditionné, Nicolaides n'emploie pas les mêmes moyens que ses prédécesseurs ; là où le préconditionnement vise à multiplier le système $Ax = b$ par une matrice "proche de A^{-1}" de façon à diminuer le conditionnement du système avant sa résolution, l'idée première de la déflation est de projeter le problème $Ax = b$ sur un sous-espace de $\mathbb{R}^n$. L'expression mathématique de cette méthode permet d'en faire une méthode à deux niveaux pour l'application, entre autres, aux problèmes associés à la simulation d'écoulements incompressibles [3, 56].

5.2.1 Développement de la méthode

Il est intéressant de noter que la méthode du Gradient Conjugué avec Déflation (DCG), développée dans un premier temps par Nicolaides [57] à partir d'une expression de l'algorithme CG faisant apparaitre une récurrence à trois termes, a été dérivée quelques années plus tard par Saad et Yeung pour la version courante de l'algorithme CG, avec la collaboration des chercheurs français Erhel et Guyomarc'h [68] et en s'appuyant sur leurs recherches antérieures concernant le développement de méthodes de Gradient Conjugué dites "augmentées" [19].

5.2.1.1 Récurrence à trois termes

L'idée de base de la méthode développée par Nicolaides est d'ôter une partie des composantes du résidu à chaque itération. A cet effet, une matrice W est créée, de dimension $n \times m$, avec $m \ll n$; les colonnes de W forment une famille libre de $\mathbb{R}^n$, c'est-à-dire que $S_m = \mathrm{Span}\,(w_1, \ldots, w_m)$ est un sous-espace de R^n de dimension m. Ces composantes seront ôtées au résidu à chaque itération, de façon à résoudre le système d'équations sur un espace de dimension $n - m$ A-orthogonal à S_m.

La version de l'algorithme CG considérée par Nicolaides est celle exploitant la relation de récurrence d'ordre 2

$$x_{k+1} = [\omega_k x_k + (1 - \omega_k)\, x_{k-1}] + \mu_{k+1} r_k\,, \tag{5.1}$$

c'est-à-dire que la nouvelle solution est une combinaison convexe des deux solutions précédentes, à laquelle s'ajoute une composante colinéaire au résidu à l'itération précédente. Précisons que cette expression est obtenue en faisant disparaitre les directions de descente p_k de l'expression de l'algorithme CG [26], ce qui transforme l'itération sur les x_i de l'algorithme 1, page 53, en

$$x_{k+1} = \left[\left(1 + \frac{\beta_k}{\alpha_k}\right) x_k - \frac{\beta_k}{\alpha_k} x_{k-1}\right] + \alpha_{k+1} r_k\,. \tag{5.2}$$

On obtient directement une expression similaire du résidu à la même itération en développant $r_{k+1} = b - Ax_{k+1}$ et en utilisant le fait que la combinaison linéaire de x_k et x_{k-1} qui y figure est une combinaison convexe. On obtient l'expression

$$r_{k+1} = [\omega_k r_k + (1 - \omega_k)\, r_{k-1}] + \mu_{k+1} A r_k \,. \tag{5.3}$$

La déflation consiste à projeter r_k sur le sous-espace orthogonal à S_m, c'est-à-dire à lui enlever ses composantes selon les vecteurs $\{w_i\}_{i=1}^m$, avant d'effectuer l'itération décrite par l'équation (5.1). Les coefficients d_k^i du vecteur d_k sont choisis de façon à minimiser la A-norme de $r_k - \sum_{i=1}^m d_k^i w_i$, c'est-à-dire que l'on veut résoudre

$$\min_{d_k} \|r_k - W d_k\|_A \,. \tag{5.4}$$

La solution de ce problème de minimisation est celle du système linéaire à m inconnues

$$\hat{A} d_k = W^T A r_k \,, \tag{5.5}$$

où

$$\hat{A} = W^T A W \,, \tag{5.6}$$

projection de A sur le sous-espace S_m, est une matrice symétrique semi-définie positive si A l'est : ce système peut être résolu lui aussi par une méthode itérative telle que la méthode CG, bien que Nicolaides lui-même n'évoque pas directement ce sujet. On transforme alors l'itération (5.1) en

$$x_{k+1} = [\omega_k x_k + (1 - \omega_k)\, x_{k-1}] + \mu_{k+1} (r_k - W d_k) \,, \tag{5.7}$$

d'où il découle que l'itération sur le résidu, décrite par l'équation (5.3), devient à son tour

$$r_{k+1} = [\omega_k r_k + (1 - \omega_k)\, r_{k-1}] + \mu_{k+1} A (r_k - W d_k) \,. \tag{5.8}$$

Mentionnons qu'il s'agit bien d'une méthode de projection, en ce que, si le résidu initial est orthogonal à S_m, alors tous les résidus le seront. En effet, à partir des équations (5.8) et (5.5), on obtient

$$W^T r_{k+1} = \omega_k W^T r_k + (1 - \omega_k)\, W^T r_{k-1} \,. \tag{5.9}$$

La nullité de $W^T r_0$ rend d_0 nul, et $W^T r_1 = W^T r_0 - \mu_1 W^T A r_0 = W^T r_0 - \mu_1 \hat{A} d_0 = 0$. Une simple récurrence sur (5.9) permet de conclure. On peut d'ailleurs expliciter les projecteurs utilisés implicitement par la méthode de déflation, à savoir

$$Q = W \hat{A}^{-1} W^T \text{ et } P^\perp = I - AQ \,. \tag{5.10}$$

La matrice $P^{\perp}$ est donc la matrice de projection d'un vecteur de $\mathbb{R}^n$ quelconque sur le sous-espace de $\mathbb{R}^n$ A-orthogonal à S_m. Il s'agit bien d'un projecteur au sens strict du terme, puisqu'un calcul rapide nous donne $\left(P^{\perp}\right)^2 = P^{\perp}$.

Pour avoir un r_0 orthogonal à S_m, Nicolaides propose un calcul effectué à partir d'un vecteur arbitraire. En choisissant de partir du vecteur nul, cette méthode revient à choisir Qb, que l'on peut également écrire

$$x_0 = Wd_{-1} \text{ avec } \hat{A}d_{-1} = W^T b\,. \tag{5.11}$$

Il est clair que $W^T r_0 = W^T b - W^T AWd_{-1} = 0$. En posant la condition que chaque résidu soit orthogonal à tous les précédents, condition qui est respectée dès que $r_1 \perp r_0$ et $\forall k \in \mathbb{N}^*, r_{k+1} \perp r_k$, on obtient

$$\mu_{k+1} = \frac{r_k^T r_k}{r_k^T APr_k}\,, \tag{5.12}$$

que l'on peut calculer en pratique en résolvant au préalable l'équation $\hat{A}d_k = W^T Ar_k$, d'où il découle que

$$\mu_{k+1} = \frac{r_k^T r_k}{r_k^T A\left(r_k - Wd_k\right)}\,. \tag{5.13}$$

Quant au calcul de ω_k, il est inchangé par rapport à l'algorithme de départ.

5.2.1.2 Gradient Conjugué Augmenté

Un rapport de recherche publié en 1997 par deux chercheurs de l'INRIA, Erhel et Guyomarc'h [19], présente une façon d'accélérer la convergence de l'algorithme des Gradients Conjugués dans le cas de la résolution de plusieurs systèmes linéaires dont la matrice est identique. La résolution d'un système $Ay = c$ par la méthode CG est effectuée en m itérations, ce qui fait que $m+1$ directions de descente $\{w_0, w_1, \ldots, w_m\}$ sont empruntées. La première idée développée par Erhel et Guyomarc'h est de déterminer, pour la résolution du système $Ax = b$, une solution initiale orthogonale aux directions de descente déjà empruntées et une première direction de descente qui leur soit A-orthogonale, soit, avec $W = (w_0, \ldots, w_m)$,

$$W^T r_0 = 0\ ;\ W^T Ap_0 = 0\,. \tag{5.14}$$

Le choix du x_0 permettant de respecter la première condition de (5.14) est le même que celui donné pour la déflation du Gradient Conjugué par l'équation (5.11), à la différence près qu'ici, les vecteurs w_i sont A-orthogonaux les uns aux autres, ce qui fait que $\hat{A}$ est diagonale et que x_0 se calcule directement. De même, on a $p_0 = r_0 - Wd_0$ avec d_0 solution de l'équation (5.5), ce qui peut être directement écrit $p_0 = Pr_0$. La méthode ainsi créée,

qui n'est pas une méthode CG *stricto sensu* à cause de la différence dans le choix de la première descente de direction, est nommée InitCG par les auteurs.

Cette idée est ensuite poussée plus loin, de façon à ce que des conditions d'orthogonalité similaires à (5.14) soient préservées tout au long du calcul. On exige donc que le résidu r_{k+1} soit A-orthogonal, non seulement aux résidus précédents $\{r_0, r_1, \ldots, r_k\}$ comme pour le CG classique, mais également à tous les résidus $\{s_0, s_1, \ldots, s_m\}$ calculés lors de la résolution préalable du système $Ay = c$.

On peut montrer que, pour cela, la direction de descente p_{k+1} doit seulement être orthogonalisée par rapport à la direction de descente précédente p_k et au vecteur w_m. Au coefficient β_{k+1}, qui sert à rendre r_{k+1} A-orthogonal à p_k et dont l'expression est inchangée (voir équation (3.42)), s'ajoute un coefficient δ_{k+1} qui permet d'A-orthogonaliser r_{k+1} par rapport à w_m. La condition $\langle r_{k+1} - \delta_{k+1} w_m, w_m \rangle_A = 0$ donne directement

$$\delta_{k+1} = \frac{r_{k+1}^T A w_m}{w_m^T A w_m} \,. \tag{5.15}$$

De cette façon, l'itération sur la direction de descente devient

$$p_{k+1} = r_{k+1} + \beta_{k+1} p_k - \delta_{k+1} w_m \,. \tag{5.16}$$

La convergence de la méthode de Gradient Conjugué Augmentée ainsi créée, raccourcie en Aug-CG par les auteurs, est asymptotiquement plus rapide que celle de la méthode des Gradients Conjugués. En outre, l'apport de Aug-CG par rapport à InitCG est clairement montré par la comparaison des comportements "sur le long terme" des deux algorithmes.

5.2.1.3 Récurrence à deux termes

En 1998, moins d'un an après la création de la méthode Aug-CG, une nouvelle version du Gradient Conjugué avec déflation nait de la collaboration entre, d'une part Saad et Yeung de l'Université du Minnesota, et d'autre part Erhel et Guyomarc'h [68]. L'algorithme Aug-CG y est étendu, pour ne plus considérer les vecteurs $\{w_0, w_1, \ldots, w_m\}$ particuliers considérés par Erhel et Guyomarc'h mais un ensemble de vecteurs $\{w_1, w_2, \ldots, w_m\}$ quelconques, selon l'idée de Nicolaides. Ainsi, en lieu et place du terme

$$\frac{r_{k+1}^T A w_m}{w_m^T A w_m} w_m \tag{5.17}$$

de l'algorithme Aug-CG, qui correspond à la composante de r_{k+1} colinéaire à w_m, et qui doit lui être soustraite, on trouve

$$W \hat{A}^{-1} W^T A r_{k+1} \,, \tag{5.18}$$

soit un terme de la forme $W d_{k+1}$ où d_{k+1} est un vecteur à m composantes. Ce terme correspond à une somme pondérée des vecteurs $\{w_i\}_{i=1}^m$ qui, soustraite à r_{k+1}, garantira

que le résultat obtenu est A-orthogonal à ces m vecteurs. Remarquons que d_{k+1} est déterminé par la résolution de l'équation (5.5), directement héritée de Nicolaides. Le DCG s'écrit alors comme dans l'algorithme 4 ; en comprimant cet algorithme de façon à ne plus faire apparaitre explicitement les directions de descente p_k, on obtient exactement l'algorithme de Gradient Conjugué avec déflation développé par Nicolaides onze ans plus tôt. Cette formulation de la méthode de déflation appliquée aux Gradients Conjugués est plus proche de l'algorithme CG usuel et, par là-même, plus facilement compréhensible, grâce à l'apparition explicite des directions de descente.

Algorithme 4 Algorithme du DCG tel que défini par Saad, Yeung, Erhel et Guyomarc'h, pour la résolution de $Ax = b$ avec la matrice de déflation W.

Entrées : A, b, W
$\hat{A} \leftarrow W^T AW$
Résoudre $\hat{A}d_{-1} = W^T b$
$x_0 = Wd_{-1}$
$r_0 = b - Ax_0$
Résoudre $\hat{A}d_0 = W^T Ar_0$
$p_0 = r_0 - Wd_0$
Pour $k = 0, 1, 2 \ldots$ jusqu'à obtenir la convergence souhaitée **Faire**
$\alpha_{k+1} = \frac{r_k^T r_k}{r_k^T A(r_k - Wd_k)}$
$x_{k+1} = x_k + \alpha_{k+1} p_k$
$r_{k+1} = r_k - \alpha_{k+1} Ap_k$
$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
Résoudre $\hat{A}d_{k+1} = W^T Ar_{k+1}$
$p_{k+1} = r_{k+1} - Wd_{k+1} + \beta_{k+1} p_k$
Fin Pour

5.2.2 Variante A-DEF2

Dans un article de 2009 [75], Tang *et al.* fournissent une étude abstraite des méthodes à deux niveaux que sont la déflation, la décomposition de domaine et la méthode multigrilles, les ramenant toutes sur le plan théorique à la résolution d'un système linéaire de la forme

$$\mathcal{P}\mathcal{A}\mathbf{x} = \mathbf{b}\,, \tag{5.19}$$

les différences entre ces méthodes s'affirmant sur le plan applicatif. Pour certaines méthodes à deux niveaux, on a $\mathcal{A} = A$ et $\mathbf{x} = x$; la matrice de projection $\mathcal{P}$ associée à la méthode peut ainsi être vue d'un point de vue théorique comme une matrice de préconditionnement, et on a évidemment $\mathbf{b} = \mathcal{P}b$. Pour la déflation classique, par exemple,

cette matrice vaut $\mathcal{P} = I - AW\hat{A}^{-1}W^T$, qui est le projecteur sur le sous-espace de $\mathbb{R}^n$ orthogonal aux vecteurs $\{w_i\}_{i=1}^m$.

On peut constater d'ailleurs que cette écriture permet de retrouver rapidement la forme usuelle de l'algorithme DCG, en écrivant la solution x du système $Ax = b$ sous la forme

$$x = \left(I - P^T\right) x + P^T x \,. \tag{5.20}$$

On peut alors calculer directement une partie de cette solution, car

$$\left(I - P^T\right) x = W\hat{A}^{-1}W^T b \tag{5.21}$$

ce qui fait que le terme $\left(I - P^T\right) x$ est la solution initiale de la méthode de déflation donné par l'équation (5.11). En remplaçant ensuite x par l'expression donnée par (5.20) dans $Ax = b$, et en utilisant la symétrie de A d'où découle $AP^T = PA$, on obtient qu'après résolution du système

$$PA\tilde{x} = Pb \,, \tag{5.22}$$

la solution du système $Ax = b$ est donnée par

$$x = x_0 + P^T\tilde{x} \,. \tag{5.23}$$

Il est alors possible de combiner plusieurs préconditionneurs, qu'ils soient des préconditionneurs "classiques" ou des préconditionneurs à deux niveaux, et ce de façon additive ou multiplicative. La combinaison additive consiste tout simplement à créer, à partir de deux préconditionneurs P_1 et P_2, la matrice

$$\mathcal{P} = P_1 + P_2 \,. \tag{5.24}$$

La combinaison dite multiplicative, quant à elle, est expliquée par le biais des méthodes itératives stationnaires induites par les deux matrices P_1 et P_2. Comme décrit en sous-section 3.1.4, une méthode itérative stationnaire est une méthode de résolution dans laquelle les solutions successives sont obtenues par une itération telle que celle de l'équation (3.20). A partir d'un préconditionneur P, on peut définir l'itération

$$x_{k+1} = x_k + Pr_k \,, \tag{5.25}$$

qui équivaut à $x_{k+1} = (I - PA)\, x_k + Pb$. Cette méthode est cohérente avec une des définitions d'une bonne matrice de préconditionnement, qui veut que la matrice P soit "la plus proche possible" de A, ce qui fait que la matrice $I - PA$ est censé avoir un rayon spectral très petit.

A partir des deux préconditionneurs P_1 et P_2, on peut alors créer les deux demi-itérations $x_{k+\frac{1}{2}} = x_k + P_1 r_k$ et $x_{k+1} = x_{k+\frac{1}{2}} + P_2 r_{k+\frac{1}{2}}$, que l'on peut condenser en $x_{k+1} = x_k + \mathcal{P} r_k$ avec

$$\mathcal{P} = P_1 + P_2 - P_2 A P_1 \,. \tag{5.26}$$

Ces deux modes de combinaison sont appliquées par Tang et ses collaborateurs à différents "préconditionneurs", la notion de préconditionnement étant ici étendue à la projection sur un sous-espace effectuée par une méthode comme la déflation. Il s'avère que la combinaison multiplicative, dans cet ordre, d'un préconditionneur classique et de la matrice Q de projection de la méthode de déflation, crée la méthode la plus rapide et robuste parmi les neuf méthodes comparées dans l'article. Cette méthode, introduite sous le nom de Deuxième Variante Adaptée de la Déflation ou A-DEF2, repose sur le "préconditionneur"

$$\mathcal{P}_{A-DEF2} = M^{-1} + Q - QAM^{-1} = M^{-1}P + Q \tag{5.27}$$

où P et Q sont les matrices définies à partir de la matrice W comme dans l'équation (5.10). La méthode se présente alors sous la forme donnée par l'algorithme 5. Dans tout ce qui suit, la matrice de préconditionnement utilisée est l'inverse de la diagonale de A, c'est-à-dire la matrice de préconditionnement de Jacobi.

Algorithme 5 Algorithme du solveur linéaire A-DEF2, pour la résolution de $Ax = b$ grâce au préconditionneur M^{-1} et à la matrice de déflation W).

Entrées : A, b, M^{-1}, W

$\hat{A} \leftarrow W^T A W$
Résoudre $\hat{A} d_{-1} = W^T b$
$x_0 = W d_{-1}$
$r_0 = b - A x_0$
Résoudre $\hat{A} d_0 = W^T (A M^{-1} - I) r_0$
$w_0 = M^{-1} r_0 - W d_0$
$p_0 = w_0$
Pour $k = 0, 1 \ldots$ jusqu'à ce que $\|r_{k+1}\|_\infty \leq \gamma$ **Faire**
  $\alpha_{k+1} = \frac{r_k^T w_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_{k+1} p_k$
  $r_{k+1} = r_k - \alpha_{k+1} A p_k$
  Résoudre $\hat{A} d_{k+1} = W^T (A M^{-1} - I) r_{k+1}$
  $w_{k+1} = M^{-1} r_{k+1} - W d_{k+1}$
  $\beta_{k+1} = \frac{r_{k+1}^T w_{k+1}}{r_k^T w_k}$
  $p_{k+1} = w_{k+1} + \beta_{k+1} p_k$
Fin Pour

5.2.3 Mise en oeuvre

La méthode utilisée dans le solveur YALES2 pour la mise en oeuvre de la méthode de déflation consiste à créer la matrice de déflation W à partir de vecteurs indicateurs de groupes de volumes de contrôle, construits à partir des groupes de cellules créés lors du partitionnement du domaine avec la méthodologie de double décomposition de domaine. Cette méthode, très commune pour l'application de la méthode de déflation à la simulation physique (voir par exemple [23, 81]), fait que le solveur sur le maillage grossier, dans notre cas le maillage créé par les groupes, renvoie une solution qui est une approximation constante par morceaux du champ solution. Deux exemples en deux dimensions, donnés par un solveur structuré présenté en sous-section 6.2.2 et par le solveur non structuré YALES2, sont donnés par les figures 5.2 et 5.3.

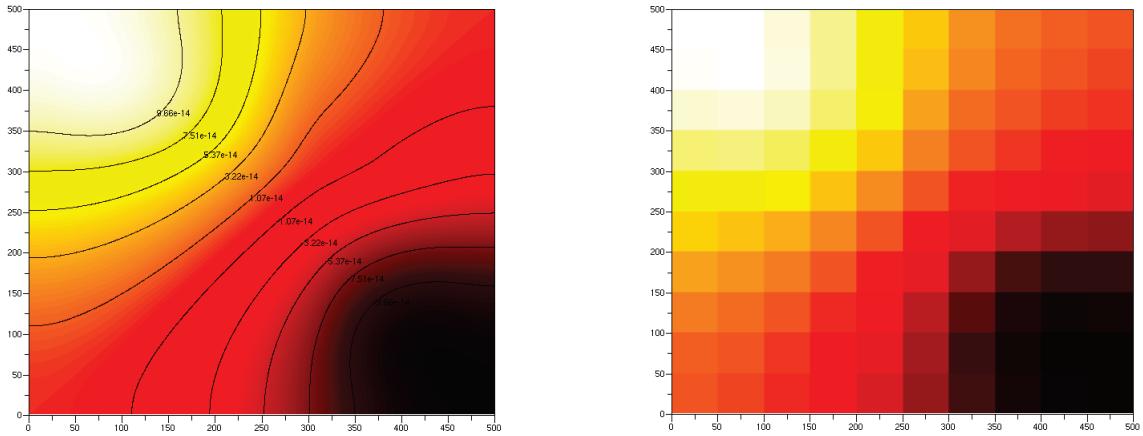

FIGURE 5.2 – *Comparaison entre la solution d'une équation de Poisson sur un maillage cartésien en deux dimensions et sa première estimation sur le maillage grossier par un solveur DPCG.*

Mentionnons cependant que, grâce à la structure de données de YALES2, la matrice W n'est jamais déclarée explicitement. En effet, les opérations impliquant la matrice W peuvent être effectuées uniquement grâce à la connaissance des groupes de volumes de contrôle. La multiplication à gauche d'un vecteur sur le maillage fin par W^T consiste à sommer ses composantes sur chaque groupe pour créer le vecteur résultant ; d'une façon similaire, la multiplication à gauche d'un vecteur sur le maillage grossier par W consiste à créer un vecteur sur le maillage fin dans lequel chaque noeud reçoit la valeur affectée à son groupe dans le vecteur original.

L'opération la plus délicate consiste à créer l'opérateur correspondant à la matrice $\hat{A}$, et puisque l'opérateur laplacien sur le maillage fin A est déclaré sur les paires de noeuds, l'opérateur sur le maillage grossier $\hat{A}$ est déclaré sur les paires de groupes.

Pour créer cet opérateur, on peut considérer directement le résultat de l'opération $\hat{y} = W^T A W \hat{x}$.

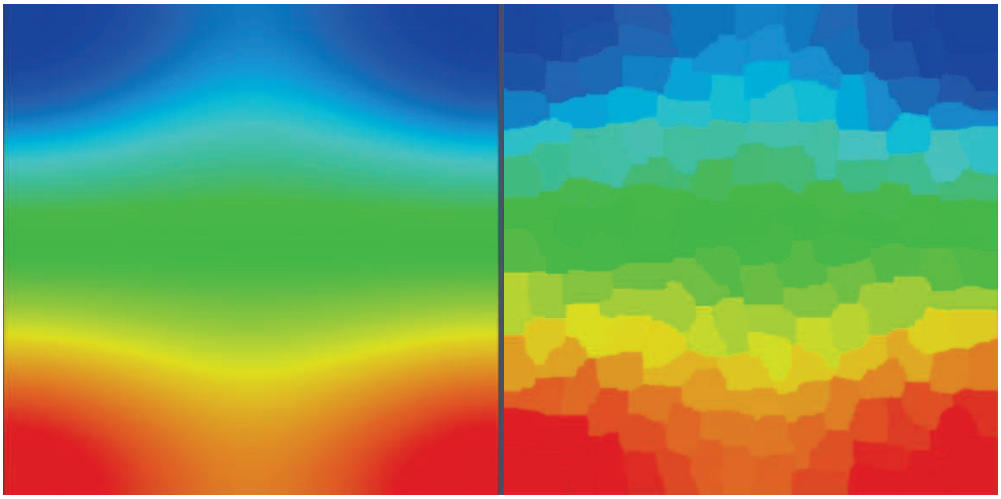

FIGURE 5.3 – *Comparaison entre la solution d'une équation de Poisson sur un maillage non structuré en deux dimensions et sa première estimation sur le maillage grossier par le solveur A-DEF2 utilisé dans YALES2.*

1. Lors de la multiplication du vecteur $\hat{x}$ par W, les valeurs de $\hat{x}$ seront copiées dans un vecteur sur le maillage fin : la i-ième composante du vecteur recevra la valeur $\hat{x}_{g_i}$, où g_i est le groupe auquel le noeud i appartient.

2. Lors du produit avec A, un coefficient de la forme $a_{ij}\left(\hat{x}_{g_i} - \hat{x}_{g_j}\right)$ sera ajouté et enlevé aux composantes i et j du vecteur résultat, et ce pour toutes les paires de noeuds i et j du maillage. Si i et j appartiennent au même groupe, cette contribution est nulle ; par conséquent, seules les contributions des paires de noeuds à la frontière entre deux groupes seront prises en compte dans le calcul de $AW\hat{x}$.

3. Le produit par W^T sommera les valeurs sur chaque groupe. Par conséquent, on peut écrire chaque valeur de $\hat{y}$, c'est-à-dire la valeur prise par $\hat{y}$ sur chaque groupe de volumes de contrôle, comme une somme de contributions attribuées aux groupes voisins, sous la forme

$$\hat{y}_{g_i} = \sum_{g_j \leftrightarrow g_i} \left(\sum_{j \in g_j} a_{ij} \right) \left(\hat{x}_{g_i} - \hat{x}_{g_j} \right) . \tag{5.28}$$

Ainsi, la valeur de la contribution d'une paire de groupes à l'opérateur $\hat{A}$ est la somme des contributions des paires de noeuds à la frontière de ces deux groupes à l'opérateur A, et l'application de l'opérateur $\hat{A}$ sur le maillage grossier se fait d'une façon similaire à l'application de A sur le maillage fin.

Maillage		MP1		MP2	
Critère de convergence		10^{-8}	10^{-10}	10^{-8}	10^{-10}
Itérations	PCG	983	1121	12374	16717
	A-DEF2	40	49	42	61

TABLE 5.2 – *Comparaison des nombres d'itérations sur la simulation du premier pas de temps d'un écoulement incompressible dans le brûleur PRECCINSTA sur deux maillages différents et avec deux critères de convergence différents, pour le solveur linéaire PCG et pour le solveur A-DEF2.*

5.3 Résultats et nouvelles limites

La table 5.2 reprend les résultats exposés dans la table 5.1 pour les simulations d'un écoulement à température constante dans le brûleur PRECCINSTA en utilisant le solveur linéaire PCG, et les compare avec les résultats obtenus grâce à l'application d'une méthode de déflation géométrique avec 200 éléments par groupe. On voit clairement les effets de l'application de la méthode de déflation, non seulement sur la réduction du nombre d'itérations par rapport à la méthode PCG, mais également sur la stabilisation de ce nombre d'itérations lors d'un raffinement homogène de maillage. D'importantes réductions de temps de calcul s'ensuivent, de l'ordre d'une division par 15 du temps de calcul pour la résolution de l'équation de Poisson pour la pression sur le maillage **MP1**, et d'une division par 200 sur le maillage **MP2.** De manière générale, l'algorithme des Gradients Conjugués est mal adapté au calcul massivement parallèle, à cause de son nombre d'itérations quasi-linéaire par rapport à la taille du problème.

Cependant, malgré la réduction importante du temps de simulation due à la déflation de l'algorithme des Gradients Conjugués, la proportion de temps de calcul dédiée à la résolution des équations de Poisson pour la pression successives reste très important, de l'ordre de 55 à 72 % du temps de simulation global, comme montré en table 5.3. De plus, la proportion du temps de simulation passée à faire communiquer les processeurs est relativement élevée, pour la simple raison que de nouvelles communications sont requises à chaque appel du solveur sur le maillage grossier, qui intervient à chaque itération du solveur sur le maillage fin.

Ces deux valeurs vont en augmentant lorsque la taille du maillage croît, ce qui parait naturel puisque l'on conserve la même taille de groupes de cellules, et donc que la taille du maillage grossier augmente également. Cependant, une augmentation trop importante de la taille des groupes de cellules est exclue, d'une part puisqu'elle réduirait la qualité des solutions sur le maillage grossier, d'autre part parce qu'elle effacerait les avantages, en terme de mémoire, de la Double Décomposition de Domaine (voir sous-section 4.2.2).

Par conséquent, bien que l'application de la méthode de déflation permette de dépasser certaines des limites de l'algorithme PCG qui ont été présentées en début de ce chapitre, il est possible de nuancer ces résultats, qui laissent entendre que de nouvelles améliora-

Maillage	MP1		MP2	
Critère de convergence	10^{-8}	10^{-10}	10^{-8}	10^{-10}
Proportion du Poisson	54,98 %	61,58 %	65,27 %	72,72 %
Communications	13,91 %	14,94 %	15,61 %	16,83 %

TABLE 5.3 – *Proportions de temps de simulation consacrées à la résolution de l'équation de Poisson pour la pression, et proportions globales de temps de communications, sur la simulation du premier pas de temps d'un écoulement incompressible dans le brûleur PRECCINSTA sur deux maillages différents et avec deux critères de convergence différents, pour le solveur linéaire A-DEF2.*

tions sont possibles, et qu'elles pourraient être cruciales pour la réduction du temps de simulation. La problématique liée aux travaux effectués lors de cette thèse apparait donc clairement : comment améliorer un algorithme de type DPCG de façon à le rendre plus performant, particulièrement dans le cadre de la simulation massivement parallèle ? Le chapitre suivant présente les recherches effectuées lors de ces trois années de thèse à ce sujet et les résultats obtenus.

Chapitre 6

Développement de nouvelles méthodes de déflation

Lors de la simulation d'écoulements incompressibles, la résolution à chaque pas de temps de l'équation de Poisson peut représenter la plus grande partie des temps de calcul globaux. C'est pourquoi nos efforts se sont focalisés, dans le contexte de cette thèse, sur l'accélération des solveurs linéaires utilisés pour cette résolution.

Comme montré dans le chapitre précédent, l'utilisation d'une méthode DPCG donne déjà des résultats satisfaisants, en ce que la méthode de déflation réduit et stabilise le nombre d'itérations du Gradient Conjugué sur maillage fin, diminuant donc de beaucoup les temps de calcul sur des grands maillages. Cependant, deux problématiques sont à considérer.

La première est que la résolution ne peut pas être assurée sur le maillage fin si les solutions calculées sur le maillage grossier sont imprécises. Or, à cause de l'accumulation d'erreurs d'arrondi, il se peut que le critère de convergence requis pour l'arrêt de l'algorithme sur le maillage grossier ne soit jamais atteint ; dans ce cas, une valeur-seuil du résidu est atteinte, puis l'on observe non pas une stagnation, mais une augmentation du résidu, et donc une détérioration de la solution qui sera renvoyée à l'algorithme A-DEF2 sur le maillage fin. La section 6.1 présente les travaux effectués sur le développement d'une méthode de redémarrage de l'algorithme PCG et du critère utilisé pour ce redémarrage, ainsi que la stabilisation du résidu qui en découle.

La seconde problématique prise en compte dans nos travaux est que la résolution de l'équation de Poisson reste coûteuse devant les autres calculs, et les itérations sur le maillage grossier engendrent des surcoûts de communication et rendent le solveur de moins en moins efficace à mesure que les tailles de maillage augmentent. Ces résultats, exposés en section 5.3, révèlent un goulet d'étranglement dans le développement du code YALES2 : pour rendre plus performantes les simulations d'écoulements incompressibles, l'accélération du solveur linéaire est une étape indispensable.

Une première idée d'amélioration à ce niveau a été que le solveur sur le maillage

grossier pourrait renvoyer des solutions plus élaborées, qui approximeraient un plus large éventail de composantes de basse fréquence du champ de pression recherché ; cette méthode de déflation, son application sur maillages structurés en deux dimensions, puis ses dysfonctionnements après mise en oeuvre dans le solveur non-structuré YALES2, sont détaillés dans la section 6.2.

Notre second axe de recherche a concerné une application récursive de la déflation, afin de créer un solveur sur trois niveaux de maillage. Cette méthode a été conçue particulièrement pour être appliquée à des simulations massivement parallèles (plusieurs milliers, voire dizaines de milliers de processeurs) sur des maillages dépassant le milliard d'éléments. Nous présentons en section 6.3 la conception de ce solveur et ses différences formelles et conceptuelles avec les méthodes multigrilles géométriques.

L'explosion du nombre d'itérations sur les maillages grossiers due à l'application de cette méthode multi-niveaux nous a incité à rechercher des solutions performantes de réduction du nombre total d'itérations. Pour cela, nous nous sommes dans un premier temps replacés dans le contexte d'un DPCG classique, et avons cherché à déterminer les possibles voies de réduction du nombre d'itérations sur le maillage grossier.

La section 6.4 présente une nouvelle méthode, paramétrable et adaptée à l'algorithme CG, de création de solutions initiales par recyclage des solutions précédentes ; nous détaillons son application sur le niveau grossier. La section 6.5, quant à elle, explore la possibilité d'adapter intelligemment le critère de convergence sur le niveau grossier, de manière à arrêter plus tôt le solveur itératif sur le maillage grossier sans que cela n'impacte significativement le comportement du solveur sur le maillage fin.

La mise en oeuvre conjointe de ces deux méthodes dans le solveur A-DEF2, version stabilisée du DPCG utilisée dans YALES2, résulte en un nouveau solveur nommé RA-DEF2 dont la section 6.6 montre l'efficacité, à la fois pour le nombre d'itérations, le temps de calcul, la proportion du temps CPU passé en communications, et le scaling du solveur incompressible de YALES2. Les premiers résultats, très encourageants, obtenus en implantant ces techniques dans la déflation à trois niveaux sont donnés en section 6.7.

6.1 Redémarrage de la méthode CG

Lorsque le critère d'arrêt d'un solveur itératif devient proche de la précision machine et/ou lors de la résolution d'un système particulièrement mal conditionné, les erreurs d'arrondi peuvent rendre la convergence désirée impossible. Dans ce cas de figure, un algorithme comme celui du Gradient Conjugué se mettra à lentement diverger après avoir atteint un résidu minimum supérieur au critère d'arrêt.

Cette divergence est due à l'accumulation des erreurs d'arrondi, qui est rendue inévitable par le calcul itératif de la solution approchée et du résidu après chaque itération. Pour annihiler ses effets, le redémarrage de l'algorithme des Gradients Conjugués est requis [4, 14, 62]. Cette technique permettra de stabiliser le critère de convergence aux

environs de la valeur minimale qu'il peut atteindre en arithmétique finie.

Le redémarrage le plus simple consiste à annuler la valeur de β_k de l'algorithme 2 ; de cette façon, le solveur CG efface la mémoire des itérations précédentes et choisit la direction de descente r_k, comme il l'aurait fait pour résoudre le problème $Ax = b$ avec la solution initiale x_k.

En revanche, la stratégie à adopter pour le redémarrage est moins évidente. Un choix basique est de redémarrer l'algorithme toutes les K itérations, avec K fixé *a priori*, ce qui peut s'avérer être une décision risquée : si le redémarrage est trop fréquent, la convergence de la solution sera ralentie ; s'il est trop rare, la variation du résidu entre deux redémarrages sera grande. Nous avons donc cherché un nouveau paramètre de redémarrage qui tienne compte du comportement effectif du solveur ; une étude simple des symptomes de la divergence du solveur nous a aiguillé dans nos recherches.

6.1.1 Divergence de l'algorithme des Gradients Conjugués

On peut facilement deviner que la divergence du résidu se produit quand le coefficient d' A-orthogonalisation β_k devient trop important ; alors, puisque p_{k-1} et r_k sont orthogonaux d'après l'équation (3.46), l'angle entre p_k et r_k augmente tellement, c'est-à-dire que la direction de descente suivie diffère tellement de la direction du résidu, que le fait de suivre cette direction de descente ne réduira plus la valeur de la fonction de coût C définie par l'équation (3.1).

