Développement et validation d'un solveur linéaire hybride RSVD-SOR pour les équations de Navier-Stokes 3D incompressibles

par

Alexis MALAVAL

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE AVEC CONCENTRATION PERSONNALISÉE
M. Sc. A.

MONTRÉAL, LE 15 MAI 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Mme. Camille Coti, directrice de mémoire
Département du génie logiciel et des TI à l'École de Technologie Supérieure

M. Damien Pham Van Bang, codirecteur
Département du génie de la construction à l'École de Technologie Supérieure

M. Kim Khoa Nguyen, président du jury
Département de génie électrique à l'École de Technologie Supérieure

M. Sheldon Andrews, membre du jury
Département de génie logiciel et des TI à l'École de Technologie Supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 20 AVRIL 2026

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens tout d'abord à exprimer ma reconnaissance envers l'École de technologie supérieure qui m'a permis de réaliser cette maîtrise. Ce parcours au sein d'un environnement de recherche stimulant a été une étape déterminante de mon cheminement.

Je souhaite adresser mes plus sincères remerciements aux professeurs Mme. Camille Coti et M. Damien Pham Van Bang pour avoir dirigé ce travail. Je les remercie pour la confiance qu'ils m'ont témoignée et pour leur disponibilité constante tout au long de ce projet. Nos rencontres régulières ont été particulièrement utiles pour affiner ma réflexion et assurer la progression constante de mes recherches. Je leur suis également reconnaissant pour le financement qu'ils m'ont accordé par le biais de l'ÉTS. Ce soutien financier a été un levier essentiel pour mener à bien ce projet de recherche dans les meilleures conditions.

Toute ma gratitude va également à M. Abdelkader Hammouti, associé de recherche, dont l'expertise technique a été d'une aide précieuse. Nos échanges réguliers m'ont permis de surmonter les obstacles rencontrés lors de ce projet.

Enfin, je tiens également à souligner l'appui de M. Assè-fètou Simon Yaka, doctorant au laboratoire. Je le remercie pour le temps qu'il m'a accordé, pour ses conseils et pour l'entraide constante qui a régné entre nous.

Ce travail a également bénéficié de l'accès aux ressources de calcul de haute performance fournies par Calcul Québec et l'Alliance de recherche numérique du Canada. Je tiens à les remercier pour la mise à disposition du supercalculateur Narval, sans lequel les tests de performance à grande échelle n'auraient pu être réalisés.

Développement et validation d'un solveur linéaire hybride RSVD-SOR pour les équations de Navier-Stokes 3D incompressibles

Alexis MALAVAL

RÉSUMÉ

Les équations de Navier–Stokes incompressibles peuvent être résolues par différentes stratégies numériques, dont les méthodes de projection. Ces dernières nécessitent la résolution répétée d'une équation de Poisson pour la pression afin de satisfaire la contrainte de divergence nulle. Cette opération elliptique constitue, de manière générale, la principale étape de cout de calcul des simulations. Le code NSMP utilisé dans ce projet n'échappe pas à cette contrainte, la résolution du problème de Poisson y représentant le goulot d'étranglement dominant.

Ce mémoire présente le développement et la validation d'un solveur linéaire hybride conçu pour améliorer cette résolution. L'approche proposée combine la décomposition en valeurs singulières aléatoire (RSVD) avec un lisseur de type sur-relaxation successive (SOR). L'algorithme RSVD est utilisé pour générer rapidement une approximation de rang faible du système, fournissant un point de départ robuste qui capture les modes dominants du champ de pression. Le lisseur SOR intervient ensuite pour éliminer les résidus restants et assurer la convergence vers la solution exacte du système original.

La précision de la méthode a été validée sur le cas test analytique du vortex de Taylor-Green, montrant un ordre de convergence spatiale de 2. Les performances ont été évaluées par une étude de passage à l'échelle forte allant de 1 à 256 processeurs sur le supercalculateur Narval. Les résultats démontrent qu'en ajustant le seuil d'énergie, le solveur hybride surpasse significativement l'algorithme MultiSOR classique. La flexibilité paramétrique de cet algorithme permet d'optimiser le temps de calcul sans compromettre la rigueur numérique.

Mots-clés: Navier-Stokes, CFD, RSVD, SOR, Calcul haute performance, Équation de Poisson

Development and Validation of a Hybrid RSVD-SOR Linear Solver for 3D Incompressible Navier-Stokes Equations

Alexis MALAVAL

ABSTRACT

Incompressible Navier-Stokes equations can be solved using various numerical strategies, including projection methods. These approaches require the repeated resolution of a Poisson equation for pressure to satisfy the divergence-free constraint. This elliptic operation generally constitutes the primary computational cost of the simulations. The NSMP code used in this project is no exception to this constraint, as the resolution of the Poisson problem represents the dominant bottleneck.

This thesis presents the development and validation of a hybrid linear solver designed to improve this resolution. The proposed approach combines Randomized Singular Value Decomposition (RSVD) with a Successive Over-Relaxation (SOR) smoother. The RSVD algorithm is used to quickly generate a low-rank approximation of the system, providing a robust initial guess that captures the dominant modes of the pressure field. The SOR smoother then intervenes to eliminate the remaining residuals and ensure convergence toward the exact solution of the original system.

The accuracy of the method was validated using the Taylor-Green vortex analytical test case, demonstrating a spatial convergence order of 2. Performance was evaluated through a strong scaling study ranging from 1 to 256 processors on the Narval supercomputer. The results demonstrate that by adjusting the energy threshold, the hybrid solver significantly outperforms the classic MultiSOR algorithm. The parametric flexibility of this algorithm allows for computational time optimization without compromising numerical rigor.

Keywords: Navier-Stokes, RSVD, SOR, CFD, High-Performance Computing, Poisson equation

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AGMG	Aggregation-based Algebraic Multigrid (Méthode multigrille algébrique agglomérative)
BiCGStab	Biconjugate Gradient Stabilized (Algorithme itératif)
BLAS	Basic Linear Algebra Subprograms (Routines optimisées pour les opérations de base en algèbre linéaire)
CALU	Communication Avoiding LU (Algorithme de factorisation LU minimisant les communications inter-processeurs)
CFD	Computational Fluid Dynamics (Simulation numérique de dynamique des fluides)
CPU	Central Processing Unit (Unité centrale de traitement ou processeur)
CUDA	Compute Unified Device Architecture (Plateforme de calcul parallèle et interface de programmation de NVIDIA pour l'utilisation des GPU)
DF	Différences Finies
DNN	Deep Neural Network (Réseau de neurones profonds)
DNS	Direct Numerical Simulation (Simulation numérique directe des équations de Navier–Stokes sans modélisation de la turbulence)
EDP	Équations aux Dérivées Partielles
EF	Éléments Finis
ÉTS	École de Technologie Supérieure
GNN	Graph Neural Networks (Réseaux de neurones graphiques)
GMRES	Generalized Minimal RESidual (Méthode de Krylov itérative)
GPU	Graphics Processing Unit (Processeur graphique)
HPC	High-Performance Computing (Calcul haute performance)
ILU(k)	Incomplete Lower-Upper (Factorisation LU incomplète de niveau de remplissage k)
IMEX	Implicit–Explicit (Schéma numérique combinant des traitements temporels implicites et explicites)

LAPACK	Linear Algebra PACKage (Bibliothèque standard pour les opérations d'algèbre linéaire complexe)
LES	Large-Eddy Simulation (Simulation des grandes échelles turbulentes)
LSFD-FV	Least Square-based Finite Difference-Finite Volume (Schéma hybride)
LU	Lower-Upper (Méthode de décomposition d'une matrice en produit de matrices triangulaires)
ML	Machine Learning (Apprentissage machine)
MPI	Message Passing Interface (Standard de communication pour le calcul distribué)
MUMPS	Multifrontal Massively Parallel Sparse (Solveur direct parallèle pour systèmes linéaires creux)
NCCL	NVIDIA Collective Communications Library (Bibliothèque de communications collectives optimisée pour les multi-GPU)
NSMP	Navier-Stokes Multi Phase (Code de calcul tridimensionnel utilisé et optimisé dans cette étude)
OpenMP	Open Multi-Processing (Interface pour le parallélisme à mémoire partagée)
PD	Parabolic Diffusion (Méthode d'accélération par introduction d'un temps fictif)
RANS	Reynolds-Averaged Navier–Stokes (Modélisation statistique des effets de la turbulence)
RSVD	Randomized Singular Value Decomposition (Décomposition en valeurs singulières aléatoire)
ScaLAPACK	Scalable Linear Algebra PACKage (Version parallèle de LAPACK pour mémoire distribuée)
SOR	Successive Over-Relaxation (Méthode itérative de résolution)
SVD	Singular Value Decomposition (Décomposition en valeurs singulières)
TAU	Tuning and Analysis Utilities (Suite logicielle utilisée pour le profilage et l'analyse de performances HPC)

TGV	Taylor-Green Vortex (vortex de Taylor-Green)
VF	Volumes Finis
WENO	Weighted Essentially Non-Oscillatory (Schéma numérique)

LISTE DES SYMBOLES ET UNITÉS DE MESURE

A	Matrice du système linéaire (Ax=b)
b	Vecteur des termes source du système linéaire
f	Forces volumiques
H	Hauteur totale de la colonne d'eau
h	Bathymétrie ou profondeur au repos
L	Longueur caractéristique ou matrice triangulaire inférieure (LU)
p	Pression
Pe	Nombre de Péclet
Q	Matrice orthogonale (QR) ou base réduite (RSVD)
R	Matrice triangulaire supérieure (QR)
t	Temps physique
u, v, w	Composantes du champ de vitesse
U	Vitesse caractéristique ou matrice triangulaire supérieure (LU)
x	Vecteur des inconnues
α_k	Fraction volumique de la phase k (code NSMP)
Δt	Pas de temps de la simulation
η	Déplacement de la surface libre
$\kappa(A)$	Conditionnement de la matrice A
μ	Viscosité dynamique
ν	Viscosité cinématique ou coefficient de diffusion
ρ	Masse volumique du fluide
σ	Coordonnée verticale réduite
σ_i	Valeur singulière d'ordre i
Σ	Somme ou matrice diagonale des valeurs singulières (RSVD)
τ	Temps fictif pour la méthode de diffusion parabolique

ϕ	Variable générique ou composante de vitesse
ψ	Variable scalaire pour l'approximation aux interfaces
ω	Paramètre de relaxation de la méthode SOR
∇	Opérateur nabla (gradient ou divergence)

INTRODUCTION

L'étude de la dynamique des fluides trouve son fondement mathématique dans les équations de Navier-Stokes. À travers la CFD (*Computational Fluid Dynamics*), ces équations deviennent un outil de modélisation puissant, capable de simuler fidèlement des écoulements réels en s'appuyant sur les capacités croissantes du calcul haute performance. L'évolution de la discipline a permis de passer de solutions analytiques simplifiées à des méthodes numériques capables de traiter des géométries de plus en plus complexes. Dans ce contexte, la performance d'un solveur se définit par sa capacité à naviguer entre deux exigences souvent opposées : la précision numérique, indispensable pour bien capturer les phénomènes physiques, et la rapidité d'exécution, qui détermine la viabilité pratique du code.

Dans le cadre de cette étude, le code NSMP (Navier-Stokes Multi Phase) sera utilisé. Il s'agit d'un solveur tridimensionnel développé pour simuler des écoulements incompressibles turbulents avec transport de sédiments, sur des géométries complexes. Il repose sur la résolution des équations de Navier–Stokes pour la phase fluide et de l'équation d'Exner pour la phase solide (sédiments) (Uh Zapata, 2012). Une transformation de coordonnées σ est utilisée pour prendre en compte la topographie du fond et la déformation de la surface libre. Une méthode de projection est utilisée pour traiter le couplage pression-vitesse (Chorin, 1968), imposant la résolution d'une équation de Poisson à chaque pas de temps.

Malgré ses capacités, le code fait face à des défis computationnels majeurs. La résolution de l'équation de Poisson constitue le principal goulot d'étranglement, représentant la majorité du temps de calcul total. Historiquement, le code NSMP s'est appuyé sur les méthodes SOR (*Successive Over-Relaxation*) et BiCGStab (*Biconjugate Gradient Stabilized*). Si ces outils ont permis d'aborder des simulations complexes, leur exploitation en environnement parallèle a révélé des limites. Le solveur BiCGStab, présente des problèmes de divergence critiques dans certaines configurations distribuées. À l'inverse, si le solveur SOR s'avère performant et stable,

la résolution de l'équation de Poisson pour la pression y est extrêmement coûteuse en temps, particulièrement lors des premiers pas de temps.

Ce constat constitue le cœur du problème. L'analyse de la matrice du système, suggère qu'une résolution purement itérative n'est sûrement pas la solution la plus optimale. En utilisant la décomposition en valeurs singulières aléatoire (*Randomized Singular Value Decomposition*), il devient possible de générer une approximation de rang faible de cette matrice complexe (Martinsson, Rokhlin & Tygert, 2011). Cette étape permet de capturer rapidement l'essentiel de l'information du système pour accélérer le début de la résolution.

C'est dans cette perspective d'optimisation que s'inscrit le travail. Un solveur hybride est développé : la RSVD fournit une solution approchée globale qui sert de point de départ robuste, tandis qu'un lisseur SOR se charge d'éliminer les erreurs restantes pour atteindre la précision requise.

Ce manuscrit est structuré en trois chapitres principaux :

Le **Chapitre 1** dresse un état de l'art des stratégies numériques pour les fluides incompressibles. Le cas d'étude est présenté en détails et des stratégies de parallélisation y sont développées.

Le **Chapitre 2** expose le développement méthodologique. Après une analyse des performances du code initial et de la matrice du système, divers algorithmes ont été étudiés et essayés, menant au choix du solveur RSVD+SOR.

Enfin, le **Chapitre 3** présente les résultats obtenus. Nous y évaluons la précision de la méthode sur le cas test du vortex de Taylor-Green et présentons une étude de passage à l'échelle allant de 1 à 256 processeurs pour valider l'efficacité de notre approche en environnement HPC (Calcul Haute Performance).

CHAPITRE 1

REVUE DE LITTÉRATURE ET PRÉSENTATION DU CAS D'ÉTUDE

1.1 Contexte des écoulements incompressibles et stratégies numériques

Les problèmes d'écoulement incompressible représentent un domaine d'étude fondamental en mécanique des fluides, en ingénierie et en physique. Ils sont régis par les équations de Navier-Stokes, qui décrivent la dynamique du fluide, en tenant compte de sa pression, de sa vitesse, de sa masse volumique et de sa viscosité. L'incompressibilité du fluide est exprimée par l'équation de continuité, stipulant que la divergence du champ de vitesse est nulle ($\nabla \cdot \mathbf{u} = 0$) (Perron, Boivin & Hérard, 2004; Busto, Dumbser & Río-Martín, 2023).
Ces équations, qui expriment la conservation de la masse et de la quantité de mouvement pour un fluide Newtonien incompressible, s'écrivent sous forme vectorielle :

$$\nabla \cdot \mathbf{u} = 0 \tag{1.1}$$

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1.2}$$

Où :

- $\mathbf{u}$ représente le champ de vitesse du fluide ;
- p est la pression statique ;
- ρ désigne la masse volumique du fluide ;
- μ est la viscosité dynamique ;
- $\mathbf{f}$ représente les forces de volume (comme la gravité).

Dans les méthodes numériques dédiées aux fluides incompressibles, un enjeu fondamental réside dans le découplage entre la pression et la vitesse. Diverses approches ont été développées

pour traiter ce problème, parmi lesquelles les méthodes dites de projection occupent une place centrale.

Le code NSMP (*Navier-Stokes Multi Phase*) est un solveur tridimensionnel s'appuyant sur une méthode de projection pour le couplage pression-vitesse et une transformation de coordonnées σ pour la gestion des géométries variables. Bien que ce code constitue le support applicatif de nos travaux, la revue de littérature présentée ci-après s'étend au-delà du cadre spécifique de NSMP pour aborder les avancées globales en mécanique des fluides numérique et en stratégies de calcul haute performance.

1.1.1 Méthodes de projection

Les méthodes de projection sont largement utilisées pour découpler le calcul des champs de vitesse et de pression dans les schémas numériques, en particulier dans les contextes de volumes finis non structurés (Zhang *et al.*, 2020). Ces techniques reposent généralement sur une stratégie de pas fractionnaires, permettant de résoudre successivement les différentes composantes du problème sans imposer simultanément toutes les contraintes physiques. Leur objectif principal est d'imposer, à chaque pas de temps, la condition d'incompressibilité du champ de vitesse (Guermond, 1996).

La procédure typique se compose de deux étapes. Tout d'abord, une étape de prédiction permet de résoudre les équations de quantité de mouvement sans inclure le gradient de pression. Cette étape produit un champ de vitesse intermédiaire, qui ne respecte pas nécessairement la contrainte de divergence nulle. Vient ensuite une étape de correction, ou projection proprement dite, qui consiste à résoudre une équation de Poisson pour la pression. La solution obtenue permet de corriger le champ de vitesse intermédiaire afin d'obtenir un champ finalement de divergence nulle, en conformité avec l'incompressibilité. Mathématiquement, cette opération peut être interprétée comme une projection orthogonale du champ de vitesse intermédiaire sur l'espace des champs solénoïdaux (Guermond & Shen, 2006). La pression agit comme un multiplicateur de Lagrange imposant la contrainte de divergence nulle sur le champ de vitesse (Perron *et al.*,

2004). Plusieurs variantes de cette méthode existent, selon la manière dont la pression et les termes visqueux sont traités. Dans notre cas, nous utilisons la méthode de projection standard, où la prédiction est effectuée sans gradient de pression, puis la correction est obtenue via la résolution de l'équation de Poisson. À l'inverse, les méthodes dites de correction de pression consistent à calculer une approximation de la pression dans l'étape de prédiction, puis à la corriger ultérieurement. Ces dernières peuvent cependant introduire des conditions aux limites artificielles (souvent de type Neumann sur la pression), qui ne correspondent pas toujours aux conditions physiques réelles par exemple en surface libre ou au fond (Guermond & Shen, 2006). Une alternative plus précise est la forme rotationnelle de la correction de pression, qui inclut une correction du champ de divergence dans le calcul, assurant ainsi une meilleure cohérence aux limites et une précision accrue (Guermond & Shen, 2003). Par ailleurs, certaines méthodes dites de correction de vitesse inversent l'ordre des traitements : elles prennent en compte les termes visqueux dès la première étape, pour ne les corriger qu'ensuite dans la seconde.

La stabilité et la précision de ces méthodes reposent fortement sur leur capacité à maintenir l'incompressibilité du champ de vitesse, même après discrétisation (Guermond, 1996). En projetant systématiquement le champ de vitesse intermédiaire sur un sous-espace sans divergence, les algorithmes de projection garantissent une solution stable et satisfaisant exactement la contrainte d'incompressibilité à chaque pas de temps (Guermond & Shen, 2006). La simulation numérique de ces écoulements présente plusieurs défis importants. L'un d'eux est le coût computationnel élevé, particulièrement pour les géométries complexes et les régimes turbulents qui nécessitent l'utilisation de maillages très fins (Hanna *et al.*, 2020; Uh Zapata *et al.*, 2016). La complexité des géométries et la nature turbulente des écoulements rendent la simulation directe des équations de Navier-Stokes particulièrement ardue (Hanna *et al.*, 2020). Pour relever ces défis, diverses approches numériques ont été développées. Les méthodes des volumes finis (VF) sont très répandues pour intégrer les équations aux dérivées partielles ainsi que les conditions initiales et aux limites. D'autres familles de méthodes existent, comme les éléments finis (EF) ou les différences finies (DF), utilisant des schémas d'ordre élevé, qui sont souvent recherchés afin

d'améliorer la précision (Perron *et al.*, 2004; Liu, Miao & Zhang, 2023; Boivin, Cayré & Hérard, 2000).

Lorsque la pression et la vitesse sont définies sur les mêmes points du maillage (grilles colocalisées), cela peut engendrer un phénomène de damier (checkerboard effect), caractérisé par une oscillation alternée des valeurs de pression entre mailles adjacentes ; sans une discrétisation adaptée, cela génère des instabilités numériques dépourvues de sens physique (Perron *et al.*, 2004; Ma, Shi & Kirby, 2012).

1.1.2 Choix du maillage

La qualité du maillage constitue un facteur déterminant pour garantir l'exactitude, la stabilité et l'efficacité des simulations d'écoulement incompressible. Dans la configuration de géométries complexes, les maillages non structurés constituent un avantage majeur (Perron *et al.*, 2004; Liu *et al.*, 2023; Boivin *et al.*, 2000; Li *et al.*, 2025). Toutefois, cette liberté géométrique peut introduire des distorsions de mailles susceptibles de nuire à la précision du calcul des flux. Pour assurer la stabilité numérique, des schémas adaptés, tels que les méthodes à bords centrés pour les problèmes de diffusion anisotropes, sont mis en œuvre (Liu *et al.*, 2023). Par ailleurs, la capture des structures fines de l'écoulement nécessite souvent un raffinement local du maillage. Pour gérer efficacement le volume de calcul qui en résulte sans dégrader les performances, le recours à des méthodes multigrilles algébriques s'avère particulièrement robuste (Notay & Napov, 2015). Enfin, ce choix de maillage complexe justifie l'utilisation d'une grille colocalisée stabilisée par interpolation de quantité de mouvement, évitant ainsi la complexité algorithmique des grilles décalées en 3D non structuré (Busto *et al.*, 2023; Zhang *et al.*, 2020).

L'un des enjeux majeurs réside dans l'arrangement spatial des variables, où deux stratégies principales s'affrontent. Historiquement, les grilles décalées (*staggered grids*) ont été privilégiées pour leur capacité à prévenir naturellement le phénomène d'oscillation numérique, dit "effet de damier", en stockant la pression au centre des cellules et les composantes de la vitesse sur leurs faces respectives (Perron *et al.*, 2004; Ma *et al.*, 2012). Cependant, cette configuration impose

une complexité algorithmique importante, rendant son implémentation particulièrement difficile lors du traitement de géométries complexes ou de domaines irréguliers (Hammouti, 2009; Busto *et al.*, 2023).

À l'inverse, les grilles colocalisées se distinguent par une plus grande simplicité de mise en œuvre, car toutes les variables sont définies aux mêmes points du maillage. Bien que cette approche soit intrinsèquement sujette au découplage entre la pression et la vitesse, la stabilité numérique peut être rétablie par l'usage de techniques spécifiques. Des méthodes telles que l'interpolation de la quantité de mouvement ou la pondération par la pression sont alors indispensables pour assurer la cohérence du couplage physique et éviter les instabilités (Zhang *et al.*, 2020; Uh Zapata, 2012; Perron *et al.*, 2004).

L'amélioration de la rentabilité computationnelle du solveur repose sur des stratégies avancées visant à maximiser l'exploitation des ressources matérielles et la précision des modèles. Sur le plan du parallélisme et de la performance, le réordonnancement de la grille constitue un levier essentiel pour optimiser l'agencement des données en mémoire, ce qui permet d'accélérer significativement la convergence des algorithmes dans un contexte de calcul distribué (Grigori, Demmel & Xiang, 2011).

De plus, l'émergence de couplages hybrides ouvre des perspectives innovantes pour la simulation numérique. L'intégration d'approches mêlant l'intelligence artificielle au calcul classique permet notamment de corriger les erreurs de maillage ou de mieux appréhender les phénomènes complexes de turbulence (Hanna *et al.*, 2020).

En définitive, la structuration du maillage pour les fluides incompressibles est dictée par la nécessité de concilier une représentation fidèle des géométries irrégulières avec la robustesse du couplage numérique entre la pression et la vitesse.

1.1.3 La méthode des volumes finis

La méthode des volumes finis est intéressante pour plusieurs raisons, notamment ses propriétés de conservation, sa flexibilité et son adaptabilité à divers types de problèmes et de maillages, ainsi que sa capacité à gérer les discontinuités. La figure 1.1 illustre un exemple de discrétisation tridimensionnelle utilisée dans la méthode des volumes finis, avec stockage des variables (vitesse, pression) au centre des cellules sur un maillage colocalisé. Les figures (b) et (c) montrent respectivement les relations géométriques entre voisins horizontaux et verticaux, essentielles pour la reconstruction des gradients dans les maillages non structurés.

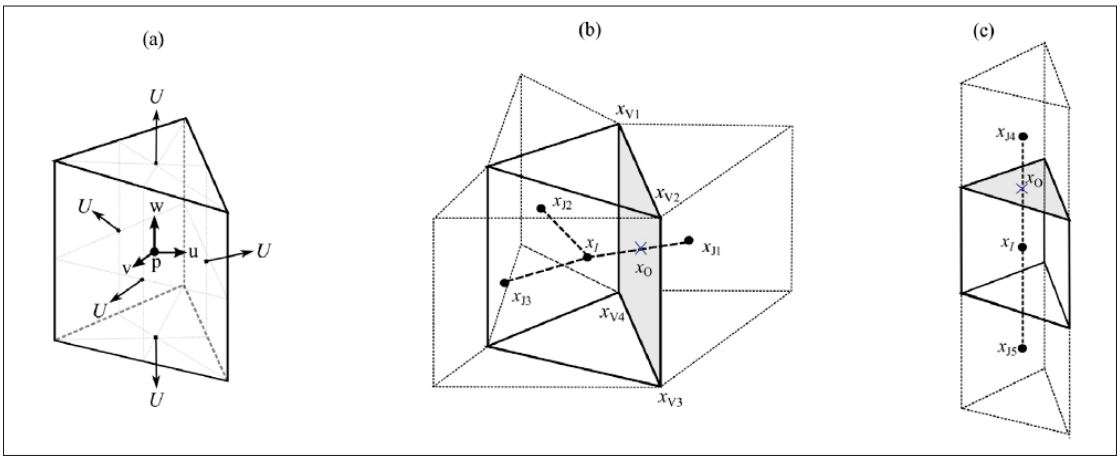


Figure 1.1 (a) Variables u, v, w, p stockées au centre des cellules sur une grille colocalisée 3D. (b) Connexions horizontales entre éléments voisins dans le plan. (c) Connexions verticales entre éléments superposés. Ce type de géométrie est typique des volumes finis sur maillages non structurés à 5 faces
Tirée par Zhang *et al.* (2020)

La méthode des volumes finis est largement utilisée pour ses propriétés de conservation et sa forte capacité d'adaptation à la géométrie (Liu *et al.*, 2023). Elle est appliquée à la résolution de lois de transport (convection–diffusion) sur des géométries arbitraires. La méthode des volumes finis exploite le théorème de Green-Ostrogradski, qui permet de transformer une divergence volumique en flux surfacique sur le contour de la cellule de contrôle. Cette propriété est particulièrement utile pour traiter le terme non linéaire de convection, et rend possible

l'utilisation de maillages non structurés pour résoudre les équations de Navier–Stokes sur des domaines complexes (Liu *et al.*, 2023; Zhang *et al.*, 2020).