Pour se convaincre de ce point, on peut considérer le cas où β_k tend vers l'infini, ce qui équivaut à choisir une direction de descente orthogonale au résidu $r_k = b - Ax_k$. L'incidence sur la fonction de coût de l'ajout à x_k d'un δ_x donné, colinéaire à cette direction de descente, donc orthogonale à r_k, est donnée par

$$\begin{aligned} C(x_k + \delta_x) &= \frac{1}{2}\left(x_k + \delta_x\right)^T A \left(x_k + \delta_x\right) - b\left(x_k + \delta_x\right) \\ &= \frac{1}{2} x_k^T A x_k + \delta_x^T A x_k + \frac{1}{2}\delta_x^T A \delta_x - b^T x_k - b^T \delta_x \qquad (6.1) \\ C(x_k + \delta_x) &= C(x_k) + \delta_x^T \left(A x_k - b\right) + \frac{1}{2}\delta_x^T A \delta_x \,. \end{aligned}$$

Comme δ_x est orthogonal à r_k, soit $\delta_x^T \left(b - Ax_k\right) = 0$, on déduit facilement

$$C(x_k + \delta_x) = C(x_k) + \frac{1}{2}\delta_x^T A \delta_x \,. \qquad (6.2)$$

Comme $\delta_x^T A \delta_x$ est strictement positif, l'équation (6.2) prouve qu'une contribution à x_k orthogonale à r_k ne peut qu'augmenter la valeur de la fonction de coût. Pour éviter la non-convergence d'un solveur de Gradients Conjugués, nous devons donc empêcher l'angle entre r_k et p_k de devenir trop important.

Dans la figure 6.1, qui illustre ce cas-limite en deux dimensions, les courbes sont des iso-valeurs de la fonction C. Le point central est $A^{-1}b$ et correspond donc à la solution

du système linéaire $Ax = b$, point en lequel la valeur de la fonction C est minimale. Deux exemples de résidus r_k sont donnés ; ils pointent dans la direction de descente la plus raide. Dans les deux cas, la direction orthogonale δ_x, tangente à la courbe d'iso-valeur passant par x_k, mène inévitablement vers des points où la valeur de la fonction de coût est plus élevée.

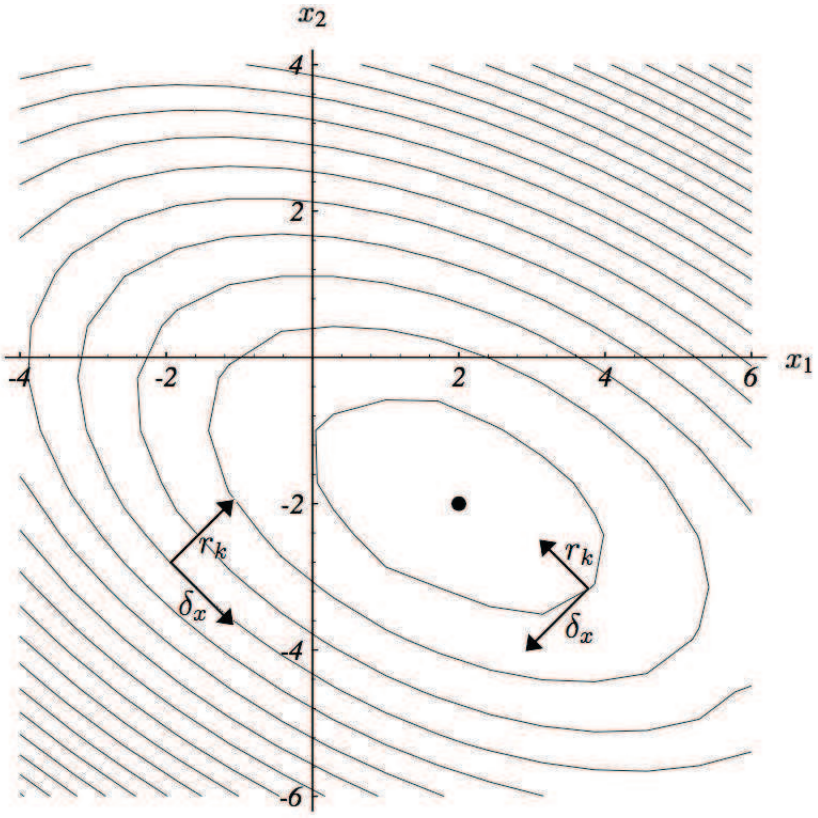

FIGURE 6.1 – *Illustration de l'augmentation de la valeur de la fonction de coût lors de l'ajout à x_k d'une composante orthogonale au résidu*

6.1.2 Création d'un indicateur du besoin de redémarrage

Dans la méthode CG,

$$p_k = r_k + \beta_k p_{k-1} \tag{6.3}$$

d'après l'algorithme 1, et r_k et p_{k-1} sont orthogonaux. Par conséquent, on peut évaluer simplement le cosinus de l'angle entre r_k et p_k :

$$\cos\theta_k = \frac{|r_k|}{|p_k|}, \tag{6.4}$$

ce qui donne

$$\cos^2 \theta_k = \frac{\langle r_k, r_k \rangle}{\langle p_k, p_k \rangle} \,. \tag{6.5}$$

Lorsque cette valeur devient proche de 0, l'angle entre r_k et p_k devient proche de l'angle droit ; cependant, la question de la valeur-seuil en-dessous de laquelle le redémarrage devient nécessaire reste ouverte. Bien que, comme montré par l'équation (6.2), la fonction de coût ne puisse qu'augmenter lorsque la direction de descente est orthogonale au résidu, l'angle minimal entre r_k et un δ_x de norme fixée tel que $C(x_k + \delta_x) > C(x_k)$ reste en revanche inconnu a priori, puisqu'il diffère selon la position de x_k, même à coût fixé.

Un indicateur "idéal" de l'augmentation de la fonction de coût devrait donc prendre en compte le véritable profil de la fonction C, qui est fortement lié aux valeurs et vecteurs propres de la matrice A, pour déterminer selon la valeur de x_k la valeur-seuil à choisir pour le redémarrage. Notre choix, plus pratique bien qu'imprécis, a été de déterminer empiriquement ce critère en testant un large panel de valeurs-seuil de $s_k = \cos^2 \theta_k$ sur différents cas-tests ; ce point est traité plus en détail dans la sous-section suivante.

Remarquons que, dans l'algorithme des Gradients Conjugués usuel, $\rho_k = \langle r_k, r_k \rangle$ est calculé à chaque itération ; on peut alors écrire le critère s_k comme ρ_k / p_k^2, où les valeurs successives de p_k^2 satisfont elles aussi une relation de récurrence très simple. En repartant à nouveau de l'équation (3.40), on obtient

$$p_{k+1}^2 = (r_{k+1} + \beta_{k+1} p_k)^2 = r_{k+1}^2 + 2\beta_{k+1} \langle r_{k+1}, p_k \rangle + \beta_{k+1}^2 p_k^2 \,. \tag{6.6}$$

Comme r_{k+1} et p_k sont orthogonaux, on aboutit directement à la relation

$$p_{k+1}^2 = \rho_{k+1} + \beta_{k+1}^2 p_k^2 \,. \tag{6.7}$$

Le choix le plus simple pour l'implémentation dans ce contexte est de stocker la valeur de p_k^2, qui est initialisée à ρ_0 (car $p_0 = r_0 \Rightarrow p_0^2 = r_0^2 = \rho_0$), puis modifiée à chaque itération par un simple calcul scalaire.

6.1.3 Choix de la valeur-seuil de l'indicateur

Pour des raisons déjà explicitées dans la sous-section 6.1.1, notre indicateur du besoin de redémarrage ne peut prétendre à aucune exactitude. En effet, les couples de valeurs (α_{k+1}, β_k) pour lesquels la valeur de $C\,(x_{k+1})$ est supérieure, ou égale, ou proche de $C\,(x_k)$ dépend du profil local de C autour du point x_k. Une première simplification du problème, inspirée par la figure 6.1, pourrait prendre en compte la courbure locale de l'isocontour de C passant par x_k pour estimer une valeur critique de l'angle entre le résidu et la direction de descente, mais l'approche adoptée dans ces travaux est résolument plus simple encore. Une étude empirique s'est donc avérée indispensable pour estimer la ou les valeurs-seuil

ad hoc de l'indicateur du besoin de redémarrage, également nommé critère de skewness dans ce qui suit.

Deux cas-tests d'écoulements turbulents tridimensionnels sont considérés dans cette sous-section ; il s'agit des cas-tests désignés par **Cylindre_3D** et **Preccinsta**. Le premier, présenté plus en détail en sous-section 6.6.1, correspond à un écoulement autour d'un obstacle cylindrique dans une boîte parallélépipédique, tandis que le second, introduit en sous-section 5.1.1, reprend la géométrie complète d'un brûleur réaliste.

Le maillage utilisé dans le cas-test **Cylindre_3D** compte 491 000 éléments tétraédriques, celui utilisé sur **Preccinsta** en compte 13,7 millions. Le comportement du solveur n'est ici étudié que pour le premier pas de temps de chaque simulation, pour lequel le nombre d'itérations nécessaires à la résolution de l'équation de Poisson pour la pression est généralement très important. Nous avons étudié le comportement du solveur PCG en suivant, durant les 5000 premières itérations, la valeur de la norme infinie du résidu. Dans les deux cas, différentes valeurs-seuil de l'indicateur allant de 0 à 0,03 ont été testées, la valeur 0 correspondant au cas où aucun redémarrage n'est effectué.

Les figures 6.2 et 6.3 montrent les résultats obtenus avec quatre critères particuliers. Il est clair dans les deux cas que, sans redémarrage, le résidu atteint une valeur minimale puis recroît plus ou moins rapidement : environ un ordre de grandeur en 3500 itérations dans le cas-test **Preccinsta**, contre quatre ordres de grandeur en 3000 itérations pour le cas **Cylindre_3D**, dans lequel le résidu semble se stabiliser ensuite.

Cependant, malgré ce que pourrait laisser penser ce premier résultat, le cas-test **Cylindre_3D** semble plus facile à traiter par le solveur. En effet, l'adoption d'une valeur-seuil de la skewness, fut-elle aussi petite que 10^{-5}, permet une relative stabilisation du résidu. On voit clairement sur la courbe correspondante de la figure 6.2 le comportement du solveur, puisque chaque redémarrage est suivi d'une nouvelle descente rapide de la norme du résidu, de l'ordre de deux ordres de grandeur en une centaine d'itérations. Il apparait également que l'augmentation, dans une certaine mesure, de la valeur-seuil du critère de skewness permet de conserver la norme du résidu dans un intervalle plus restreint, mais que pour une valeur-seuil trop grande, le redémarrage devient trop fréquent et ralentit la convergence.

D'un point de vue théorique, un cas-limite apparait pour une valeur-seuil fixée à 1 ou plus ; dans ce cas de figure, la direction de descente correspondrait exactement au résidu à chaque itération, ce qui efface la particularité de la méthode PCG en la remplaçant par une méthode de descente de résidu basique (voir partie 3.1.4.2).

La figure 6.3 montre que la situation est plus délicate pour la simulation du cas-test **Preccinsta**. D'une part, il est évident qu'aucun redémarrage ne se produit avec la valeur-seuil de skewness 10^{-5}, puisque la courbe correspondante est exactement superposée à la courbe du PCG sans redémarrage. D'autre part, la convergence est empêchée par l'adoption d'une valeur du critère de skewness trop haute (0,03 dans notre cas). Dans ce dernier cas, le redémarrage du solveur ne se produit pas à chaque itération, ce qui transformerait la méthode utilisée en une simple méthode de descente de résidu, mais toutes les 20

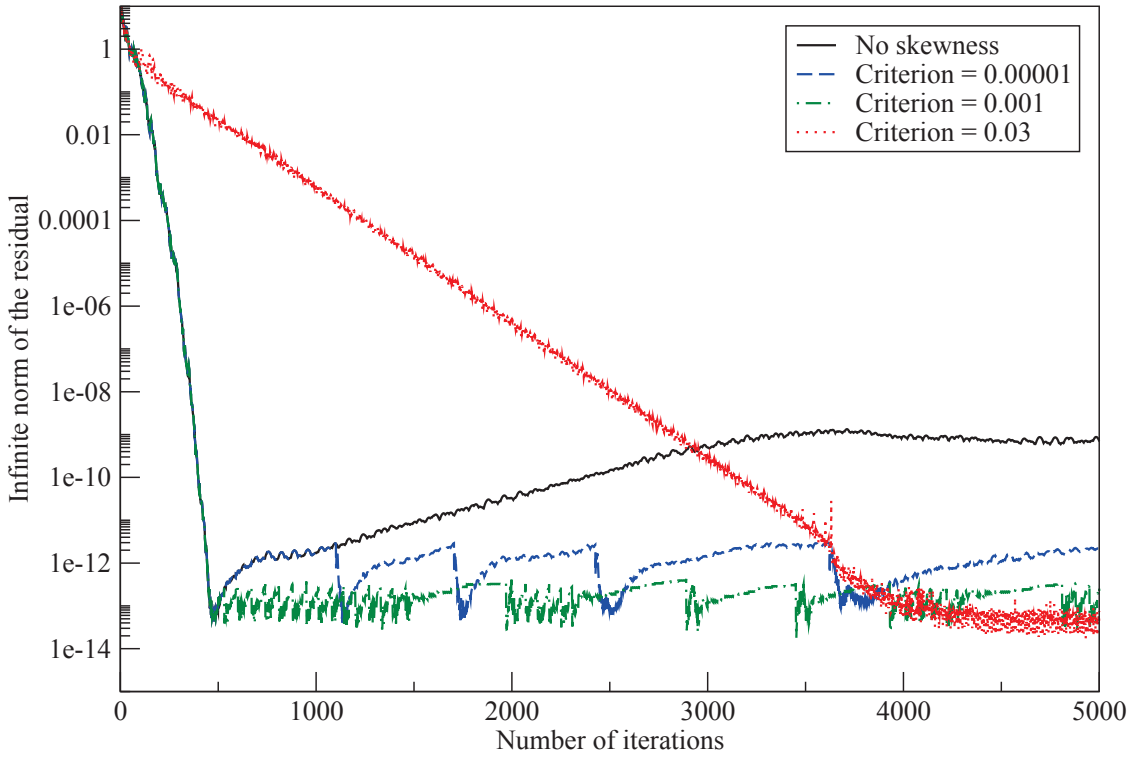

FIGURE 6.2 – *Influence de la valeur-seuil de skewness pour le premier pas de temps de la simulation du cas-test* **Cylindre_3D** *sur un maillage de 491 000 éléments tétraédriques.*

itérations en moyenne, rendant ainsi le comportement du solveur hautement erratique.

Rappelons que le redémarrage a pour but de stabiliser autant que possible la norme du résidu aux alentours de la valeur minimale qu'elle peut atteindre en arithmétique exacte ; un critère de skewness trop contraignant aura donc pour effet de ralentir la convergence vers ce minimum, en provoquant des redémarrages du solveur avant qu'ils ne soient effectivement requis, ce qui se produit avec la valeur 0,03. A l'inverse, un critère de skewness trop lâche fera que les redémarrages ne seront pas assez fréquents, d'où une trop grande amplitude des valeurs de la norme du résidu ; l'arrêt des itérations après un nombre maximal imposé par le solveur fait donc prendre le risque de conserver une solution approchée pour laquelle la norme du résidu est relativement éloignée de l'optimum. C'est le cas pour le critère de skewness 10^{-5} avec le nombre maximal d'itérations imposé par le solveur, 5000 dans notre cas : la norme du résidu pour la solution approchée conservée au terme de ces itérations est supérieure de deux ordres de grandeur à celle des meilleures solutions approchées en arithmétique finie (environ 10^{-12} alors que les meilleurs solutions correspondent à une norme du résidu voisine de 10^{-14}).

Suite à ces tests, la valeur 10^{-3} a été adoptée comme valeur par défaut du critère de skewness. Cette valeur assure la stabilité de l'algorithme des Gradients Conjugués pour

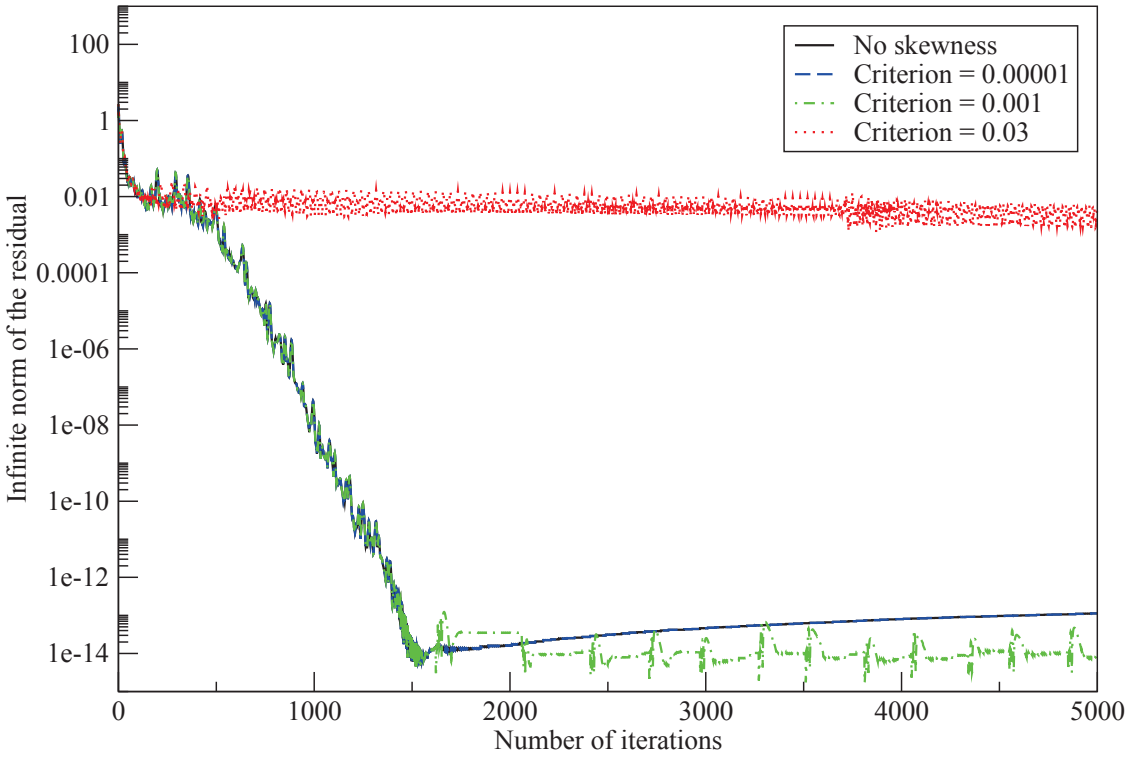

FIGURE 6.3 – *Influence de la valeur-seuil de skewness pour le premier pas de temps de la simulation du cas-test* **Preccinsta** *sur un maillage de 13,7 millions d'éléments tétraédriques.*

toutes les simulations effectuées depuis 2010 grâce au solveur YALES2.

6.2 Déflation plane et quadrique

Derrière les abus de langage "déflation plane" et "déflation quadrique" réside l'idée que le sous-espace formé par les vecteurs de déflation, dont le choix est en théorie laissé à l'utilisateur sans aucune contrainte, pourrait contenir non plus l'ensemble des solutions constantes par morceaux, c'est-à-dire dont la valeur est constante sur chaque groupe de volumes de contrôle, mais l'ensemble des solutions planes par morceaux voire quadriques par morceaux. Nous détaillons dans cette section la procédure de création d'une telle méthode, qui revient principalement à la création de nouveaux vecteurs de déflation, puis nous montrons les premiers résultats obtenus lors de l'implémentation dans un solveur structuré en deux dimensions. Nous évoquons ensuite l'implémentation de la méthode dans YALES2 et les problèmes qui en ont découlé.

6.2.1 Principe

Comme expliqué dans la section 5.2, la méthode de déflation, telle qu'elle a été originellement développée par R. A. Nicolaides [57], consiste à calculer la projection de la solution d'un système linéaire à n inconnues dans un sous-espace, de dimension k très petite devant n, de l'espace de travail initial $\mathbb{R}^n$. En théorie, l'implémentation d'une méthode de déflation dans un solveur itératif nécessite donc uniquement une base $\{w_i\}_{i=1}^k$ de vecteurs linéairement indépendants, les vecteurs de déflation, à partir de quoi le reste de la méthode est déroulé naturellement.

Les solutions calculées sur l'espace projeté, c'est-à-dire le sous-espace de $\mathbb{R}^n$ de dimension k dont les vecteurs de base sont les $\{w_i\}_{i=1}^k$, sont donc sur $\mathbb{R}^n$ des combinaisons linéaires des vecteurs de déflation de la forme $\sum_{i=1}^k \alpha_i w_i$ avec $\forall i \in \{1, 2, \ldots, k\}, \alpha_i \in \mathbb{R}$.

L'application initialement visée par Nicolaides était le choix de vecteurs de déflation qui soient les vecteurs propres associés aux k plus petites valeurs propres de la matrice A, de façon à réduire le conditionnement du problème à résoudre et donc le nombre d'itérations pour la résolution [68]. Cependant, une version de la déflation qui a été rapidement privilégiée consiste à créer une partition du maillage en k groupes de cellules de taille similaire, puis à créer pour chaque groupe de volumes de contrôle correspondant un vecteur de déflation qui en est un indicateur, c'est-à-dire valant 1 sur chaque groupe de contrôle appartenant au groupe et 0 sinon, avec éventuellement des valeurs différentes à la proximité de la frontière entre deux groupes de cellules [23, 81, 3].

Dans le cas où les vecteurs de déflation sont exactement des indicateurs des groupes de cellules, les solutions calculées sur le sous-espace ainsi défini correspondent à des solutions constantes par morceaux sur le maillage fin, et captent donc une première estimation des composantes de plus basse fréquence de la solution. Conceptuellement, il y a donc une relative similarité avec l'application utilisant les vecteurs propres correspondant aux plus petites valeurs propres dans le cas de l'équation de Poisson : l'opérateur laplacien atténuant les composantes de basse fréquence du champ de pression, celles-ci sont associées aux plus petites valeurs propres de la matrice du laplacien.

Une idée d'amélioration de cette méthode a été de chercher à construire une approximation de la solution qui soit plane par morceaux, voire quadrique par morceaux. Pour cela, des vecteurs de déflation doivent être ajoutés pour chaque groupe de volumes de contrôle, afin que les combinaisons linéaires des vecteurs de déflation associés à ce groupe décrivent l'ensemble des polynômes de degré un (pour une solution plane par morceaux) ou deux (pour une solution quadrique par morceaux) selon les coordonnées spatiales des points.

6.2.2 Première application : solveur linéaire 2D structuré

Lors des premiers mois de cette thèse, un solveur structuré de l'équation de Poisson en deux dimensions a été codé intégralement. L'équation $\nabla \cdot \nabla P = \nabla \cdot \vec{u}$, avec la condition

aux limites $\frac{\partial P}{\partial n} = 0$, y est résolue sur un maillage régulier de $n_X \times n_Y$ cellules rectangulaires sur un rectangle de côtés X_M et Y_M. Les groupes de cellules sont créés par conglomération de $L_X \times L_Y$ cellules, avec $n_X = L_X \times k_X$ et $n_Y = L_Y \times k_Y$; le nombre de groupes de cellules est donc $k = k_X \times k_Y$. Dans ce solveur, contrairement à YALES2, les groupes de contrôle sont directement les cellules elles-mêmes.

Comme montré en sous-section 5.2.3, la déflation géométrique standard est appliquée en créant un vecteur de déflation $W_{d,1}$ pour chaque groupe de cellules d ; il s'agit d'un vecteur indicateur valant 1 en chaque cellule appartenant au groupe et 0 en toutes les autres. Pour un second membre de test r_t donné, obtenu par produits de fonctions sinusoïdales, la comparaison entre la solution exacte et sa première approximation par déflation géométrique est donnée par la figure 6.4.

On peut créer, par la même méthode de déflation, des solutions planes par morceaux, en ajoutant deux vecteurs de déflation $W_{d,2}$ et $W_{d,3}$ correspondant à chaque groupe de cellules d, l'un de pente constante selon un des deux axes du repère, l'autre de pente constante selon l'autre axe. Dans $W_{d,2}$, les valeurs

$$\left\{1; 1-2\frac{1}{L_x-1}; 1-2\frac{2}{L_x-1}; \ldots; 1-2\frac{L_x-2}{L_x-1}; -1\right\}$$

sont alors associées dans cet ordre aux cellules de chaque "ligne" de cellules du groupe concerné ; dans $W_{d,3}$, ce sont les valeurs

$$\left\{1; 1-2\frac{1}{L_y-1}; 1-2\frac{2}{L_y-1}; \ldots; 1-2\frac{L_y-2}{L_y-1}; -1\right\}$$

qui sont attribuées dans cet ordre aux cellules de chaque "colonne" du groupe. Ainsi, la solution calculée par le solveur projeté à chaque appel prend sur chaque groupe de cellules les valeurs d'une fonction du type $a + bx + cy$. Elle est donc plane par morceaux, et constitue une meilleure approximation de la solution recherchée, comme illustré par la figure 6.5.

Pour que le solveur projeté renvoie une solution, non plus plane, mais quadrique par morceaux, cette solution doit suivre sur chaque groupe de cellules une fonction de la forme $a + bx + cy + dx^2 + ey^2 + fxy$. Trois vecteurs de déflation sont ainsi ajoutés pour chaque sous-groupe : les valeurs de $W_{d,4}$ et de $W_{d,5}$ sont obtenues en élevant les valeurs de $W_{d,2}$ et $W_{d,3}$ respectivement au carré, et $W_{d,6}$ est obtenu par produit terme à terme de $W_{d,2}$ et $W_{d,3}$. La qualité de cette nouvelle approximation de la solution de l'équation de Poisson, toujours pour le second membre r_t, est illustrée par la figure 6.6.

Une comparaison des nombres d'itérations et des temps de calcul pour chaque méthode de déflation, sur une station locale, a été effectuée avec les paramètres suivants :

- champ de vitesse d'entrée donné par $U(x,y) = \begin{pmatrix} cos(x)sin(y) \\ -sin(x)cos(y) \end{pmatrix}$;
- domaine : $[0; 2\pi] \times [0; 2\pi]$;

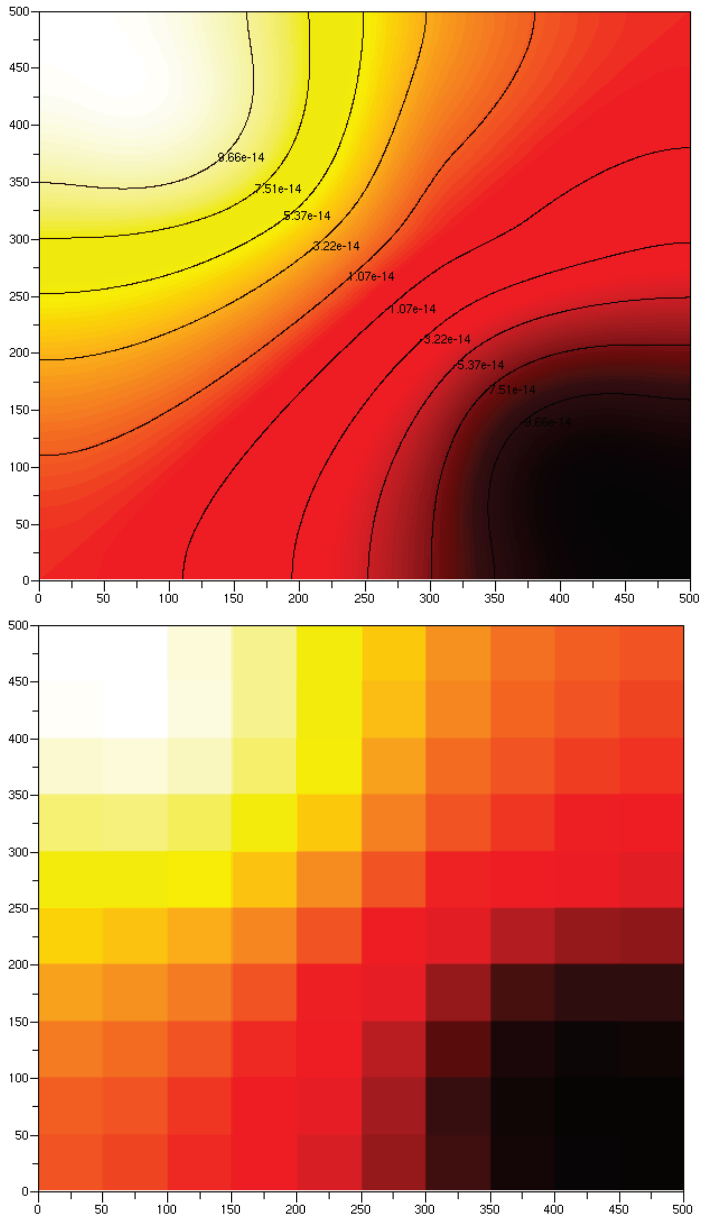

FIGURE 6.4 – *Comparaison entre la solution attendue de l'équation de Poisson pour le second membre r_t (en haut) et sa première approximation constante par morceaux de la solution (en bas), donnée par la déflation géométrique classique (norme infinie de l'erreur :* $1,157.10^{-13}$*).*

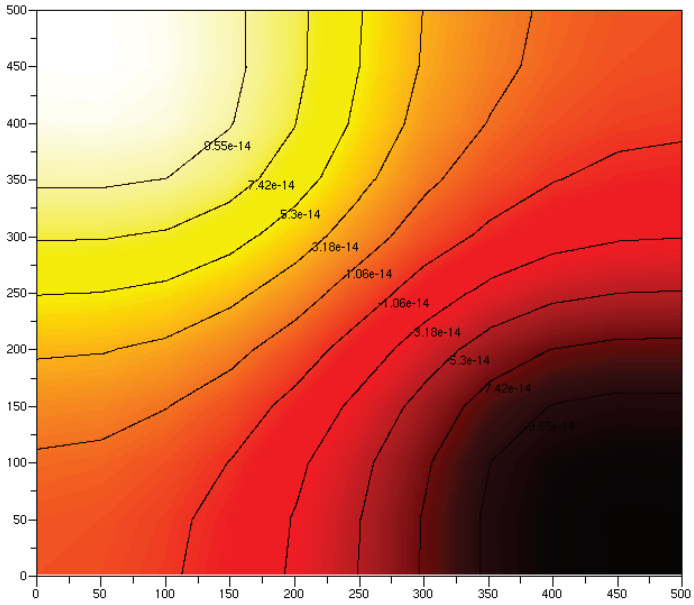

FIGURE 6.5 – *Première approximation plane par morceaux de la solution pour le second membre r_t, donnée par la déflation à trois vecteurs par groupe de cellules (norme infinie de l'erreur :* $6,258.10^{-15}$*).*

- critère de convergence $\gamma = 10^{-7}$;
- maillages réguliers de 200×200 à 600×600 par pas de 100.

Les résultats obtenus sont donnés dans la table 6.1. Les temps de calcul, bien qu'obtenus sur une machine non dédiée, suggèrent que la précision accrue des solutions données par le solveur projeté est à privilégier lorsque la taille du maillage augmente.

Taille	**PCG**		**D1**		**D3**		**D6**	
	N_{iter}	T (s)	N_{iter}	T (s)	N_{iter}	T (s)	N_{iter}	T (s)
200	610	0,587	130	**0,302**	84	0,504	58	1,195
300	931	2,027	186	**0,892**	120	1,171	83	2,377
400	1235	6,402	242	**2,552**	154	2,603	107	4,286
500	1529	14,248	302	4,665	192	**4,496**	130	6,643
600	1840	25,65	366	8,347	226	**7,373**	158	10,043

TABLE 6.1 – *Comparaison des nombres d'itérations N_{iter} et temps de calcul T, en secondes, requis sur une station locale pour la résolution de l'équation de Poisson, pour la méthode PCG classique et pour les déflations standard, plane et quadrique. Pour chaque taille de maillage, le plus petit temps de calcul a été mis en gras.*

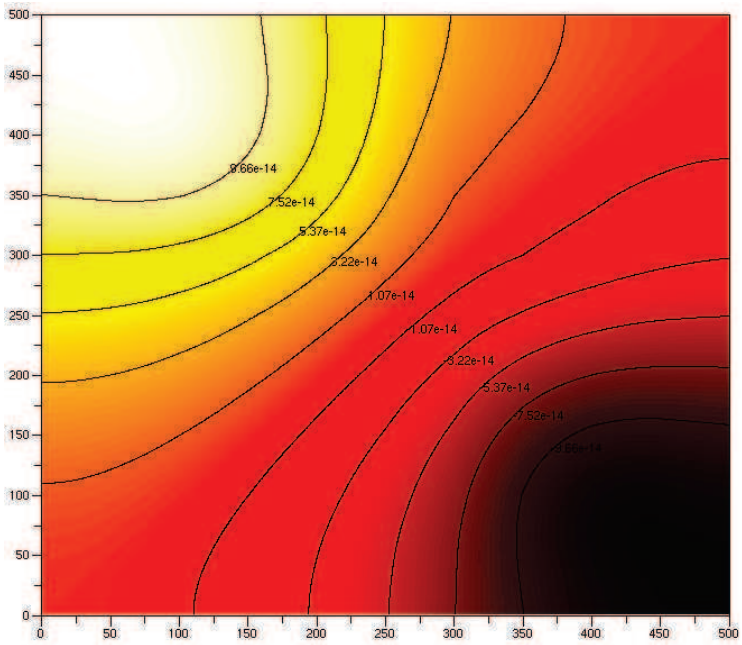

FIGURE 6.6 – *Première approximation quadrique par morceaux de la solution pour le second membre r_t, donnée par la déflation à six vecteurs par groupe de cellules (norme infinie de l'erreur :* $4,28.10^{-16}$*).*

6.2.3 Mise en oeuvre dans YALES2

La mise en oeuvre de cette méthode dans le solveur non structuré YALES2 fut un peu plus complexe. D'une part, la création des vecteurs de déflation a nécessité en particulier de récupérer des valeurs des coordonnées extrémales des groupes de cellules ; d'autre part, la projection des solutions "grossières" sur l'espace initial a demandé un développement algorithmique plus important.

Le premier point est dû au fait que les valeurs non nulles de chaque vecteur de déflation sont calculées en fonction de la position relative d'une cellule au sein de son groupe. Dans le cas d'un maillage structuré, il suffit de répartir des valeurs allant de -1 à 1 sur chaque rangée d'éléments, comme pour le solveur 2D dont la construction est décrite dans la sous-section précédente. Pour un maillage non structuré, en revanche, le calcul est à faire en fonction des coordonnées réelles. Par exemple, considérons un volume de contrôle autour d'un noeud de coordonnées (x_P, y_P, z_P), situé dans un groupe dont la *bounding box* est $[x_m; x_M] \times [y_m; y_M] \times [z_m; z_M]$: il s'agit du plus petit parallélépipède rectangle contenant le groupe de cellules, et ses coordonnées correspondent aux coordonnées extrémales de ce groupe de cellules. Les composantes des vecteurs de "déflation plane" associées au volume de contrôle seront alors

$$\begin{cases} W_{d,2}^P = \dfrac{2x_P - \Sigma x}{\Delta x} \\ W_{d,3}^P = \dfrac{2y_P - \Sigma x}{\Delta y} \\ W_{d,4}^P = \dfrac{2z_P - \Sigma x}{\Delta z} \end{cases} \tag{6.8}$$

avec $\Delta x = x_M - x_m$, $\Sigma x = x_M + x_m$, et idem pour les valeurs de Δy, Σy, Δz et Σz. Seules ces contributions sont stockées dans YALES2, pour rester dans la logique de déclaration déjà appliquée à l'opérateur discrétisé A, qui contient les contributions de chaque paire de noeuds à l'opérateur. Un développement algorithmique supplémentaire a donc été requis, notamment pour la création et l'application de l'opérateur $\hat{A} = W^T A W$.