Cette méthode des volumes finis permet également de conserver une propriété essentiellement non oscillatoire près des chocs forts ou des discontinuités de contact sur des maillages décalés, et est conçue pour résoudre des problèmes d'écoulement visqueux unidimensionnels et bidimensionnels contenant des chocs ou des discontinuités de contact (Cui & Zhu, 2022). Elle favorise le respect des contraintes physiques locales (Boivin *et al.*, 2000). Cette propriété est essentielle pour garantir la stabilité globale de la simulation sans introduire de diffusion numérique excessive. Parmi les méthodes numériques d'ordre élevé sur les maillages non structurés, les méthodes de volumes finis ont aussi une construction plus simple et une mise en œuvre plus facile (Liu, Shu, Zhang, Yang & Lee, 2021). Pour éviter le problème de damier causé par les maillages colocalisés, une méthode d'interpolation de la quantité de mouvement est utilisée, en introduisant des vitesses normales aux faces aux points médians (Zhang *et al.*, 2020). Les méthodes de type Godunov à ordre élevé nécessitent de connaître les variables conservées directement sur les faces entre cellules. Pour cela, ces valeurs sont généralement obtenues en reconstruisant les données disponibles aux centres des cellules adjacentes. (Ma *et al.*, 2012).

Applications pratiques de la méthode des volumes finis :

Elle est utilisée pour résoudre les équations de Navier-Stokes en 3D (Perron *et al.*, 2004), les équations compressibles sur maillages non structurés (Liu *et al.*, 2021). Des modèles non hydrostatiques tridimensionnels s'appuient aussi sur cette méthode (Zhang *et al.*, 2020). Elle est efficace pour la simulation d'écoulements faiblement compressibles à faible nombre de Mach (Busto *et al.*, 2023), et a été validée par des cas tests comme l'écoulement de cavité à couvercle (Liu *et al.*, 2021; Li *et al.*, 2025).

1.1.4 Hybridation numérique et intelligence artificielle

Au-delà des méthodes classiques de projection, d'autres stratégies numériques ont été explorées pour améliorer la précision, la flexibilité ou l'efficacité des simulations de fluides incompressibles.

Parmi elles, les approches hybrides exploitent les avantages respectifs de deux formulations : les volumes finis sont souvent utilisés pour la discrétisation des termes de convection et de diffusion, tandis que les éléments finis continus sont appliqués à la résolution de l'équation de pression. Ce couplage permet une meilleure adaptabilité aux géométries complexes, tout en maintenant un bon niveau de conservation locale.

Une autre classe d'approches repose sur les méthodes dites LSFD-FV (*Least Square-based Finite Difference-Finite Volume*). Ces schémas hybrides combinent la conservation locale des volumes finis avec une reconstruction de gradients par moindres carrés, permettant d'atteindre un ordre spatial élevé sur maillages non structurés. Ils ont été développés initialement pour les écoulements compressibles et visqueux, afin de mieux capturer les discontinuités et les chocs. En revanche, leur usage pour les fluides incompressibles reste plus limité à ce jour, car la résolution de la pression et le maintien de la contrainte de divergence nulle sont généralement traités efficacement par les volumes finis classiques (Liu *et al.*, 2021).

La résolution de l'équation de Poisson pour la pression reste l'un des goulots d'étranglement majeurs en termes de performance. De récents travaux ont proposé d'utiliser des réseaux de neurones pour accélérer ou remplacer certaines étapes de cette résolution. Des méthodes dites informées par la physique permettent de résoudre cette équation sur différents types de maillages, sans recourir à des données d'apprentissage explicites (Li *et al.*, 2025). L'objectif est de réduire le coût d'inversion des matrices associées à la pression non hydrostatique, en s'appuyant sur des schémas de discrétisation locaux plus compacts et optimisés (Cui, Pietrzak & Stelling, 2012). Ce type d'intégration vise à améliorer à la fois la précision locale et la scalabilité du solveur, en tirant parti des perspectives d'évaluation rapide sur GPU (Margenberg *et al.*, 2024).

En effet, l'intégration de l'apprentissage machine (*Machine Learning, ML*) et des réseaux de neurones (*Neural Networks, NN*) dans le domaine de la simulation numérique, gagne en importance en raison de leur potentiel à améliorer l'efficacité et la précision.

Plusieurs approches sont explorées à cette fin :

- Prédiction d'erreurs et modélisation de la turbulence :
  Une approche basée sur l'apprentissage machine est développée pour la prédiction des erreurs dans les simulations CFD sur maillage grossier (Hanna *et al.*, 2020). Le ML offre une voie pour prédire les erreurs dans les simulations à grande échelle mais repose souvent sur de nombreuses corrélations empiriques. L'apprentissage profond par renforcement pour la mécanique des fluides est également un sujet d'étude (Novati & Koumoutsakos, 2021).
- Résolution d'EDP basée sur les réseaux de neurones :
  Une méthode hybride, DNN-MG (*Deep Neural Network Multigrid*), combinant les réseaux de neurones profonds et la méthode des éléments finis est développée pour les simulations 3D des équations de Navier-Stokes (Margenberg *et al.*, 2024). L'objectif est d'augmenter une simulation par éléments finis sur des maillages grossiers avec des informations à fine échelle obtenues via des réseaux de neurones profonds. Cette approche locale permet au réseau de rester relativement petit et d'être appliqué en parallèle sur tous les *patchs* (cellules et leur voisinage local) de maillage, améliorant ainsi la généralisabilité. La figure 1.2 illustre ce cycle de traitement local par le réseau DNN dans DNN-MG :

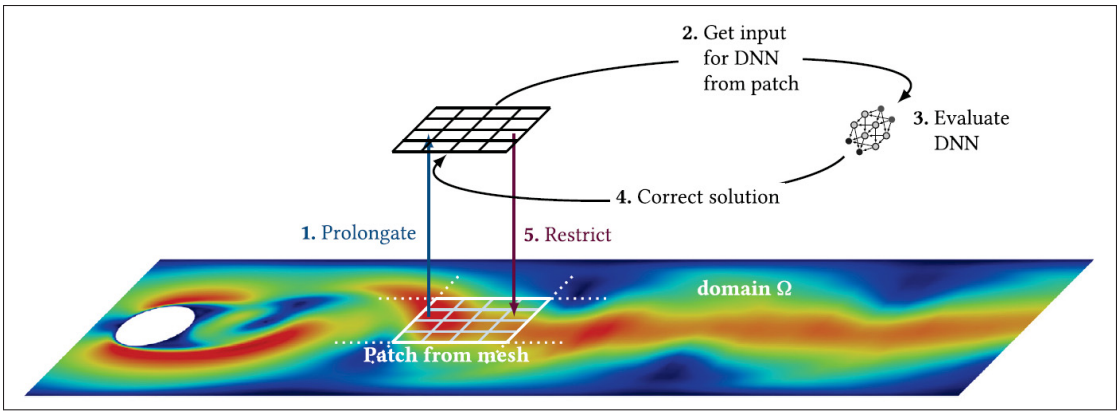


Figure 1.2 Cycle local du DNN-MG : interpolation d'un patch vers une échelle fine (1), extraction des données (2), évaluation du réseau (3), correction (4) puis restriction vers le maillage grossier (5). Une fois les patchs interpolés, le résidu du maillage fin est calculé, puis découpé en sous-patchs à traiter en parallèle par le réseau. Les mises à jour sont combinées pour corriger la solution du champ de vitesse. Ce processus est illustré en Fig. 1.3
Tirée par Margenberg *et al.* (2024)

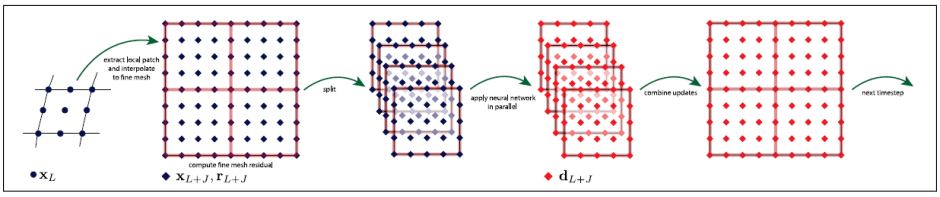


Figure 1.3 Intégration parallèle de DNN-MG : à partir de la solution sur maillage grossier x_L, une interpolation sur le maillage fin est effectuée, puis les patchs sont traités indépendamment par le réseau de neurones pour calculer l'update d_{L+J}
Tirée par Margenberg *et al.* (2024)

L'avantage principal de DNN-MG est la capacité d'atteindre une très haute précision avec un maillage grossier, réduisant ainsi considérablement le temps de calcul. La parallélisation est un élément clé, avec l'évaluation du réseau de neurones effectuée sur un GPU et l'utilisation de directives OpenMP pour la parallélisation des opérateurs de pré-traitement et de post-traitement (Margenberg *et al.*, 2024).

D'autres approches utilisent des réseaux de neurones récurrents pour simuler des EDP évolutives en les traitant comme des séries temporelles. Des modèles hybrides Élément Fini–Réseau de Neurones ont également été explorés (Margenberg *et al.*, 2024).

- Méthodes hybrides innovantes :

  Un cadre computationnel combine la méthode des volumes finis avec les réseaux de neurones graphiques (*Graph Neural Networks*) pour construire une fonction de perte basée sur l'EDP, permettant la résolution directe d'EDP paramétriques pendant l'entraînement sans nécessiter de données précalculées. Ce cadre vise à atténuer la dépendance vis-à-vis des ensembles de données de haute précision, en tirant parti du calcul parallèle sur GPU et en mettant en œuvre une méthode des volumes finis complète et différentiable (Li *et al.*, 2025). Cette approche hybride a le potentiel d'intégrer des algorithmes d'ordre réellement supérieur à l'avenir. Les comparaisons de performances ont été faites sur différents types de maillages, y compris des maillages quadrilatéraux, triangulaires anisotropes et mixtes triangle-quadrilatère (Li *et al.*, 2025).

En résumé, l'apprentissage automatique et les réseaux de neurones sont activement explorés pour des applications variées en simulation numérique, allant de la prédiction des erreurs et la modélisation de la turbulence à la résolution directe d'EDP. Les approches hybrides combinant des méthodes numériques traditionnelles avec le ML, ainsi que l'utilisation de calcul parallèle sur GPU et des techniques de parallélisation comme OpenMP, sont des pistes prometteuses pour surmonter les défis de la complexité computationnelle et de la généralisation (Novati & Koumoutsakos, 2021; Hanna *et al.*, 2020; Li *et al.*, 2025; Margenberg *et al.*, 2024).

1.2 Parallélisation des solveurs

1.2.1 Nécessité de la parallélisation pour les simulations CFD

La parallélisation est indispensable dans la simulation des écoulements CFD : les coûts numériques augmentent fortement avec l'extension spatiale du domaine, la nécessité d'une

résolution fine pour la turbulence, la prise en compte de l'instationnarité ou encore la considération d'effets non hydrostatiques comme les interactions fluide–structure.

Au cœur de ces simulations réside la résolution des équations de Navier-Stokes et en particulier du problème de Poisson associé pour la résolution de la pression sur un grand nombre de points et de pas de temps, des tâches extrêmement coûteuses en temps de calcul (Perron *et al.*, 2004). La résolution de ces systèmes linéaires de grande taille constitue le segment le plus lent du code et la principale cause de goulots d'étranglement des performances sur les ordinateurs parallèles (Chowdhury & L'Excellent, 2010). À titre d'exemple, une simulation d'écoulement thermique dans des enceintes de confinement a exigé une semaine de calcul avec 128 processeurs pour simuler seulement 10 secondes de dépressurisation de vapeur, même avec une grille grossière d'environ 1 million de mailles (tailles de cellules de 3 à 14 cm) et un modèle RANS (Reynolds-Averaged Navier–Stokes, famille de modèles de turbulence où l'on ne résout pas directement toutes les fluctuations turbulentes), ce qui illustre la lourdeur numérique de telles simulations (Hanna *et al.*, 2020). Ce défi computationnel est d'autant plus grand pour les écoulements complexes avec des transitoires prolongés. De surcroît, les modèles diphasiques, essentiels pour simuler le transport de sédiments dans les écoulements CFD, sont particulièrement coûteux en temps de calcul. En effet, certains nécessitent la résolution de deux systèmes d'équations de Navier–Stokes, l'un pour la phase fluide et l'autre pour la phase solide, couplés par des termes de transfert inter-phase de quantité de mouvement (Nguyen *et al.*, 2012).

Une autre raison fondamentale réside dans l'exigence d'une haute résolution pour capturer fidèlement les phénomènes complexes, en particulier les écoulements turbulents. Ces derniers, très fréquents dans les milieux fluviaux, imposent des résolutions spatiales et temporelles fines, dont l'échelle dépend fortement du nombre de Reynolds. Dans le cas des simulations numériques directes (DNS) d'écoulements turbulents tridimensionnels, le nombre de points de grille croît drastiquement avec le nombre de Reynolds, rendant la simulation séquentielle impossible et imposant le recours à des clusters parallèles (Bolis, Cantwell, Moxey, Serson & Sherwin, 2016). Les simulations des grands tourbillons (LES), souvent utilisées pour les écoulements géophysiques turbulents, nécessitent elles aussi un parallélisme massif (Zhang *et al.*, 2020). La

finesse de la grille conduit en outre à des pas de temps réduits, ce qui rend la résolution de problèmes transitoires de longue durée particulièrement coûteuse (Hanna *et al.*, 2020). Enfin, les géométries complexes des rivières, avec leurs obstacles et des fonds hétérogènes, accentuent encore ces exigences de calcul (Zhang *et al.*, 2020).

Il est donc nécessaire d'utiliser plusieurs processus travaillant chacun sur un sous-domaine du maillage, comme illustré ci-dessous.

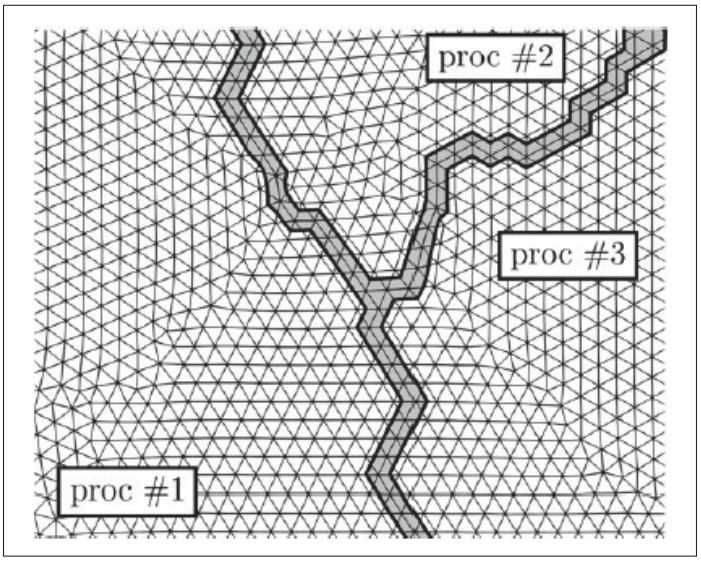


Figure 1.4 Exemple de décomposition de domaine en trois sous-domaines, chacun étant traité par un processus distinct (proc #1, proc #2 et proc #3). Cette stratégie permet de paralléliser les calculs sur un maillage non structuré. Voir Figure 1.6 pour plus de détails
Tirée par Gava (2022)

L'intégration de modèles et de fonctionnalités avancés dans les simulations amplifie également la charge de calcul. Les modèles non-hydrostatiques, qui prennent en compte des effets de dispersion essentiels pour des phénomènes tels que les tsunamis, sont plus complexes et plus intensifs que les modèles hydrostatiques (Ma *et al.*, 2012). L'utilisation de méthodes d'ordre élevé pour les équations de Navier-Stokes (Liu *et al.*, 2021), de schémas implicite-explicite

(IMEX) pour une meilleure stabilité et précision (Li *et al.*, 2025), ou encore des schémas WENO (*Weighted Essentially Non-Oscillatory*) (Cui & Zhu, 2022), augmente considérablement la demande en ressources.

Enfin, la parallélisation est cruciale pour exploiter pleinement les architectures matérielles modernes et optimiser l'efficacité des simulations. Les systèmes de calcul haute performance (HPC) contemporains sont conçus pour une intensité plus élevée, ce qui rend la parallélisation indispensable pour maximiser les gains de performance (Eichstädt, Vymazal, Moxey & Peiró, 2020). Elle permet des temps d'exécution réduits, une meilleure efficacité énergétique et une évolutivité accrue (Bolis *et al.*, 2016; Gander & Vandewalle, 2007). Des techniques d'optimisation, telles que la diagonalisation approximative des matrices (*lumping*), permettent de réduire le temps de calcul CPU d'environ 30% (Cui *et al.*, 2012), les techniques de *lumping* peuvent toutefois générer des diffusions d'erreurs numériques. L'utilisation de bibliothèques BLAS multithreadées (technique de parallélisation permettant à un processus unique d'exécuter simultanément plusieurs fils d'exécution (threads) au sein d'un espace mémoire partagé) et l'insertion de directives OpenMP dans les routines coûteuses, permettent de tirer parti des architectures multi-cœurs sans restructuration profonde du code (Chowdhury & L'Excellent, 2010). La parallélisation permet ainsi de dépasser les limitations de performance des codes purement multithreadés (Chowdhury & L'Excellent, 2010).

En somme, la parallélisation est essentielle pour gérer les gros volumes de calcul, avec des bibliothèques comme MPI pour la mémoire distribuée et OpenMP pour la mémoire partagée (Chowdhury & L'Excellent, 2010; Eijkhout, 2016; Bolis *et al.*, 2016; Eichstädt *et al.*, 2020). Elle est une condition fondamentale pour que les simulations d'écoulements CFD puissent relever les défis de la complexité des modèles, de la résolution des détails physiques et de l'efficacité computationnelle, permettant ainsi de réaliser des prédictions précises et opportunes dans un temps convenable.

Il convient également de souligner que les problématiques de haute résolution et d'efficacité de calcul ne sont pas propre à la CFD traditionnelle. Le domaine de l'infographie (*computer*

graphics) a développé des méthodes de simulation de fluides à très grande échelle dont la complexité égale ou dépasse souvent les standards industriels. Des approches basées sur des stratégies multigrilles hautes performances (McAdams *et al.*, 2011) ou des solveurs exploitant le complément de Schur, une technique d'algèbre linéaire permettant de réduire la taille du système global en éliminant un sous-ensemble d'inconnues pour se concentrer sur une interface réduite, sur des structures de données spatiales optimisées (Setaluri, Aanjaneya, Bauer & Fedkiw, 2014) permettent d'atteindre de très grands niveaux de détail. L'efficacité de ces méthodes repose fréquemment sur des algorithmes de distribution de charge et de parallélisation massive (Shah, Cohen, Tarini & Panozzo, 2018), ainsi que sur des techniques de lissage avancées pour accélérer la convergence, comme les approches de type Gauss-Seidel sensibles à la connectivité (Aanjaneya, Gao, Liu, Batty & Sifakis, 2017).

1.2.2 Bibliothèques et techniques pour la parallélisation des solveurs

Pour paralléliser les solveurs, diverses bibliothèques et techniques de programmation parallèle sont exploitées, visant à tirer parti des architectures multicœurs et distribuées. Le solveur direct creux MUMPS utilise la bibliothèque MPI pour le parallélisme (Chowdhury & L'Excellent, 2010). Le solveur MUMPS est développé en Fortran 90 et C, et repose sur les bibliothèques BLAS et ScaLAPACK pour les calculs sur matrices denses (Golub & Van Loan, 2013). L'archive ouverte pluridisciplinaire HAL est également mentionnée comme dépôt de publications scientifiques en lien avec ces développements (Chowdhury & L'Excellent, 2010).

Pour les problèmes de type Poisson, un solveur massivement parallèle comme AGMG utilise un gradient conjugué flexible avec un préconditionneur multigrille algébrique basé sur l'agrégation. Ce solveur fait appel à MUMPS (en version séquentielle ou parallèle) pour les systèmes linéaires au niveau le plus grossier. Le code CALU, optimisé pour minimiser les communications interprocesseurs, vise une factorisation LU efficace sur architectures multicœurs (Grigori *et al.*, 2011). Pour l'équation de Poisson 3D sur géométries non structurées, les méthodes SOR parallèles utilisent une décomposition de domaine horizontale par blocs et des échanges via MPI.

Sur maillages non structurés, le réordonnancement des grilles est essentiel pour les solveurs à mémoire partagée. Des stratégies sont proposées pour les solveurs centrés sur les arêtes, avec utilisation ciblée d'OpenMP et de méthodes de coloration multi-couleurs (Cheng, Wang & Mian, 2015).

La précision mixte peut être utilisée pour accélérer les solveurs linéaires denses tout en garantissant la fiabilité des résultats. Elle consiste à effectuer les opérations les plus coûteuses en simple précision, puis à appliquer des corrections en double précision par raffinement itératif (Buttari, Dongarra, Langou, Luszczek & Kurzak, 2007). Les bibliothèques LAPACK et BLAS fournissent les outils essentiels à cette approche.

D'un point de vue plus général, MPI et OpenMP restent les deux standards principaux de programmation parallèle (Eijkhout, 2016). Des bibliothèques comme CUDA ou NCCL sont utilisées pour tirer profit des GPU (Eichstädt *et al.*, 2020). Des outils comme hwloc servent à détecter la topologie des machines, et les modèles OpenACC ou Kokkos offrent une portabilité améliorée des performances (Eichstädt *et al.*, 2020).

1.2.3 Études de cas et enseignements issus de la littérature

Les études de cas et les benchmarks existants dans la littérature sur la parallélisation offrent des enseignements précieux sur les stratégies, les optimisations et les défis rencontrés.

L'une des leçons empiriques les plus importantes est que l'approche hybride, combinant des modèles de programmation tels que MPI (*Message Passing Interface*) pour le parallélisme distribué et OpenMP/*threads* pour le parallélisme partagé, est souvent plus performante que des implémentations purement MPI (Chowdhury & L'Excellent, 2010). Cette combinaison permet d'accroître le parallélisme et de réduire les temps de calcul sur des architectures matérielles modernes, notamment les machines à mémoire distribuée avec des architectures multicœurs, qui sont les plus couramment rencontrées (Chowdhury & L'Excellent, 2010; Bolis *et al.*, 2016).

Par exemple, le solveur MUMPS, un solveur direct creux, a montré que l'intégration de bibliothèques BLAS multithreadées et de directives OpenMP pouvait exploiter efficacement les architectures multicœurs, avec un bon compromis souvent atteint avec 4 threads par processus MPI (Chowdhury & L'Excellent, 2010). Pour des cas de problème très volumineux, 8 threads par processus MPI peuvent même offrir les meilleures performances (Chowdhury & L'Excellent, 2010).

Un enseignement crucial est la nécessité de minimiser la communication entre les processeurs sur les machines parallèles, car des communications excessives peuvent entraîner des pertes d'efficacité significatives (Grigori *et al.*, 2011). La localité des données est également d'une importance capitale pour les calculs matriciels à haute performance ; un accès distant aux données peut fortement dégrader l'efficacité (Golub & Van Loan, 2013).

Le parallélisme élémentaire, basé sur la décomposition de maillage, peut parfois mal s'adapter en raison d'un rapport communication/calcul élevé (Bolis *et al.*, 2016). Les messages MPI, par exemple, prennent quelques microsecondes, ce qui implique que la quantité de travail entre les messages doit être suffisamment importante pour compenser ce coût (Eijkhout, 2016).

Enfin, la conception des algorithmes doit être adaptée à la topologie de la machine et codée avec soin dans un langage proche de la machine comme Fortran ou C pour les architectures massivement parallèles, afin d'éviter des pertes d'efficacité importantes (Luckáč *et al.*, 1996). L'exploitation de la structure inhérente aux problèmes (par exemple, la symétrie) est un principe fondamental pour rationaliser les procédures de résolution (Golub & Van Loan, 2013). Concrètement, cela peut consister à tirer parti de la symétrie d'une matrice pour simplifier les calculs, ou de sa sparsité (parcimonie) si elle est creuse pour réduire le coût de stockage et de traitement. Pour les solveurs directs de systèmes linéaires creux, le pivotage dans l'élimination de Gauss peut détruire la structure exploitable et est une préoccupation majeure en calcul haute performance en raison du coût du déplacement des données (Golub & Van Loan, 2013).

Les méthodes de parallélisation temporelle, comme l'algorithme Parareal pour l'intégration temporelle, peuvent être considérées comme des implémentations pratiques de la méthode de tir

multiple (*multiple shooting*) ou des méthodes multigrilles temporelles (Gander & Vandewalle, 2007). Elles peuvent atteindre d'excellentes accélérations, bien que ce gain puisse être compensé par une augmentation du travail computationnel due à la propagation lente de l'information dans le temps (Bolis *et al.*, 2016).

L'utilisation d'outils de profilage (comme TAU Paraprof) est essentielle pour identifier les goulots d'étranglement et optimiser la parallélisation (Chowdhury & L'Excellent, 2010).

1.3 Présentation du cas d'étude

Ce travail s'inscrit dans la modélisation des écoulements tridimensionnels dans des milieux naturels tels que les rivières, estuaires ou zones côtières. L'objectif est de développer des modèles non-hydrostatiques performants, dans lesquels la résolution de l'équation de Poisson constitue l'étape la plus coûteuse en temps de calcul (Uh Zapata *et al.*, 2016).

1.3.1 Géométrie du domaine et transformation σ

Le domaine physique est tridimensionnel et présente une géométrie irrégulière, maillée par des triangles non structurés dans le plan horizontal. Les frontières verticales sont définies par un fond topographiquement complexe et une surface libre mobile. Pour traiter cette variabilité, une transformation verticale σ est utilisée, projetant la colonne d'eau $H = h + \eta$ sur un domaine vertical fixe (Uh Zapata, 2012; Uh Zapata *et al.*, 2016).

Cette transformation permet de simplifier le calcul numérique en éliminant le déplacement des mailles lié à la variation de la surface libre, tout en conservant la précision de la représentation géométrique.