Les tests sur des maillages structurés ont confirmé l'efficacité de la méthode de déflation dite LDPCG, qui calcule à chaque itération sur le maillage fin une nouvelle composante plane par morceaux de la solution. Les figures 6.7 et 6.8 donnent une comparaison entre les efficacités réduites des solveurs DPCG et LDPCG sur des maillages cartésiens en 2D de 6 et 25 millions de cellules respectivement, sur 4 processeurs, selon le nombre de cellules par groupe. L'équation résolue est du type $Ax = b$, où A est le laplacien discrétisé par différences finies à l'ordre deux, et b un second membre prenant en chaque point la valeur d'une fonction définie par produits de fonctions sinusoïdales dans chaque direction. On y constate clairement une amélioration des performances, de l'ordre d'une division par deux de l'efficacité réduite entre les meilleures performances du solveur DPCG et du LDPCG, et il apparait que le nombre "idéal" de cellules par groupe augmente avec le nombre de vecteurs de déflation utilisés sur chaque groupe.

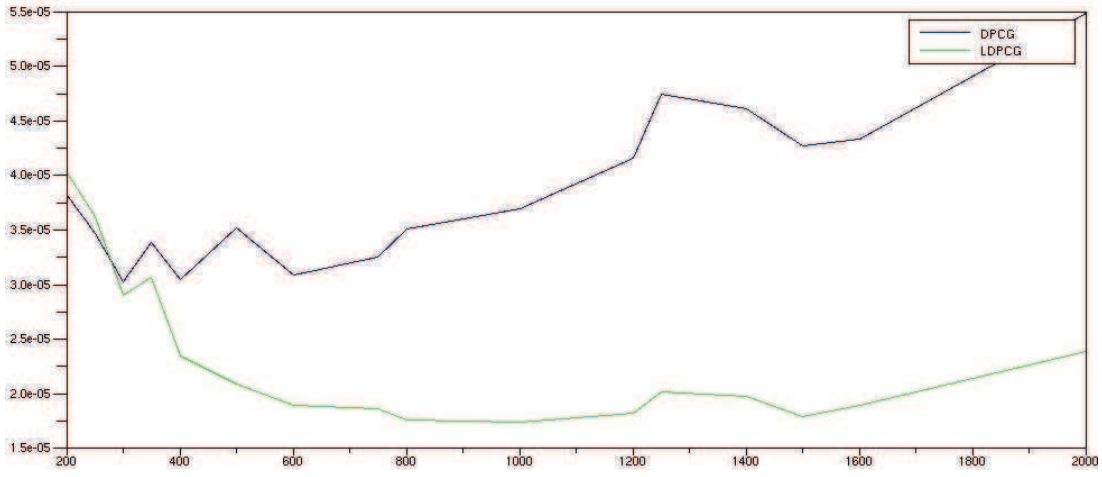

FIGURE 6.7 – *Efficacités réduites des solveurs DPCG et LDPCG pour une résolution sur 4 processeurs sur un maillage cartésien en deux dimensions comptant 6 millions de cellules, selon le nombre de cellules par groupe.*

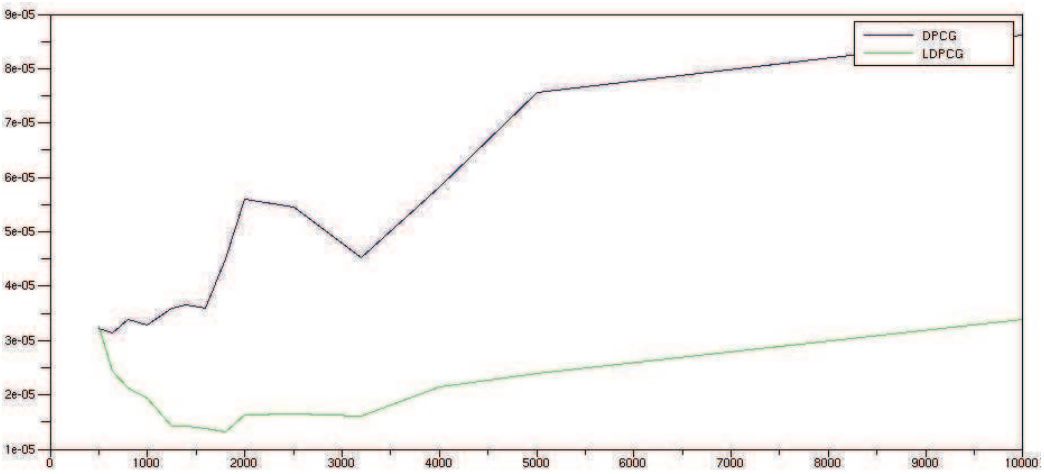

FIGURE 6.8 – *Efficacités réduites des solveurs DPCG et LDPCG pour une résolution sur 4 processeurs sur un maillage cartésien en deux dimensions comptant 25 millions de cellules, selon le nombre de cellules par groupe.*

Cependant, lors des premiers tests sur des maillages non structurés, il s'est avéré que le solveur était incapable de converger vers la solution de la plupart des problèmes. Le code est passé par de nombreuses phases de diagnostic et de dépistage des erreurs, jusqu'à ce que soit acquise la certitude que le solveur était idéalement codé, et donc que sa non-convergence était due à un manque de stabilité intrinsèque de l'algorithme. Le remplacement de l'algorithme DPCG classique par le plus stable A-DEF2 (voir sous-section 5.2.2) n'a pas permis de résoudre ce problème.

Nos recherches de raisons à ces instabilités ont été contrecarrées par le fait que la littérature concernant la déflation argumente la stabilité théorique des méthodes DPCG et A-DEF2 [57, 68, 75] mais manque d'informations sur sa stabilité numérique ; les informations que nous avons pu y trouver ne suffisent pas à expliquer le problème rencontré dans notre nouvelle application de la méthode de déflation. L'annexe de ce manuscrit, en page 169, présente différentes stratégies abordées pour la résolution de ce problème.

6.3 Déflation multi-niveaux

Après le relatif insuccès des méthodes de déflation dites "planes" et "quadriques", nous n'avons pu que repartir sur une nouvelle idée, cette fois-ci plus spécifique car plus particulièrement adaptée à la simulation massivement parallèle. En nous inspirant du fonctionnement des méthodes multigrilles, nous avons voulu créer une méthode multi-niveaux reposant sur la méthode de déflation ; ainsi, le seul choix de l'utilisateur ou du programmeur serait la taille des macrocellules sur chaque niveau, contrairement aux méthodes multigrilles pour lesquelles le programmeur est confronté au choix des opérateurs de pro-

jection entre chaque niveau. Dans cette section, après avoir expliqué plus en détail le principe d'une telle méthode et ce qui la différencie d'une méthode multigrilles, nous évoquerons le développement de cette méthode dans YALES2. Finalement, les résultats des tests effectués sur cette méthode mettront en évidence les besoins pratiques engendrés par un tel solveur multiniveaux.

6.3.1 Principe

Tandis qu'une méthode de déflation telle que décrite ci-dessus aurait pu permettre de rendre plus rapide la résolution de systèmes linéaires de taille relativement modeste, une amélioration destinée particulièrement aux simulations massivement parallèles, sur des maillages comptant plusieurs milliards d'éléments, restait à concevoir. Nos recherches se sont donc orientées vers la conception d'un solveur de type DPCG, mais avec plusieurs niveaux de projection.

Deux idées principales motivent cette direction de recherche. D'une part, l'utilisation d'une méthode de type DPCG pour des simulations sur des maillages de plusieurs milliards d'éléments passe par la résolution d'un système linéaire "grossier" pouvant compter plusieurs millions d'inconnues, et ce à chaque itération sur le maillage fin ; le remplacement du PCG par une méthode utilisant la déflation permettrait de réduire le nombre d'itérations à ce niveau, avec l'espoir de diminuer le temps de calcul global. D'autre part, la conception même de cette méthode nous assure sa stabilité, puisque sur chacun des deux niveaux les plus raffinés, un DPCG "géométrique" classique est utilisé. Ce mode de fonctionnement, schématiquement proche du W-cycle des méthodes multigrilles, est illustré par la figure 6.9.

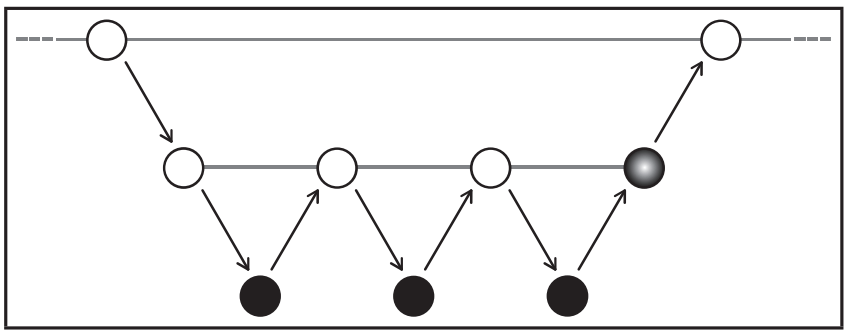

FIGURE 6.9 – *Illustration du V-cycle utilisé par un solveur utilisant une déflation à trois niveaux, sur une itération du solveur sur le maillage fin. Les niveaux sont représentés dans l'ordre standard, du plus raffiné (en haut) vers le plus grossier (en bas). Les cercles blancs représentent le début d'une itération ; le cercle dégradé du blanc au noir représente la dernière itération du solveur pour le problème donné, c'est-à-dire celle lors de laquelle le critère de convergence est atteint ; les cercles noirs représentent de façon condensée une résolution itérative complète du problème.*

6.3.2 Différences avec les méthodes multigrilles

Les méthodes multigrilles [29, 78, 86] sont une classe de méthodes régulièrement utilisées pour la résolution de problèmes liés à la CFD. Dans la pratique, tout comme la méthode de déflation, les méthodes multigrilles géométriques font entrer en jeu un ou plusieurs maillages grossiers sur lesquels le problème originel est projeté et résolu, au moins en partie, et ce afin d'accélérer la convergence générale de l'algorithme. Pour éviter au lecteur toute assimilation trop rapide, cette sous-section présente brièvement les méthodes multigrilles et leurs différences avec les méthodes de déflation.

6.3.2.1 Méthodes multigrilles géométriques

Le principe des méthodes multigrilles s'appuie sur une constatation qui prend tout son sens dans le contexte de la simulation physique : la convergence des modes élevés, c'est-à-dire les composantes de la solution correspondant aux hautes fréquences, est généralement rapide avec les méthodes itératives conventionnelles, mais la convergence est lente sur les composantes à basse fréquence [1, 10, 11].

Les problèmes physiques liés aux écoulements turbulents étant des problèmes multi-échelles, la constatation précédente permet d'expliquer, tout du moins en partie, la convergence relativement lente des algorithmes itératifs de résolution des systèmes linéaires liés à la CFD. L'idée d'introduire un ou plusieurs maillages plus grossiers est alors relativement naturelle, pour faciliter la convergence des composantes à basse fréquence de la solution.

L'approche classique des multigrilles géométriques sur un maillage structuré est de multiplier le pas par deux dans chaque direction de l'espace pour créer une grille plus grossière. Il est possible d'itérer ce procédé pour créer une hiérarchie de maillages sur le même domaine, ce que la couverture du livre "Multigrid" d'Ulrich Trottenberg *et al.* reproduite en figure 6.10 illustre d'une façon imagée.

La hiérarchie des grilles est représentée classiquement en descendant des maillages les plus fins vers les maillages les plus grossiers, et des opérateurs sont créés pour se déplacer dans cette hiérarchie. Plus formellement, il s'agit d'opérateurs de restriction et de prolongation ; les premiers permettent de projeter un vecteur, défini sur un des maillages, sur le maillage immédiatement inférieur ; les seconds, à l'inverse, créent à partir d'un vecteur sur un des maillages un vecteur de dimension supérieure défini sur le maillage immédiatement supérieur. Nous ne détaillerons pas ici les différentes stratégies de restriction et de prolongation. Il va de soi que la stratégie de restriction la plus naturelle passe par le calcul de moyennes pondérées, tandis que la prolongation fait généralement appel à des interpolations plus ou moins sophistiquées.

A partir de la matrice du système initial $Ax = b$ et des opérateurs de restriction et de prolongation, on peut construire par calcul direct les matrices correspondant au même problème physique, mais ramené aux niveaux plus grossiers. En pratique, une fois définis les opérateurs de prolongation P et de restriction R qui permettent de faire communiquer le niveau fin et celui immédiatement inférieur, la matrice du problème sur ce deuxième

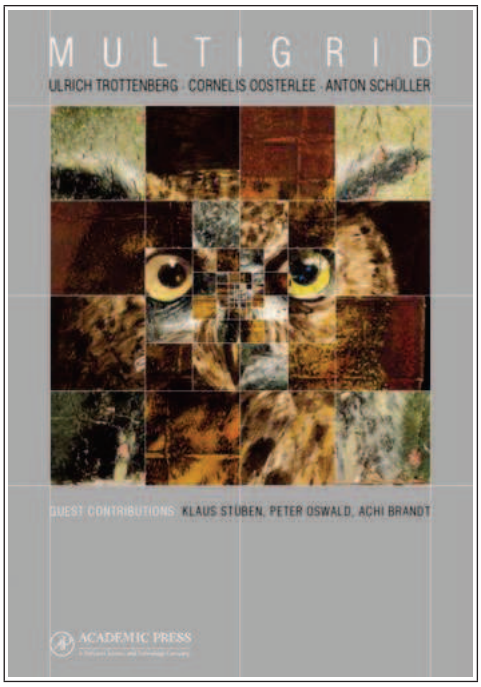

FIGURE 6.10 – *Illustration schématique du principe de la méthode multigrilles sur la couverture de "Multigrid" par Ulrich Trottenberg et al. [79].*

maillage est obtenue par l'approximation de Galerkin $\hat{A} = RAP$ [87], et un calcul similaire est effectué à chaque niveau pour déterminer la matrice du problème sur le niveau inférieur.

Avant d'aborder les différences entre la méthode de déflation et la méthode multigrilles, précisons que nous n'abordons pas ici l'utilisation de la méthode multigrilles dans un but de préconditionnement d'un problème [2, 59, 76] mais uniquement son usage pour la résolution complète d'un système linéaire.

6.3.2.2 Différence conceptuelle

Le but premier des méthodes multigrilles (MG) est d'améliorer la convergence des méthodes numériques telles que Jacobi ou Gauss-Seidel, dites "smoothing methods" ou méthodes de lissage ; ces méthodes diminuent toutes les composantes de l'erreur $x - x_k$ à chaque itération lorsque cette erreur est exprimée dans la base des valeurs propres de la matrice A (voir partie 3.1.4.1, page 47). La convergence est rapide sur les modes élevés, c'est-à-dire sur les composantes de la solution correspondant aux hautes fréquences, associées aux plus petites valeurs propres ; en revanche, elle est lente voire inexistante sur les composantes à basse fréquence, ce qui correspond au lissage évoqué par le nom de cette famille de méthodes.

A l'inverse, la déflation n'est pas une méthode corrective conçue pour une famille donnée de méthodes numériques, mais une technique indépendante pouvant être implémentée

dans un solveur itératif quelconque. Les solveurs dans lesquels la technique de déflation est mise en oeuvre reposent d'ailleurs très souvent sur des méthodes numériques telles que GMRES ou PCG, la seconde étant un "rougher", c'est-à-dire un algorithme itératif ne garantissant pas la réduction de toutes les composantes de l'erreur à chaque itération [1]. Ces choix de méthodes reposent principalement sur leur efficacité, la déflation ayant été construite comme une méthode générale d'accélération de convergence des solveurs itératifs qui laisse, en plus, une grande marge de manoeuvre concernant le choix des vecteurs de déflation - lesquels définissent les opérateurs de restriction et de prolongation.

L'usage de la déflation et des méthodes MG est donc différent : la première permet d'estimer les composantes de basse fréquence de la solution sur un maillage grossier à chaque itération du solveur sur le maillage fin ; les secondes commencent par plusieurs itérations sur le maillage fin pour lisser le résidu, lequel sera ensuite projeté sur un maillage plus grossier.

6.3.2.3 Différence fonctionnelle

Lors de l'utilisation d'une méthode multigrilles, les choix des méthodes de restriction et de prolongation rendant la méthode la plus efficace possible dépendent particulièrement des équations à résoudre, et de ces choix découle l'approximation de Galerkin de la matrice sur le niveau inférieur. Au contraire, le choix des vecteurs de déflation, lesquels peuvent aussi bien être "géométriques" (tel le choix de vecteurs indicateurs de groupes de cellules dans notre cas) qu'analytiques (tels les vecteurs propres associés aux valeurs propres les plus basses, comme suggéré initialement par Nicolaides [57]), crée directement les opérateurs de prolongation et de restriction, à savoir W et W^T respectivement. L'équivalent direct de l'approximation de Galerkin $\hat{A} = RAP$ utilisé dans certaines méthodes multigrilles est donc l'opérateur exact $\hat{A} = W^T AW$ sur le sous-espace engendré par les vecteurs de déflation. L'efficacité d'une méthode de déflation repose sur le choix des vecteurs de déflation et de leur stratégie de construction, lesquels créent directement les opérateurs de restriction et de prolongation de la méthode, tandis qu'une méthode multigrilles s'appuie autant sur la construction du maillage grossier que sur les opérateurs de restriction et de prolongation créés pour faire le lien entre ce maillage et celui du niveau supérieur.

6.3.3 Développement et mise en oeuvre

Le solveur YALES2 est structurellement basé sur la Double Décomposition de Domaine décrite dans la sous-section 4.2.2 : la grille sur laquelle la simulation est effectuée est déclarée dans le solveur comme un ensemble de groupes de cellules liés par des communicateurs

[1] Précisons néanmoins que la convergence globale vers la solution est assurée dans le contexte classique d'utilisation de ces méthodes ; dans le cas d'une méthode de Gradients Conjugués, une des composantes de l'erreur dans la base propre peut augmenter lorsqu'une direction de descente particulière est suivie, mais cette augmentation sera compensée par la suite.

(internes et externes). Pour la mise en oeuvre d'une méthode de déflation à trois niveaux, il nous faut donc, soit créer un maillage grossier par conglomération des groupes de cellules, soit conserver le maillage formé par les groupes de cellules et décomposer chacun de ces groupes en sous-groupes de façon à introduire un maillage intermédiaire. Il s'est avéré plus pratique de s'orienter vers cette deuxième solution.

Un niveau intermédiaire a donc été introduit en scindant chaque groupe de cellules en un nombre fixe n_{sub} de sous-groupes. La structure du solveur ne permettant pas de créer une nouvelle structure de données "à moindre frais" pour les sous-groupes, le choix d'une simple coloration a été fait : chaque volume de contrôle d'un groupe se voit attribuer une couleur, c'est-à-dire un entier compris entre 1 et n_{sub}, qui correspond au numéro du sous-groupe auquel elle appartient. Cette numérotation locale est amplement suffisante, puisque chaque groupe de volumes de contrôle a d'ores et déjà une couleur unique attribuée par le code ; deux volumes de contrôle appartiennent donc au même sous-groupe *si et seulement si* leurs couleurs de groupes et leurs couleurs de sous-groupes coincident.

La projection de l'opérateur laplacien A sur ce maillage intermédiaire se fait d'une façon similaire à celle sur le maillage des groupes de volumes de contrôle, décrite dans la sous-section 5.2.3. Si une arête d'un élément lie les noeuds de deux sous-groupes distincts, une contribution est ajoutée à l'opérateur laplacien sur le maillage formé par les sous-groupes.

Le PCG sur le maillage intermédiaire est une version modifiée du PCG sur le maillage grossier, qui prend en compte les nouvelles structures de données déclarées à ce niveau : les vecteurs utilisés à ce niveau sont déclarés dans YALES2 comme des tableaux à deux dimensions, contenant n_{sub} valeurs pour chaque groupe de volumes de contrôle ; de même pour les opérateurs associés au laplacien, qui comptent eux aussi une dimension de plus.

Quant à la technique de déflation, elle est aisée à mettre en place sur le niveau intermédiaire, puisque les données sur un groupe sont directement obtenues par congrégation des données sur les n_{sub} sous-groupes qui le partitionnent.

6.3.4 Tests et comparaison avec le solveur DPCG

Afin d'obtenir une première idée de l'efficacité du solveur à trois niveaux, que nous nommons 3LDPCG dans ce qui suit, les premiers tests ont été effectués sur un cas-test structuré consistant en la résolution d'une équation de Poisson basique, de type $\Delta P = R$ avec R un champ imposé comme paramètre du calcul. Il a été empiriquement observé que les seconds membres présentant des discontinuités ou des sauts de gradients rendent la convergence d'une méthode de Gradients Conjugués moins efficace ; nos premiers tests se sont donc orientés dans ce sens pour vérifier la stabilité de la méthode développée.

Le second membre considéré dans ce qui suit est donné par $|x|\sin(\pi y)$, avec x et y les coordonnées de chaque point du domaine, sur un maillage cartésien en deux dimensions de 400 mailles de côté, puis de 1000 mailles de côté. Dans les deux cas, le domaine maillé est le carré de coordonnées $[-1;1]^2$, et les groupes de cellules contiennent 1000 éléments

chacun. Le maillage intermédiaire est créé en scindant les groupes de cellules en quatre. Deux simulations ont été effectuées pour chaque maillage : dans l'une, le maillage formé par les sous-groupes est utilisé comme maillage grossier d'une méthode de déflation classique ; dans l'autre, une déflation à trois niveaux est utilisée, en utilisant respectivement le maillage des sous-groupes et celui des groupes comme niveaux intermédiaire et fin. On obtient simplement la première méthode à partir de la deuxième, en désactivant les appels au solveur sur le maillage grossier. Les résultats en termes de nombre d'itérations sont donnés par les figures 6.11 et 6.12.

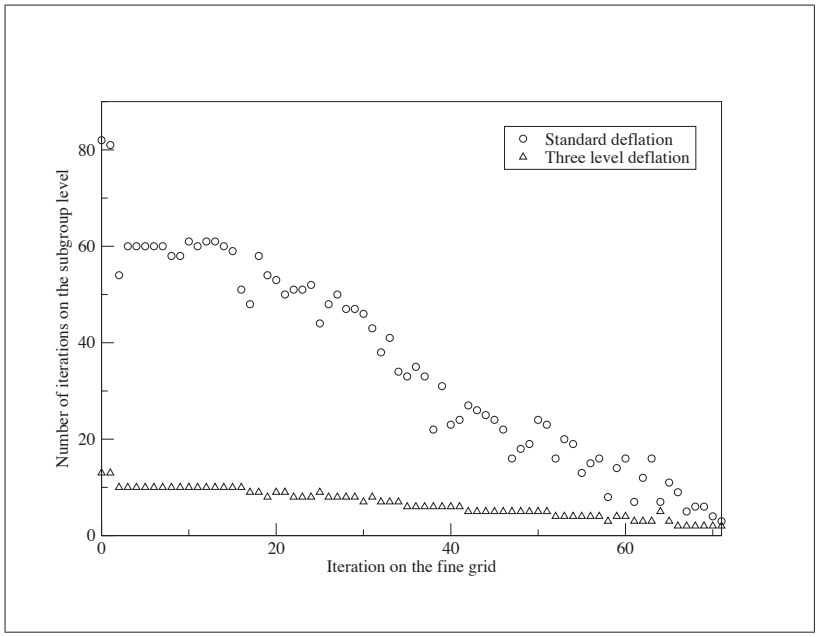

FIGURE 6.11 – *Nombre d'itérations sur le maillage intermédiaire, avec et sans application d'une méthode de déflation à ce niveau, sur le cas-test structuré décrit ci-dessus, sur un maillage régulier de 400 mailles de côté.*

Les premiers résultats sont très encourageants quant au temps de calcul requis pour la résolution de problèmes modèles. En revanche, l'application de cette méthode à des simulations concrètes, comme celle d'un écoulement turbulent autour d'un obstacle cylindrique, décrit plus en détail dans la sous-section 6.6.1, a résulté en une explosion des temps de calcul, qui peuvent même doubler voire tripler sur des maillages comptant entre 3,9 et 31,5 millions d'éléments.

En outre, lors de la simulation massivement parallèle d'écoulements à température constante dans la géométrie du brûleur PRECCINSTA (voir sous-section 5.1.1), sur des maillages comptant entre 110 et 877 millions d'éléments, la convergence peut ne pas se produire du tout : après un certain nombre d'itérations, certains scalaires utilisés par l'algorithme sur le maillage grossier deviennent trop petits, et le passage de leur valeur à

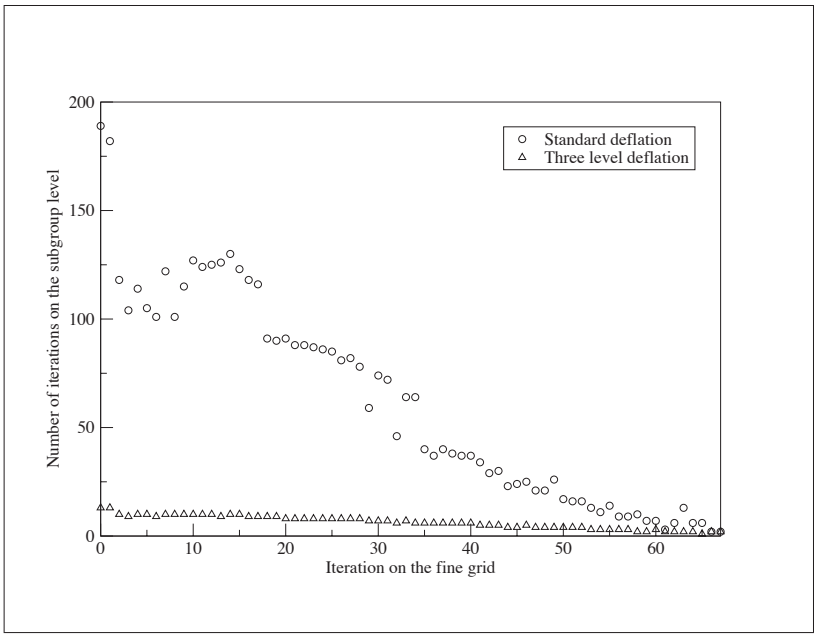

FIGURE 6.12 – *Nombre d'itérations sur le maillage intermédiaire, avec et sans application d'une méthode de déflation à ce niveau, sur le cas-test structuré décrit ci-dessus, sur un maillage régulier de 1000 mailles de côté.*

0 entraîne le crash pur et simple du programme.

6.3.5 Besoins pratiques

Dans les cas où le fonctionnement de la méthode de déflation à trois niveaux est assuré, la raison principale de sa relative inefficacité est la multiplication du nombre total d'itérations, tous niveaux confondus. En effet, l'usage d'un troisième niveau de maillage réduit le nombre d'itérations nécessaires pour le solveur sur le niveau supérieur, mais à chacune de ses itérations, une résolution complète d'un système linéaire est effectuée sur le maillage le plus grossier. A chaque itération du solveur sur ce niveau, comme pour les solveurs sur les niveaux supérieurs, une communication de données entre les processeurs est indispensable, et le temps nécessaire pour cette communication est loin d'être négligeable, puisqu'il comprend un temps incompressible de latence qui peut s'avérer important devant le temps de calcul sur un maillage comptant peu d'éléments.

L'ajout d'un troisième niveau, en multipliant le nombre de communications requises par les solveurs sur les différents niveaux, augmente donc considérablement le temps de communication total. Cette situation est pénalisante, car la proportion du temps de communications sur le temps de calcul global est déjà importante pour le solveur DPCG

classique : sur le supercalculateur Babel[2], par exemple, elle se stabilise aux alentours de 8 % pour la simulation d'un écoulement turbulent dans le brûleur PRECCINSTA sur un maillage de 110 millions d'éléments sur 512 processeurs, s'élève à 16 % pour la simulation de l'écoulement autour d'une pale de turbine sur 276 millions d'éléments et 2048 processeurs, et frôle les 60 % pour la simulation, sur un maillage de 1,6 milliards d'éléments répartis sur 16384 processeurs, de l'atomisation d'un jet liquide en sortie d'un injecteur triple disque.

Par conséquent, l'approche que nous avons favorisée dans la suite de nos travaux a été de viser la réduction du nombre d'itérations sur le(s) maillage(s) grossier(s), et le comportement numérique de la méthode des Gradients Conjugués nous y a aidé. En effet, bien que la méthode CG utilisée à chaque niveau ne soit pas un *smoother* [70], on peut observer que le nombre d'itérations requis pour faire descendre la norme du résidu d'un ordre de grandeur est sensiblement constant, aussi longtemps que cette norme n'atteint pas des valeurs trop faibles ; dans ce dernier cas, l'erreur numérique devient trop importante pour que le solveur garde un comportement convenable, problème abordé dans la section 6.1.

En partant de ce constat, il devient clair que le nombre d'itérations requis par un solveur itératif comme le CG est directement impacté par deux paramètres auxquels le programmeur a un accès direct. Le premier est la solution initiale : *a priori*, plus elle est proche de la solution recherchée, plus la convergence sera rapide. Le second est bien entendu le critère de convergence, qui peut être assoupli en fonction des besoins de précision sur la solution calculée. Considérer ces deux paramètres dans le même temps revient en quelque sorte à attaquer le problème du nombre d'itérations du solveur "sur deux fronts", comme illustré par la figure 6.13 ; cette méthode double a pour but de réduire le nombre d'ordres de grandeur entre la norme du résidu initial et le critère de convergence à atteindre, ce qui réduira le nombre d'itérations en conséquence.

Ces méthodes ont un intérêt direct, puisqu'elles sont également applicables à une méthode de déflation standard. Par son application à la déflation à trois niveaux, nous espérons, d'une part parvenir à rendre la méthode à trois niveaux plus efficace que celle à deux niveaux dans des cas massivement parallèles, d'autre part parvenir à rendre robuste le solveur à trois niveaux. Concernant ce dernier point, il faut rappeler que le crash du solveur à trois niveaux est dû au fait que certains scalaires sur le niveau le plus grossier peuvent voir leur valeur diminuer considérablement au fil des itérations, jusqu'à ce qu'il soit numériquement impossible de les distinguer de 0 ; l'utilisation de ces scalaires dans des opérations de division entraine alors l'arrêt prématuré du programme. Si le nombre d'itérations à ce niveau est considérablement réduit, on peut s'attendre à ce que les valeurs de ces scalaires restent largement supérieures à la précision machine, et donc à ce que le solveur devienne robuste.

[2]Nous introduisons les caractéristiques techniques de ce supercalculateur dans la sous-section 6.6.1.

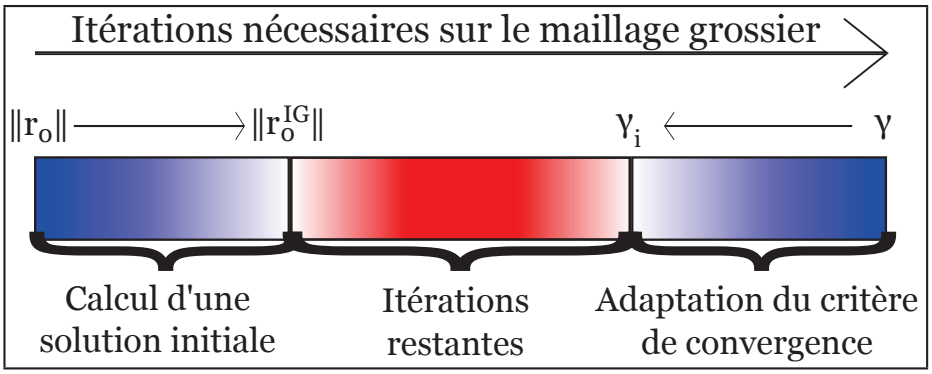

FIGURE 6.13 – *Illustration schématique, et limitée à deux niveaux de maillages, de la stratégie de réduction du nombre d'itérations sur maillage grossier. Le calcul d'une solution initiale adaptée permet de remplacer le résidu initial r_0 par un résidu r_0^{IG} de norme strictement inférieure ; l'adaptation du critère de convergence consiste à remplacer le seuil fixe γ, qui est également imposé à la norme du résidu sur le maillage fin, par une valeur γ_i recalculée à chaque itération sur le maillage fin, c'est-à-dire à chaque appel du solveur sur le maillage grossier.*

6.4 Création d'une solution initiale adaptée

Premier besoin mis en évidence par le solveur trois niveaux, la création de solutions initiales sur les maillages grossiers a été développée en prenant en compte le fonctionnement de l'algorithme de Gradients Conjugués utilisé. La méthode de recyclage des solutions précédentes que nous présentons dans cette thèse est ainsi adaptée à l'algorithme dans lequel elle sera appliquée. Nous présentons ici le principe général de la création de solutions initiales à partir de vecteurs quelconques, puis en particularisons le développement au cas du recyclage de solutions précédentes dans le contexte et avec les notations du solveur avec déflation standard à deux niveaux. L'indépendance linéaire de ces solutions n'étant a priori pas assurée, nous montrons l'algorithme qui nous a permis de résoudre ce problème en écartant les vecteurs indésirables. Enfin, nous abordons le problème du choix du nombre de vecteurs à recycler.

6.4.1 Principe

L'idée générale de la méthode utilisée pour la création d'une solution initiale est de déterminer une combinaison linéaire de n_{ig} vecteurs donnés, que l'on suppose indépendants dans un premier temps. La résolution itérative du système $\hat{A}d_i = r_i^h$ se fait alors à partir d'une solution initiale $\sum_{j=1}^{n_{ig}} \nu_j^i u_j^i$, avec $\{u_j^i\}_{j=1}^{n_{ig}}$ une famille de vecteurs indépendants. Comme on peut écrire $d_i = \sum_{j=1}^{n_{ig}} \nu_j^i u_j^i + \tilde{d}_i$, il devient clair que les coefficients ν_j^i doivent être choisis de façon à rendre $\tilde{d}_i$ orthogonal aux vecteurs $\{u_j^i\}_{j=1}^{n_{ig}}$.

En d'autres termes, $\tilde{d_i}$ est obtenu par orthogonalisation de d_i par rapport aux vecteurs $\{u_j^i\}_{j=1}^{n_ig}$. Cette base, introduite par Fischer dans [22], a été ici particularisée : comme l'algorithme du Gradient Conjugué, utilisé sur le niveau fin comme sur le niveau grossier, est basé sur l'utilisation de produits scalaires définis en fonction de la matrice du laplacien sur ce niveau, les coefficients ν_j^i doivent être déterminés par l'$\hat{A}$-orthogonalisation de d_i par rapport aux $\{u_j^i\}_{j=1}^{n_ig}$, soit

$$\forall k \in \{1; \ldots; n_{ig}\}, \langle \tilde{d_i}, u_k^i \rangle_{\hat{A}} = 0\,. \tag{6.9}$$

6.4.2 Recyclage des solutions précédentes

Le choix *ex nihilo* des vecteurs $\{u_j^i\}_{j=1}^{n_ig}$ aurait pu diminuer l'impact de la solution initiale sur le comportement du solveur sur le niveau grossier, mais surtout, il aurait introduit un élément pseudo-aléatoire dans le comportement global du solveur. Nous avons donc préféré déterminer la solution initiale en fonction de vecteurs déjà calculés par le code.

A ce titre, les solutions calculées pour les systèmes projetés précédents semblent des candidats idéaux, puisqu'on peut s'attendre à ce que les solution successives de ces systèmes aient des directions proches les unes des autres. L'exploration de cette méthode pour différents nombres de solutions recyclées a été accompagnée d'une réflexion sur la façon d'éviter l'instabilité numérique causée par une trop grande similitude des vecteurs recyclés.

6.4.2.1 Utilisation d'une seule solution

On peut écrire $d_i = \nu_i d_{i-1} + \tilde{d_i}$, avec $\nu_i \neq 0$ sous réserve de non-orthogonalité entre d_i et d_{i-1}, et $\tilde{d_i}$ orthogonal à d_{i-1}. Cette condition d'orthogonalité s'écrit $\langle d_i - \nu_i d_{i-1}, d_{i-1} \rangle = 0$, soit

$$\nu_i = \frac{\langle d_i, d_{i-1} \rangle}{\langle d_{i-1}, d_{i-1} \rangle}\,. \tag{6.10}$$

Le calcul de cette valeur nécessiterait de connaître d'avance la valeur de d_i, ce qui serait une raison suffisante pour utiliser l'$\hat{A}$-produit scalaire au lieu du produit scalaire usuel. Une autre raison, bien moins empirique, pour ce choix est qu'il a déjà été fait pour l'algorithme du Gradient Conjugué qui est utilisé à chaque niveau. Il est donc plus que raisonnable de garder à chaque niveau un unique produit scalaire et une définition unique de l'orthogonalité des vecteurs.