1.3.2 Formulation mathématique

Le système physique est modélisé par un problème diphasique comprenant une phase fluide (f) et une phase solide (s) (Uh Zapata, 2012). Chaque phase satisfait les lois de conservation :

Équation de continuité transformée :

On note α_k la fraction volumique de la phase k (avec $k = f$ pour la phase fluide et $k = s$ pour la phase solide), et u_k, v_k, ω_k les trois composantes de vitesse associées. Dans le cas d'un liquide pur, on a $\alpha_f = 1$ et $\alpha_s = 0$. L'équation de continuité transformée s'écrit alors :

$$\frac{\partial(H\alpha_k)}{\partial t^*} + \frac{\partial(H\alpha_k u_k)}{\partial x^*} + \frac{\partial(H\alpha_k v_k)}{\partial y^*} + \frac{\partial(H\alpha_k \omega_k)}{\partial \sigma} = 0 \tag{1.3}$$

Équation de quantité de mouvement transformée :

On note ϕ une composante de la vitesse (u, v ou ω). Le terme H provient de la transformation en coordonnées σ, où la coordonnée verticale physique z est transformée en une coordonnée réduite σ suivant :

$$\sigma = \frac{z - \eta(x, y, t)}{H(x, y, t)}, \quad H(x, y, t) = \eta(x, y, t) + h(x, y), \tag{1.4}$$

avec $\eta(x, y, t)$ la surface libre et $h(x, y)$ la bathymétrie.

L'équation de quantité de mouvement transformée s'écrit alors :

$$\frac{\partial(H\alpha\phi)}{\partial t^*} + \frac{\partial(H\alpha u\phi)}{\partial x^*} + \frac{\partial(H\alpha v\phi)}{\partial y^*} + \frac{\partial(H\alpha\omega)\phi}{\partial \sigma} = HT(\text{source}_\phi) \tag{1.5}$$

Les termes considérés incluent : l'advection, la diffusion visqueuse, la turbulence, la pression dynamique (non hydrostatique), la gravité et les transferts de quantité de mouvement inter-phases.

1.3.3 Discrétisation numérique

Le domaine est discrétisé selon la méthode des volumes finis sur des cellules prismatiques non structurées (Uh Zapata *et al.*, 2016). La projection de Chorin est utilisée pour isoler l'équation

de Poisson à chaque pas de temps, pouvant être portée à un schéma temporel du second ordre grâce à Runge–Kutta (Chorin, 1968; Uh Zapata, 2012).

La matrice obtenue est creuse et non symétrique en raison de la transformation σ et de l'interpolation. Les flux d'advection sont traités par un schéma *upwind*, la diffusion par une approche semi-implicite avec découplage fluide/solide.

1.3.4 Conditions aux limites et approximations aux interfaces

Pour l'équation de Poisson, différentes conditions aux limites sont imposées selon la position dans le domaine (Uh Zapata *et al.*, 2016; Uh Zapata, 2012) :

Tableau 1.1 Conditions aux limites typiques pour l'équation de Poisson en coordonnées σ

Limite	**Conditions**
Fond ($\sigma = -1$)	Neumann sur la pression : $\dfrac{\partial p_k}{\partial \sigma} = -\rho g$, Vitesse nulle : $u = 0$
Surface libre ($\sigma = 0$)	Dirichlet sur la pression : $p_k = 0$, Condition cinématique : $\omega_k = \dfrac{\partial \eta}{\partial t^*} + u_k \dfrac{\partial \eta}{\partial x^*} + v_k \dfrac{\partial \eta}{\partial y^*}$

Les équations de conservation sont intégrées sur chaque volume de contrôle, et le théorème de Green-Ostrogradski est appliqué pour transformer les termes divergents en intégrales de surface sur les faces du volume de contrôle, qui représentent les interfaces entre mailles adjacentes.

Le calcul des flux advectifs à travers les faces d'interface (S_{e-j}) séparant deux volumes de contrôle (V_e et V_j) repose sur le produit d'un terme $H\alpha\phi$ évalué à l'interface et du flux de masse U_{e-j} traversant cette surface (Uh Zapata, 2012). Afin d'obtenir une précision de second ordre, la valeur d'une variable scalaire (ϕ ou ψ) sur l'interface X^*_{e-j} est estimée par une expansion de Taylor autour des centres de cellule. Cette approche s'appuie sur un modèle linéaire par

morceaux, reconstruit à partir des valeurs moyennes, où le gradient de la fonction à l'interface est déterminé par une méthode des moindres carrés utilisant les centres des mailles voisines horizontalement et verticalement (Uh Zapata, 2012).

Cette reconstruction est d'autant plus essentielle que le nombre de Péclet (Pe) est élevé. Ce nombre adimensionnel mesure le rapport entre le transport convectif et la diffusion :

$$Pe = \frac{UL}{\nu} \tag{1.6}$$

où U représente une vitesse caractéristique, L une longueur caractéristique et ν la viscosité cinématique ou le coefficient de diffusion. Dans les régimes où la convection domine largement ($Pe \gg 1$), un schéma insuffisamment précis peut engendrer des oscillations numériques indésirables. Ainsi, pour garantir la stabilité du calcul, l'approximation d'une variable ψ sur l'interface suit la formulation suivante :

$$\psi(X^*_{e-j}) \approx \psi(X_e) + \nabla\psi(X_e) \cdot (X^*_{e-j} - X_e) \tag{1.7}$$

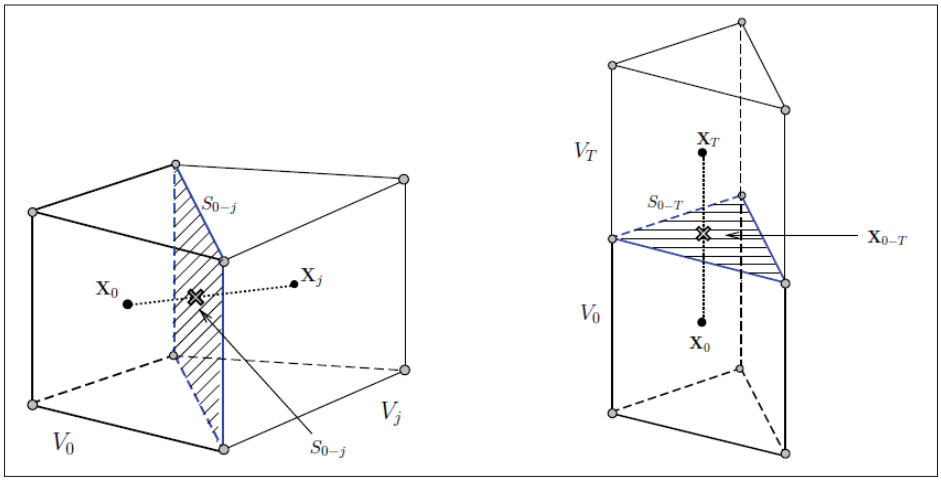


Figure 1.5 (a) Points horizontaux et (b) points verticaux situés à l'interface entre deux volumes de contrôle prismatiques
Tirée par Uh Zapata (2012)

La gestion des termes de diffusion constitue un autre enjeu majeur, nécessitant une approximation précise des dérivées partielles aux interfaces. La méthode employée consiste à effectuer une intégration sur un volume de contrôle artificiel construit autour de la face d'interface, en utilisant le gradient moyen calculé sur ce volume comme approximation (Uh Zapata, 2012). Les valeurs nécessaires aux sommets des triangles de base sont alors obtenues par une moyenne des valeurs nodales des cellules adjacentes.

1.3.5 Méthode d'accélération par diffusion parabolique (PD)

La méthode de diffusion parabolique (PD) consiste à transformer l'équation elliptique de Poisson en une équation parabolique artificielle en introduisant un terme de diffusion additionnel. Cette équation est ensuite intégrée dans un temps fictif τ jusqu'à convergence vers l'état stationnaire, qui correspond à la solution de l'équation initiale. (Uh Zapata *et al.*, 2016) :

$$\frac{\partial \phi}{\partial \tau} = A\phi + \mathbf{b_v} - f \tag{1.8}$$

Cette méthode améliore la convergence en limitant l'application répétée des conditions aux limites et des interpolations, contrairement au SOR classique.

1.3.6 Parallélisation du solveur

La parallélisation repose sur une décomposition horizontale du domaine entre plusieurs processeurs, utilisant une architecture à mémoire distribuée (MPI) (Uh Zapata *et al.*, 2016). Bien que le calcul local diminue avec plus de processeurs, le coût des communications reste constant, ce qui limite l'efficacité parallèle pour les petits problèmes. Les maillages sont générés avec BlueKenue et partitionnés par Chaco pour équilibrer la charge (Uh Zapata, Zhang, Pham Van Bang & Nguyen, 2019). Deux méthodes SOR parallèles sont testées :

- MCSOR : méthode de coloration multiple (jusqu'à 4 couleurs) (Uh Zapata *et al.*, 2016; Uh Zapata, Hernández-López & Itzá Balam, 2023) .

- PSOR : méthode partitionnée avec mise à jour JSOR sur les bords puis SOR central (Uh Zapata *et al.*, 2016).

Ces méthodes sont combinées à la PD et permettent d'être 36,7 fois plus rapide que la version séquentielle (Uh Zapata *et al.*, 2016).

Cette efficacité repose sur une stratégie de parallélisation rigoureuse qui débute par un découpage du domaine en sous-domaines de tailles équivalentes en nombre de cellules. Dans cette phase de décomposition initiale, la séparation est stricte : aucun chevauchement n'est autorisé au niveau des centres de cellules, bien que certains sommets puissent être partagés entre domaines adjacents (Uh Zapata *et al.*, 2016).

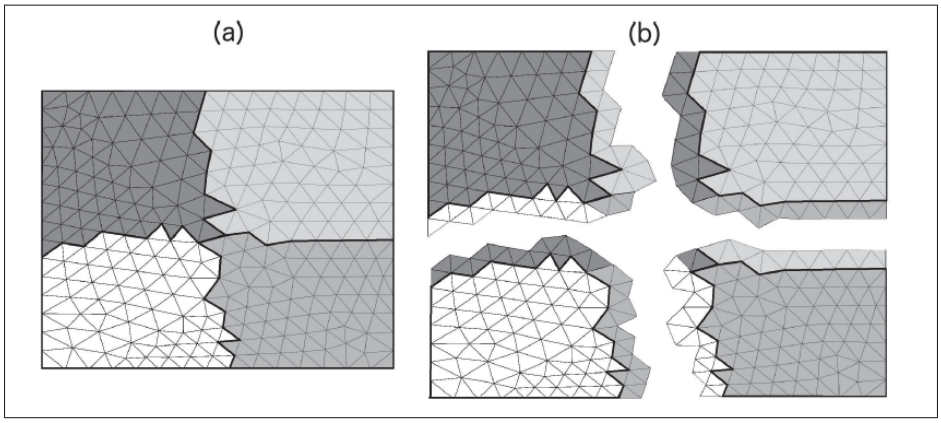


Figure 1.6 (a) Exemple de décomposition initiale du domaine horizontal en quatre sous-domaines disjoints. (b) Introduction de régions de chevauchement autour des frontières, permettant de prendre en compte toutes les inconnues nécessaires au schéma de volumes finis
Tirée par Uh Zapata *et al.* (2016)

Afin de satisfaire les exigences du schéma des volumes finis et de disposer de toutes les inconnues nécessaires localement, des régions de chevauchement sont introduites autour de chaque sous-domaine, tel qu'illustré sur la partie (b) de la Figure 1.6. Ces zones englobent les

éléments entourant les sommets partagés, réduisant la communication nécessaire(Uh Zapata *et al.*, 2016).

La synchronisation des variables sur ces frontières artificielles s'effectue par des échanges inter-processus à chaque itération des solveurs parallèles. Dans le cas spécifique de l'algorithme MCSOR, cette étape de communication est optimisée en parallélisant les échanges pour les éléments appartenant à une même couleur (Uh Zapata *et al.*, 2016). Cette approche garantit la cohérence des données sur l'ensemble du domaine tout en maximisant l'utilisation des ressources de calcul.

CHAPITRE 2

DÉVELOPPEMENT MÉTHODOLOGIQUE

2.1 Diagnostic des coûts de calcul

TAU (*Tuning and Analysis Utilities*) est une suite logicielle d'analyse de performances conçue pour les applications de calcul haute performance. Elle permet de combiner à la fois le profilage (mesures agrégées des temps passés dans chaque fonction ou phase de calcul) et le traçage (enregistrement temporel détaillé des événements lors de l'exécution). L'outil instrumente le code et collecte des métriques telles que le temps CPU, le temps d'attente dans les communications MPI, ou encore l'utilisation mémoire. Ces informations sont ensuite visualisables sous forme de profils ou de chronogrammes, ce qui permet d'identifier précisément les portions du programme responsables de la majorité du coût de calcul. Dans ce mémoire, nous utiliserons TAU afin de détecter ces phases critiques et de guider les stratégies d'optimisation de notre algorithme.

L'application de TAU sur le code de départ, basé sur l'algorithme multiSOR, met en évidence la répartition du temps d'exécution entre les différents nœuds de calcul. L'analyse des données de performance a été réalisée via l'interface de visualisation ParaProf, l'outil d'analyse graphique de la suite TAU. On observe notamment que les charges ne sont pas parfaitement équilibrées : certains processus passent plus de temps que d'autres dans les mêmes phases de calcul, ce qui induit des désynchronisations et limite l'efficacité parallèle globale. La figure 2.1 illustre cette première analyse de performance.

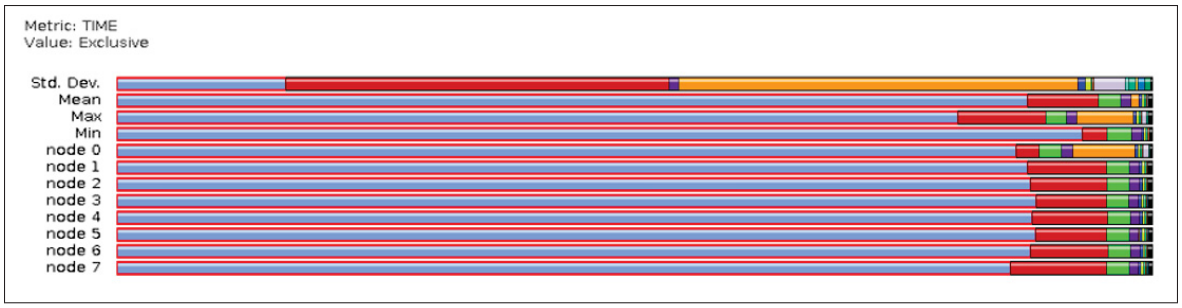


Figure 2.1 Analyse de performance du code NSMP initial utilisant l'algorithme multiSOR. Les mesures de temps sont réparties par nœud et permettent d'identifier les déséquilibres de charge ainsi que les phases les plus coûteuses en calcul

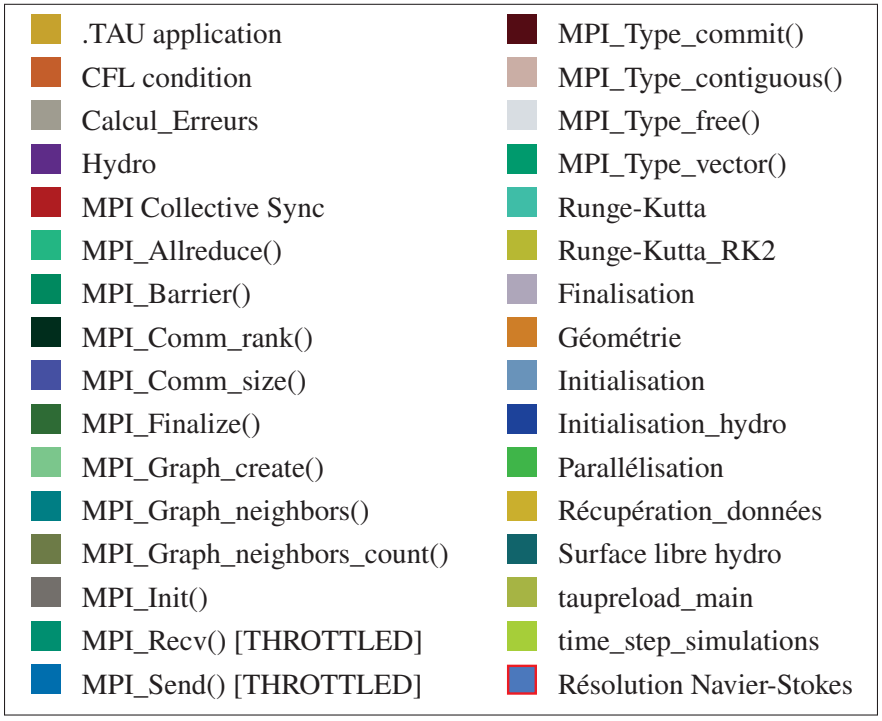


Figure 2.2 Légende des fonctions instrumentées par TAU

On constate que, comme attendu, la partie "Résolution Navier–Stokes", qui correspond à l'étape de prédiction de la vitesse ainsi que la résolution de l'équation de Poisson et de la correction du champ de vitesse, concentre la part la plus importante du temps d'exécution et constitue le cœur du calcul. En effet, cette partie représente 85% du temps total d'exécution en moyenne.

La résolution des équations de Navier-Stokes incompressibles dans le code NSMP repose sur une méthode à pas fractionnaire décomposée en trois étapes principales :

Étape de prédiction

On calcule d'abord un champ de vitesse intermédiaire $\mathbf{u}^*$ en résolvant les termes de convection et de diffusion sans contrainte d'incompressibilité :

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \nu \nabla^2 \mathbf{u}^n + \mathbf{f}^n \tag{2.1}$$

Équation de pression (Le système $A\mathbf{x} = \mathbf{b}$)

Pour projeter $\mathbf{u}^*$ sur un espace à divergence nulle, on résout l'équation de Poisson pour la pression p^{n+1} :

$$\nabla^2 p^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{2.2}$$

Après discrétisation sur le maillage non structuré, cette équation se transforme en un système linéaire algébrique :

$$A\mathbf{x} = \mathbf{b} \tag{2.3}$$

Où :

- $\mathbf{A}$ est la matrice de discrétisation du Laplacien.
- $\mathbf{x}$ est le vecteur des pressions inconnues p^{n+1} à déterminer.
- $\mathbf{b}$ est le terme source calculé à partir de la divergence de la vitesse intermédiaire $\mathbf{u}^*$.

C'est ce système spécifique que le solveur résout et c'est cette étape que l'on cherche à optimiser pour accélérer la convergence globale du code.

Étape de correction

Enfin, le champ de vitesse finale $\mathbf{u}^{n+1}$ est obtenu en corrigeant la vitesse intermédiaire par le gradient de la pression calculée :

$$\mathbf{u}^{n+1} = \mathbf{u}^{*} - \frac{\Delta t}{\rho} \nabla p^{n+1} \tag{2.4}$$

Description des variables :

$\mathbf{u}^{n}$: Vecteur vitesse au pas de temps actuel t.

$\mathbf{u}^{*}$: Vecteur vitesse intermédiaire (vitesse prédite).

$\mathbf{u}^{n+1}$: Vecteur vitesse corrigé au pas de temps futur $t + \Delta t$.

p^{n+1} : Pression scalaire au pas de temps futur $t + \Delta t$.

Δt : Pas de temps de la simulation.

ν : Viscosité cinématique du fluide.

ρ : Masse volumique du fluide.

$\mathbf{f}^{n}$: Terme source représentant les forces extérieures.

On observe que ce temps est significatif non seulement par sa durée totale, mais aussi par les déséquilibres entre processus : certains nœuds passent plus de temps que d'autres dans cette phase, en effet les hauteurs de barres varient nettement d'un rang MPI à l'autre, ce qui génère des attentes lors des synchronisations MPI. Dans cette perspective, nous chercherons à développer un algorithme de résolution plus performant, c'est à dire capable d'effectuer les calculs plus rapidement, en préservant une précision qui soit au moins du même ordre que dans l'algorithme existant, tout en optimisant la parallélisation des calculs.

La *Message Passing Interface* (MPI) est la norme de référence pour la programmation parallèle sur architectures à mémoire distribuée. Chaque processus dispose de sa propre mémoire et n'échange des informations qu'au moyen de messages explicites. On distingue les communications point à

point (MPI_Send, MPI_Recv) utilisées pour transmettre les valeurs aux processus voisins, et les communications collectives (MPI_Bcast, MPI_Allreduce, etc.) qui assurent une synchronisation ou une réduction globale. Dans un solveur CFD, ces mécanismes sont indispensables : ils servent à échanger les valeurs situées sur les interfaces entre sous-domaines, à sommer des résidus ou encore à imposer des conditions de convergence. L'utilisation de MPI est ainsi très intéressante car une fois le maillage découpé en plusieurs blocs, chacun est traité par un processus et la cohérence numérique de la solution est maintenue.

Description de la plateforme de calcul

Dans le cadre de cette étude, Narval, une plateforme hétérogène et polyvalente de Calcul Québec, a joué un rôle central en fournissant une infrastructure de calcul haute performance (HPC) adaptée aux exigences du projet. Narval, située à l'École de Technologie Supérieure, est conçue pour une grande variété de calculs scientifiques.

Les principales caractéristiques matérielles de Narval sont les suivantes :

- Nombre de nœuds de calcul : Narval dispose de 807 nœuds de calcul CPU et de 161 nœuds GPU, interconnectés par un réseau InfiniBand HDR à 200 Gbit/s. La topologie réseau permet une communication rapide entre nœuds, notamment avec des îlots de 32 nœuds pouvant communiquer sans blocage.
- Processeurs CPU : Les nœuds de calcul sont équipés de processeurs AMD EPYC 7763 (Zen 3) à 64 cœurs par processeur (soit 128 cœurs par nœud). Ces unités affichent une fréquence de base de 2,45 GHz.
- Cartes graphiques (GPU) : Les nœuds GPU sont dotés de cartes graphiques NVIDIA A100 (40 Go de mémoire). Ces GPU peuvent être utilisés en mode complet ou partagés grâce à la technologie MIG (Multi-Instance GPU).
- Mémoire : La majorité des nœuds CPU disposent de 512 Go de mémoire vive, tandis que certains nœuds dits "large mémoire" atteignent 1 To de RAM.
- Stockage : Narval utilise un système de fichiers Lustre haute performance comprenant :

  - HOME : Espace dédié aux scripts et codes sources, avec sauvegarde automatique.
  - SCRATCH : Environ 6,5 Po pour les fichiers temporaires de calcul (sans sauvegarde, avec purge automatique des fichiers anciens).
  - PROJECT : Espace de stockage partagé pour les données à long terme liées au projet, avec sauvegarde régulière.
- Évolutivité et interconnexion : L'infrastructure est hautement évolutive. Chaque nœud offre jusqu'à 128 cœurs physiques, et l'interconnexion rapide permet d'exploiter efficacement plusieurs nœuds simultanément pour des tâches MPI.

Pour ce projet, Narval a été utilisé pour exécuter des simulations parallèles sur plusieurs nœuds, exploitant la densité de cœurs des processeurs AMD. Grâce à la puissance de calcul et à la flexibilité offertes par cette plateforme, il a été possible de réaliser des simulations de grande envergure dans des délais optimisés.

2.2 Analyse numérique des matrices

Dans un premier temps, le code a été exécuté avec l'algorithme Multicolor SOR existant, en utilisant 32 processus parallèles (MPI). Le calcul modélise l'interaction entre un écoulement incident et un pilier de pont, afin d'analyser le comportement du fluide. L'objectif était d'observer la structure et les propriétés numériques des matrices locales générées par chaque processus au cours de la résolution. Pour chaque rang MPI, la matrice associée au sous-domaine a été extraite pour chaque itération de l'algorithme. La Figure 2.3 illustre la répartition des matrices pour une exécution sur 8 rangs MPI. Chaque matrice locale A_i correspond à la discrétisation de l'opérateur Laplacien restreinte au sous-domaine physique attribué au processeur i.

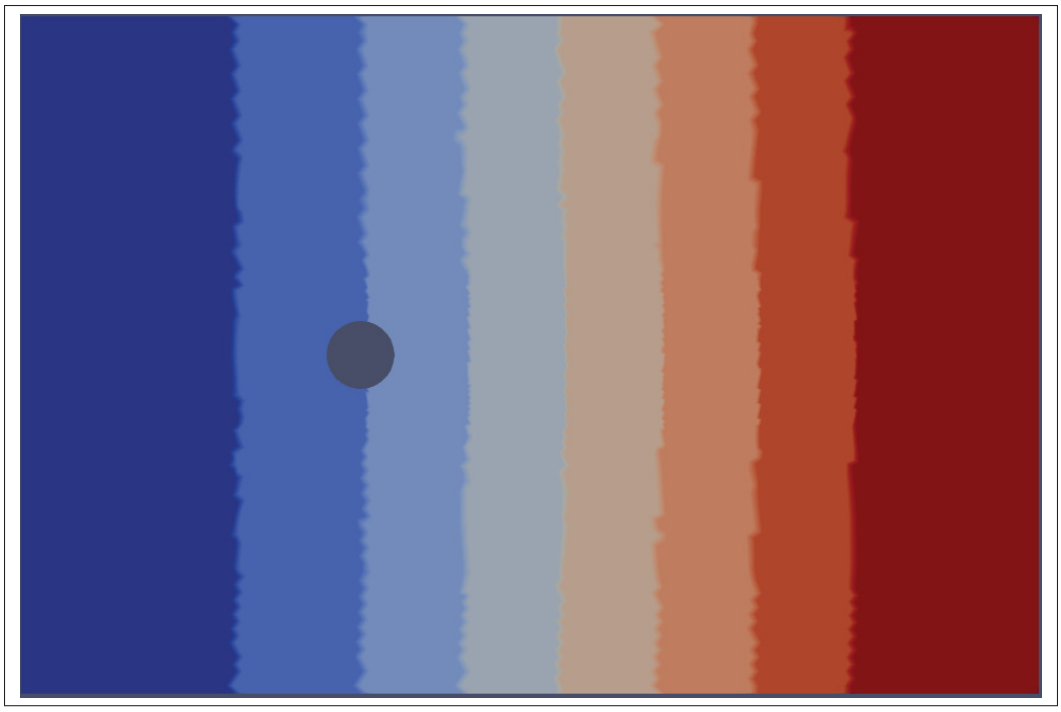


Figure 2.3 Décomposition en 8 sous-domaines MPI, du rang 0 à 7

L'examen de ces matrices a montré qu'elles demeurent identiques d'une itération à l'autre. Autrement dit, une fois la discrétisation fixée et la répartition des données effectuée entre processus, les matrices locales restent inchangées au cours du calcul. Cette invariance permet d'effectuer une analyse approfondie sur un échantillon représentatif, sans devoir répéter les extractions à chaque étape temporelle.

Afin de mieux comprendre les caractéristiques numériques de ces matrices, nous avons ensuite calculé leur conditionnement. C'est un indicateur clé car il mesure la sensibilité de la solution aux perturbations des données et influe directement sur la rapidité et la stabilité des méthodes itératives. Des valeurs élevées traduisent en général des difficultés accrues de convergence, tandis que des valeurs plus modérées témoignent d'un système mieux conditionné (voir Annexe I pour plus de détails). La mesure du conditionnement permet d'évaluer l'efficacité du schéma de résolution choisi, de quantifier la difficulté intrinsèque du problème linéaire associé au code,

lequel résout les equations de Navier-Stokes. discrétisées par volumes finis. Elle offre en outre une base pour évaluer l'apport de futurs préconditionneurs ou de méthodes alternatives.