On peut ajouter à cet argument que l'algorithme des Gradients Conjugués sur le maillage projeté calcule une solution en y ajoutant à chaque itération une composante $\hat{A}$-orthogonale à toutes les précédentes ; par conséquent, seule la construction d'une solution initiale par ce procédé d'$\hat{A}$-orthogonalisation, c'est-à-dire en calculant préalablement l'$\hat{A}$-projection de la solution attendue sur un sous-espace vectoriel, permet effectivement

de diminuer le nombre maximal théorique d'itérations requises pour atteindre la solution. Cet argument peut être considéré comme faible, dans le sens où il ne considère que le comportement théorique de l'algorithme, mais il mène lui aussi à une construction cohérente de l'algorithme modifié.

En remplaçant le produit scalaire usuel, implicitement utilisé dans l'équation (6.10), par l'$\hat{A}$-produit scalaire, et en utilisant la commutativité de ce produit scalaire et la définition $\hat{A}d_i = r_i^h$, on obtient

$$\nu_i = \frac{d_{i-1}^T r_i^h}{d_{i-1}^T r_{i-1}^h} \,. \tag{6.11}$$

6.4.2.2 Utilisation de deux solutions

La méthode décrite ci-avant peut être étendue, de façon à créer une solution initiale par combinaison linéaire des solutions des deux systèmes précédents. Il s'agit alors de créer une solution initiale $\nu_0^i d_{i-2} + \nu_1^i d_{i-1}$ pour la résolution du système $\hat{A}d_i = r_i^h$, de façon à ce que $\tilde{d}_i = d_i - (\nu_0^i d_{i-2} + \nu_1^i d_{i-1})$ soit $\hat{A}$-orthogonal à la fois à d_{i-2} et d_{i-1}. On suppose dans un premier temps que d_{i-2} et d_{i-1} ne sont pas colinéaires afin d'assurer l'unicité des coefficients ν_0^i et ν_1^i ; le problème de l'éventuelle colinéarité des vecteurs recyclés sera traité dans la sous-section 6.4.3.

La condition d'$\hat{A}$-orthogonalité énoncée ci-avant peut s'écrire sous la forme d'un système de deux équations linéaires d'inconnues ν_0^i et ν_1^i, à savoir

$$\begin{cases} \left\langle d_i - \nu_0^i d_{i-2} - \nu_1^i d_{i-1}, d_{i-2} \right\rangle_{\hat{A}} = 0 \\ \left\langle d_i - \nu_0^i d_{i-2} - \nu_1^i d_{i-1}, d_{i-1} \right\rangle_{\hat{A}} = 0 \end{cases} \Leftrightarrow \begin{cases} d_i^T \hat{A} d_{i-2} - \nu_0^i d_{i-2}^T \hat{A} d_{i-2} - \nu_1^i d_{i-1}^T \hat{A} d_{i-2} = 0 \\ d_i^T \hat{A} d_{i-1} - \nu_0^i d_{i-2}^T \hat{A} d_{i-1} - \nu_1^i d_{i-1}^T \hat{A} d_{i-1} = 0 \end{cases} . \tag{6.12}$$

Ce système correspond à l'équation matricielle

$$\begin{pmatrix} a_1^1 & a_1^0 \\ a_1^0 & a_0^0 \end{pmatrix} \begin{pmatrix} \nu_1^i \\ \nu_0^i \end{pmatrix} = \begin{pmatrix} b_1 \\ b_0 \end{pmatrix} \tag{6.13}$$

avec

$$\begin{cases} a_0^0 = d_{i-2}^T r_{i-2}^h \\ a_1^0 = d_{i-1}^T r_{i-2}^h \\ a_1^1 = d_{i-1}^T r_{i-1}^h \\ b_0 = d_{i-2}^T r_i^h \\ b_1 = d_{i-1}^T r_i^h \end{cases} . \tag{6.14}$$

La solution exacte de ce système se calcule directement par

$$\nu_0^i = \frac{a_1^1 b_0 - a_1^0 b_1}{a_1^1 a_0^0 - (a_1^0)^2} \quad ; \quad \nu_1^i = \frac{a_0^0 b_1 - a_1^0 b_0}{a_0^0 a_1^1 - (a_1^0)^2} \,. \tag{6.15}$$

Le PCG utilisé pour la résolution du système projeté est alors modifié comme indiqué par l'algorithme 6, après la résolution des deux premiers systèmes projetés $\hat{A}d_{-1} = W^T b$ et $\hat{A}d_0 = r_0^h$, ainsi que le calcul de la valeur initiale $a_0^0 = d_0^T r_0^h$.

Algorithme 6 Algorithme pour la résolution du système projeté $\hat{A}d_i = r_i^h$ avec solution initiale $\nu_0^i d_{i-2} + \nu_1^i d_{i-1}$

Entrées : $\hat{A}, r_{i-2}^h, r_{i-1}^h, r_i^h, d_{i-2}, d_{i-1}, d_i, a_0^0$
$a_1^1 \leftarrow d_{i-1}^T r_{i-1}^h$
$a_1^0 \leftarrow d_{i-1}^T r_{i-2}^h$
$b_1 \leftarrow d_{i-1}^T r_i^h$
$b_0 \leftarrow d_{i-2}^T r_i^h$
$\nu_d \leftarrow a_0^0 a_1^1 - (a_1^0)^2$
$\nu_0^n \leftarrow a_1^1 b_0 - a_1^0 b_1$
$\nu_1^n \leftarrow a_0^0 b_1 - a_1^0 b_0$
$\nu_0 \leftarrow \nu_0^n / \nu_d$
$\nu_1 \leftarrow \nu_1^n / \nu_d$
Résoudre $\hat{A}d_i = r_i^h$ à partir de la solution initiale $\nu_0 d_{i-2} + \nu_1 d_{i-1}$
$a_0^0 \leftarrow a_1^1$

6.4.2.3 Généralisation à plusieurs solutions

Il est possible d'aller plus loin en utilisant les vecteurs $\{d_{i-j}\}_{j=1}^{n_{ig}}$ pour le calcul de la solution initiale du système $\hat{A}d_i = r_i^h$. Bien entendu, jusqu'à l'itération $n_{ig} - 1$, il n'y a pas encore n_{ig} solutions utilisables, mais seulement $i+1$ vecteurs d_k, pour k allant de -1 à $i-1$. On peut alors facilement écrire le nombre de vecteurs à utiliser pour le calcul de la solution initiale à l'itération i comme

$$n_{vect} = \min\{i+1, n_{ig}\} \,. \tag{6.16}$$

Les contraintes d'orthogonalité s'écrivent sous la forme d'un système linéaire $C^i \nu^i = s^i$, avec

$$C^i = \left((u_j^i)^T r_k^i\right)_{j,k=1}^{n_{vect}} \quad ; \quad \nu^i = \left(\nu_j^i\right)_{j=1}^{n_{vect}} \quad ; \quad s^i = \left((u_j^i)^T r_i^h\right)_{j=1}^{n_{vect}} \tag{6.17}$$

où

$$\forall j \in \{1; \ldots; n_{vect}\} \ , \ u_j^i = d_{i+j-n_{vect}-1} \,. \tag{6.18}$$

Cette équation est simplement une formalisation du fait que les vecteurs d_k utilisés sont pour k allant de -1 à $i-1$ tant que $i < n_{ig} - 1$, et de $i - n_{ig}$ à $i-1$ ensuite.

La matrice U^i contenant n_{ig} vecteurs est définie comme

$$U^i = \left(u_j^i\right)_{j=1}^{n_{ig}} \tag{6.19}$$

et, si $i < n_{ig} - 1$, c'est-à-dire $n_{vect} < n_{ig}$, les vecteurs u_j^i avec $j > n_{vect}$ sont nuls.

Pour la mise en oeuvre de ce calcul, on peut prendre en compte le fait que tous ou presque tous les vecteurs qui composent C^i seront présents dans C^{i+1} : tous tant que $n_{vect} < n_{ig}$, tous sauf un ensuite. L'implémentation peut alors se faire comme suit :

Après le calcul de d_{-1} : comme illustré par la figure 6.14, la valeur de d_{-1} est stockée dans la première colonne de U^0, et $C^0(1,1) = d_{-1}^T r_{-1}$.

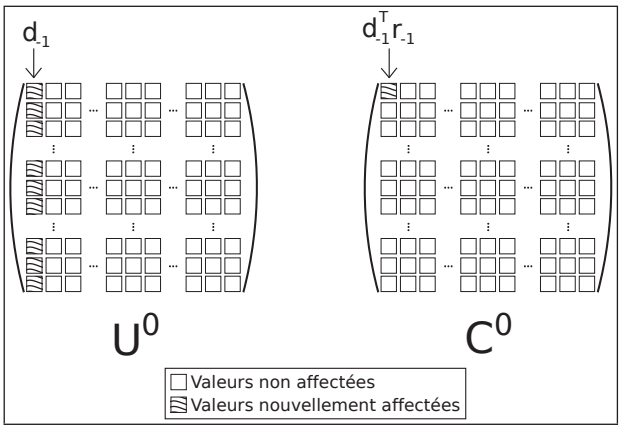

FIGURE 6.14 – *Schéma représentant la création des matrices U^0 et C^0 après le calcul de d_{-1}.*

Après le calcul de r_0 : la solution initiale est calculée comme dans la sous-section 6.4.2.1, c'est-à-dire

$$d_0^{IG} = \frac{(u_1^0)^T r_0}{C^0(1,1)} u_1^0 \,. \tag{6.20}$$

Après le calcul de d_0 : comme montré en figure 6.15, U^1 est en fait U^0 auquel on ajoute la valeur de d_0 en seconde colonne ; de la même façon, C^1 est une copie de C^0 à laquelle sont ajoutées, dans la deuxième colonne, les valeurs sur-diagonales $C^1(1,2) = d_{-1}^T r_0$ et $C^1(2,2) = d_0^T r_0$, soit

$$\begin{cases} C^1(1,2) = (u_1^1)^T r_0 \\ C^1(2,2) = (u_2^1)^T r_0 \end{cases} . \tag{6.21}$$

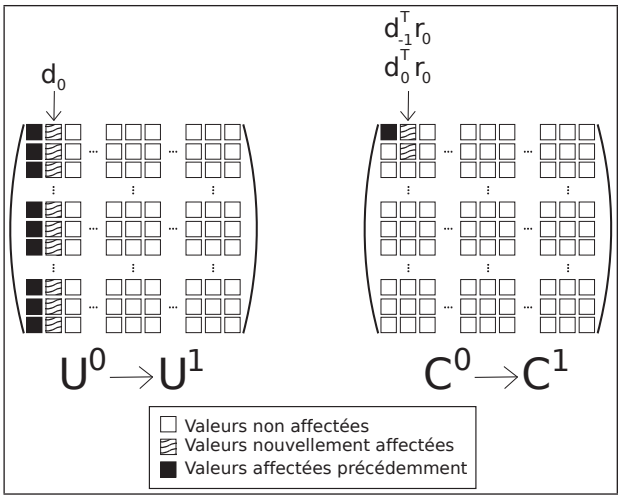

FIGURE 6.15 – *Schéma représentant la création des matrices U^1 et C^1 à partir de U^0 et C^0 après le calcul de d_0.*

Après le calcul de r_i, $i > 0$: la valeur de n_{vect} pour l'itération en cours est calculée selon l'équation (6.16), et les valeurs de s^i sont calculées selon les formules données par l'équation (6.17). Une fois le système linéaire $C^i\nu^i = s^i$ résolu, la solution initiale est calculée comme étant $d_i^{IG} = \sum_{j=1}^{n_{vect}} \nu_j^i u_j^i$.

Après le calcul de d_i, $i > 0$: selon le nombre d'itérations déjà réalisées, deux cas sont à distinguer.

- Si $i < n_{ig} - 1$, soit $n_{vect} < n_{ig}$, alors, comme illustré en figure 6.16, U^{i+1} et C^{i+1} sont des copies de U^i et C^i complétées par quelques nouvelles valeurs : le vecteur $u^i_{n_{vect}+1}$, égal à d_i, est ajouté à U^{i+1}, et la partie sur-diagonale de la colonne d'indice $n_{vect}+1$ de C^{i+1} reçoit les produits scalaires

$$\forall k \in \{1, \cdots, n_{vect}+1\}\ ,\ C^{i+1}(k, n_{vect}+1) = (u_k^{i+1})^T r_i\,. \tag{6.22}$$

- Si $i \geq n_{ig} - 1$, soit $n_{vect} \geq n_{ig}$, alors les matrices U^{i+1} et C^{i+1} sont générées à partir de U^i et C^i comme représenté sur la figure 6.17. Les vecteurs u_k^i deviennent les vecteurs u_{k-1}^{i+1} pour k allant 2 à n_{ig}, et le vecteur $u_{n_{ig}}^{i+1} = d_i$ est ajouté. D'une façon similaire, C^{i+1} reçoit des valeurs de C^i, car $C^{i+1}(j,k) = C^i(j+1, k+1)$ pour $k < n_{ig}$ et $j \leq k$. Ce décalage des éléments de la matrice C^i a pour résultat d'en effacer la première ligne et de créer une matrice dont la dernière colonne est vide. Il ne reste plus qu'à affecter aux éléments de la colonne n_{ig} de C^{i+1} les produits scalaires

$$\forall k \in \{1, \cdots, n_{ig}\}, C^{i+1}(k, n_{ig}) = (u_k^{i+1})^T r_i\,. \tag{6.23}$$

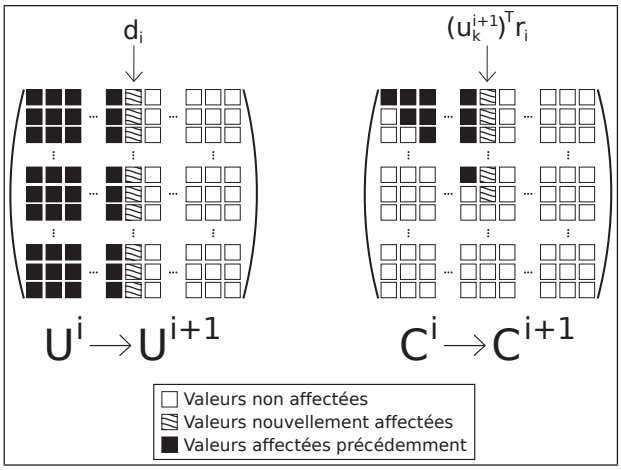

FIGURE 6.16 – *Schéma représentant la création des matrices U^{i+1} et C^{i+1} à partir de U^i et C^i après le calcul de d_i, pour $n_{vect} < n_{ig}$.*

6.4.3 Dépendance linéaire des solutions

La quasi-dépendance linéaire des solutions recyclées est le principal problème pouvant apparaître dans l'algorithme décrit ci-dessus. Certes, l'indépendance linéaire *stricto sensu* d'un ensemble de solutions successives est quasiment assurée en arithmétique exacte, tant que le nombre de solutions considérées reste petit devant leur dimension ; en revanche, les approximations commises par un calculateur, même en double précision, peuvent rendre l'algorithme de recyclage très sensible à une quasi-dépendance linéaire.

Une solution, presque naturelle, à ce problème serait de considérer les vecteurs recyclés l'un après l'autre ; ainsi, il devient possible de vérifier si un vecteur est "proche" d'une certaine combinaison linéaire des vecteurs précédents, pour ne pas le prendre en considération dans le calcul de la solution initiale le cas échéant – cette "proximité" est définie ci-après. Dans cette perspective, le choix de l'orthogonalisation successive des solutions recyclées semble s'imposer naturellement, puisqu'elle permet d'isoler de chaque vecteur sa composante orthogonale aux vecteurs précédents. Une décomposition QR utilisant la méthode de Gram-Schmidt est ainsi utilisée dans YALES2 pour calculer le pseudo-inverse de la matrice C^i, grâce auquel la solution initiale peut être calculée.

Reste à définir la notion de quasi-dépendance. Le plus simple est de considérer qu'un vecteur est une quasi-combinaison linéaire d'un ensemble de vecteurs si, une fois orthogonalisé en fonction de ces vecteurs, sa norme passe en-dessous d'un certain seuil. En pratique, on préfèrera considérer, non pas un seuil fixe, mais une valeur qui dépend des normes des vecteurs initiaux.

On calcule la racine carrée de la plus grande valeur diagonale, en valeur absolue, de la matrice $(C^i)^T C^i$. Cette valeur est en fait une approximation de $\|C^i\|_2$, qui se révèlerait exacte si C^i était diagonale (c'est-à-dire si les solutions recyclées étaient déjà orthogonales

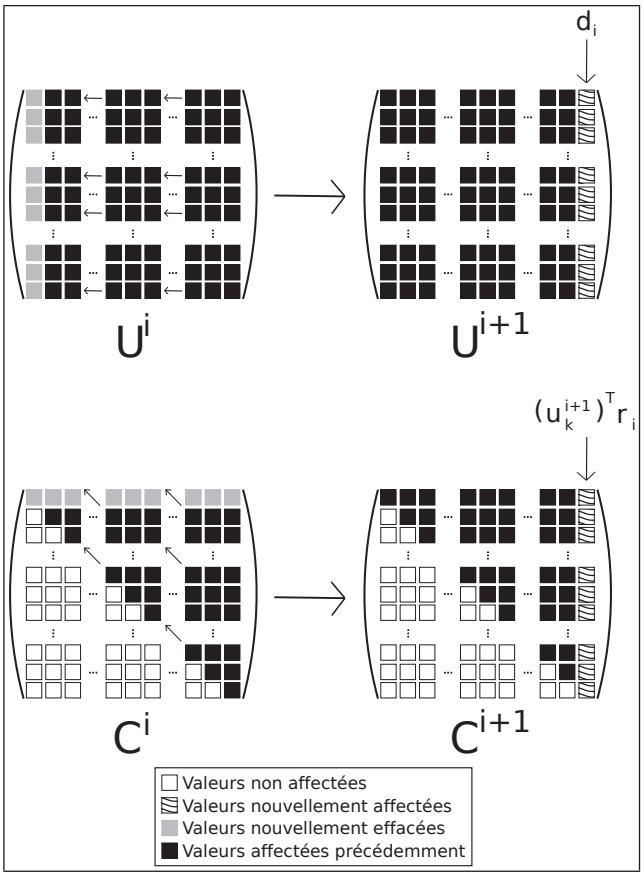

FIGURE 6.17 – *Schéma représentant la création des matrices U^{i+1} et C^{i+1} à partir de U^i et C^i après le calcul de d_i, pour $n_{vect} \geq n_{ig}$.*

les unes aux autres). On peut alors considérer que le vecteur w_j^i, résultat de la projection de u_j^i sur l'espace orthogonal à Vect $\left\{u_1^i, u_2^i, \ldots, u_{j-1}^i\right\}$, est négligeable si sa norme est plus petite que celle de C^i d'au moins μ ordres de grandeur, c'est-à-dire

$$\left\|w_j^i\right\| < 10^{-\mu}\rho_i\,. \tag{6.24}$$

De manière empirique, le choix de $\mu = 8$ a été fait après les premiers tests du recyclage de résidu dans YALES2. En outre, la matrice est réordonnancée de façon à ce que les vecteurs à recycler soient considérés dans l'ordre antichronologique, soit du plus récent $u_{n_{ig}}^i = d_{i-1}$ au plus ancien $u_1^i = d_{\max(-1,i-n_{ig})}$.

6.4.4 Choix du nombre de solutions recyclées

Le procédé décrit ci-dessus mène à la création "à la volée" d'une famille de vecteurs indépendants, par élimination éventuelle de certaines des solutions recyclées ; il assure

la stabilité de l'algorithme de recyclage des solutions précédentes, et ce quel que soit le nombre n_{ig} de solutions recyclables. Il semble donc que ce paramètre puisse être choisi aussi grand que possible, mais l'amélioration de la solution initiale calculée par l'algorithme de recyclage est de plus en plus faible à mesure que n_{ig} augmente, tandis que les coûts de calcul continuent d'augmenter ; cet effet est illustré en figure 6.18.

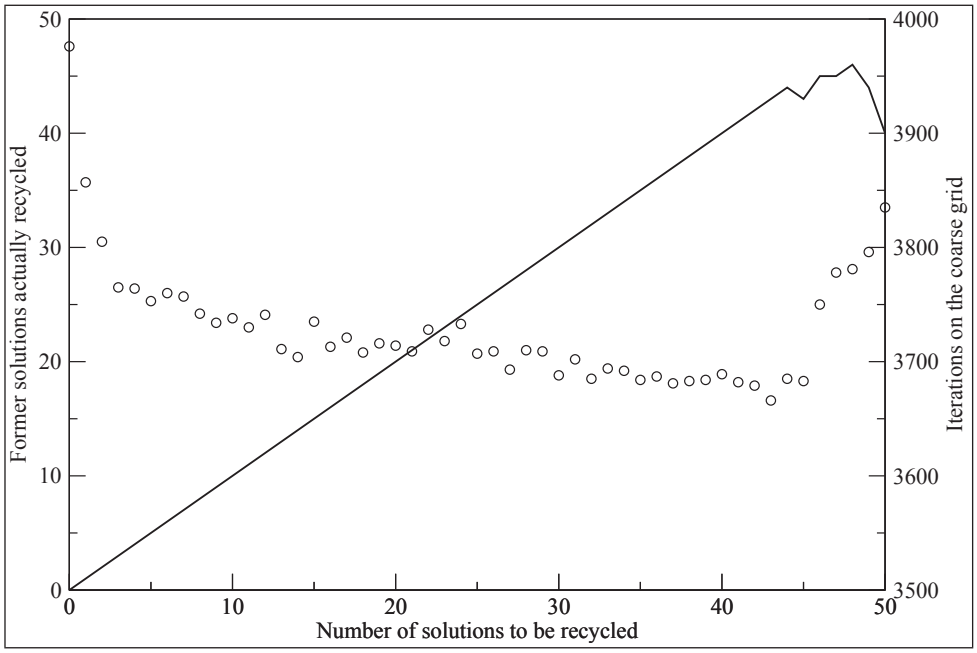

FIGURE 6.18 – *Influence de la valeur du paramètre n_{ig} sur le nombre total d'itérations sur le maillage grossier lors de la résolution de l'équation de Poisson pour la pression associée au premier pas de temps de la simulation du cas-test* **Cylindre_3D**. *Le nombre réel de vecteurs effectivement recyclés lors du cinquante-et-unième appel du PCG sur le maillage grossier est également représenté. Le trait plein représente le nombre de solutions réellement recyclées, et la graduation correspondante est à gauche du graphique ; les cercles donnent les nombres d'itérations sur le maillage grossier, et la graduation correspondante est sur l'axe de droite du graphique.*

Le maillage non structuré utilisé est constitué de 135000 cellules tétraédriques, réparties en groupes de 1000 cellules distribués sur 4 processeurs. On considère le premier pas de temps de la simulation, lors duquel l'équation de Poisson pour la pression est résolue par un DPCG, qui appelle à chaque itération le PCG sur le maillage grossier. Lors de la cinquantième itération du solveur sur le maillage fin, on appelle le solveur sur le maillage grossier pour la cinquante-et-unième fois ; chaque valeur du paramètre n_{ig} comprise entre 0 et 50 donne donc des résultats différents à ce stade.

On observe que la diminution du nombre d'itérations sur le maillage grossier, particulièrement marquée en début de courbe, devient plus faible et erratique pour des valeurs

de n_{ig} augmentant au-delà de quelques unités. D'autre part, la condition d'indépendance imposée aux solutions recyclées devient visible, dans ce cas de figure, pour des valeurs de n_{ig} au-dessus de 40 ; des solutions sont alors éliminées, et ce d'une façon très variable puisque dépendant également des itérations précédentes. En effet, le calcul d'une solution initiale différente lors d'un appel précédent implique le calcul d'une solution différente, bien que respectant le même critère de convergence ; le comportement du solveur est ainsi légèrement modifié sur le long terme. On peut cependant s'attendre à ce que, lors des pas de temps suivants également, le choix d'une valeur de n_{ig} supérieure à quelques dizaines ne se répercute pas sur le véritable nombre de solutions recyclées, mais seulement sur le léger surcoût nécessaire aux calculs des solutions initiales. En outre, au-dessus de 20 à 30 solutions recyclées, l'impact sur les temps de calcul n'est plus visible, voire même négatif, et ce malgré l'utilisation effective d'un plus grand nombre de vecteurs pour la construction de la solution initiale.

L'impact du choix de cette valeur sur les temps de calcul a été étudié conjointement à celui du critère de négligeabilité introduit dans la section suivante, et il est ressorti de cette étude que, indépendamment de la configuration étudiée et des paramètres choisis, la valeur $n_{ig} = 25$ constitue un choix efficace et sûr.

6.5 Adaptation du critère de convergence

Tandis que la création de solutions initiales adaptées permet de se rapprocher de la solution cherchée avant même la première itération de l'algorithme CG, l'adaptation du critère de convergence vise à attaquer "sur l'autre front" (voir figure 6.13, page 116) le problème du nombre d'itérations sur le maillage grossier.

Comme nous le savons, la résolution des systèmes sur le maillage grossier est faite par un solveur PCG, qui modifie à chaque étape l'approximation d_i^* de la solution d_i, et ce jusqu'à ce que le résidu $r_i^* = r_i^h - \hat{A}d_i^*$ soit tel que

$$\|r_i^*\|_\infty < \gamma_i \,. \tag{6.25}$$

La valeur de γ_i est souvent fixée à la valeur γ utilisée pour le critère de convergence sur le maillage fin, ou bien calculée à partir de γ. L'idée principale de cette partie de notre travail de recherche est que la précision d'une solution sur maillage grossier ne devrait pas nécessairement être celle exigée de la solution sur le maillage fin ; plus précisément, il a été envisagé de considérer une valeur de γ_i différente pour chaque i, et calculée à chaque itération de façon à ne pas détériorer le comportement global du solveur. Nous développons dans cette section le principe de cette méthode et le développement qui en a été fait, puis nous abordons le problème du choix du paramètre de cette méthode, nommé "critère de négligeabilité".

6.5.1 Principe et développement

Une fois qu'une valeur de d_i^* de norme inférieure à γ_i a été déterminée, la solution exacte d_i du système peut être exprimée comme la somme de la solution calculée et d'une erreur $\Delta d_i = d_i - d_i^*$. Suivre les étapes de calcul suivantes du solveur sur le maillage fin permet de déterminer l'impact de cette erreur absolue sur le niveau fin ; une valeur de γ_i peut alors être déterminée de façon à respecter un certain critère de déviation sur le maillage fin.

Pour être plus précis, l'idée initiale de ce travail était l'étude de l'impact de Δd_i sur la nouvelle direction de descente calculée p_i^*, qu'il aurait été souhaitable de pouvoir écrire comme la somme de la direction de descente théorique p_i, calculée directement à partir de d_i, et d'une erreur Δp_i. En imposant qu'une certaine norme de Δp_i, sans doute la norme infinie pour des raisons à la fois de commodité et de cohérence, soit négligeable devant la norme de p_i, nous aurions alors pu remonter à une expression d'un γ_i assurant cette condition.

En suivant les étapes de l'algorithme 5 qui suivent immédiatement la résolution de $\hat{A}d_i = r_i^h$, et en considérant par simplicité que les calculs effectués avant la résolution de ce système sont tous exacts, on obtient directement

$$\Delta w_i = w_i^* - w_i = W \Delta d_i \,, \tag{6.26}$$

puis

$$\Delta \beta_i = \beta_i^* - \beta_i = \frac{r_i^T W \Delta d_i}{r_{i-1}^T w_{i-1}} \,, \tag{6.27}$$

et enfin l'équation souhaitée

$$\Delta p_i = p_i^* - p_i = W \Delta d_i + \frac{r_i^T W \Delta d_i}{r_{i-1}^T w_{i-1}} p_{i-1} \,. \tag{6.28}$$

Il s'avère hélas difficile d'exprimer la norme de ce Δp_i en fonction de γ_i, et encore plus difficile de trouver un minorant satisfaisant pour γ_i qui permette de respecter une majoration imposée sur une des normes usuelles de Δp_i ou, plus exactement, sur le rapport entre la norme de Δp_i et la norme de p_i. Un critère plus simple a donc dû être déterminé. Nous avons choisi de nous focaliser sur w_i, que nous appellerons dans ce qui suit la *base de descente*, car la direction de descente p_i est obtenue par l'$\hat{A}$-orthogonalisation de w_i par rapport à p_{i-1}.

L'équation (6.26) permet de réécrire le résidu sur le maillage grossier comme

$$\tilde{r}_i = \hat{A} \, (d_i - d_i^*) = W^T A \Delta w_i \,. \tag{6.29}$$

La valeur $W^T A \Delta w_i$ est ainsi explicitée comme nouvelle expression du résidu sur le maillage grossier, et c'est sur elle que s'appliquera le critère de convergence que l'on peut

réécrire $\left\|W^T A\Delta w_i\right\|_\infty < \gamma_i$. Limiter la déviation, non pas de $\|\Delta w_i\|_\infty$ par rapport à $\|w_i\|_\infty$, mais de $\left\|W^T A\Delta w_i\right\|_\infty$ par rapport à $\left\|W^T A w_i\right\|_\infty$, permet donc de déterminer directement une valeur de γ_i.

Comme $w_i = M^{-1} r_{i-1} - W d_i$, on peut écrire

$$W^T A w_i = W^T A M^{-1} r_{i-1} - \hat{A} d_i = \left(r_i^h + W^T r_i\right) - r_i^h \,. \tag{6.30}$$

Souhaiter que $\left\|W^T A\Delta w_i\right\|_\infty$ soit négligeable devant $\left\|W^T A w_i\right\|_\infty$, c'est-à-dire que la déviation sur $W^T A w_i$ soit négligeable au sens de la norme infinie, crée une condition très difficile à respecter, car la norme infinie de $W^T r_i$ a tendance à être très petite. On peut par contre imposer que $W^T A\Delta w_i$ soit négligeable devant chacun des deux termes de $W^T A w_i$ obtenus dans l'équation (6.30), soit

$$\begin{cases} \left\|W^T A\Delta w_i\right\|_\infty \leq C_N \left\|r_i^h + W^T r_i\right\|_\infty \\ \left\|W^T A\Delta w_i\right\|_\infty \leq C_N \left\|r_i^h\right\|_\infty \end{cases} , \tag{6.31}$$

et cette condition peut s'écrire sous la forme

$$\gamma_i \leq C_N \min\left(\left\|r_i^h + W^T r_i\right\|_\infty, \left\|r_i^h\right\|_\infty\right) . \tag{6.32}$$

En pratique, on évite des valeurs de γ_i trop petites en les ramenant au critère de convergence sur le maillage fin γ si la condition exprimée par l'équation (6.32) donne une valeur plus petite ; le calcul de γ_i dans YALES2 est donc effectué selon la formule

$$\gamma_i = \max\left[\gamma, C_N \min\left(\left\|r_i^h + W^T r_i\right\|_\infty, \left\|r_i^h\right\|_\infty\right)\right] . \tag{6.33}$$

6.5.2 Choix du critère de négligeabilité

La définition du critère de négligeabilité ne donne *a priori* qu'un encadrement grossier, à savoir $0 < C_N < 1$, auquel vient s'ajouter une conception plus empirique de ce que peut être un nombre négligeable devant un autre. On serait ainsi tenté d'imposer une valeur de C_N de l'ordre de 10^{-3} ou 10^{-4}, mais la question d'une valeur optimale de ce critère reste ouverte, ainsi que celle, plus importante encore, de la dépendance de cette valeur au problème à résoudre. En effet, rien ne semble garantir qu'une valeur de C_N adaptée à la résolution d'un système linéaire donné sera adéquate pour un système plus grand ou de conditionnement plus élevé, par exemple.

Cependant, les résultats de différents tests ont semblé montrer une certaine constance des valeurs de C_N à choisir. Les cas-tests étudiés sont très variés, de la résolution d'une équation de Poisson modèle obtenue sur un maillage en deux dimensions à celle de l'équation de Poisson pour la pression à chaque pas de temps lors de la simulation d'un écoulement tridimensionnel dans des configurations réalistes. Les résultats concernant le choix de C_N sont présentés dans la partie 6.6.2.2.

6.6 Performances du solveur RA-DEF2

Avant la mise en oeuvre dans le solveur à trois niveaux des deux méthodes décrites ci-dessus, c'est-à-dire le calcul de solutions initiales *ad hoc* et l'adaptation du critère de convergence sur les maillages grossiers, nous avons voulu constater l'efficacité de ces méthodes en les implantant dans le DPCG du solveur YALES2. En greffant ces deux techniques dans le solveur bi-niveaux A-DEF2, nous en avons fait un solveur plus général, nommé RA-DEF2(n_{ig} ;C_N), dont l'efficacité dépend du choix des deux paramètres n_{ig} et C_N, et tel que RA-DEF2(0 ;0) est strictement équivalent au solveur A-DEF2 originel.

Une batterie de tests a été effectuée pour étudier l'efficacité de ce solveur et déterminer ses paramètres optimaux. Dans cette section, après avoir détaillé les configurations étudiées, nous déterminerons des paramètres quasi-optimaux pour le solveur RA-DEF2 et étudierons l'impact de ce nouveau solveur sur le nombre d'itérations sur le maillage grossier, sur les temps de calcul et de communications, ainsi que sur le *weak scaling* du solveur. ENous nous intéresserons enfin brièvement à la robustesse du solveur sur une simulation massivement parallèle effectuée sur un maillage de 17,8 milliards d'éléments tétraédriques.

6.6.1 Configurations étudiées

Comme précédemment mentionné, divers cas-tests ont été étudiés, allant de simples tests de validation à des simulations d'écoulements turbulents, parfois diphasiques, dans des géométries réalistes complexes. Sont listées ci-après les dénominations des cas étudiés et leurs descriptions.

Poisson_2D est un cas-test correspondant à la résolution d'une équation de Poisson sur un carré unité centré en l'origine, maillé avec des éléments triangulaires. Le membre de droite de l'équation associée à ce cas-test est une combinaison linéaire de signaux sinusoïdaux de fréquences différentes ; une fonction f est définie par

$$\begin{aligned} f(x,y) = 500\sin(\pi x)\sin(\pi y) + 1000\sin\left(\frac{\pi x}{0,2}\right)\sin\left(\frac{\pi y}{0,2}\right) \\ + 6000\sin\left(\frac{\pi x}{0,033}\right)\sin\left(\frac{\pi y}{0,041}\right) \end{aligned} \tag{6.34}$$

et la valeur du second membre sur un volume de contrôle donné est le produit de la valeur de f sur le noeud correspondant par le volume du volume de contrôle. Le maillage est plus raffiné au centre et s'élargit vers les bords, comme représenté par la figure 6.19, ce qui donne un membre de droite similaire à celui de la figure 6.20.