Les résultats de cette étude sont présentés sous deux formes complémentaires. La figure 2.4 est un graphique du conditionnement, où l'axe des abscisses correspond au numéro du processus MPI et l'axe des ordonnées à la valeur du conditionnement de la matrice correspondante. Ce graphique révèle les disparités entre sous-domaines : certains processus manipulent des matrices mieux ou moins bien conditionnées que d'autres, ce qui peut avoir une incidence directe sur la vitesse de convergence et sur le déséquilibre de charge dans le solveur parallèle. La figure 2.5 est une bitmap illustrant la structure creuse d'une de ces matrices, qui met en évidence la répartition des coefficients non nuls et confirme la nature *sparse* du système, ce qu'on peut voir plus clairement sur la figure 2.6. Une bitmap est une représentation graphique d'une matrice où chaque coefficient est traduit en pixel, permettant de visualiser la répartition des zéros et des non-zéros sans tenir compte de leur valeur numérique.

Dans l'ensemble, ces analyses constituent une étape essentielle pour orienter les futures optimisations, que ce soit dans l'introduction de préconditionneurs adaptés ou dans l'amélioration du schéma de résolution.

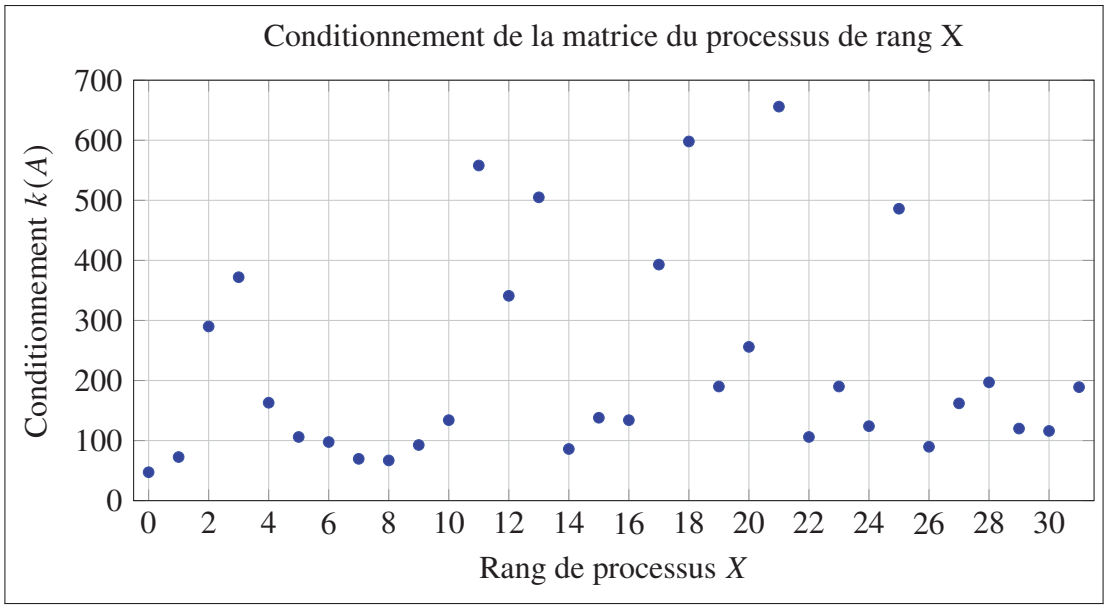


Figure 2.4 Conditionnement de la matrice associée à chaque processus MPI (rang allant de 0 à 31). Les variations de conditionnement selon le rang MPI traduisent des différences de propriétés numériques entre sous-domaines

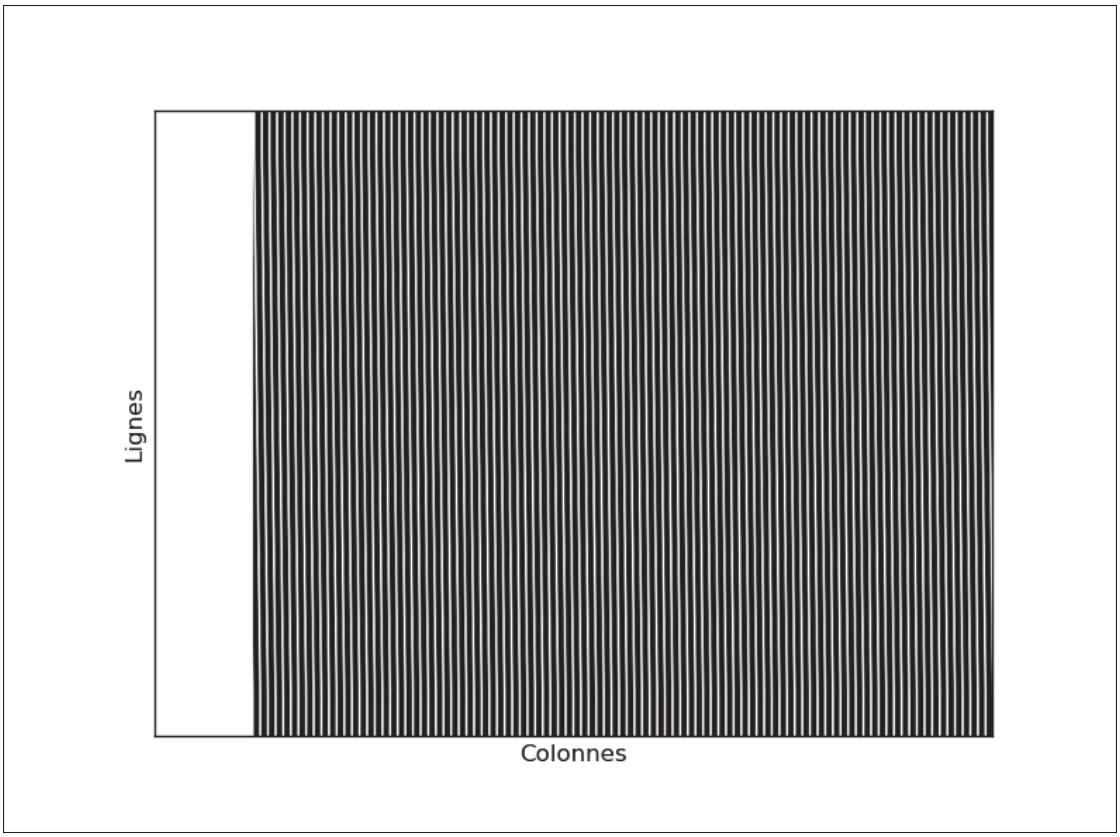


Figure 2.5 Bitmap de la matrice du processus de rang 10, les zéros étant représentés en noir. Cette visualisation illustre la structure creuse et régulière de la matrice issue de la discrétisation par volumes finis

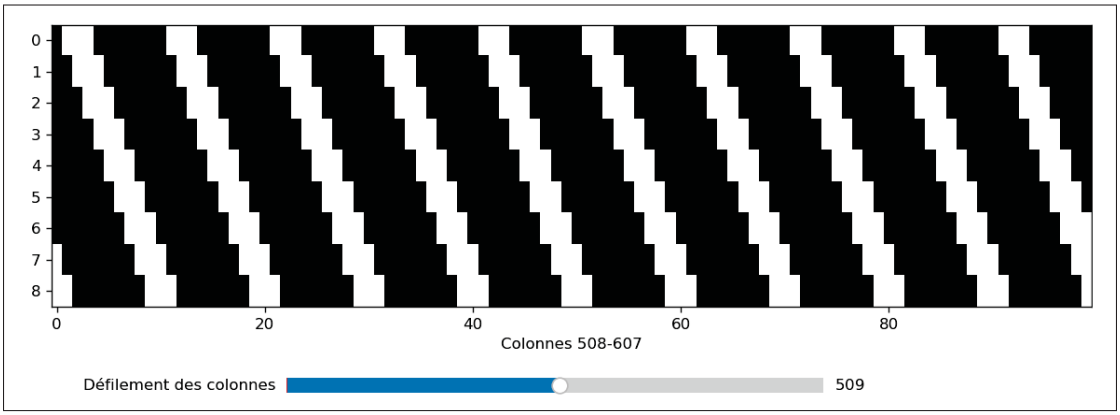


Figure 2.6 Vue à l'échelle de la matrice du processus de rang 10, seules certaines colonnes sont visibles (ici 508 à 607) car la matrice est rectangulaire telle que $n_{\text{lignes}} \ll n_{\text{colonnes}}$. Cette représentation binaire (zéros en noir) met en évidence la structure régulière des bandes non nulles

La matrice locale $\mathbf{A}_i$ associée à chaque processeur présente une structure creuse et rectangulaire. Ses dimensions sont définies par $(N_{\text{cell0},i} \times N_Z) \times (N_{\text{total},i} \times N_Z)$, où $N_{\text{cell0},i}$ est le nombre de cellules intérieures (locales) et N_Z le nombre de niveaux verticaux.

Le nombre total de cellules physiques uniques dans le domaine global est donné par la somme des cellules intérieures de chaque sous domaine :

$$N_{\text{global}} = \sum_i N_{\text{cell0},i} \tag{2.5}$$

En revanche, la mémoire allouée localement sur chaque processeur doit tenir compte des cellules fantômes (N_{CELLF_i}), qui sont des copies de cellules appartenant à des processeurs voisins. Bien que ces cellules soient nécessaires, elles ne sont pas comptabilisées dans la taille du problème global pour éviter tout double comptage :

$$N_{\text{mémoire\_local},i} = (N_{\text{cell0},i} + N_{\text{CELLF},i}) \times N_Z \tag{2.6}$$

Ces cellules permettent de gérer les échanges de données entre les frontières des sous-domaines locaux et sont essentielles pour le calcul des dérivées ou des opérations qui impliquent des voisins directs.

Cette structure imposée par les dimensions de A présente à la fois des avantages et des inconvénients pour la résolution du système. D'une part, l'aspect creux de la matrice permet une gestion plus efficace de la mémoire et des calculs en exploitant les zones non nulles. D'autre part, la forme rectangulaire de la matrice pose un défi pour l'utilisation de certains solveurs classiques. Pour rendre la matrice carrée et permettre l'application de solveurs standards, il existe des approches comme la multiplication de A^T et A (formant ainsi une matrice carrée symétrique), ou AA^T dans le cas d'un autre type de transformation. Cependant, dans le cas où l'on utilise AA^T, il faut résoudre le système $AA^T y = b$, puis calculer $x = A^T y$. Cette approche est moins courante car elle nécessite deux étapes successives, ce qui peut augmenter considérablement le temps de calcul.

2.3 Méthodes numériques de résolution existantes

Afin de déterminer la stratégie de résolution la plus adaptée à notre système, plusieurs algorithmes ont été mis mis à l'essai dont les avantages et inconvénients sont détaillés ci-après, justifiant ainsi le choix final.

2.3.1 Méthode de Jacobi, Gauss-Seidel, SOR et MultiSOR

Méthode de Jacobi

La méthode de Jacobi est une méthode itérative utilisée pour résoudre des systèmes linéaires du type $Ax = b$. Elle repose sur l'itération suivante :

$$x_i^{(k+1)} = \frac{1}{A_{ii}} \left(b_i - \sum_{j \neq i} A_{ij} x_j^{(k)} \right), \tag{2.7}$$

où $x^{(k)}$ représente l'approximation de la solution au k-ième pas, et A_{ii} est l'élément diagonal de la matrice A.

Cette méthode est simple à implémenter, mais elle converge lentement si la matrice est mal conditionnée.

Algorithme 2.1 Méthode de Jacobi

Algorithme : Méthode de Jacobi

Input : Matrice $A \in \mathbb{R}^{n\times n}$ (carrée), vecteur $b \in \mathbb{R}^n$, tolérance $\varepsilon > 0$

Output : Vecteur x approximant la solution de $Ax = b$

Initialiser $x^{(0)} \leftarrow 0$, $k \leftarrow 0$;

repeat

  for $i = 1$ **to** n **do**

    $x_i^{(k+1)} \leftarrow \frac{1}{A_{ii}} \left(b_i - \sum_{j\neq i} A_{ij} x_j^{(k)} \right)$;

  end for

  $k \leftarrow k + 1$;

until $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$;

return $x^{(k)}$;

Méthode de Gauss–Seidel

La méthode de Gauss–Seidel est une amélioration naturelle de la méthode de Jacobi pour la résolution de systèmes linéaires du type $Ax = b$. Elle repose sur une mise à jour séquentielle des composantes de x, utilisant immédiatement les nouvelles valeurs dès qu'elles sont disponibles dans une itération.

Principe :

On décompose la matrice A en :

$$A = D + L + U, \tag{2.8}$$

où D est la diagonale, L la partie strictement inférieure et U la partie strictement supérieure. L'itération de Gauss–Seidel s'écrit :

$$x_i^{(k+1)} = \frac{1}{A_{ii}} \left(b_i - \sum_{j<i} A_{ij} x_j^{(k+1)} - \sum_{j>i} A_{ij} x_j^{(k)} \right) \tag{2.9}$$

Contrairement à la méthode de Jacobi, ici chaque nouvelle composante $x_i^{(k+1)}$ est calculée en utilisant immédiatement les dernières mises à jour des x_j déjà calculés au cours de l'itération $k+1$. Cette méthode est moins facile à paralléliser que Jacobi car les mises à jour sont séquentielles. La méthode de Gauss–Seidel converge généralement plus rapidement que Jacobi, surtout lorsque la matrice A est :

- diagonale dominante,
- ou symétrique définie positive.