Cylindre_3D est un cas-test correspondant à la simulation d'un écoulement à température constante dans un parallélépipède rectangle, autour d'un obstacle cylindrique

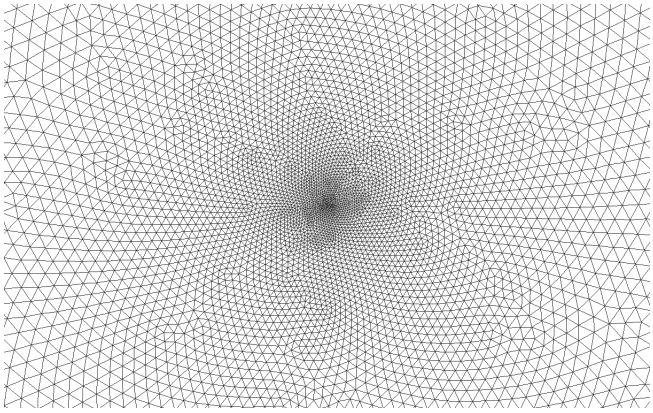

FIGURE 6.19 – *Zoom sur le centre du maillage du cas-test* **Poisson_2D**. *Le ratio entre le plus grand et le plus petit volume de cellule du maillage est d'environ 80000.*

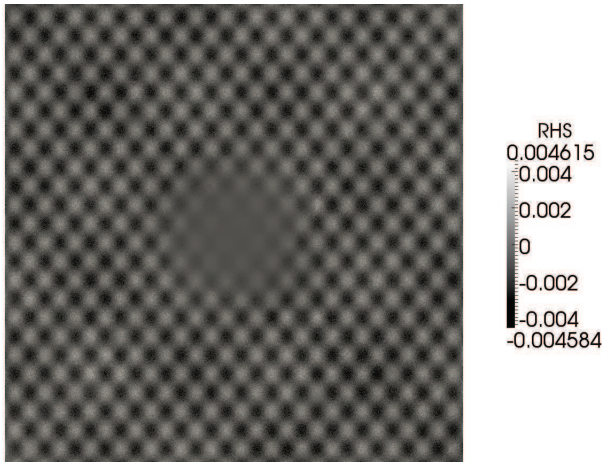

FIGURE 6.20 – *Membre de droite de l'équation de Poisson résolue pour le cas-test* **Poisson_2D**.

dont l'axe central est orthogonal à la vitesse d'entrée de l'écoulement. Le domaine est de longueur 60 cm, de largeur 30 cm et de hauteur 5 cm, avec des conditions de périodicité s'appliquent sur la hauteur. Le fluide a une vitesse d'entrée de 15 $m.s^{-1}$, constante sur tout le plan d'entrée, qui est la paroi gauche du domaine, dans la direction orthogonale au plan d'entrée. Le cylindre, de diamètre 1 cm, est placé à 10 cm de l'entrée, à mi-largeur, parallèle à un des trois axes du repère, et traverse le domaine sur toute sa hauteur. Des murs glissants délimitent le domaine. Le maillage le moins raffiné pour cette configuration, qui compte 491 000 éléments, est représenté en figure 6.21.

Preccinsta est présenté en sous-section 5.1.1, et les références bibliographiques corres-

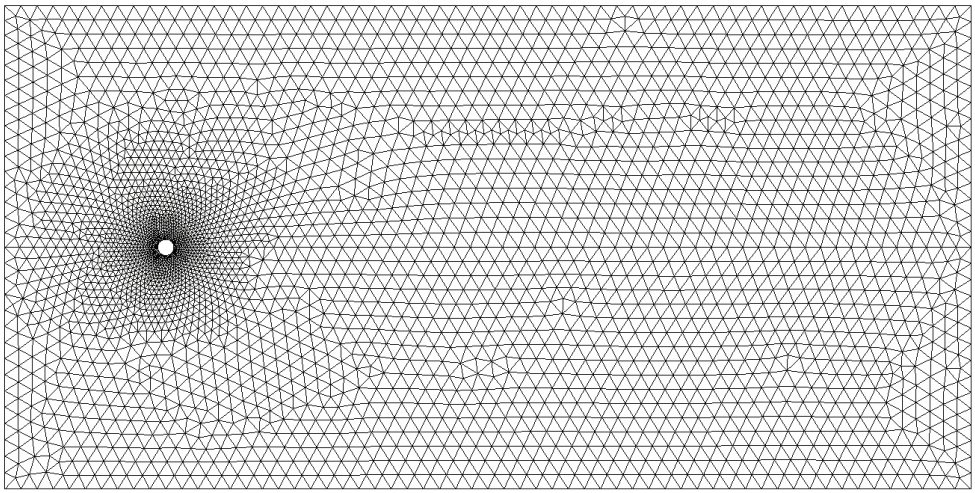

FIGURE 6.21 – *Vue du dessus d'un maillage de 491 000 éléments sur le domaine du cas-test* **Cylindre_3D.**

pondantes y sont données. Les deux maillages considérés sur cette géométrie dans cette partie comptent respectivement 1,7 millions et 110 millions d'éléments tétraédriques. Une représentation d'isocontours de critère Q colorés par la vitesse, obtenue grâce à une simulation massivement parallèle de cette configuration sur un maillage de 2,6 milliards d'éléments tétraédriques, est donnée par la figure 6.22.

Pale_TC est un cas-test correspondant à la simulation d'un écoulement turbulent autour de la pale de turbine T7.2 issue du projet de recherche européen AITEB2. Cette simulation est réalisée pour l'étude des effets de la turbulence sur les transferts de chaleur. La pale T7.2 a été conçue par MTU, pour représenter un design archétypique d'un rotor de turbine refroidie à basse pression ; pour des conditions d'entrée données, la forme de son intrados crée une importante zone de recirculation qui induit de hautes valeurs d'intensité turbulente sous la pale.

Cette configuration a été étudiée expérimentalement par Ladisch *et al.* [45], et numériquement par Lutum et Cottier [50]. Cependant, la méthode RANS utilisée par ces derniers ne permet pas de capturer correctement la transition à la turbulence sur l'extrados, ce qui a d'importantes conséquences sur l'intensité des transferts de chaleur ; par conséquent, une étude LES est actuellement réalisée par Nicolas Maheu et Vincent Moureau, au sein du CORIA, pour améliorer les résultats numériques sur cette configuration [51].

L'épaisseur maximale de la pale vaut 16.53 mm et sa longueur de corde est de 73.93 mm. Pour les simulations, la vitesse d'entrée a été fixée à 29.16 $\mathrm{m.s^{-1}}$; le fluide est à une température de 350K et la pale est refroidie à 290K. Une condition de non-glissement est imposée sur la paroi de la pale. Le nombre de Reynolds résultant est

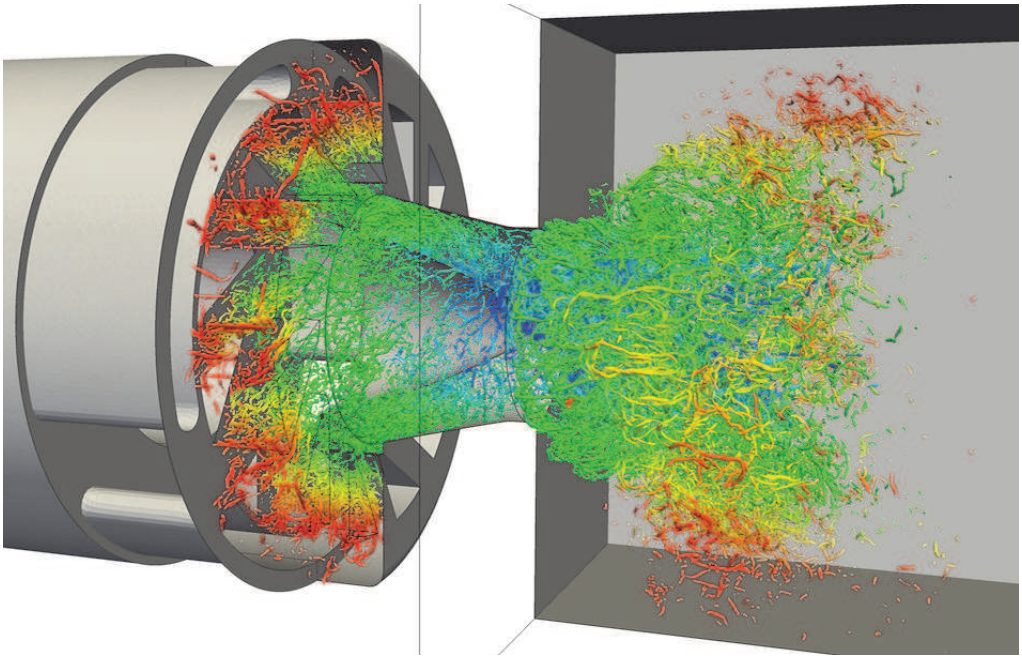

FIGURE 6.22 – *Iso-contours de critère Q colorés par la norme de la vitesse, obtenus par une simulation du cas-test* **Preccinsta.**

de 150 000 en sortie de domaine.

Un des maillages utilisés pour ces simulations est représenté en figure 6.23 ; tous les maillages de cette configuration sont sur le même domaine et respectent les mêmes ratios de tailles de mailles, notamment un rapport de 100 entre les plus grandes et les plus petites arêtes du maillage. Les plus petites arêtes de mailles mesurent entre 30 microns environ, dans le maillage de 34,5 millions d'éléments étudié ci-après, le moins fin pour cette configuration, et 2 microns environ pour le maillage le plus raffiné, qui compte 17,8 milliards d'éléments. La bounding box de chacun de ces maillages mesure 18.1 cm par 9.87 cm par 2.85 cm, et une périodicité est appliquée dans les directions y et z. Une section d'un champ de vitesse instantané pour la simulation sur un maillage de 17,8 milliards d'éléments est montré en figure 6.24, et un isocontour de critère Q autour de la pale, coloré par la magnitude de la vitesse, calculé sur le même maillage, est représenté sur la figure 6.25. Le supercalculateur Curie, sur lequel cette simulation a été effectuée, est présentée ci-après en même temps que le supercalculateur Babel.

L'Injecteur à Triple Disque représenté par l'acronyme **ITD** dans ce qui suit est un cas-test qui consiste à modéliser l'atomisation primaire d'un fuel liquide dans de l'air sous conditions atmosphériques, avec des propriétés réalistes pour le liquide et le gaz. L'injecteur étudié est formé par trois disques décentrés de diamètres différents, ce

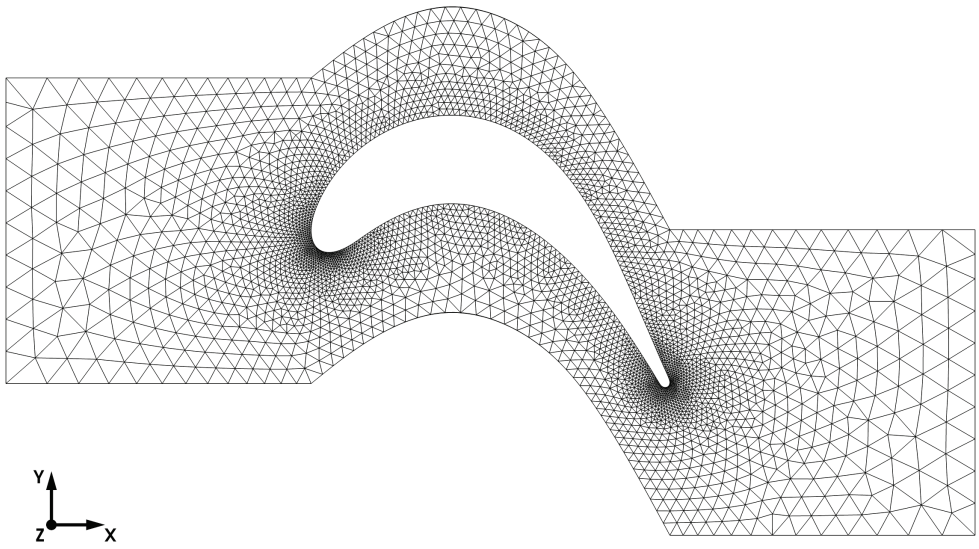

FIGURE 6.23 – *Section longitudinale d'un maillage autour de la pale T7.2.*

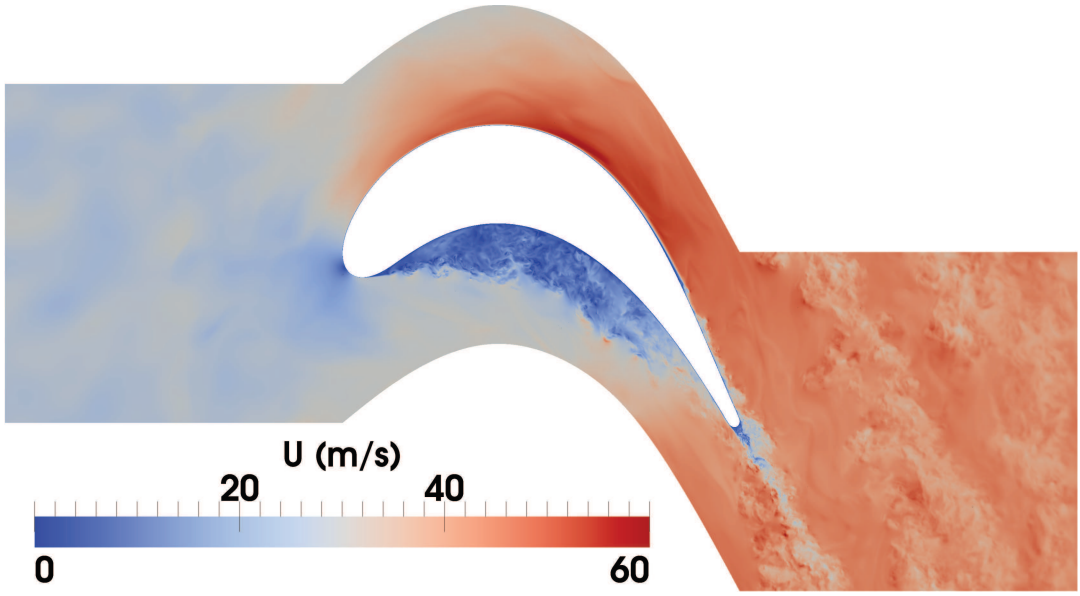

FIGURE 6.24 – *Section longitudinale d'un champ de vitesse instantané autour de la pale T7.2, calculé sur un maillage de 17,8 milliards d'éléments tétraédriques sur 16 384 processeurs de la machine Curie.*

qui crée des zones de forte recirculation dans le liquide afin de former une nappe à la sortie de l'injecteur. Cette nappe se désintègre rapidement pour former de petites

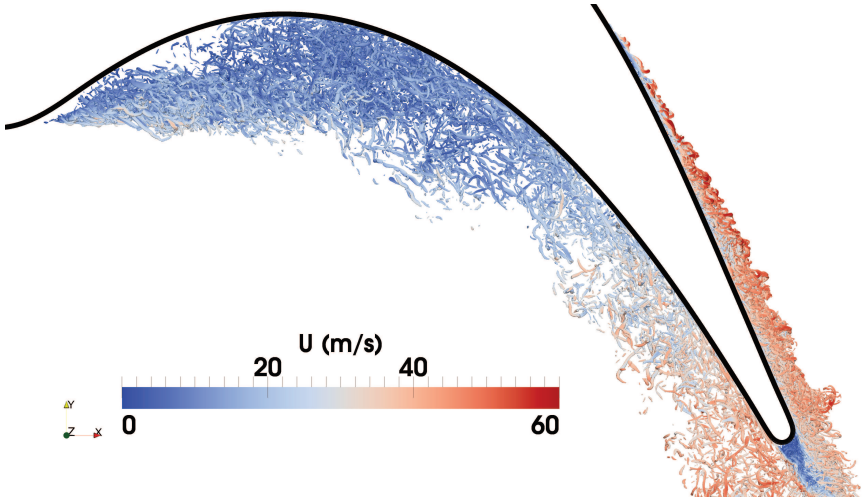

FIGURE 6.25 – *Isocontours de critère Q coloré par la magnitude de la vitesse autour de la pale T7.2, calculé sur un maillage de 17,8 milliards d'éléments tétraédriques sur 16 384 processeurs de la machine Curie.*

gouttelettes, même pour des pertes de charge modérées.

Plusieurs campagnes de mesures ont été conduites par Grout *et al.* [28] pour l'établissement de diagnostics optiques avancés. Dans les simulations décrites ici, l'intérieur de l'injecteur et la base de la nappe liquide sont tous les deux simulés. La géométrie de l'injecteur, dont la sortie circulaire a un diamètre de 180 μm, et la nappe liquide obtenue par la simulation sont montrées en figure 6.26. Le nombre de Reynolds du fluide est d'environ 3700, et le nombre de Weber fluide vaut $We_L = 1061$.

Cet écoulement diphasique est modélisé grâce à une méthode Ghost-Fluid [20] et le suivi d'interface est basé sur l'approche Levelset conservative [16]. Ce type d'écoulement représente un défi particulier pour les solveurs linéaires, en ce que l'équation de Poisson qui découle des équations qui gouvernent l'écoulement est de la forme

$$\nabla \cdot \left(\frac{1}{\rho} \nabla P \right) = RHS \,, \tag{6.35}$$

avec une densité ρ dont les valeurs peuvent être séparées de trois ordres de grandeur sur l'interface suivant la phase considérée.

Selon les cas étudiés et les tailles de maillage, plusieurs machines ont été utilisées pour les simulations.

Une machine Intel Xeon X5680 à 12 coeurs à 3,33GHz a été utilisée pour les plus petits cas-tests, pour lesquels seuls les nombres d'itérations ont une importance pour

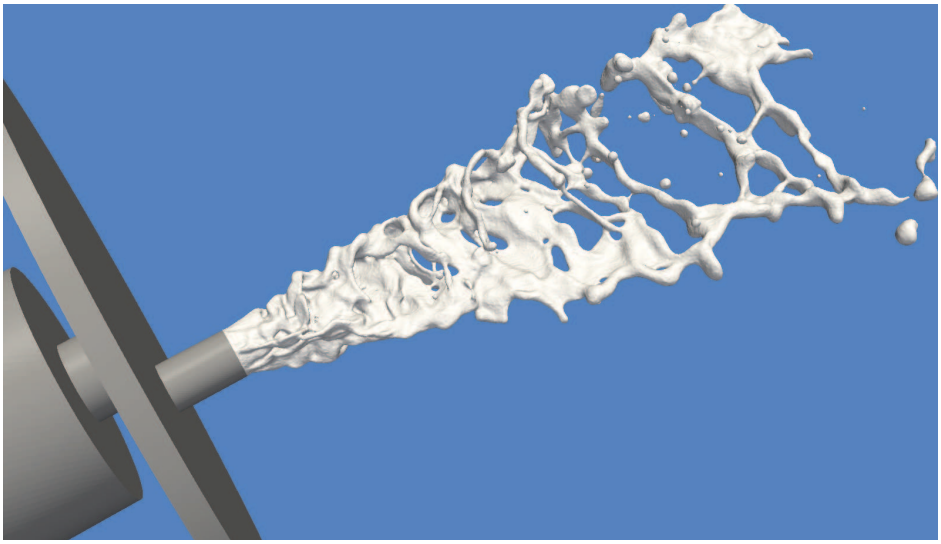

FIGURE 6.26 – *Géométrie de l'injecteur à triple disque et nappe liquide simulée sur un maillage de 1,6 milliard de cellules tétraédriques.*

notre étude. En effet, contrairement aux autres machines utilisées, celle-ci n'est pas dédiée aux simulations : les temps de calcul obtenus ne sont donc pas forcément représentatifs, ni comparables entre eux.

Le cas-test **Poisson_2D**, sur un maillage de 3,23 millions d'éléments triangulaires avec 200 cellules par groupe, a été simulé sur 4 coeurs de cette machine, et le même cas-test sur un maillage de 12,9 millions d'éléments avec 400 cellules par groupe a été simulé sur 8 coeurs de cette même machine.

Le cas-test **Cylindre_3D**, sur un maillage de 491 000 éléments tétraédriques et avec 500 cellules par groupe, et le cas-test **Preccinsta**, sur un maillage de 1,7 million d'éléments tétraédriques et avec 2000 cellules par groupe, ont été simulés sur 4 coeurs de calcul sur cette machine.

Babel est la machine IBM Blue Gene/P utilisée pour la plus grande partie des tests. Cette machine, détenue par l'Institut du Développement et des Ressources en Informatique Scientifique (IDRIS), compte 10 racks de 1024 noeuds de calcul, chacun constitué de 4 coeurs à 850MHz.

Le cas-test **Preccinsta**, sur un maillage de 110 millions d'éléments tétraédriques, avec 2000 cellules par groupe, a été simulé par 512 coeurs (128 noeuds) de cette machine.

Le cas-test **Pale_TC**, sur un maillage de 34,5 millions de tétraèdres et avec 500 cellules par groupe, a été simulé sur 512 coeurs (128 noeuds) de cette machine. Le même cas-test sur un maillage de 2,2 milliards d'éléments, avec 5000 cellules par

groupe, a été simulé sur 16384 coeurs (4096 noeuds) de cette machine.

Le cas-test **ITD**, sur un maillage de 1,6 milliards d'éléments tétraédriques, avec 1000 cellules par groupe, a été simulé sur 16384 coeurs (4096 noeuds) de cette machine.

Curie est un supercalculateur Tier-0 Bull-X, situé au Très Grand Centre de Calcul (TGCC) de Bruyères-le-Châtel et géré par le CEA. Démarré le premier mars 2012, il est séparé en trois différentes ressources visant des types de calculs variés, pour des codes hybrides OpenMP/MPI ou des codes purement MPI. Les "noeuds fins" dédiés aux codes MPI sont constitués de 9648 octocores Intel Xeon de nouvelle génération, pour une vitesse de calcul totale de 1,36 pétaFLOPS, chaque coeur de calcul ayant une fréquence de 2,7 GHz.

Le cas-test **Pale_TC** sur des maillages de 35 millions, 280 millions, 2,2 milliards et 17,9 milliards de mailles tétraédriques, chacun d'eux avec 5000 cellules par groupe, a été simulé sur respectivement 128, 1024, 8192 et 8192 coeurs (soit 16, 128, 1024 et 1024 noeuds) de cette machine. Pour distinguer ces simulations de celles effectuées sur Babel, nous les désignerons par **Pale_TC_Curie** dans cette section.

Il faut également préciser que les valeurs moyennes de la sous-section 6.6.3 ont été calculées sur différents nombres d'étapes temporelles selon les cas étudiés. Les 100 premiers pas de temps sont considérés pour le cas-test **Cylindre_3D** et pour toutes les simulations du brûleur PRECCINSTA. En ce qui concerne le cas-test **Pale_TC**, les 400 premiers pas de temps ont été pris en compte pour la simulation sur le maillage de 34,5 millions d'éléments sur 256 coeurs de calcul, et les 750 premiers pour la même simulation sur 512 coeurs ; 45 pas de temps ont été simulés sur le maillage de 2,2 milliards d'éléments pour la **Pale_TC**, et 23 seulement sur le cas-test **ITD**, car les simulations sans solutions initiales ni critère de convergence adaptatif ne sont pas allées au-delà de ce nombre de pas de temps avec les limites de temps de calcul imposées. Tous les résultats pour le cas-test **Pale_TC_Curie** prennent en compte les 200 premiers pas de temps, sauf mention contraire dans la sous-section 6.6.5.

Enfin, le critère de convergence γ sur maillage fin a été fixé à 10^{-12} pour le cas-test **Poisson_2D**, 10^{-10} pour toutes les simulations du cas-test **Cylindre_3D** et du brûleur **Preccinsta**, 10^{-9} pour toutes les simulations sur l'**Injecteur Triple Disque**, et 5×10^{-9} pour toutes les simulations autour de la **Pale_TC**.

6.6.2 Réduction du nombre d'itérations

Nous présentons dans cette partie l'influence de chacune des deux techniques sur la réduction effective du nombre d'itérations sur le maillage grossier pour plusieurs cas-tests, puis leur effet conjoint.

6.6.2.1 Influence du calcul de solutions initiales

Le premier paramètre dont nous avons étudié l'impact est le nombre n_{ig} de solutions recyclées pour les calculs de solutions initiales sur le maillage grossier. La table 6.2 donne, pour des cas-tests dont les tailles de maillage vont de 490 000 à 110 millions d'éléments, les gains en termes de nombre d'itérations pour les valeurs de n_{ig} donnant les meilleurs résultats parmi celles testées. Pour les cas-tests **2D_Poisson** et **Cylindre_3D**, ainsi que le maillage à 1,7 millions d'éléments sur le brûleur **Preccinsta,** les valeurs de l'ensemble $\{5; 10; 25; 50\}$ ont été testées ; pour le maillage à 110 millions d'éléments de **Preccinsta**, l'ensemble de test est $\{1; 2; 5; 10; 20\}$.

Cas-test	2D_Poisson		Cylindre_3D	Preccinsta	
Eléments ($\times 10^6$)	3,23	12,9	0,49	1,7	110
Coeurs	4	8	4	4	512
Pourcentage de gain	2,67 %	2,19 %	11,8 %	8,11 %	9,67 %

TABLE 6.2 – *Gains sur les nombres moyens d'itérations sur le maillage grossier des cas-tests* **2D_Poisson**, **Cylindre_3D** *et* **Preccinsta** *induits par le calcul de solutions initiales adaptées.*

On peut préciser, à ce stade, que les meilleurs résultats n'ont pas été systématiquement obtenus pour les plus hautes valeurs de n_{ig} ; dans le cas de la simulation du brûleur **Preccinsta** sur un maillage de 110 millions d'éléments, par exemple, la valeur $n_{ig} = 1$ a donné la plus importante réduction du nombre d'itérations. Les autres valeurs non-nulles testées, cependant, n'ont pas notablement détérioré le comportement du solveur sur le maillage grossier, avec des écarts relatifs de nombres d'itérations compris entre 0,5 et 2,6 % par rapport au cas $n_{ig} = 1$. A l'impact de l'erreur numérique près, le solveur sur maillage grossier est resté, dans ce cas précis, insensible à l'ajout de nouvelles solutions recyclées.

Ces premiers résultats semblent montrer que le calcul de solutions initiales sur le maillage grossier, décrit en section 6.4, peut présenter un intérêt limité pour des systèmes "simples", mais se révéler bien plus utile pour des simulations dans des géométries plus complexes. Le choix d'une valeur efficace pour n_{ig} s'annonce très dépendant du cas étudié, ce pourquoi il semble que cette méthode utilisée seule ne soit pas particulièrement avantageuse. L'utilisation conjointe de cette méthode et de l'adaptation du critère de convergence présentée en section 6.5, en revanche, s'est montrée particulièrement efficace.

6.6.2.2 Influence du critère de convergence adaptatif

Le second paramètre dont nous avons étudié l'importance et recherché des valeurs optimales a été le critère de négligeabilité C_N. Comme mentionné dans la sous-section 6.5.2, cette valeur doit être choisie de façon à éviter deux effets opposés : une valeur très basse

n'aurait qu'une incidence très réduite sur le comportement du solveur, jusqu'à revenir à un DPCG classique en faisant tendre cette valeur vers 0, et une valeur trop grande détériorerait le comportement du solveur sur le niveau fin, qui redeviendrait un PCG classique à partir d'un certain seuil de valeurs de C_N.

Ces effets extrémaux sont illustrés par les figures 6.27 et 6.28, qui montrent le comportement du solveur sur les niveaux fin et grossier pour différentes valeurs de C_N, sur un maillage du brûleur PRECCINSTA comptant 1,7 million d'éléments. Pour des raisons de commodité de lecture, les valeurs qui ont été conservées sur les deux figures ne sont pas les mêmes ; un nombre réduit de courbes pertinentes a été conservé à chaque fois. Par exemple, la valeur $C_N = 0,02$ donne une courbe d'itérations sur le maillage grossier très similaire à celle obtenue avec $C_N = 0,01$, tandis que le nombre d'itérations sur le maillage fin diffère grandement entre ces deux valeurs. A l'inverse, le fait d'activer l'adaptation du critère de convergence avec $C_N = 0,0001$ ne modifie qu'infimement les nombres d'itérations sur le maillage fin, tandis que le nombre d'itérations sur le maillage grossier chute visiblement.

Ces deux figures n'ont pas prétention à l'exhaustivité, mais elles illustrent une tendance générale suivie par les nombres d'itérations sur chaque maillage lorsque la valeur de C_N change : le nombre moyen d'itérations sur le maillage grossier réduit à mesure que C_N augmente, mais le choix d'une valeur au-delà d'un seuil situé aux alentours de 0,01 fait augmenter le nombre d'itérations sur le maillage fin de plus de 10 %, pourcentage à partir duquel nous considérons que cette augmentation ne pouvait plus être liée uniquement au hasard de l'approximation numérique.

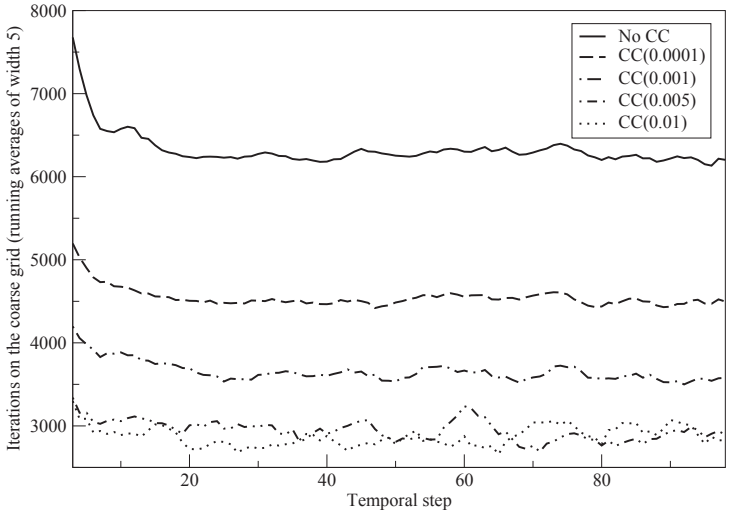

FIGURE 6.27 – *Nombre total d'itérations du solveur sur le maillage grossier à chaque pas de temps, pour différentes valeurs de C_N, pour une simulation sur 4 coeurs de calcul sur un maillage à 1,7 millions d'éléments du brûleur* **Preccinsta.**

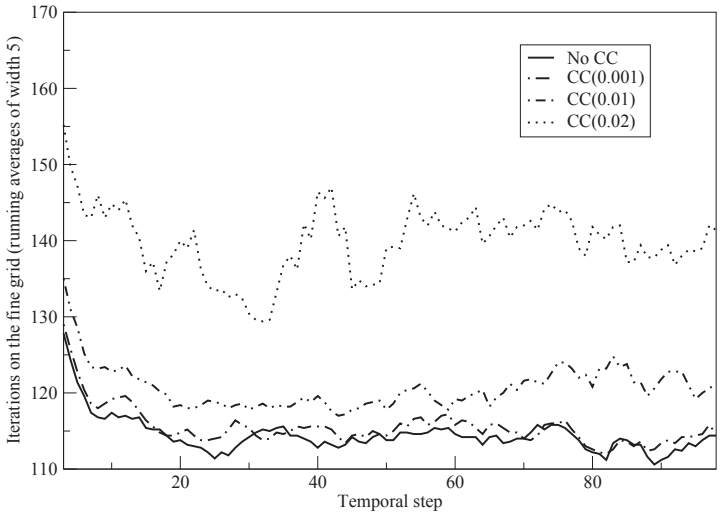

FIGURE 6.28 – *Nombre d'itérations du solveur sur le maillage fin à chaque pas de temps, pour différentes valeurs de C_N, pour une simulation sur 4 coeurs de calcul sur un maillage à 1,7 millions d'éléments du brûleur* **Preccinsta**.

Le comportement du solveur lorsque la valeur de C_N est modifiée est qualitativement le même pour n'importe quelle configuration et n'importe quelle taille de maillage, mais la valeur-seuil de C_N, définie en fonction de l'augmentation du nombre d'itérations sur le maillage fin comme décrit ci-dessus, ainsi que les gains en itérations sur le maillage grossier correspondants peuvent changer d'un cas à un autre. C'est pourquoi une série de tests a été effectuée sur les mêmes cas-tests que dans la sous-section précédente, pour des valeurs de C_N allant de 0,0001 à 0,05. Le seul cas dans lequel le nombre d'itérations sur le maillage fin reste bas même pour les plus hautes valeurs de C_N est celui du **Cylindre_3D** ; pour les valeurs testées, le facteur de réduction sur le nombre d'itérations va de 1,29 pour $C_N = 0,0001$ à 2,43 pour $C_N = 0,02$. Pour chaque autre cas-test étudié, la table 6.3 donne la plus grande valeur de C_N, parmi celles testées, pour laquelle le nombre d'itérations sur le maillage fin est augmenté de moins de 10 % par rapport au DPCG utilisé précédemment, ainsi que le facteur de réduction du nombre d'itérations sur le maillage grossier qui a été observé avec cette valeur de C_N.

Ces résultats nous donnent une idée plus pécise des valeurs de C_N à utiliser, qui semblent être les valeurs entre $0,001$ et $0,01$. Comme la valeur $0,005$ nous a semblé être un excellent candidat lors de toutes les simulations effectuées, nous l'avons conservée comme paramètre par défaut du solveur RA-DEF2 mis en oeuvre dans YALES2.

6.6.2.3 Solveur RA-DEF2

Nous nous attendions, depuis le début de ces travaux, à ce que l'adaptation du critère de convergence et le calcul de solutions initiales fonctionnent particulièrement bien lors

Cas-test	2D_Poisson		Preccinsta	
Eléments ($\times 10^6$)	3,23	12,9	1,7	110
Coeurs	4	8	4	512
Seuil de C_N	0,001	0,001	0,01	0,005
Facteur de réduction	3,76	4,27	2,2	1,95

TABLE 6.3 – *Valeur-seuil de C_N et facteur de réduction du nombre d'itérations sur le maillage grossier, pour les cas-tests* **Poisson_2D** *et* **Preccinsta** *avec différentes tailles de maillages.*

de leur usage conjoint dans le même solveur, comme expliqué en sous-section 6.3.5. Pour confirmer cette intuition, nous avons testé sur l'ensemble des cas décrits ci-avant chaque possibilité pour le solveur, avec ou sans chacune des deux méthodes développées lors de cette thèse.

Les figures 6.29 et 6.30 montrent des exemples des comportements du solveur sur le maillage grossier selon les valeurs de C_N et n_{ig}. Ces exemples concernent tous deux le cas-test **Pale_TC**, et nous avons choisi de ne montrer que trois courbes sur chaque figure de façon à ce que la lecture de ces figures soit simplifiée ; précisons néanmoins que, d'un point de vue qualitatif, le comportement du solveur lorsque les valeurs de ces paramètres sont modifiées est le même pour chaque cas-test et chaque taille de maillage.

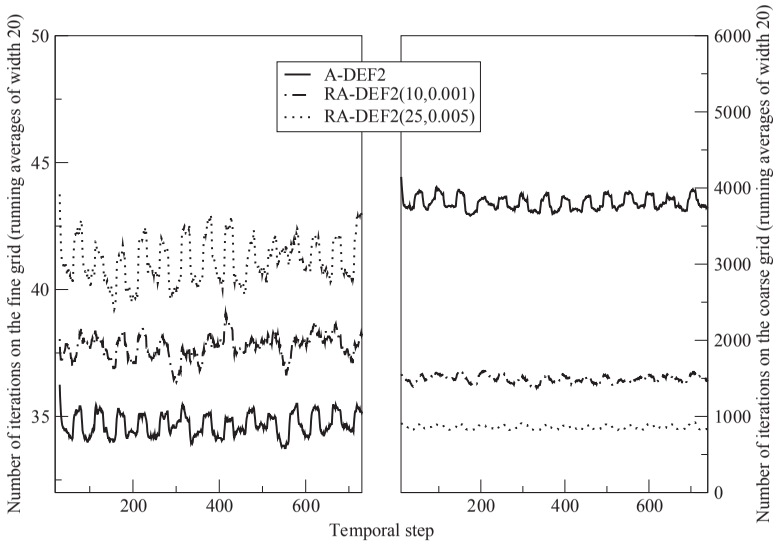

FIGURE 6.29 – *Nombres d'itérations sur le maillage grossier à chaque pas temporel, pour différents couples de paramètres, pour le cas-test* **Pale_TC**, *sur un maillage de 35 millions d'éléments sur 512 processeurs.*

Deux effets positifs sont induits par la combinaison du calcul de solutions initiales et de l'adaptation du critère de convergence sur le maillage grossier. D'une part, l'application

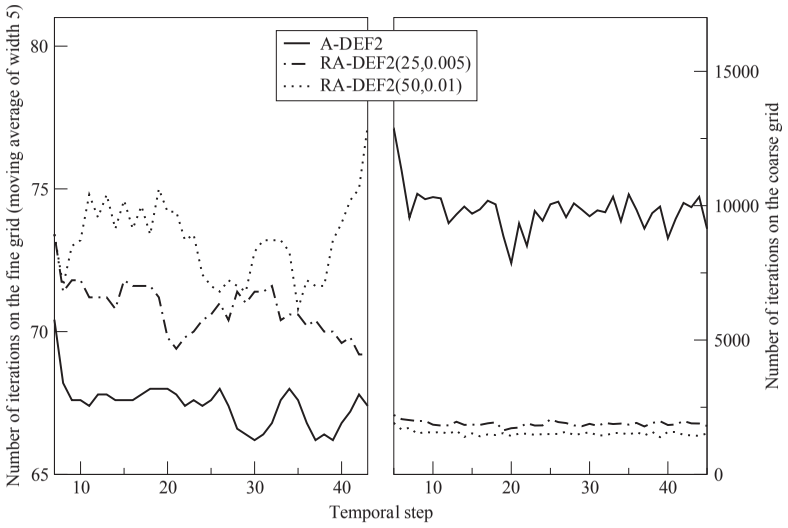

FIGURE 6.30 – *Nombres d'itérations sur le maillage grossier à chaque pas temporel, pour différents couples de paramètres, pour le cas-test* **Pale_TC**, *sur un maillage de 2,2 milliards d'éléments sur 16384 processeurs.*

simultanée de ces deux techniques rend le nombre d'itérations sur le maillage grossier plus petit encore que l'utilisation de l'adaptation du critère de convergence seule. D'autre part, nous avons pu observer que le calcul de solutions initiales permet de stabiliser le solveur lors de l'utilisation de valeurs élevées du paramètre de négligeabilité C_N. Cet effet est illustré par la figure 6.31, qui donne les nombres d'itérations sur un maillage de 1,75 million d'éléments du cas-test **Preccinsta** ainsi que sur le maillage grossier associé, pour différentes valeurs de n_{ig} et une valeur de C_N constante et élevée, en l'occurrence $C_N = 0,02$. La constante de négligeabilité élevée augmente le nombre d'itérations sur le maillage fin, mais le calcul de solutions initiales sur le maillage grossier semble compenser en partie cette augmentation. Cette réduction est sensiblement la même pour toute valeur de n_{ig} supérieure ou égale à 5, bien que la figure ne représente que l'impact des valeurs $n_{ig} = 5$ et $n_{ig} = 50$ pour la praticité de lecture. Mentionnons que cet effet est clairement visible sur l'intégralité des cas-tests présentés dans cette thèse, ce qui exclut la possibilité d'un effet "accidentel".

Le nombre d'itérations sur le maillage grossier diminue au fur et à mesure que la valeur de C_N augmente, mais il a été constaté dans chaque cas-test que la valeur $C_N = 0,005$ est proche d'un seuil. Lorsque la valeur de C_N est choisie au-dessus de cette valeur-seuil, l'imprécision des solutions calculées sur le maillage grossier provoque une augmentation considérable du nombre d'itérations sur le maillage fin, qui peut à son tour augmenter le temps de calcul nécessaire. Le choix de la valeur $C_N = 0,005$, en revanche, permet dans presque tous les cas de conserver un nombre d'itérations sur le maillage fin très convenable, c'est-à-dire très proche du nombre d'itérations du solveur A-DEF2 pour le même calcul.

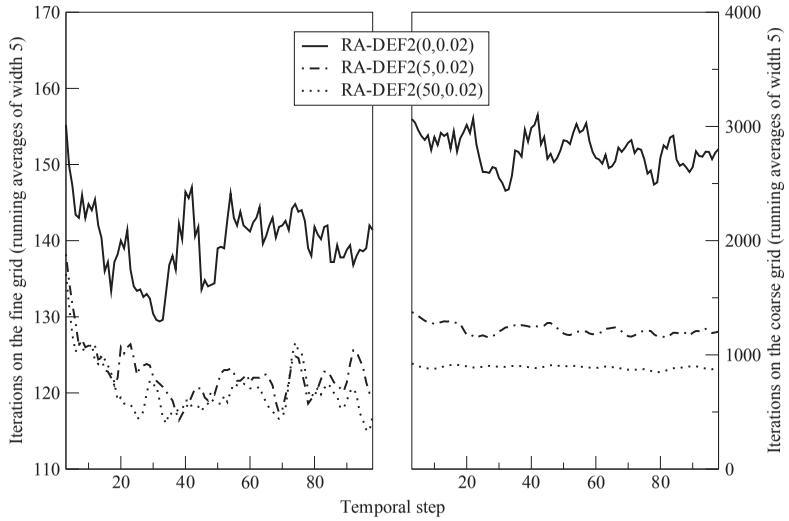

FIGURE 6.31 – *Nombres d'itérations sur les maillages fin et grossier pour le solveur RA-DEF2, avec* $C_N = 0.02$ *et différentes valeurs de* n_{ig}*, pour le cas-test* **Preccinsta** *sur un maillage de 1,75 millions d'élements sur 4 processeurs.*

Parmi les cas-tests étudiés dans cette thèse, la seule exception est le cas de l'**Injecteur Triple Disque**, pour lequel même de petites valeurs de C_N peuvent modifier notablement le comportement du solveur sur le maillage fin ; cette observation peut être expliquée par le fait que le problème linéaire à résoudre pour cette simulation est particulièrement raide, avec des coefficients de la matrice du système distants de plusieurs ordres de grandeur. Le nombre total d'itérations sur le maillage grossier est malgré tout divisé par un facteur important, comme montré par la figure 6.32.

Dans un certain nombre de cas-tests parmi ceux que nous avons lancés sur moins de 1000 processeurs et avec la valeur $C_N = 0,005$, le recyclage de plus de 20 ou 25 solutions sur le maillage fin peut représenter un gaspillage de ressources, dans la mesure où le faible gain de précision supplémentaire ne permet pas de réduire le nombre d'itérations. C'est pourquoi le couple de valeurs $(n_{ig} = 25; C_N = 0,005)$ a été choisi comme couple de valeurs par défaut du solveur RA-DEF2 ; pour des raisons pratiques, RA-DEF2(25 ;0,005) est renommé RA-DEF2(d) dans ce qui suit, ainsi que dans la figure 6.32.

La table 6.4 fournit une comparaison détaillée des nombres moyens d'itérations sur le maillage grossier par pas de temps pour les solveurs A-DEF2 et RA-DEF2(d), pour tous les cas-tests étudiés ici à l'exception de **Poisson_2D**. Ces résultats montrent clairement que la méthode que nous avons développée est encore plus rentable pour les simulations les plus complexes, avec un facteur de réduction sur le nombre d'itérations sur le maillage grossier allant de 2,62 pour le cas-test **Cylindre_3D** à un frappant 12,5 pour le cas-test de l'**Injecteur Triple Disque**.

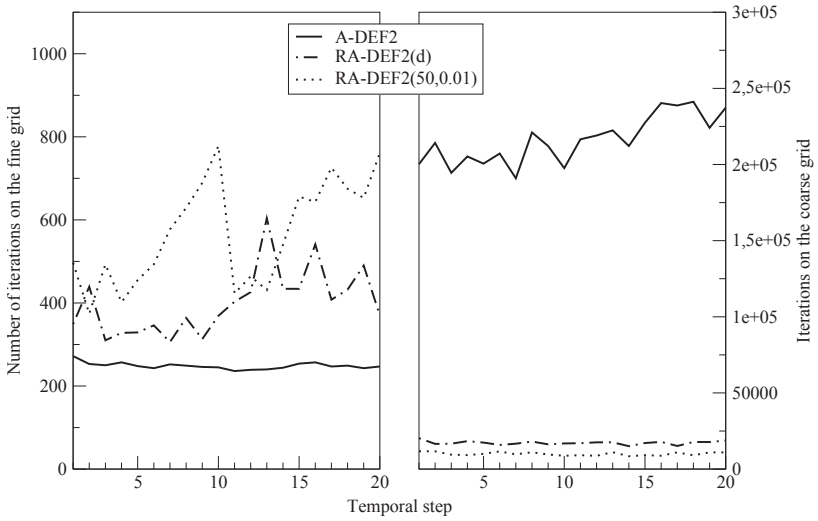

FIGURE 6.32 – *Nombres d'itérations sur les maillages fin et grossier à chaque pas de temps, pour les solveurs A-DEF2, RA-DEF2(d) et RA-DEF2(50;0,01), pour le cas-test* **ITD** *sur un maillage de 1,6 milliards d'éléments et 16,384 processeurs.*

Cas-test	Cylindre_3D	Preccinsta	Pale_TC		ITD
Eléments ($\times 10^6$)	0,49	110	34,5	2200	1600
Processeurs	4	512	512	16384	16384
A-DEF2	2468	13203	3813	10364	216153
RA-DEF2(d)	941	3044	859	1948	17227
Facteur de réduction	**2,62**	**4,34**	**4,44**	**5,32**	**12,5**

Cas-test	Pale_TC_Curie		
Eléments ($\times 10^6$)	34,5	276	2200
Processeurs	128	1024	8192
A-DEF2	4002	6233	9593
RA-DEF2(d)	1059	1670	2293
Facteur de réduction	**3,78**	**3,73**	**4,18**

TABLE 6.4 – *Nombre moyen d'itérations sur le maillage grossier par pas de temps pour les solveurs A-DEF2 et RA-DEF2(d), suivi par le facteur de réduction correspondant, pour tous les cas-tests sauf* **Poisson_2D**.

6.6.3 Impact sur les temps de calcul et de communication

Le temps passé sur le niveau grossier est principalement constitué de temps de communications interprocesseurs, eux-mêmes constitués en partie de temps de latence incompressibles en raison du très faible volume de données échangé. Par conséquent, la réduction du nombre d'itérations sur le maillage grossier, clairement montré en table 6.4, ne peut

qu'avoir un impact important à la fois sur le temps de calcul global et la proportion de communications sur ce temps de calcul. Ces deux informations nous sont données par le solveur YALES2 à chaque pas de temps. Au lieu des temps de calcul bruts, nous nous intéressons aux efficacités réduites, présentées en sous-section 4.1.4. Pour nos cas-tests massivement parallèles, c'est-à-dire les simulations effectuées sur 512 processeurs ou plus, les comparaisons entre les efficacités réduites des solveurs A-DEF2 et RA-DEF2(d) sont données par la table 6.5.

Cas-test	Preccinsta	Pale_TC		ITD
Eléments ($\times 10^6$)	110	34,5	2200	1600
Processeurs	512	512	16384	16384
A-DEF2	646,7	776,6	862,5	9023
RA-DEF2(d)	527,7	685,2	660	2533
Gain	**18,4 %**	**11,8 %**	**23,5 %**	**71,9 %**

Cas-test	Pale_TC_Curie		
Eléments ($\times 10^6$)	34,5	276	2200
Processeurs	128	1024	8192
A-DEF2	77,2	96,21	243,7
RA-DEF2(d)	70,4	76,51	128
Gain	**8,81 %**	**20,5 %**	**47,5 %**

TABLE 6.5 – *Efficacités réduites moyennes par pas de temps pour les solveurs A-DEF2 et RA-DEF2(d), en microsecondes-processeurs par volume de contrôle et par itération, pour les cas-tests massivement parallèles.*

Bien évidemment, la réduction des temps de calcul n'est pas directement proportionnelle à la réduction du nombre d'itérations. Néanmoins, les résultats obtenus montrent clairement l'efficacité de la nouvelle méthode RA-DEF2, particulièrement lors de l'application aux simulations massivement parallèles pour des géométries complexes. Une fois de plus, les résultats sur le cas-test complexe **ITD** sont particulièrement marquants, puisque le temps de calcul global est pratiquement divisé par quatre. Ce résultat répond à celui concernant la réduction du nombre d'itérations sur le maillage grossier, puisque la résolution de l'équation de Poisson pour la pression par le solveur A-DEF2 représente 58,9 % du temps de calcul global pour cette simulation.

La figure 6.33 représente les efficacités réduites des vingt premiers pas de temps du cas-test **ITD**, pour les trois mêmes solveurs que dans la figure 6.32, de façon à relier la réduction du nombre d'itérations sur le maillage grossier entre les solveurs A-DEF2 et RA-DEF2 avec la réduction correspondante des temps de calcul. L'augmentation non négligeable du nombre d'itérations sur le maillage fin pour les simulations les plus complexes aurait pu être une raison de nuancer l'efficacité du solveur RA-DEF2, mais il apparait clairement que la très importante réduction du nombre d'itérations sur le maillage grossier compense largement cet effet négatif. La qualité des simulations obtenues, dictée par le

critère de convergence imposé sur le maillage fin, n'est bien évidemment pas impactée par le comportement du solveur, du moment qu'il finit par converger ; le solveur RA-DEF2 remplit donc largement ses objectifs, en atteignant bien plus rapidement que le solveur A-DEF2 des résultats de la même qualité.

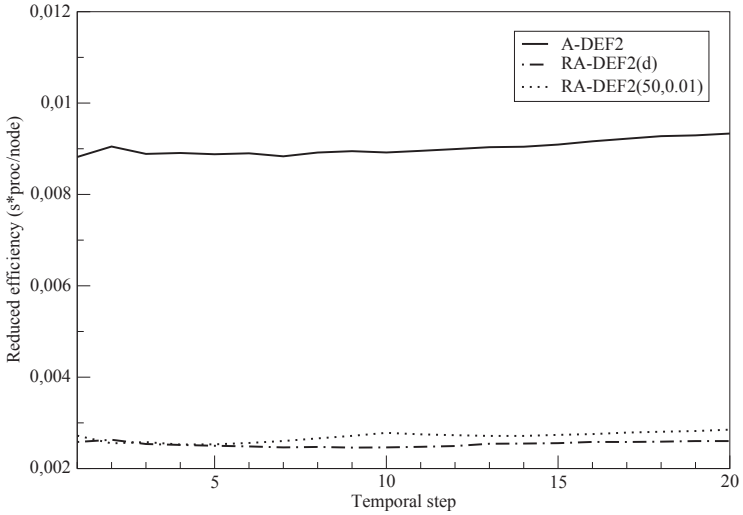

FIGURE 6.33 – *Efficacités réduites à chaque pas de temps des solveurs A-DEF2, RA-DEF2(d) et RA-DEF2(50 ;0,01), pour le cas-test* **ITD** *sur un maillage de 1,6 milliards d'éléments sur 16384 processeurs.*

La réduction du temps de calcul est fortement liée à la réduction des communications, puisqu'elle est due à la suppression d'itérations sur le maillage grossier qui sont constituées en grande partie de temps de communications incompressibles. Par conséquent, la proportion des temps de communications sur les temps de calcul globaux est forcément réduite par le passage du solveur A-DEF2 au solveur RA-DEF2. La table 6.6 donne ces proportions, et les gains réalisés sur ces proportions, pour la résolution massivement parallèle de plusieurs cas-tests en utilisant les solveurs A-DEF2 et RA-DEF2(d).

6.6.4 Weak scaling du solveur RA-DEF2(d)

Comme nous l'avons vu dans la sous-section précédente, le nombre d'itérations sur le maillage grossier, et donc le nombre de communications interprocesseurs, requises pour une simulation donnée, baisse considérablement lors du passage du solveur A-DEF2 au solveur RA-DEF2(d). Pour mesurer l'impact de la réduction du nombre de communications sur la vitesse d'exécution massivement parallèles du solveur YALES2, nous avons étudié son *weak scaling* (voir sous-section 4.1.4) avec le solveur A-DEF2 et avec le solveur RA-DEF2(d) sur trois maillages du cas-test **Pale_TC**. Les simulations sur le maillage de 34,5 millions d'éléments ont été effectuées sur 256 processeurs de la machine Babel, et les

Cas-test	Preccinsta	Pale_TC		ITD
Eléments ($\times 10^6$)	110	34,5	2200	1600
Processeurs	512	512	16384	16384
A-DEF2	11,35 %	16,15 %	27,35 %	58,9 %
RA-DEF2(d)	9,015 %	9,496 %	12,71 %	36,28 %
Gain	**20,6 %**	**41,2 %**	**53,5 %**	**38,4 %**

Cas-test	Pale_TC_Curie		
Eléments ($\times 10^6$)	34,5	276	2200
Processeurs	128	1024	8192
A-DEF2	14,24 %	19,46 %	18,92 %
RA-DEF2(d)	10,96 %	13,16 %	13,83 %
Gain	**23 %**	**32,4 %**	**26,9 %**

TABLE 6.6 – *Proportions moyennes de temps de communications par rapport aux temps de calcul globaux pour les solveurs A-DEF2 et RA-DEF2(d), pour les cas-tests sur 512 processeurs et plus.*

maillages de 276 millions et 2,2 milliards d'éléments, obtenus par raffinements homogènes non-dégénératifs [63] successifs du premier maillage, ont été utilisés pour les simulations sur 2048 et 16384 processeurs de la même machine, respectivement.

Les temps de calcul de quarante itérations successives ont été moyennées, et les dix premières itérations n'ont pas été prises en compte dans le calcul, de façon à se focaliser sur les itérations qui ont lieu après que le régime de l'écoulement est devenu proche d'un régime statistiquement stationnaire. La figure 6.34 montre les *weak scalings* qui en résultent, en prenant pour chacun des deux solveurs les efficacités réduites sur 256 processeurs comme références. Il apparait que les modifications apportées au solveur A-DEF2, de façon à en faire le solveur RA-DEF2(d), ont considérablement amélioré le *weak scaling* du solveur incompressible de YALES2, puisque le speed-up entre les simulations sur 256 et 16384 processeurs passe de 75,44 % avec le solveur A-DEF2 à 90,57 % avec le solveur RA-DEF2(d).

6.6.5 Robustesse du solveur sur maillage de très grande taille

Une simulation supplémentaire a été lancée sur un maillage de 17,8 milliards de tétraèdres pour le cas-test **Pale_TC**. Pour cela ont été mis à contribution 8192 processeurs de la machine Curie, plus puissants que ceux de la machine Babel. Les statistiques concernant les temps de calcul n'ont été calculées par YALES2 que tous les dix pas de temps, mais les moyennes globales sur la simulation, qui compte 1600 pas de temps au total, peuvent être considérées comme quasiment exactes. La table 6.7 fournit les résultats de cette simulation et les compare avec ceux obtenus lors du calcul, sur les mêmes processeurs, sur un maillage de 2,2 milliards d'éléments.

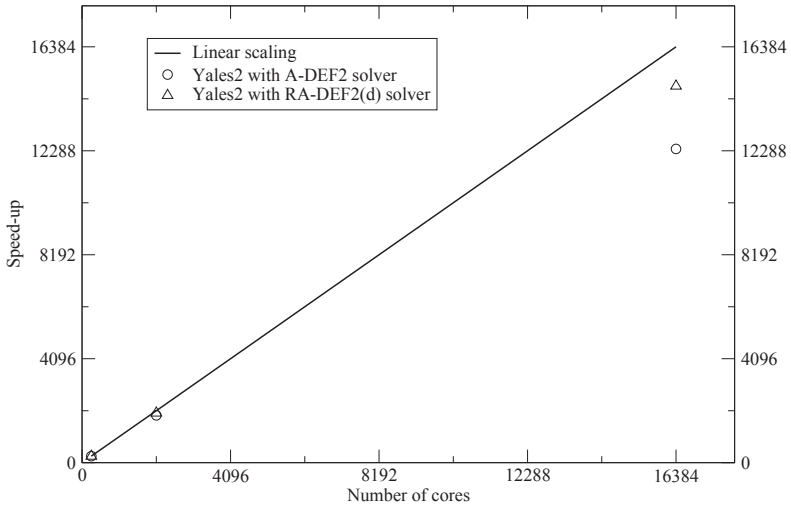

FIGURE 6.34 – *Weak scaling de YALES2 sur Babel, pour des maillages du cas-test* **Pale_TC** *atteignant 2,2 milliards d'éléments, sur un nombre de processeurs atteignant 16384.*

Cas-test	Pale_TC_Curie	
Eléments ($\times 10^9$)	2,2	17,8
Efficacité réduite moyenne	128	110,3
Proportion de communications moyenne	13,83 %	9,55 %
Itérations sur le maillage grossier	2293	4436

TABLE 6.7 – *Efficacités réduites moyennes d'un pas de temps, en microsecondes-processeurs par volume de contrôle ; proportions moyennes de temps de communications par rapport aux temps de calcul globaux ; et nombres moyens d'itérations sur le maillage grossier par pas de temps ; pour le cas-test* **Pale_TC_Curie** *sur 2,2 milliards et 17,8 milliards d'éléments, sur 8192 processeurs. La dernière ligne donne plus précisément le nombre moyen d'itérations sur le maillage grossier par pas de temps.*

L'efficacité réduite moyenne sur le cas-test à 17,8 milliards d'éléments est plus petite de quasiment 15 % que celle obtenue avec le maillage de 2,2 milliards d'éléments sur le même nombre de processeurs, à savoir 110 microsecondes-processeurs par volume de contrôle et par pas de temps au lieu de 128. Il s'agit très probablement d'une conséquence directe de la réduction des communications, due à la plus grande quantité de données sur chaque processeur. La proportion de communications moyenne passe d'ailleurs de 13,8 % à 9,55 % lorsque le maillage est raffiné de façon homogène, passant de 2,2 à 17,8 milliards d'éléments.

Ces résultats sont donc moins significatifs qu'une étude de *scaling* à proprement parler, mais ils indiquent que le code YALES2, muni du solveur linéaire RA-DEF2(d), est capable

de mener à bien des simulations massivement parallèles sur des maillages de plusieurs dizaines de milliards d'éléments, le tout avec des résultats très satisfaisants en termes de temps de calcul.

6.7 Application à la déflation à trois niveaux

Une fois effectuée la mise en oeuvre de l'adaptation du critère de convergence sur les trois niveaux de maillage de l'algorithme 3LDPCG, et du recyclage des solutions précédentes sur les niveaux intermédiaire et grossier, l'efficacité de la nouvelle méthode ainsi créée, nommée R3LDPCG dans ce qui suit, a été testée sur plusieurs simulations. Nous présentons dans cette section les résultats en termes d'itérations, d'efficacités réduites et de proportions de communications sur les cas-tests décrits ci-après.

Précisons auparavant que, puisque le solveur RA-DEF2(d) a comme paramètres $n_{ig} = 25$ et $C_N = 0,005$, nous avons décidé de garder des paramètres similaires pour le solveur à trois niveaux R3LDPCG(d) : 25 solutions au maximum sont recyclées sur chacun des trois niveaux, et le critère de négligeabilité utilisé pour le calcul du critère de convergence est fixé à $0,005$ à la fois pour le niveau intermédiaire et pour le niveau grossier. Les autres valeurs possibles du couple $(n_{ig}; C_N)$ ne sont pas considérées dans cette section, puisqu'il est prévisible que les valeurs optimales pour la déflation à deux niveaux conviendront à la déflation à trois niveaux : nous sommes déjà assurés qu'elles sont quasi-optimales pour le solveur implanté sur le niveau intermédiaire, puisque ce solveur est un A-DEF2 classique.

6.7.1 Cas-tests étudiés

Les cas-tests étudiés dans cette section sont décrits plus en détail dans la sous-section 6.6.1. Nous ne nous intéressons ici qu'aux cas suivants :

Cylindre_3D sur un maillage de 3,9 millions d'éléments tétraédriques. Ces simulations sont effectuées sur 4 processeurs d'une station locale ; par conséquent, seuls les nombres d'itérations sont significatifs pour ce cas-test. Pour les solveurs A-DEF2 et RA-DEF2(d), les groupes sont formés de 200 cellules ; pour les solveurs 3LDPCG et R3LDPCG(d), les groupes sont formés de 4000 cellules regroupées en 20 sous-groupes de 200 cellules chacun. Le critère de convergence sur le maillage fin est fixé à 10^{-8}.

Preccinsta sur des maillages de 110 et 877 millions d'éléments tétraédriques respectivement, le second ayant été obtenu par raffinement homogène du premier. Toutes les simulations sur le premier maillage ont été effectuées sur 512 processeurs de la machine Babel (voir sous-section 6.6.1 pour la description de cette machine), et toutes les simulations sur le second maillage sur 4096 processeurs de cette même machine, de façon à ce que le nombre d'éléments par processeur soit le même pour toutes les

simulations. Les tailles de groupes et de sous-groupes sont les mêmes que pour le cas-test **Cylindre_3D** : 200 cellules par groupe pour les solveurs A-DEF2 et RA-DEF2(d), 4000 cellules et 20 sous-groupes par groupe pour les solveurs 3LDPCG et R3LDPCG(d). Les critères de convergence 10^{-8} et 10^{-10} ont été utilisés sur le maillage fin.

Notons que pour le choix des nombres de cellules par groupe et par sous-groupe, nous avons veillé à ce que le nombre d'éléments sur le maillage grossier de la déflation classique soit le même que le nombre d'éléments sur le maillage intermédiaire de la déflation à trois niveaux, de façon à obtenir des comparaisons fiables.

6.7.2 Réduction du nombre d'itérations

Le solveur 3LDPCG ne s'est pas révélé indéfectible, puisque lors des tests effectués sur la géométrie complexe du brûleur PRECCINSTA, aucune simulation effectuée avec ce solveur n'est parvenue à dépasser le troisième pas de temps. Le premier avantage du solveur R3LDPCG(d) a donc été immédiatement visible, puisqu'il s'est montré robuste sur tous ces cas-tests.

Nous ne disposons de comparaisons directes entre les solveurs 3LDPCG et R3LDPCG(d) que pour le maillage à 3,9 millions d'éléments du cas-test **Cylindre_3D**. Les résultats obtenus pour ce cas-test sont illustrés par la figure 6.35, qui montre clairement le gain obtenu grâce à l'ajout d'un troisième niveau de déflation et aux méthodes de réduction du nombre d'itérations.

Plus spécifiquement, une comparaison entre les nombres d'itérations sur les niveaux intermédiaire et grossier pour les solveurs 3LDPCG et R3LDPCG(d) est présentée en figure 6.36. Le gain obtenu sur ces deux niveaux par la mise en oeuvre de méthodes de réduction du nombre d'itérations y est mis en évidence.

La table 6.8 résume ces résultats sous forme numérique, en donnant les nombres moyens d'itérations à chaque niveau et pour chaque solveur. Le très faible nombre d'itérations sur le second niveau de maillage pour le solveur R3LDPCG(d) est imputable à la fois à l'ajout d'un troisième niveau de maillage et à l'utilisation de méthodes de réduction du nombre d'itérations :

- L'ajout d'un troisième niveau de maillage, avec ou sans mise en oeuvre des méthodes de réduction présentées ci-avant, divise le nombre d'itérations sur le deuxième niveau de maillage par un facteur proche de 6 : 6,65 entre A-DEF2 et 3LDPCG ; 5,24 entre RA-DEF2(d) et R3LDPCG(d).

- La mise en oeuvre des méthodes de réduction du nombre d'itérations, dans le solveur A-DEF2 comme dans le solveur 3LDPCG, divise le nombre d'itérations sur le deuxième niveau de maillage par un facteur proche de 3 : 3,43 entre A-DEF2 et RA-DEF2(d) ; 2,7 entre 3LDPCG et R3LDPCG(d).

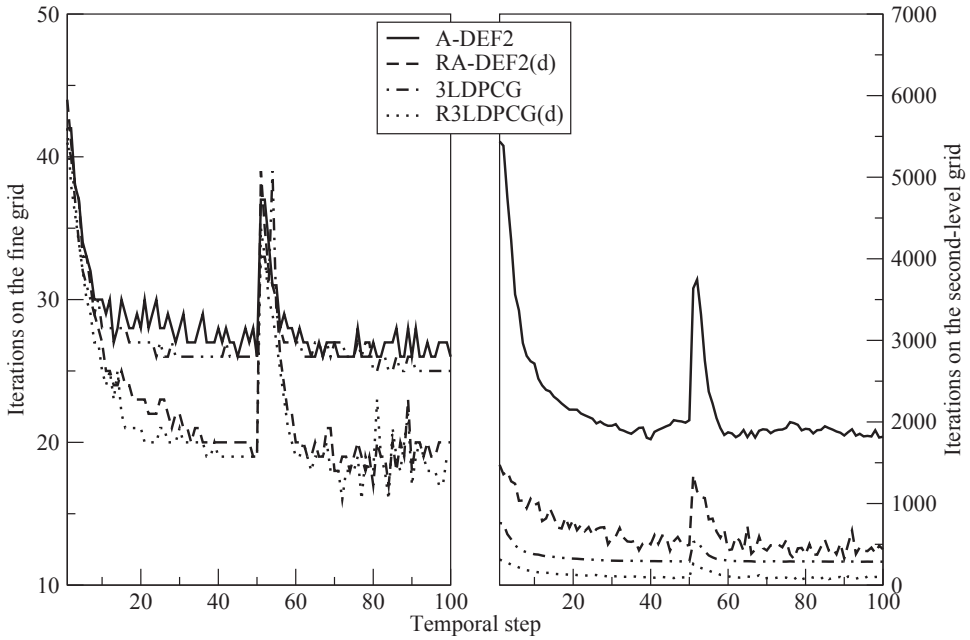

FIGURE 6.35 – *Nombres d'itérations sur les deux premiers niveaux de maillage des solveurs A-DEF2, RA-DEF2(d), 3LDPCG et R3LDPCG(d), pour la simulation du cas-test* **Cylindre_3D** *sur un maillage de 3,9 millions d'éléments sur quatre processeurs.*

Le nombre d'itérations sur le niveau le plus grossier est quant à lui divisé par 6,95 lors du passage du solveur 3LDPCG au solveur R3LDPCG(d), ce qui laisse espérer une réduction importante du temps de simulation global.

Solveur	A-DEF2	RA-DEF2(d)	3LDPCG	R3LDPCG(d)
Premier niveau de maillage	28,4	22,6	27,3	21,4
Second niveau de maillage	2209	644	332	123
Troisième niveau de maillage	-	-	8958	1289

TABLE 6.8 – *Nombres moyens d'itérations par pas de temps sur tous les niveaux de maillage pour les solveurs A-DEF2, RA-DEF2(d), 3LDPCG et R3LDPCG(d), pour le cas-test* **Cylindre_3D** *sur 3,9 millions d'éléments et 4 processeurs.*

Concernant le cas-test **Preccinsta**, la figure 6.37 donne le nombre d'itérations sur les deux premiers niveaux de maillage (fin et grossier pour la déflation standard, fin et intermédiaire pour la déflation à trois niveaux) lors de la simulation sur un maillage de 110 millions d'éléments sur 512 processeurs et avec un critère de convergence de 10^{-10}, pour les solveurs A-DEF2, RA-DEF2(d) et R3LDPCG(d). Une moyenne flottante de largeur 5 de ces valeurs est donnée au lieu des valeurs ponctuelles, de façon à lisser les irrégularités obtenues sur le maillage fin avec le solveur A-DEF2 pour des raisons de lisibilité des

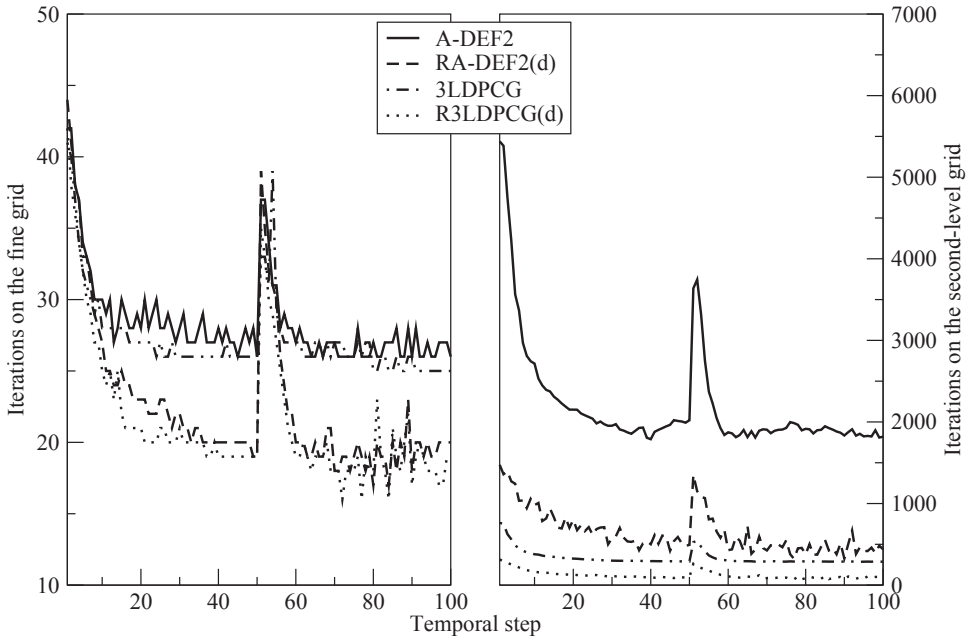

FIGURE 6.36 – *Nombres d'itérations sur les maillages intermédiaire et grossier des solveurs 3LDPCG et R3LDPCG(d), pour la simulation du cas-test* **Cylindre_3D** *sur un maillage de 3,9 millions d'éléments sur quatre processeurs.*

courbes.

La comparaison simple des nombres d'itérations des solveurs RA-DEF2(d) et R3LDPCG(d), pour les 30 premiers pas de temps, sur le maillage de 877 millions d'éléments sur 4096 processeurs, est donnée par les figures 6.38 et 6.39. Les quelques sautes de valeur qui apparaissent dans la première figure pour le solveur R3LDPCG(d) sont encore un sujet ouvert, qui semble demander des efforts de stabilisation de l'algorithme ; cependant, ces effets ne semblent pas affecter la robustesse du solveur.

Comme prévu, le nombre d'itérations sur le deuxième niveau de maillage est considérablement réduit par l'ajout d'un troisième niveau. Nous avons comparé les nombres moyens d'itérations pour les quatre cas-tests étudiés sur le brûleur PRECCINSTA, sur les 100 premiers pas de temps pour les simulations sur le maillage de 110 millions d'éléments, sur les 30 premiers pour les simulations sur le maillage de 877 millions d'éléments ; les résultats sont exposés en table 6.9.

Notons que les résultats du solveur R3LDPCG(d) sur le maillage de 877 millions d'éléments sont légèrement faussés. En effet, lors de la simulation avec le critère de convergence 10^{-8}, de légers problèmes de convergence sur le maillage intermédiaire ont artificiellement gonflé le nombre d'itérations à ce niveau lors de certaines itérations sur le maillage fin ; il en résulte un nombre d'itérations sur le maillage intermédiaire plus élevé pour ce critère de convergence que pour le critère de convergence 10^{-10}. Cette anomalie, qui ne semble pas

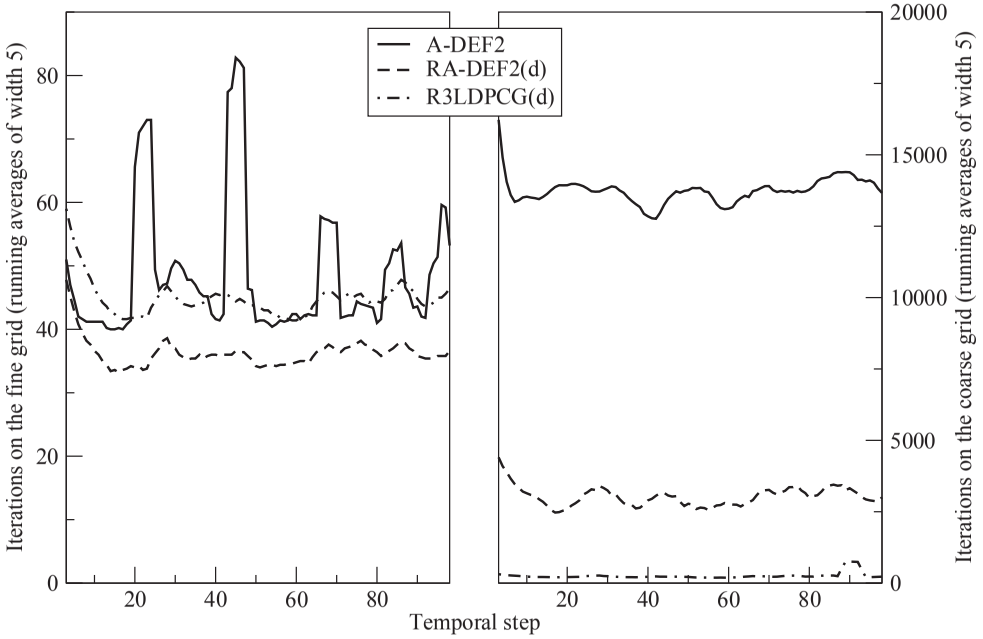

FIGURE 6.37 – *Moyennes flottantes de largeur 5 des nombres d'itérations sur les deux premiers niveaux de maillages des méthodes A-DEF2, RA-DEF2(d) et R3LDPCG(d), pour la simulation de 100 pas de temps d'un écoulement à température constante dans le brûleur PRECCINSTA, sur un maillage de 110 millions d'éléments sur 512 processeurs, avec un critère de convergence de* 10^{-10}.

Eléments ($\times 10^6$)	110		877	
Processeurs	512		4096	
Critère de convergence	10^{-8}	10^{-10}	10^{-8}	10^{-10}
RA-DEF2(d)	1954	3071	4101	5261
R3LDPCG(d)	158	257	420	338
Gain	**91,9 %**	**91,6 %**	**89,7 %**	**93,6 %**

TABLE 6.9 – *Nombres moyens d'itérations sur le deuxième niveau de maillage pour les solveurs RA-DEF2(d) et R3LDPCG(d), sur les 100 premiers pas de temps de la simulation sur le maillage à 110 millions d'éléments du cas-test* **Preccinsta** *et sur les 30 premiers pas de temps de la simulation sur le maillage à 877 millions d'éléments du même cas-test, avec des critères de convergence de* 10^{-8} *et* 10^{-10} *sur le maillage fin.*

empêcher la convergence ni impacter fortement les temps de simulation, reste néanmoins à étudier.

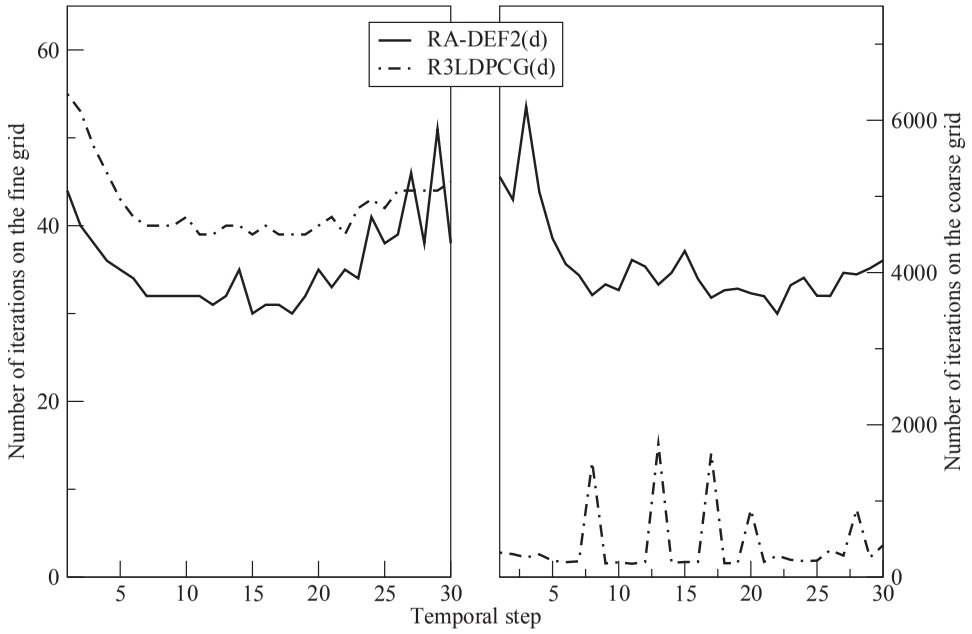

FIGURE 6.38 – *Nombres d'itérations sur les deux premiers niveaux de maillages des méthodes RA-DEF2(d) et R3LDPCG(d), pour la simulation de 30 pas de temps d'un écoulement à température constante dans le brûleur PRECCINSTA, sur un maillage de 877 millions d'éléments sur 4096 processeurs, avec un critère de convergence de* 10^{-8}.

6.7.3 Impact sur les temps de calcul et de communication

Le revers de la médaille, concernant l'ajout d'un troisième niveau de maillage, est bien évidemment un surcoût qui s'exprime particulièrement au niveau des temps de communications. La table 6.10 révèle cet inconvénient en mettant en parallèle les proportions moyennes de temps de communication obtenues pour les simulations sur le cas-test **Preccinsta** avec les solveurs RA-DEF2(d) et R3LDPCG(d).

Cependant, cet effet négatif ne contrebalance pas l'impact de la réduction très importante du nombre d'itérations sur le second niveau de maillage, ce qui induit des réductions de temps de simulation effectifs de 16,6 % à 20,3 %, comme le montre la table 6.11. A en croire ces premiers tests, la méthode de déflation à trois niveaux améliorée R3LDPCG(d) développée dans le contexte de cette thèse est donc plus efficace encore que la méthode RA-DEF2(d), dont les performances sont excellentes comparées à celles de la méthode de déflation A-DEF2 standard.

En outre, les scalabilités faibles des deux méthodes semblent comparables au vu de ces premiers résultats : les efficacités réduites sur le maillage de 110 millions d'éléments sur 512 processeurs dans ce cas valent 82,22 % et 80,34 % de celles sur le maillage de 877 millions d'éléments sur 4096 processeurs pour la simulation avec le solveur RA-DEF2(d), contre 78,52 % et 79,31 % pour le solveur R3LDPCG(d).

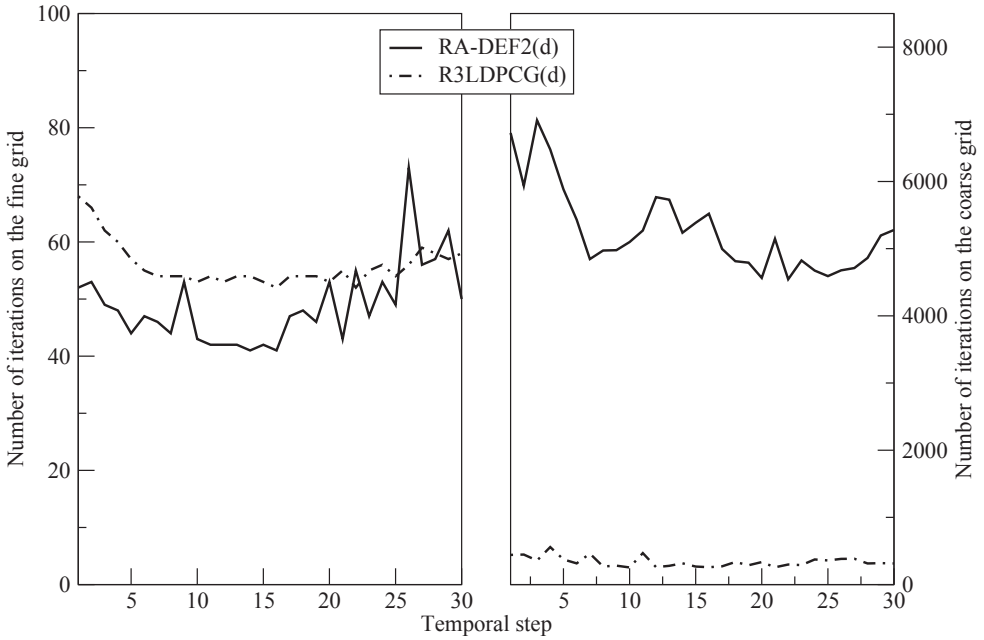

FIGURE 6.39 – *Nombres d'itérations sur les deux premiers niveaux de maillages des méthodes RA-DEF2(d) et R3LDPCG(d), pour la simulation de 30 pas de temps d'un écoulement à température constante dans le brûleur PRECCINSTA, sur un maillage de 877 millions d'éléments sur 4096 processeurs, avec un critère de convergence de* 10^{-10}.

Cas-test	**Preccinsta**			
Eléments ($\times 10^6$)	110		877	
Processeurs	512		4096	
Critère de convergence	10^{-8}	10^{-10}	10^{-8}	10^{-10}
RA-DEF2(d)	9,16 %	10,03 %	12,53 %	14,13 %
R3LDPCG(d)	10,86 %	12,72 %	17,29 %	19,96 %

TABLE 6.10 – *Proportions moyennes des temps de communication sur les temps de simulations globaux, pour les 100 premiers pas de temps de la simulation sur le maillage à 110 millions d'éléments du cas-test* **Preccinsta** *et les 30 premiers pas de temps de la simulation sur le maillage à 877 millions d'éléments du même cas-test, avec des critères de convergence de* 10^{-8} *et* 10^{-10} *sur le maillage fin.*

Eléments ($\times 10^6$)	110		877	
Processeurs	512		4096	
Critère de convergence	10^{-8}	10^{-10}	10^{-8}	10^{-10}
RA-DEF2(d)	580,9	656,3	706,4	816,4
R3LDPCG(d)	462,8	530,1	589,5	668,4
Gain	**20,3 %**	**19,2 %**	**16,6 %**	**18,1 %**

TABLE 6.11 – *Efficacités réduites moyennes, en microsecondes-processeur par volume de contrôle et par pas de temps, pour les 100 premiers pas de temps de la simulation sur le maillage à 110 millions d'éléments du cas-test* **Preccinsta** *et les 30 premiers pas de temps de la simulation sur le maillage à 877 millions d'éléments du même cas-test, avec des critères de convergence de* 10^{-8} *et* 10^{-10} *sur le maillage fin.*

Chapitre 7

Conclusion et perspectives

Ce dernier chapitre est divisé en deux sections. Dans la première, nous présentons une conclusion générale du travail mené lors de ces trois années de thèse ; les développements réalisés sont résumés, ainsi que les résultats principaux qui en sont ressorti et les quelques points restant à améliorer ou à creuser. Dans la seconde, nous abordons des perspectives plus générales, dont celle de l'équilibrage dynamique de charges de calcul au sujet duquel des recherches préliminaires ont déjà été menées.

7.1 Conclusion

Dans cette section, nous présentons un résumé des travaux effectués et des résultats liés, puis des pistes d'approfondissement de ces recherches.

7.1.1 Travail effectué

La thèse présentée dans ce manuscrit avait pour but général d'accroitre l'efficacité du solveur non-structuré massivement parallèle YALES2. Le sujet prévisionnel comprenait trois thématiques distinctes, dont celle de l'amélioration des solveurs linéaires utilisés pour la résolution de l'équation de Poisson pour la pression dans le cadre de la simulation d'écoulements incompressibles ; différentes approches ont été explorées pour cette thématique, qui a fini par devenir de fait le fil conducteur de cette thèse.

Un solveur de Gradient Conjugué avec déflation étant déjà implanté et communément utilisé par YALES2, nous avons choisi de rester dans le cadre de cette méthode et de nous attacher à chercher comment la rendre plus robuste et plus performante. Ainsi, l'étude de nouveaux solveurs haute performance, tels que le solveur de Chebyshev préconditionné récursif introduit par Koutis *et al.* en 2010 [41], a été écartée.

La méthode DPCG déjà mise en oeuvre dans le solveur YALES2 se basait sur une déflation géométrique standard, dont la conception s'appuyait sur la double décomposition de domaine en usage dans le solveur, les groupes de cellules de la DDD permettant la

création du maillage grossier sur le domaine. Notre première approche fut donc l'ajout de vecteurs de déflation, de façon à ce que les solutions calculées sur le niveau grossier ne correspondent plus à des solutions constantes par morceaux sur le maillage fin, mais linéaires voire quadratiques. La mise en oeuvre de cette technique sur un solveur structuré en deux dimensions a donné des résultats très encourageants, qui n'ont hélas pas été répercutés sur le solveur YALES2 suite à des problèmes de stabilité que la littérature existante sur les aspects numériques des solveurs avec déflation ne nous a pas aidé à résoudre.

Notre seconde approche, basée sur l'idée des méthodes multigrilles géométriques, a été la création d'une méthode de déflation à trois niveaux de maillage, en se servant à nouveau d'une méthode avec déflation sur le deuxième niveau de maillage afin de créer un troisième niveau de maillage. En pratique, nous avons préféré créer un maillage intermédiaire par scission des groupes de cellules de la DDD, plutôt que de former le maillage grossier par conglomération des groupes de cellules, pour des raisons pratiques de mise en oeuvre dans le code.

Les premiers tests effectués sur ce nouveau solveur 3LDPCG ont mis en évidence deux besoins de développement. D'une part, la robustesse du solveur est mise à rude épreuve pour des simulations dans des géométries complexes. D'autre part, la réduction du nombre d'itérations sur le second niveau de maillage ne compense pas le nombre très élevé d'itérations sur le troisième niveau de maillage nouvellement créé.

Pour rendre le solveur plus robuste et stable, nous avons choisi d'utiliser la variante A-DEF2 de la méthode de déflation avec préconditionnement introduite par Tang *et al* [75] sur le niveau fin comme sur le niveau intermédiaire, et avons mis en oeuvre sur les trois niveaux de maillage une méthode de redémarrage du Gradient Conjugué basée sur un critère géométrique, à notre connaissance absent de la littérature.

La réduction du nombre d'itérations sur les niveaux grossiers, quant à elle, a été abordée grâce au développement de deux techniques différentes. L'une entre elles vise à fournir une solution initiale, créée par une méthode de recyclage des solutions des systèmes linéaires précédemment résolus sur le même niveau de maillage, la méthode de recyclage étant conçue pour être adaptée au fonctionnement de la méthode des Gradients Conjugués. L'autre technique consiste en l'augmentation raisonnée du critère de convergence sur les niveaux intermédiaire et grossier, de façon à respecter une condition liée à l'impact de ce critère de convergence sur les valeurs calculées sur le niveau de maillage supérieur.

Ces deux techniques ont dans un premier temps été mises en oeuvre dans un solveur A-DEF2 avec déflation géométrique standard, et testées sur une grande variété de simulations différentes, notamment dans des géométries complexes et pour des écoulements diphasiques. Des paramètres quasi-optimaux ont été déterminés pour chacune de ces techniques et le solveur RA-DEF2(d) a été créé par adoption systématique de ces valeurs. Grâce à la transformation du solveur A-DEF2 en RA-DEF2(d), les nombres d'itérations sur le maillage grossier ont été divisés par des facteurs allant jusqu'à $12,5$ sur certains de ces cas-tests, d'où une réduction des temps de calcul globaux allant de 23 % à plus de

70 % pour les simulations lancées sur 8192 et 16384 processeurs. Il a également été montré que le passage du solveur A-DEF2 au solveur RA-DEF2(d) augmente considérablement la scalabilité faible de YALES2. Ces résultats ont fait l'objet d'un article scientifique dont la première révision est actuellement en cours de revue pour publication dans le *Journal of Computational Physics*.

Ces techniques ont été implantées dans le solveur à trois niveaux, avec adaptation du critère de convergence sur les niveaux intermédiaire et grossier et calcul de solutions initiales sur les trois niveaux, en conservant les paramètres quasi-optimaux déterminés grâce aux tests sur le solveur à deux niveaux. Le solveur R3LDPCG(d) ainsi créé, particulièrement conçu pour la simulation massivement parallèle, a apporté pour les premiers cas-tests étudiés sur 512 et 4096 processeurs un gain de temps de calcul supplémentaire de l'ordre de 16 à 20 % par rapport au solveur RA-DEF2(d), grâce à une division des nombres d'itérations sur le second niveau de maillage d'un facteur 10 à 15 environ.

7.1.2 Pistes d'approfondissement

A cause, entre autres, du manque de temps en fin de thèse et de dysfonctionnements temporaires de la frontale du supercalculateur Babel, l'étude du solveur à trois niveaux R3LDPCG(d) n'a pas été aussi approfondie que nous l'aurions souhaité. Plusieurs pistes restent donc à explorer.

- D'une part, la multiplication des cas-tests permettrait de s'assurer que ce solveur est robuste, contrairement au solveur 3LDPCG qui n'a pas permis la simulation de certains écoulements dans des géométries complexes.
- En outre, bien que les paramètres par défaut des techniques de réduction du nombre d'itérations appliquées au solveur à deux niveaux conviennent très probablement au solveur à trois niveaux, comme évoqué en introduction de la section 6.7, il reste probable que ces paramètres soient ajustables de façon à obtenir des résultats encore meilleurs. Une étude complémentaire pourrait être à réaliser à ce niveau, avec la possibilité que les valeurs des paramètres puissent être calculées directement par le solveur "en boîte noire" en fonction des données du problème à résoudre, comme sa taille ou le cas physique simulé.
- Finalement, sur certains des cas-tests, certaines instabilités ponctuelles apparaissent, sous la forme de "pics" de nombres d'itérations en apparence injustifiés et imprévisibles. Bien que ces instabilités n'aient pas semblé nuire à la simulation des cas-tests étudiés, ils restent un phénomène marginal qu'il serait souhaitable d'éviter, ne serait-ce que pour le léger gain en termes de performances qui s'ensuivrait.

Parallèlement, il pourrait être bon de revenir sur la piste de la déflation dite "plane" ou "quadrique" pour chercher les raisons numériques du défaut de convergence de cette méthode dans YALES2, d'autant que les premiers résultats sur un solveur structuré en deux

dimensions ont montré des résultats plus qu'encourageants. Cependant, la convergence de la méthode est assurée en arithmétique exacte quel que soit le choix des vecteurs de déflation, et l'étude numérique de la convergence des méthodes avec déflation semble être un domaine rare, voire presque inexistant, de la littérature. Plus de pistes de recherche à ce sujet sont données en annexe de ce manuscrit.

7.2 Perspectives

Dans cette section, nous évoquons d'autres perspectives liées au besoin d'efficacité des solveurs de dynamique des fluides massivement parallèles, et particulièrement les recherches préliminaires effectuées au sujet du rééquilibrage dynamique de charges, et l'intérêt multiple de la mise en oeuvre d'une telle fonctionnalité dans un solveur de CFD tel que YALES2.

7.2.1 Equilibrage de charges

Un travail préliminaire concernant l'équilibrage dynamique des charges de calcul a été effectué lors des derniers mois de cette thèse. La mise en oeuvre d'un tel algorithme dans YALES2 permettrait deux applications cruciales : d'une part, le suivi de particules lagrangiennes dans un écoulement provoque d'importantes variations des charges de calcul associées à chaque groupe de cellules, d'où l'importance de redistribuer ces groupes sur les différents processeurs en cours de calcul afin de réduire le temps de simulation ; d'autre part, la possibilité d'un raffinement local du maillage en cours de simulation ne devient pleinement avantageuse que si les groupes concernés par le raffinement sont repartitionnés à leur tour et que l'ensemble des groupes est redistribué sur les processeurs.

Ce second point mérite de plus amples explications. L'approche envisagée dans les développements à venir du solveur YALES2 consiste à faire évoluer le code de façon à en faire un outil de simulation "intelligente", c'est-à-dire adaptative. Ainsi, le raffinement local du maillage en cours de simulation, en fonction de paramètres tels que les gradients locaux de pression ou de vitesse ou les valeurs de viscosité turbulente, permettrait l'obtention de résultats similaires à ceux obtenus par une LES très raffinée à partir de maillages relativement grossiers, et sur des maillages en fin de simulation comptant relativement peu d'éléments.

Il est évident qu'il reste de nombreuses étapes à franchir avant d'atteindre un tel objectif. La création d'une procédure de rééquilibrage dynamique des charges de calcul au sein du code YALES2 est actuellement prise en considération, en tant que première brique de ce travail. Cette sous-section a pour objectif, après un bref rappel de notions importantes de théorie des graphes, de résumer le travail préliminaire effectué sur ce sujet et les premiers résultats obtenus, puis d'ouvrir à des perspectives concernant cet aspect du développement de YALES2. Les éléments bibliographiques mentionnés dans

la présente partie n'ont pas prétention à l'exhaustivité, mais visent à proposer quelques articles importants autour des problématiques liés à l'équilibrage de charges.

7.2.1.1 Notions de théorie des graphes

Au maillage à partitionner est associé un graphe, dans lequel chaque noeud correspond à un élément ou un ensemble d'éléments, et où deux noeuds sont liés par un arc si et seulement si les deux ensembles d'éléments correspondants sont mitoyens. Par conséquent, le cadre de travail imposé par l'étude de méthodes de partitionnement demande la connaissance de quelques bases de théorie des graphes. Le lecteur intéressé pourra creuser le sujet grâce au livre "Graphes et Hypergraphes" de Berge [8].

Des poids peuvent être associés aux noeuds et aux arcs d'un graphe. Ici, on parlera de poids des arcs et de charge des noeuds pour éviter toute confusion. La charge d'une partie d'un graphe est tout simplement exprimée comme la somme des charges des noeuds contenus dans cette partie.

Un sous-graphe est un type particulier de partie d'un graphe, dans laquelle on conserve également tous les arcs reliant deux noeuds quelconques de cette partie.

Partitionner un graphe revient à créer un ensemble de sous-graphes disjoints qui contiennent ensemble tous les noeuds du graphe originel. Les arcs du graphe peuvent alors être séparés en deux sous-ensembles complémentaires, à savoir l'ensemble des arcs contenus par les sous-graphes, et l'ensemble des arcs liant des noeuds appartenant à des sous-graphes différents. La somme des poids des arcs de ce deuxième sous-ensemble est nommé *edgecut*, et correspond au poids total des arcs que l'on doit enlever pour séparer les sous-graphes les uns des autres.

La minimisation de l'*edgecut* est une des deux conditions prises en compte par la quasi-intégralité des logiciels de partitionnement. L'autre condition concerne la charge de chaque sous-graphe. On exprime généralement l'équilibre ou le déséquilibre entre les charges des sous-graphes d'une partition grâce à une quantité nommée ratio de déséquilibre, qui est le rapport entre la plus grande charge et la plus petite charge de la partition. Ce ratio tendant évidemment vers 1 pour des partitions bien équilibrées, on peut lui imposer une valeur à ne pas dépasser, ou bien fixer une borne supérieure sur une certaine combinaison linéaire de ce ratio et de l'*edgecut*.

Dans le cas d'un graphe associé à un maillage, le poids de chaque arête représente le volume de données à communiquer, ou le coût associé à cette communication, dans le cas où les deux ensembles d'éléments correspondants seraient sur deux processeurs différents. L'*edgecut* d'une partition est alors une mesure approximative du surcoût de communications induit par cette partition. Quant aux charges des noeuds, elles correspondent à des coûts de calcul associés aux sous-domaines correspondants du maillage : minimiser le ratio de déséquilibre revient donc à s'assurer que les calculs requis pour une simulation seront à peu près synchronisés sur les différents processeurs.

7.2.1.2 Création du graphe à partitionner

La première problématique à étudier a été celle de la création du graphe à partitionner à partir du maillage du domaine physique. Quelle que soit l'application faite du rééquilibrage dynamique de charges, on souhaite ne pas "oublier" le partitionnement précédent, afin de limiter autant que possible le volume de données à transférer pour le rééquilibrage des charges, ainsi que le coût de calcul lié à la reconstruction de la connectivité globale.

Il faut donc assurer que seuls des groupes situés près de frontières entre deux ou plusieurs sous-domaines affectés à des processeurs différents seront transférés d'un processeur à un autre après le repartitionnement. Dans cette optique, nous avons suivi l'approche dite de *levelsets* introduite par Walshaw et Berzins dans [83]. La méthode originale consiste, à partir du graphe associé au maillage, à définir une levelset L_1 contenant tous les noeuds du graphe liés à une arête qui traverse une frontière entre deux sous-domaines, puis une levelset L_2 contenant tous les noeuds liés à L_1 et n'appartenant pas à L_1, c'est-à-dire tous les noeuds n'appartenant pas à L_1 qui sont liés à une arête qui traverse la frontière de L_1, et ainsi de suite. Cette partition est illustrée par la figure 7.1, qui en donne un exemple pour un graphe planaire bipartitionné.

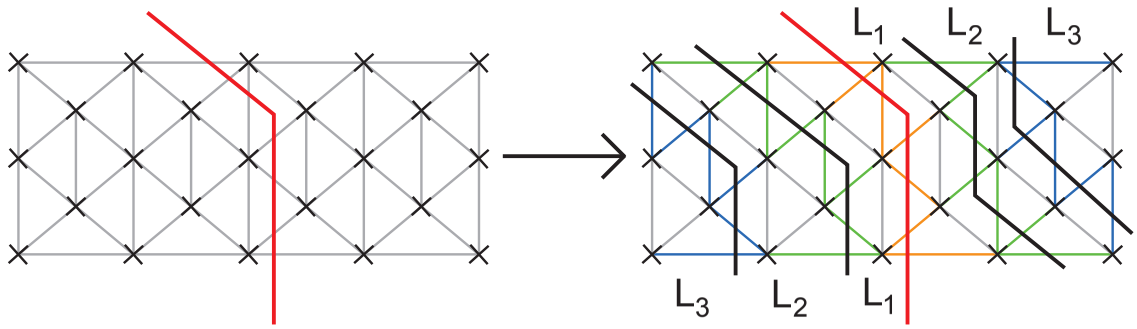

FIGURE 7.1 – *Séparation des noeuds d'un graphe planaire bipartitionné en levelsets. A gauche, le graphe original est donné, et le trait rouge épais représente la séparation entre les deux sous-graphes. Les noeuds de la première levelset L_1, dans chaque sous-graphe, sont ceux qui sont contenus par une ou plusieurs des arêtes traversant la frontière entre les deux sous-graphes, représentées en orange à droite de cette figure. D'une façon similaire, les arêtes vertes sont celles reliant un noeud de la première levelset à un noeud qui n'y appartient pas ; on détermine ainsi les noeuds de la deuxième levelset L_2 dans chaque sous-graphe. Les arêtes bleues relient les noeuds de L_2 à ceux de L_3 dans chaque sous-graphe. Dans cet exemple, les trois premières levelsets créent une partition de chaque sous-graphe.*

Lorsqu'une proportion suffisante de noeuds est contenue dans les k premières levelsets, les noeuds n'appartenant à aucune de ces levelsets sont regroupés dans des clusters. Si la partition courante du maillage est telle que chaque sous-graphe de cette partition est connexe, alors au plus un cluster sera associé à chaque sous-graphe. La valeur de k doit être choisie de façon à ce qu'exactement un cluster soit associé à chaque sous-graphe. La

figure 7.2 prolonge l'exemple de la figure 7.1 en représentant la création de ces clusters dans ce cas, avec le choix de valeur $k = 2$. Le poids de chaque cluster est la somme des poids des noeuds qu'il représente, et toutes les arêtes reliant un noeud de ce cluster à un noeud de la levelset k sont conservées ; il est alors possible que deux mêmes noeuds soient reliés dans le graphe "clusterisé" par plusieurs arêtes. En pratique, il est bien entendu possible de regrouper ces arêtes en une seule, dont le poids sera obtenu en sommant les poids des arêtes multiples, ce qui ne changera en aucun cas la solution optimale du problème de partitionnement lié.

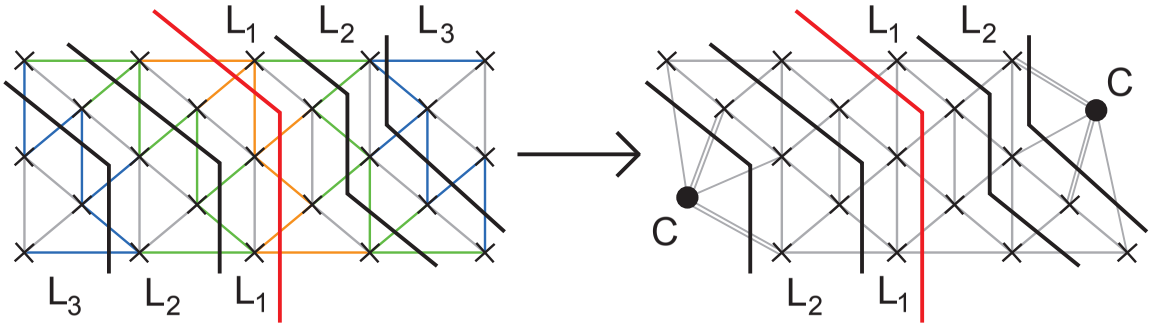

FIGURE 7.2 – *Création d'un graphe "clusterisé" à partir du graphe partitionné de la figure 7.1. On choisit ici de ne conserver que les deux premières levelsets dans chaque sous-graphe. Certaines arêtes multiples apparaissent, qu'il est possible de regrouper.*

Une telle solution est facilement adaptable à une double décomposition de domaine telle que celle utilisée dans YALES2 (voir section 4.2), à partir du graphe associé au maillage formé par les groupes de cellules. Le graphe ainsi créé est de taille très restreinte par rapport au nombre de noeuds du maillage originel et le coût de calcul requis pour sa création est très faible. D'autre part, cette méthodologie implique que seule la migration de groupes de cellules est à prévoir.

7.2.1.3 Utilisation de la librairie PT-Scotch

Une fois le graphe "clusterisé" créé, nous devons être capables de le partitionner. Ce partitionnement est pour l'instant effectué grâce à PT-Scotch, une bibliothèque permettant entre autres le partitionnement parallèle de graphes et d'hypergraphes [61]. Un effort particulier a donc été requis pour le stockage des données du graphe "clusterisé" sous la forme requise par cette bibliothèque.

Le problème principal de notre approche est que l'algorithme de partitionnement utilisé est pour ainsi dire "en boîte noire". Cet état de fait implique que nous n'avons aucun contrôle direct sur la forme finale de la partition, et que nous devons donc ajuster directement le graphe envoyé aux fonctions de la bibliothèque PT-Scotch pour que la partition renvoyée réponde à certains critères :

- chaque sous-graphe de la partition doit contenir un et un seul cluster ;
- chaque sous-graphe de la partition doit être connexe ;
- le "crénelage" aux frontières de chaque sous-graphe doit être évité autant que possible.

Nous avons mis en oeuvre dans un premier temps un clustering "artificiel" : les noeuds appartenant au cluster restaient séparés dans le graphe envoyé à la bibliothèque PT-Scotch, mais un poids énorme était affecté aux arcs reliant deux noeuds quelconques du cluster. Cette approche, plus simple à coder, a en revanche causé des problèmes dans la pratique, puisque la fonction de partition de graphe de PT-Scotch vise avant tout à faire descendre le ratio de déséquilibre de charges en-dessous d'une valeur d'environ 10/9, même si cela doit se faire au prix d'un *edgecut* démesuré. L'intégrité des clusters dans la partition résultante n'est alors pas assurée, et il est plus difficile de déterminer si la taille des clusters est trop grande ou trop petite. Il a donc fallu mettre en place un clustering véritable dans le code.

La condition selon laquelle chaque sous-graphe de la partition doit contenir un et un seul cluster est alors supposée être respectée à partir du moment où le nombre de levelsets est assez réduit pour que la charge d'un cluster représente une portion importante de la charge totale d'un sous-graphe. En pratique, on ne connait pas *a priori* le nombre idéal de levelsets à utiliser, et cette donnée dépend en grande partie du nombre de groupes par processeur. C'est pourquoi le code peut réessayer de repartitionner le domaine avec un nombre différent de levelsets si la première tentative a échoué. On distingue alors deux types d'échecs différents :

- si deux clusters se retrouvent sur un même processeur, cela signifie que les clusters sont trop petits, et donc qu'il faut réduire le nombre de levelsets ;
- si le déséquilibre de charges entre processeurs est toujours élevé, cela signifie que les clusters sont trop gros, et donc qu'il faut augmenter le nombre de levelsets.

Quand la condition de répartition des clusters est obtenue, il ne reste plus qu'à substituer les couleurs renvoyées par PT-Scotch par les rangs des processeurs en prenant en compte que les clusters ne doivent pas migrer.

Les premiers résultats obtenus par cette approche sur un cas-test en deux dimensions est donné par la table 7.1. Un léger crénelage apparaît aux frontières dans la nouvelle partition du maillage, qui est dû à la forme des groupes de cellules eux-mêmes, mais les résultats sont globalement très satisfaisants, avec un ratio de déséquilibre qui redescent en-dessous de 1,1 grâce au transfert d'un nombre réduit de groupes.

En revanche, la connexité de chaque sous-graphe n'est pas assurée dans certains cas plus complexes, notamment le cas-test ci-dessus avec périodicité, ainsi que le cas-test tridimensionnel considéré dans ce travail, constitué par un maillage de 96000 éléments

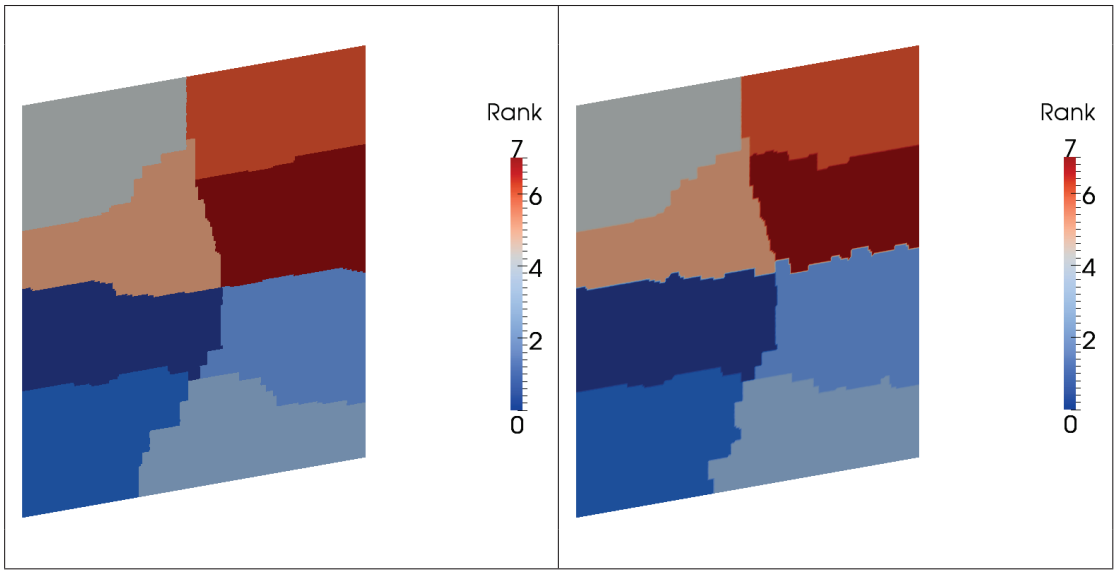

TABLE 7.1 – *Résultats d'un rééquilibrage de charges sur un maillage de 44000 éléments en deux dimensions, séparés en groupes de 100 éléments répartis sur 8 processeurs. Le poids de chaque noeud du graphe à partitionner est multiplié par* $1 + r_p/10$*, où* r_p *est le rang du processeur, allant de 0 à 7 dans ce cas. Dans chaque figure, ainsi que dans celles qui suivent, les couleurs différencient les rangs des processeurs. Le ratio de déséquilibre du graphe initial vaut 1,382 ; il est ramené à 1,095 par la repartition.*

tétraédriques sur un parallélépipède rectangle. Les fonctions de la librairie PT-Scotch ne permettant pas d'assurer directement une contrainte de connexité des sous-graphes, nous avons dû jouer avec les poids des arêtes du graphe envoyé au partitionneur. Plus précisément, ce travail est fondé sur deux idées :

- pour éviter un crénelage dans la nouvelle partition, les arêtes reliant deux noeuds d'une même levelset pourraient avoir un poids plus grand que les arêtes reliant deux noeuds de levelsets distinctes ;
- pour favoriser la migration des groupes proches des frontières entre les processeurs, les poids des arêtes entre deux levelsets distinctes pourraient être plus importants à mesure que l'on s'éloigne des frontières.

Nous avons donc introduit deux multiplicateurs w_p^1 et w_p^2 répondant respectivement à ces deux idées : le poids d'une arête entre un noeud de la levelset i et un noeud de la levelset $i+1$ est multiplié par $\left(w_p^2\right)^i$, et le poids d'une arête entre deux noeuds de la même levelset i est multiplié par $w_p^1 \left(w_p^2\right)^{i-1}$.

Nous avons testé l'influence de ce paramètre dans les trois cas-tests présentés dans cette section. Dans le cas-test en deux dimensions non périodique, l'adoption de valeurs

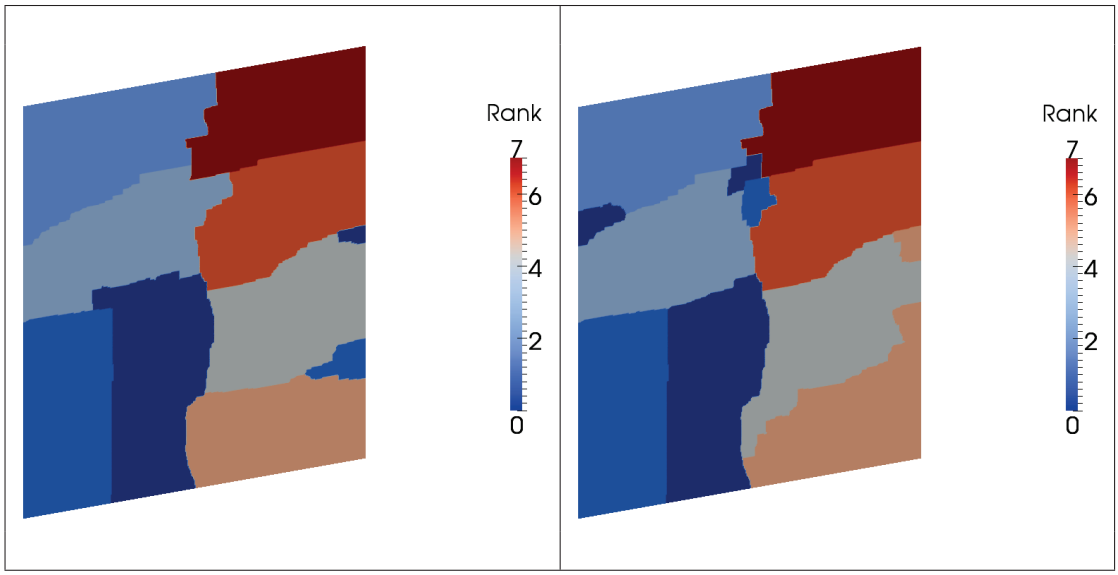

TABLE 7.2 – *Résultats du rééquilibrage de charges décrit en table 7.3, en adoptant des couples de coefficients* $\left(w_p^1; w_p^2\right)$ *valant* $(5; 2)$ *(figure de gauche) et* $(10; 20)$ *(figure de droite).*

de w_p^1 et w_p^2 dépassant quelques unités conduit à des déformations de la solution obtenue. En particulier, si la valeur de w_p^1 est trop grande, il est possible qu'une portion importante d'une levelset donnée soit transférée sur un autre processeur, ce qui cause une déformation de la partition telle que celles présentées en table 7.2.

En revanche, dans les deux autres cas-tests, les problèmes de non-connexité des sous-graphes qui émergent en temps normal disparaissent grâce à l'adoption de valeurs relativement faibles des paramètres, mais réapparaissent pour des valeurs plus basses ou plus élevées, comme détaillé en tables 7.3 et 7.4.

- Dans le cas bidimensionnel, la connexité de chaque sous-domaine est assurée pour un certain nombre de couples de valeurs $\left(w_p^1; w_p^2\right)$ relativement faibles, jusqu'à $(5; 5)$. Les solutions présentées ici sont celles pour lesquelles l'edgecut semblent "à l'oeil nu" le plus faible, grâce à une faible déformation des sous-domaines. Dans d'autres cas, des structures en P apparaissent, c'est-à-dire qu'une portion importante de la première levelset d'un sous-domaine peut se retrouver attribuée au processeur voisin.

- Dans le cas tridimensionnel, mentionnons que les solutions de la colonne de droite, obtenues avec les couples de paramètres $(3; 2)$ et $(10; 1)$, ont des ratios de déséquilibre de 1,11 et 1,107 respectivement, et donc sont jugées recevables par la fonction de partitionnement de la bibliothèque PT-Scotch. Le ratio de déséquilibre de la solution obtenue avec $w_p^1 = w_p^2 = 5$ n'est pas significativement meilleur, puisqu'il vaut 1,1 ; en revanche, la connexité des sous-domaines en fait une bien meilleure solution.

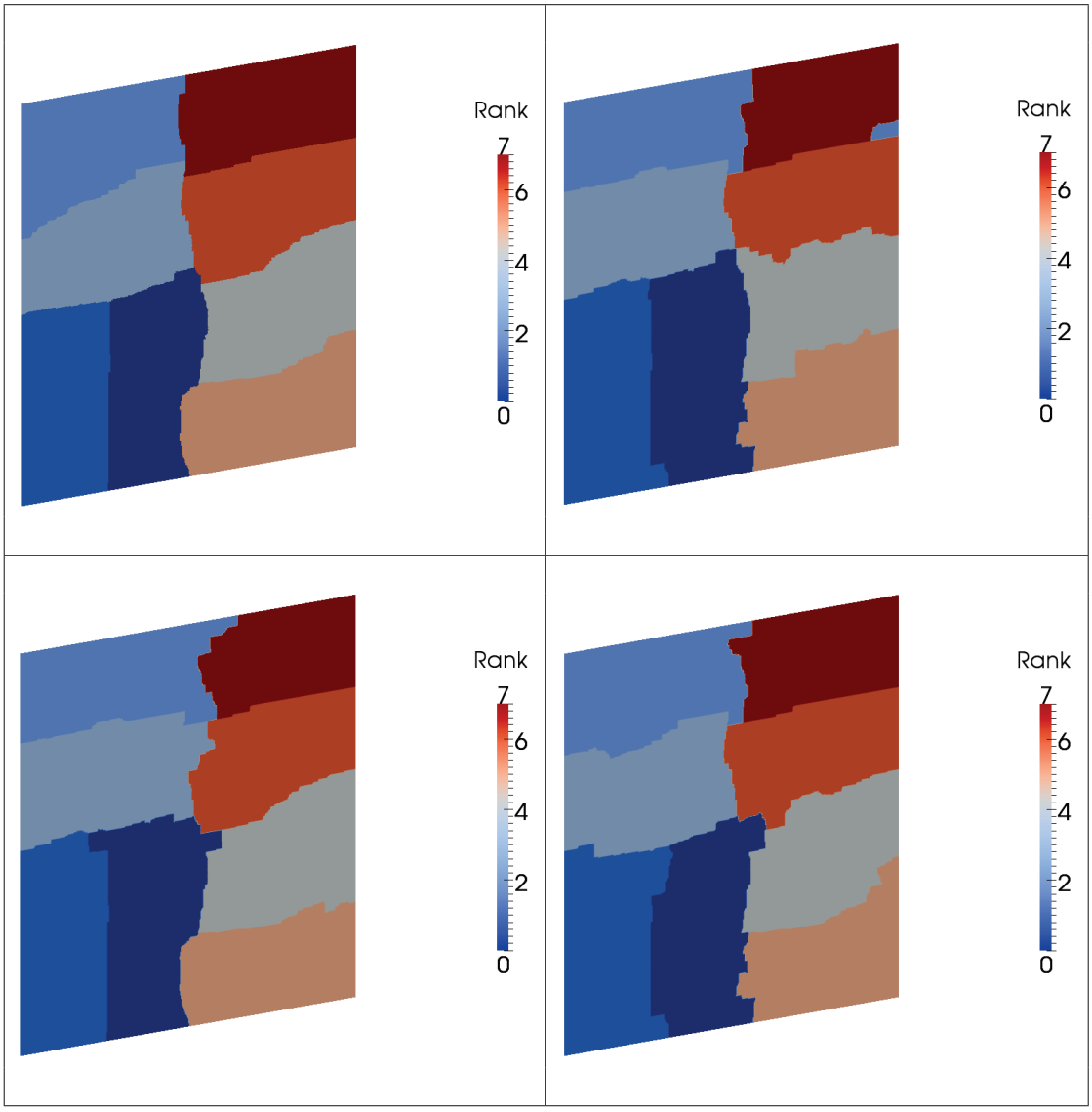

TABLE 7.3 – *Résultats d'un rééquilibrage de charges sur un maillage de 176000 éléments en deux dimensions avec conditions de périodicité aux frontières, séparés en groupes de 500 éléments répartis sur 8 processeurs. Le poids de chaque noeud du graphe à partitionner est multiplié par* $1 + r_p/10$*, où* r_p *est le rang du processeur. Le ratio de déséquilibre du graphe initial (en haut à gauche) vaut 1,384 ; il est ramené à 1,086 par la repartition de paramètres unité (*$\left(w_p^1; w_p^2\right) = (1;1)$*, en haut à droite) ; 1,101 avec* $\left(w_p^1; w_p^2\right) = (2;1)$ *(en bas à gauche) ; et 1,087 avec* $\left(w_p^1; w_p^2\right) = (1;2)$ *(en bas à droite).*

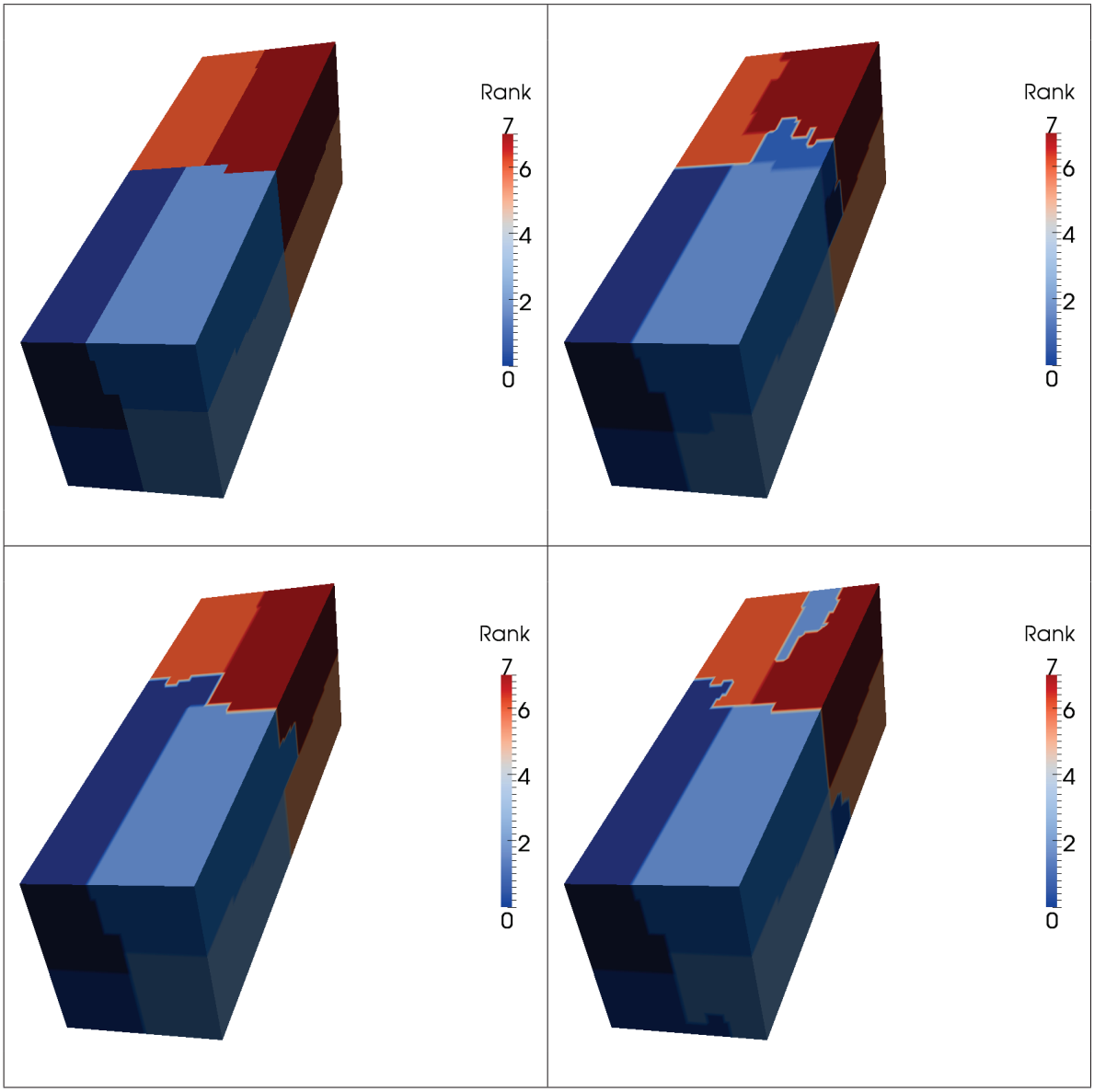

TABLE 7.4 – *Résultats d'un rééquilibrage de charges sur un maillage de 96000 éléments en trois dimensions, séparés en groupes de 200 éléments répartis sur 8 processeurs. Le poids de chaque noeud du graphe à partitionner est multiplié par* $1 + r_p/10$*, où* r_p *est le rang du processeur, allant de 0 à 7 dans ce cas. La figure en haut à gauche représente la partition initiale, dont le ratio de déséquilibre vaut 1,386. En haut à droite, la partition obtenue avec* $\left(w_p^1; w_p^2\right) = (3; 2)$ *; en bas à gauche, la partition obtenue avec le couple* $(5; 5)$ *; en bas à droite, la partition obtenue avec le couple* $(10; 1)$*.*

7.2.1.4 Ouvertures

Pour continuer la procédure de tests commencée à la fin de cette thèse, il serait utile que le code soit à même de calculer les *edgecuts* correspondant aux partitions initiale et finale. Cette information serait particulièrement précieuse pour la comparaison qualitative de différentes partitions, notamment dans des cas correspondant à des simulations sur des géométries plus complexes, et dans lesquels la vérification "à l'oeil nu" n'est plus fiable. Dans la même optique, le développement d'une fonction permettant de vérifier la connexité de chacun des sous-domaines donnés par la repartition parait indispensable si le développement d'une procédure de repartitionnement dans YALES2 reste basé sur l'utilisation de la bibliothèque PT-Scotch.

Une autre solution, bien plus coûteuse, consiste évidemment à développer un outil de partitionnement parallèle de graphe au sein de YALES2, dans lequel une condition de connexité des sous-graphes de la partition résultante serait imposée. Outre l'important coût de développement qui en résulterait, cette démarche pose la question de la méthode de partitionnement à utiliser. Le choix le plus courant est celui de la bisection récursive [5, 33], qui ne convient pas dans notre cas de figure puisque le nombre de processeurs utilisés pour une simulation par YALES2 peut ne pas être égal à une puissance de 2 : une autre solution serait donc à trouver. L'idéal reste de toute façon une méthode de partition multi-niveaux, pour des raisons de limitation du coût de calcul, mais la solution obtenue sur un niveau "grossier" de graphe doit alors être raffinée au cours de la redescente vers le graphe originel, comme suggéré par Walshaw *et al.* dans [84], en appliquant à proximité des frontières entre les nouveaux sous-graphes des méthodes comme celles de Kernighan-Lin [36] et de Fiduccia-Mattheyses [21]. Quant à l'algorithme de partitionnement *per se*, il peut être choisi parmi une vaste gamme de possibles. Le concepteur de PT-Scotch a opté pour un algorithme génétique, dont la pertinence est discutable au même titre que celle de tout autre choix. Le développement d'un outil de partition au sein de YALES2 semble donc exclu pour le moment, en raison de sa lourdeur.

Dans un second temps, une fois la procédure de repartitionnement prête à être utilisée lors de simulation réelles, un critère de déclenchement de cette procédure restera à déterminer. Le plus simple semble être de recalculer le ratio de déséquilibre régulièrement et d'imposer un rééquilibrage des charges dès que ce ratio dépasse une certaine valeur. Cependant, la procédure de rééquilibrage de charges représente un temps de calcul non négligeable, notamment à cause des nombreuses communications interprocesseurs requises pour le transfert de groupes de cellules et la reconstruction des communicateurs ; il faudra donc veiller à ne pas repartitionner trop souvent le domaine.

7.2.2 Autres perspectives

Le troisième thème évoqué dans le sujet de thèse initial était le parallélisme hybride CPU/GPU. Cette approche pose cependant des problèmes de mise en oeuvre, particuliè-

rement au niveau de la gestion de la mémoire.

En revanche, puisque les machines clusterisées sont de plus en plus répandues, représentant notamment 81,4 % des 500 plus puissants supercalculateurs du monde au mois de juin 2012, l'importance de l'utilisation d'OpenMP pour la gestion de la mémoire au sein d'un même cluster semble primordiale. L'utilisation conjointe des bibliothèques MPI et OpenMP semble donc un enjeu important pour la vitesse d'exécution d'un logiciel de simulation massivement parallèle, et l'utilisation d'OpenMP au sein du solveur YALES2 serait probablement un objectif raisonnable à intégrer à un agenda de développement déjà très chargé.

Chapitre 8

Annexe : Pistes de développement pour la déflation

Comme évoqué dans le chapitre 6 du présent manuscrit, un certain nombre de questions ont été soulevées par nos tentatives successives d'optimisation de la technique de déflation appliquée à la méthode des Gradients Conjugués. Il nous a notamment semblé que la littérature présentait relativement peu de pistes de recherches concernant le comportement numérique de la méthode de déflation en fonction des vecteurs de déflation sélectionnés.

Ce chapitre présente les différentes voies qui ont été explorées, plus ou moins en profondeur, au sujet de la stabilité numérique d'un algorithme de type Gradients Conjugués avec déflation. Certaines d'entre elles ont été proposées par Messieurs Luc Giraud et Boniface Nkonga, que nous remercions chaleureusement.

8.1 Stratégies de construction des vecteurs de déflation linéaire et quadratique

Concernant les déflations dites "plane" et "quadrique", Boniface Nkonga nous a suggéré une méthode de construction des vecteurs gradients et Hessiens qui passerait par l'utilisation d'une méthode des moindres carrés, afin notamment d'assurer l'indépendance linéaire des vecteurs de déflation. Toujours d'après lui, des techniques de limitation de pente sont également envisageables si nécessaire. Nous n'avons pas eu le temps de nous pencher sur la question, qui pourrait avoir une importante incidence sur le fonctionnement de ces méthodes de déflation géométrique non-standard, voire même les rendre effectives dans un solveur non-structuré.

8.2 Variante flexible du Gradient Conjugué

Comme la méthode de déflation peut être vue, d'un point de vue théorique, comme un préconditionnement, Luc Giraud nous a suggéré que la méthode A-DEF2 telle que nous l'avons mise en oeuvre, c'est-à-dire avec des résolutions de systèmes successifs sur le maillage grossier par un solveur itératif dont le critère de convergence est variable, peut être vue comme une méthode de Gradient Conjugué à préconditionnement variable, ou Gradient Conjugué Flexible, tel qu'introduit par Notay dans [58].

Cependant, le fait que le solveur sur le niveau grossier soit de toute façon itératif nous permet *a priori* de considérer chacune des mises en oeuvre que nous avons faites du Gradient Conjugué avec déflation comme un cas particulier du Gradient Conjugué Flexible. Le cadre de travail mis en place par Notay dans ce contexte peut donc sembler pertinent pour l'étude des problèmes liés à certains vecteurs de déflation dans le cadre de ce travail de thèse.

D'un point de vue théorique, Notay introduit la matrice de préconditionnement B, idéalement symétrique et définie positive, telle que le spectre de $B^{-1}A$ est de moins grande amplitude que celui de A et qu'un système linéaire de type $Bu = y$ est relativement rapide à résoudre. La complexité des problèmes traités fait hélas qu'au lieu d'une résolution exacte, la résolution de tout système du type $Bu = y$ se fera de manière approchée, et en lieu et place de $u = B^{-1}y$ sera obtenue l'approximation $u = \mathcal{B}(y)$ où $\mathcal{B}$ est une fonction non linéaire de $\mathbb{R}^n$ vers $\mathbb{R}^n$. L'auteur précise entre autres que sa méthode de travail, bien que pouvant être mise en parallèle avec celle de Golub et Ye [27], tend à moins surestimer les effets de la non-linéarité de $\mathcal{B}$. Dans notre cadre de travail, l'approche de Notay nous semble moins restrictive que celle de Golub et Ye, dans laquelle la conception même de l'algorithme PCG modifié dépend de conditions à imposer sur le calcul du préconditionnement, qui n'est jamais effectué de manière explicite dans un algorithme comme A-DEF2.

L'itération de l'algorithme sur lequel se base les travaux de Notay dans cet article est constituée des points suivants :

1. appliquer le préconditionnement inexact au résidu de l'itération précédente, soit $w_i = \mathcal{B}(r_i)$;
2. orthogonaliser w_i par rapport à un nombre m_i de directions de descente précédentes afin d'obtenir la direction de descente d_i ;
3. effectuer l'itération des Gradients Conjugués et recalculer le résidu pour pouvoir réitérer.

L'application d'un algorithme de type PCG revient à choisir $m_i = 1$ pour tout i, ce qui assure les propriétés de convergence optimales si $\mathcal{B} = B^{-1}$; pour respecter les conditions de convergence optimale de la méthode, il faudrait au contraire effectuer des

A-orthogonalisations complètes des directions de descente, c'est-à-dire imposer $m_i = i$. Notay suggère un calcul de la forme

$$m_i = \max(1, \mod (i, m_{max} + 1)) \tag{8.1}$$

avec m_{max} constant, de façon à imposer des redémarrages partiels réguliers. Cette stratégie d'A-orthogonalisation plus poussée pourrait être appliquée aux solveurs linéaires de type Gradient Conjugué avec Déflation étudiés au cours de cette thèse.

Notay établit par la suite des résultats de convergence moins pessimistes que ceux de Golub et Ye [27], puisque tendant exactement vers les résultats de la méthode PCG quand $\mathcal{B}$ tend vers B^{-1}. Ces résultats nécessitent cependant de connaitre les répercussions en termes de B-norme de l'estimation de $B^{-1}r_i$ par $\mathcal{B}(r_i)$ à chaque itération, ce qui est difficile pour un algorithme tel que A-DEF2. En effet, la matrice de préconditionnement d'une telle méthode est complexe et impossible à calculer directement, puisque dépendant de A^{-1}, et le solveur itératif sur le maillage grossier ne résout pas directement le système $Bw_i = r_i$. Il nous semble même difficile de vérifier la condition

$$\frac{\|\mathcal{B}(r_i) - B^{-1}r_i\|_B}{\|B^{-1}r_i\|_B} < 1\,, \tag{8.2}$$

pourtant strictement requise pour l'étude de convergence faite par Notay. Il apparait néanmoins que l'étude des valeurs propres de $B^{-1}A$ et de leur répartition soit utile à l'avancement de tels travaux, mais rappelons que la matrice de préconditionnement de l'algorithme A-DEF2 [75] vaut

$$B^{-1} = P^T M^{-1} + Q \text{ avec } P = I - AQ\ ;\ Q = W\hat{A}^{-1}W^T\,. \tag{8.3}$$

8.3 Etude du spectre de l'opérateur sur le maillage grossier

L'étude des valeurs propres de la matrice $\hat{A}$ semble utile, notamment pour envisager la possibilité d'en faire coincider certaines avec des valeurs propres de A. Il s'agirait d'un rapprochement entre les méthodes de déflation "géométriques" décrites dans ce manuscrit et la méthode de déflation d'un sous-espace propre imaginée initialement par Nicolaides.

Cette étude a échappé à notre degré de compétence, en raison de sa complexité formelle. Une collaboration est actuellement envisagée entre l'équipe de Vincent Moureau, au CORIA, et l'équipe de Luc Giraud (INRIA/CERFACS), à ce sujet.

Nous avons rapidement remarqué que, pour un vecteur propre x de la matrice A associé à la valeur propre λ, on peut écrire que si

$$\exists X \in \mathbb{R}^k,\ WX = x\,, \tag{8.4}$$

alors

$$AWX = \lambda WX\,, \tag{8.5}$$

ce qui donne

$$W^T AWX = \lambda W^T WX\,, \tag{8.6}$$

et finalement

$$W^T W = I \Rightarrow \hat{A}X = \lambda X\,. \tag{8.7}$$

Le projeté d'un vecteur propre de A est donc vecteur propre de $\hat{A}$ avec la même valeur propre associée s'il appartient au sous-espace défini par les vecteurs de W, ce qui correspond au cas d'utilisation originellement envisagé par Nicolaides, avec des vecteurs propres orthonormaux. Cependant, les conséquences de l'application de la méthode de déflation avec des vecteurs de déflation motivés par des arguments géométriques ou physiques, et non purement calculatoires, sont difficiles à évaluer d'un point de vue théorique. Il nous semble que ce champ de recherche est primordial pour toute étude complémentaire concernant les possibilités d'application "libre" de la déflation, autres que l'application géométrique standard actuellement très répandue, qui est justifiée par des arguments physiques et validée par les résultats en termes de temps de calcul.

Liste des tableaux

Table des figures

Bibliographie

[1] G. Abdelkader. Introduction à la méthode multigrille. Available at http ://www.scribd.com/doc/15098060/Introduction-a-la-Methode-MultiGrille.

[2] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1) :145–159, 1996.

[3] R. Aubry, F. Mut, R. Löhner, and J. R. Cebral. Deflated preconditioned conjugate gradient solvers for the pressure-poisson equation. *Journal of Computational Physics*, 227 :10196–10208, 2008.

[4] O. Axelsson. A restarted version of a generalized preconditioned conjugate gradient method. *Communications in Applied Numerical Methods*, 4(4) :521–530, July/August 1988.

[5] S. T. Barnard and H. Simon. A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. In *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, pages 627–632, 1995.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems : Building Blocks for Iterative Methods, 2nd Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994.

[7] T. J. Barth. Aspects of unstructured grids and finite-volume solvers for the euler and navier-stokes equations. In *Special Course on Unstructured Grid Methods for Advection Dominated Flows*, pages 18–34, 1994.

[8] C. Berge. *Graphes et Hypergraphes*. Dunod, Paris, 1970.

[9] J. Boussinesq. Essai sur la théorie des eaux courantes. *Mémoires des savants étrangers*, 2(23), 1941.

[10] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138) :333–390, 1977.

[11] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid Tutorial, Second Edition*. Society for Industrial and Applied Mathematics, 2001.

[12] A. J. Chorin. The numerical solution of the navier-stokes equations for an incompressible fluid. *Bulletin of the American Mathematical Society*, 73 :928–931, 1967.

[13] A. J. Chorin. Numerical solution of the navier-stokes equations, 1968.

[14] A. I. Cohen. Rate of convergence of several conjugate gradient algorithms. *SIAM Journal on Numerical Analysis*, 9(2), 1972.

[15] E. F. D'Azevedo, V.L. Eijkhout, and C. H. Romine. Lapack working note 56 - conjugate gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Technical report, Mathematical Sciences Section, Oak Ridge National Laboratory, December 1999.

[16] O. Desjardins, V. Moureau, and H. Pitsch. An accurate conservative level set/ghost fluid method for simulating turbulent atomization. *J. Comput. Phys.*, 227(18) :8395–8416, 2008.

[17] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9(3) :302–325, 1983.

[18] M. Engell, T. Ginsburg, H. Rutishauser, and E. Stiefel. Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems. *Birkhauser Verlag*, 1959.

[19] J. Erhel and F. Guyomarc'h. An augmented subspace conjugate gradient. Technical Report 3278, INRIA, October 1997.

[20] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152 :457–492, 1999.

[21] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[22] P. F. Fischer. Projection techniques for iterative solution of $a\underline{x} = \underline{b}$ with successive right-hand sides. *Computer methods in applied mechanics and engineering*, 163 :193–24, 1998.

[23] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2) :442–462, 2001.

[24] J. Galpin, A. Naudin, Luc Vervisch, C. Angelberger, O. Colin, and P. Domingo. Large-eddy simulation of a fuel lean premixed turbulent swirl burner. *Combustion and Flame*, 155(1) :247–266, 2008.

[25] L. Giraud, H. Baya Toda, O. Cabrit, S. Bose, and J. Lee. Using singular values to build a subgrid-scale model for large eddy simulations. *Physics of Fluids*, 23(8) :085106–12, 2011.

[26] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and lanczos algorithms : 1948-1976. *SIAM Review*, 31(1) :50–102, 1989.

[27] G. H. Golub and Q. Ye. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM Journal of Scientific Computing*, 21 :1305–1320, 1997.

[28] S. Grout, C. Dumouchel, J. Cousin, and H. Nuglisch. Fractal analysis of atomizing liquid flows. *International Journal of Multiphase Flow*, 33(9) :1023–1044, 2007.

[29] W. Hackbusch. *Multigrid Methods and Applications*. Springer-Verlag, Berlin, 1985.

[30] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12) :2182–2190, 1965.

[31] B. M. Irons. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2 :5–32, 1970.

[32] J.Crank and P.Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1) :50–67, 1947.

[33] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1) :359–392, 1998.

[34] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1) :359–392, 1999.

[35] A. J. Katz. *Meshless methods for Computational Fluid Dynamics*. PhD thesis, Department of Aeronautics and Astronautics, Stanford, Janvier 2009.

[36] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(1) :291–307, 1970.

[37] D. S. Kershaw. The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1) :43–65, 1978.

[38] J. Kim and P. Moin. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of Computational Physics*, 59(2) :308–323, 1985.

[39] M. P. Kirkpatrick, A. S. Ackerman, D. E. Stevens, and N. N. Mansour. On the application of the dynamic smagorinsky model to large-eddy simulations of the cloud-topped atmospheric boundary layer. *Journal of the Atmospheric Sciences*, 63 :526 ?546, 2005.

[40] A. N. Kolmogorov. On degeneration of isotropic turbulence in an incompressible viscous liquid. *Comptes-Rendus (Doklady) de l'Academie des Sciences de l'U.R.S.S.*, 31 :538–540, 1941.

[41] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. *51st Annual IEEE Symposium on Foundations of Computer Science*, 2010.

[42] M. Kraushaar. *Application of the compressible and low-Mach number approaches to Large-Eddy Simulation of turbulent flows in aero-engines*. PhD thesis, Institut National Polytechnique de Toulouse, Décembre 2011.

[43] A. N. Krylov. Sur la résolution numérique de l'équation séculaire des petites oscillations des systèmes matériels. *Izvestiya Akademii Nauk SSSR*, 7(4) :491–539, 1931.

[44] W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46 :435–453, 1901.

[45] H. Ladisch, A. Schulz, and H.J. Bauer. Heat transfer measurements on a turbine airfoil with pressure side separation. *ASME Conference Proceedings*, 2009(48845) :783–793, 2009.

[46] N. Lamarque. *Schémas numériques et conditions limites pour la simulation aux grandes échelles de la combustion diphasique dans les foyers d'hélicoptère*. PhD thesis, Institut National Polytechnique de Toulouse, Décembre 2007.

[47] G. Lartigue, U. Meier, and C. Bérat. Experimental and numerical investigation of self-excited combustion oscillations in a scaled gas turbine combustor. *Applied Thermal Engineering*, 24(11-12) :1583–1592, 2004.

[48] C.-J. Lin and J. J. Moré. Incomplete cholesky factorizations with limited memory. *Journal of Scientific Computing*, 21(1) :24–45, 1999.

[49] J. W. H. Liu. The multifrontal method for sparse matrix solution : theory and practice. *SIAM Review*, 34(1) :82–109, 1992.

[50] E. Lutum and F. Cottier. Aerothermal predictions on a highly loaded turbine blade including effects of flow separation. In ITU Faculty of Mechanical Engineering, editor, *9th European Turbomachinery Conference*, Istanbul, 2011.

[51] N. Maheu, V. Moureau, and P. Domingo. Large-eddy simulations of flow and heat transfer around a low-mach turbine blade. In *Turbulence, Heat and Mass Transfer 7*, pages 395–398. Begell House, Inc., 2012.

[52] W. Meier, P. Weigand, X. R. Duan, and R. Giezendanner-Thoben. Detailed characterization of the dynamics of thermoacoustic pulsations in a lean premixed swirl flame. *Combustion and Flame*, 150(1-2) :2–26, July 2007.

[53] 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, 1977.

[54] V. Moureau, C. Bérat, and H. Pitsch. An efficient semi-implicit compressible solver for large-eddy simulations. *Journal of Computational Physics*, 226 :1256–1270, 2007.

[55] V. Moureau, P. Minot, C. Bérat, and H. Pitsch. A ghost-fluid method for large-eddy simulations of premixed combustion in complex geometries. *Journal of Computational Physics*, 211 :600–614, 2007.

[56] R. Nabben and C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. Technical Report 03-10, Delft University of Technology, June 2003.

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

[58] Y. Notay. Flexible conjugate gradients. *SIAM Journal of Scientific Computing*, 22(4) :1444–1460, 2000.

[59] C. W. Oosterlee and T. Washio. On the use of multigrid as a preconditioner. In *9th Domain Decomposition Conference*, pages 441–448. P.E. Bjorstad et al., 1996.

[60] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Taylor and Francis Group, 1980.

[61] F. Pellegrini. Distillating knowledge about SCOTCH. In Uwe Naumann, Olaf Schenk, Horst D. Simon, and Sivan Toledo, editors, *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[62] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12 :241–254, 1977.

[63] M. C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21, 1984.

[64] S. Roux, G. Lartigue, T. Poinsot, U. Meier, and C. Bérat. Studies of mean and unsteady flow in a swirled combustion using experiments, acoustic analysis and large-eddy simulations. *Combustion and Flame*, 141 :40–54, 2005.

[65] C. Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2) :167–178, 1895.

[66] Y. Saad. *Iterative Methods for Sparse Linear Systems (second edition)*. WPS, 2003.

[67] Y. Saad and M. H. Schultz. Gmres : A generalized minimal residual algorithm for solving nonsymmetric linear systems. *Journal of Scientific Computing*, 7 :856–869, 1986.

[68] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21 :1909–1926, 2000.

[69] P. Sagaut. *Large Eddy Simulation for Incompressible Flows*. Springer, third edition, 2006.

[70] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Available at http ://www.cs.cmu.edu/q̃uake-papers/painless-conjugate-gradient.pdf, 1994.

[71] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91 :99–164, 1963.

[72] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI, The Complete Reference*. The MIT Press, 1996.

[73] P. Spalart, W. Jou, M. Strelets, and S. Allmaras. Comments on the feasibility of LES for wing and on a hybrid RANS/LES approach. In *First AFOSR Conference on DNS and LES, Ruston, Louisiana*, 1997.

[74] P. P. Sullivan, J. C. McWilliams, and C. H. Moeng. A subgrid-scale model for large-eddy simulation of planetary boundary-layer flows. *Boundary Layer Meteorology*, 71 :247–276, 1994.

[75] J.R. Tang, R. Nabben, C. Vuik, and Y.A. Erlangga. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *SIAM Journal on Scientific Computing*, 39 :340–370, 2009.

[76] O. Tatebe and Y. Oyanagi. Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines. In *Supercomputing '94*, pages 194–203, 1994.

[77] R. Temam. *Navier-Stokes equations : Theory and Numerical Analysis*. North-Holland Publishing Company, 1977.

[78] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[79] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[80] S. Vantieghem. *Numerical simulations of quasi-static magnetohydrodynamics using an unstructured finite-volume solver : development and applications*. PhD thesis, Université Libre de Bruxelles, January 2011.

[81] F. J. Vermolen, C. Vuik, and A. Segal. Deflation in preconditioned conjugate gradient methods for finite element problems. In *DIAM Annual Reports, available at http ://ta.twi.tudelft.nl/TWA_Reports/02/02-10new.pdf*, 2002.

[82] H. A. Van Der Vorst. Parallel iterative solution methods for linear systems arising from discretized pde's. In *Lecture Notes on Parallel Iterative Methods for discretized PDE's. AGARD Special Course on Parallel Computing in CFD, available from http ://www.math.ruu.nl/people/vorst/agard.ps.gz*, 1995.

[83] C. Walshaw and M. Berzins. Dynamic load balancing for PDE solvers on adaptive unstructured meshes. Technical report, University of Leeds, Division of Computer Science, December 1992.

[84] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47 :102–108, 1997.

[85] M. Weickert, G. Teike, O. Schmidt, and M. Sommerfeld. Investigation of the les wale turbulence model within the lattice boltzmann framework. *Computers and Mathematics with Applications*, 59(7) :2200–2214, 2010.

[86] P. Wesseling, J. Wiley, and sons. *An introduction to multigrid methods*. R. T. Edwards, Chichester, 1992.

[87] O. B. Widlund. The development of coarse spaces for domain decomposition algorithms. In *18th International Conference on Domain Decomposition*, 2008.

[88] J. H. Williamson. Low-storage runge-kutta schemes. *Journal of Computational Physics*, 35(1) :48–56, 1980.

[89] Y. F. Yao, A. M. Savill, N. D. Sandham, and W. N. Dawes. Simulation and modelling of turbulent trailing-edge flow. *Flow, Turbulence and Combustion*, 68 :313–333, 2002.

[90] P. J. Zwart, G. D. Raithby, and M. J. Raw. The integrated space-time finite volume method and its application to moving boundary problems. *Journal of Computational Physics*, 154(2) :497–519, 1999.