Méthode SOR (Successive Over-Relaxation)

La méthode SOR (*Successive Over-Relaxation*) est une amélioration de la méthode de Gauss-Seidel, conçue pour accélérer la convergence lors de la résolution d'un système linéaire $Ax = b$. Elle introduit un facteur de relaxation $\omega > 0$, permettant d'avancer plus vite (ou plus prudemment) dans la direction de la correction.

Principe :

On écrit la matrice A comme la somme de ses composantes :

$$A = D + L + U, \tag{2.10}$$

où :

- D est la diagonale de A,
- L est la partie strictement inférieure (coefficients en-dessous de la diagonale),
- U est la partie strictement supérieure (au-dessus de la diagonale).

L'itération SOR s'écrit alors :

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{A_{ii}}\left(b_i - \sum_{j<i} A_{ij}x_j^{(k+1)} - \sum_{j>i} A_{ij}x_j^{(k)}\right) \tag{2.11}$$

Autrement dit, on applique une itération de Gauss-Seidel, mais pondérée par ω. Ce facteur peut être :

- $\omega = 1$: on retrouve la méthode de Gauss-Seidel classique,
- $0 < \omega < 1$: relaxation sous-critique (plus stable mais plus lente),
- $1 < \omega < 2$: relaxation sur-critique (accélération de la convergence, si bien choisie).

Le choix optimal de ω dépend de la matrice A. Pour des matrices symétriques définies positives, il existe des bornes théoriques pour choisir un ω. En pratique, on utilise souvent $\omega \in [1; 1,9]$, ce qui donne de bons résultats pour de nombreux systèmes issus de la discrétisation d'équations aux dérivées partielles. Cette méthode améliore significativement la vitesse de convergence par rapport à Jacobi ou Gauss-Seidel.

Méthode Multi-Color SOR (parallélisée)

Une limitation majeure de la méthode SOR classique réside dans sa nature séquentielle : chaque mise à jour dépend des valeurs précédemment actualisées dans la même itération. Cela empêche une parallélisation directe. Pour pallier ce problème, la méthode Multi-Color SOR applique une technique de coloration de graphe sur la matrice du système linéaire afin de regrouper les inconnues indépendantes.

L'idée consiste à :

- associer à chaque nœud du système une couleur de telle sorte que deux nœuds adjacents n'aient jamais la même couleur,
- effectuer les mises à jour en parallèle pour tous les nœuds d'une même couleur,
- balayer l'ensemble des couleurs successivement à chaque itération.

Cette approche est particulièrement adaptée aux maillages non structurés, comme ceux issus de la discrétisation des équations de Navier–Stokes en 3D. La méthode est utilisée notamment dans des implémentations sur GPU, car elle permet d'exploiter efficacement le parallélisme des architectures modernes (Uh Zapata *et al.*, 2023).

Elle conserve la structure de SOR mais autorise une exécution concurrente des mises à jour, ce qui accélère considérablement la convergence dans des contextes de calcul intensif. Toutefois, comme pour toutes les méthodes itératives, la résolution de l'équation de Poisson y demeure un réel goulot d'étranglement.

2.3.2 Méthode BiCGStab

La méthode BiCGStab (*BiConjugate Gradient Stabilized*) est un algorithme itératif conçu pour résoudre les systèmes linéaires de la forme :

$$Ax = b, \tag{2.12}$$

où la matrice $A \in \mathbb{R}^{n\times n}$ est éventuellement non symétrique et non définie positive.

Principe

BiCGStab est une amélioration de la méthode *BiConjugate Gradient (BiCG)*, qui vise à stabiliser le comportement parfois oscillatoire de cette dernière. Elle combine deux idées :

- L'approche du gradient biconjugué pour gérer les matrices non symétriques.
- Une étape de stabilisation inspirée de la méthode GMRES (voir partie 2.3.6) pour améliorer la convergence.

Soit x_0 une approximation initiale. On pose $r_0 = b - Ax_0$ et choisit un vecteur $\tilde{r}_0$ tel que $\langle \tilde{r}_0, r_0 \rangle \neq 0$. Nous noterons dès à présent $\langle u, v \rangle$ le produit scalaire euclidien standard entre deux vecteurs u et $v \in \mathbb{R}^n$, et $\|u\| = \sqrt{\langle u, u \rangle}$ sa norme associée.

Ensuite, on génère les séquences (x_k), (r_k) et (p_k) par l'algorithme suivant :

Algorithme 2.2 Méthode BiCGSTAB

Algorithme : Méthode BiCGSTAB
Input : Matrice $A \in \mathbb{R}^{n\times n}$, vecteur $b \in \mathbb{R}^n$, approximation initiale x_0, tolérance $\varepsilon > 0$
Output : Vecteur x approchant la solution de $Ax = b$

$r_0 \leftarrow b - Ax_0$;
$\tilde{r}_0 \leftarrow r_0$;
$p_0 \leftarrow r_0$;
$k \leftarrow 0$;
repeat
  $\alpha_k \leftarrow \dfrac{\langle r_k, \tilde{r}_0\rangle}{\langle Ap_k, \tilde{r}_0\rangle}$;
  $s_k \leftarrow r_k - \alpha_k Ap_k$;
  $\omega_k \leftarrow \dfrac{\langle As_k, s_k\rangle}{\langle As_k, As_k\rangle}$;
  $x_{k+1} \leftarrow x_k + \alpha_k p_k + \omega_k s_k$;
  $r_{k+1} \leftarrow s_k - \omega_k As_k$;
  $\beta_k \leftarrow \dfrac{\langle r_{k+1}, \tilde{r}_0\rangle}{\langle r_k, \tilde{r}_0\rangle} \cdot \dfrac{\alpha_k}{\omega_k}$;
  $p_{k+1} \leftarrow r_{k+1} + \beta_k(p_k - \omega_k Ap_k)$;
  $k \leftarrow k + 1$;
until $\|r_k\| < \varepsilon$;
return x_k;

Caractéristiques

- Pas besoin de transposée : contrairement à BiCG, BiCGStab ne nécessite pas l'application de $A^\top$, ce qui simplifie le calcul et le rend plus adapté à certains systèmes pour lesquels la transposée n'est pas explicitement disponible ou coûteuse à manipuler.
- Stabilité accrue : la méthode BiCG présente parfois un comportement instable avec des résidus oscillants ou divergents, notamment lorsque les valeurs propres de A sont complexes ou mal réparties. BiCGStab corrige cela grâce à une étape de stabilisation :

$$\omega_k = \frac{\langle As_k, s_k\rangle}{\langle As_k, As_k\rangle}, \tag{2.13}$$

Cette étape réduit directement la norme du résidu, ce qui amortit les oscillations potentielles du résidu produites par BiCG. Autrement dit, on effectue une correction supplémentaire dans la direction As_k pour stabiliser la convergence.

- Efficacité : la méthode BiCGStab combine la faible complexité mémoire de BiCG avec une convergence plus fluide mais elle peut tout de même présenter des instabilités ou des problèmes de divergence critiques en environnement distribué.

2.3.3 Décomposition QR

La décomposition QR d'une matrice $A \in \mathbb{R}^{m\times n}$ avec m lignes et n colonnes ($m \geq n$) s'écrit

$$A = QR, \tag{2.14}$$

où $Q \in \mathbb{R}^{m\times m}$ est une matrice orthogonale (et ses colonnes sont orthonormées) et $R \in \mathbb{R}^{m\times n}$ est triangulaire supérieure (les lignes en trop peuvent être nulles si $m > n$). Dans la pratique, on ne stocke souvent que les n premières colonnes de Q et les n lignes de R.

L'utilisation de la procédure de Gram-Schmidt modifiée est préférable à la version classique (Gram-Schimdt standard) pour des raisons de stabilité numérique (Golub & Van Loan, 2013). En effet, la version classique souffre d'une accumulation d'erreurs d'arrondi, en particulier lorsque les vecteurs colonnes de A ont leur produit scalaire normalisé proche de l'unité (par exemple, pour $u, v \in \mathbb{R}^n$, $\frac{|\langle u,v\rangle|}{\|u\|\|v\|} \approx 1$). La version modifiée améliore cette stabilité en projetant et en ré-orthonormalisant les vecteurs un par un, en mettant à jour immédiatement le vecteur en cours. Ainsi, elle permet d'obtenir une matrice Q dont les colonnes conservent une orthogonalité numérique beaucoup plus satisfaisante.

Algorithme 2.3 Décomposition QR par Gram-Schmidt modifié

Algorithme : Décomposition QR par Gram-Schmidt modifié

Input : $A \in \mathbb{R}^{m \times n}$

Output : Matrice orthogonale $Q \in \mathbb{R}^{m \times m}$, matrice triangulaire supérieure $R \in \mathbb{R}^{m \times n}$

for $j = 1$ **to** n **do**
    $\mathbf{v}_j \leftarrow$ colonne j de A;
    for $i = 1$ **to** $j - 1$ **do**
        $R_{i,j} \leftarrow \mathbf{q}_i^\top \mathbf{v}_j$;
        $\mathbf{v}_j \leftarrow \mathbf{v}_j - R_{i,j} \cdot \mathbf{q}_i$;
    end for
    $R_{j,j} \leftarrow \|\mathbf{v}_j\|_2$;
    $\mathbf{q}_j \leftarrow \mathbf{v}_j / R_{j,j}$;
end for

Q est formée des vecteurs colonnes q_j et R contient les coefficients $R_{i,j}$. La décomposition QR par Gram-Schmidt modifié assure une excellente stabilité numérique, au prix d'un coût de calcul plus élevé que les méthodes itératives.

2.3.4 Décompositions LU et ILU(k)

Décomposition LU avec pivot

La décomposition LU avec pivot consiste à factoriser une matrice A de taille $n \times n$ où L est une matrice triangulaire inférieure et U une matrice triangulaire supérieure, et P est une matrice de permutation telle que :

$$PA = LU. \tag{2.15}$$

La permutation permet d'échanger les lignes (ou les colonnes) de la matrice A afin de maximiser les éléments sur la diagonale et améliorer la stabilité numérique. Le principe de la décomposition

LU avec pivot partiel est de permuter les lignes de A de manière à rendre les éléments pivots (sur la diagonale) plus grands en valeur absolue, ce qui aide à éviter l'instabilité numérique, surtout pour les matrices mal conditionnées (Golub & Van Loan, 2013). On distingue le pivot partiel, où seules les lignes sont échangées (en choisissant le plus grand élément de la colonne courante comme pivot), du pivot complet, où lignes et colonnes peuvent être permutées.

Algorithme 2.4 Décomposition LU avec pivot partiel

Algorithme : Décomposition LU avec pivot partiel

Input : Matrice $A \in \mathbb{R}^{n \times n}$

Output : Matrices L, U et matrice de permutation P telles que $PA = LU$

Initialiser $P \leftarrow I_n$ (matrice identité), $U \leftarrow A$, $L \leftarrow 0_{n \times n}$;
for $k = 1$ **to** $n - 1$ **do**
    Trouver l'indice p du pivot tel que : $|A_{pk}| = \max_{i=k,\dots,n} |A_{ik}|$;
    if $p \neq k$ **then**
        Permuter les lignes k et p dans A et dans P;
    end if
    for $i = k + 1$ **to** n **do**
        $L_{ik} \leftarrow \dfrac{A_{ik}}{A_{kk}}$;
        for $j = k$ **to** n **do**
            $A_{ij} \leftarrow A_{ij} - L_{ik} \cdot A_{kj}$;
        end for
    end for
end for
Placer les éléments de A au-dessus (et sur) la diagonale dans U ; et ceux en dessous dans L (avec $L_{ii} \leftarrow 1$);
return L, U, P;

De cette façon, on obtient la factorisation $PA = LU$, où L est une matrice triangulaire inférieure et U est une matrice triangulaire supérieure, et P est la matrice de permutation.

Décomposition ILU(k)

La décomposition incomplète LU de niveau k, notée ILU(k), consiste à approximer la factorisation LU d'une matrice A, tout en contrôlant le phénomène de remplissage, c'est-à-dire l'apparition de nouveaux coefficients non nuls dans L et U à des positions qui étaient nulles dans A :

- ILU(0) : aucun remplissage supplémentaire n'est autorisé. Les matrices L et U conservent exactement la structure creuse de A. C'est la version la plus rapide, mais souvent la moins précise.
- ILU(k) : autorise un remplissage de niveau k. Lors de la factorisation, chaque nouveau terme généré est associé à un niveau de remplissage ; si ce niveau est inférieur ou égal à k, le terme est conservé, sinon il est annulé.

Plus k est grand, plus ILU(k) se rapproche de la factorisation LU complète. En contrepartie, le coût en mémoire et en calcul augmente. Le choix de k dépend donc d'un compromis entre performance et qualité du préconditionneur ILU. Théoriquement, ILU($n-1$) coïncide avec la factorisation LU complète, mais en pratique on se limite à de petits ordres k (souvent 0, 1 ou 2) afin de préserver la structure creuse du préconditionneur et de réduire les coûts de calcul.

Algorithme 2.5 Factorisation incomplète ILU(0)

Algorithme : Factorisation incomplète ILU(0)

Input : Matrice creuse $A \in \mathbb{R}^{n \times n}$

Output : Matrices L_{ILU} et U_{ILU} telles que $A \approx L_{\text{ILU}} U_{\text{ILU}}$

Initialiser L_{ILU} et U_{ILU} avec la même structure creuse que respectivement les parties triangulaires inférieure et supérieure de A;
for *chaque ligne i de A* **do**
    Effectuer la factorisation LU classique sur la ligne i;
    for *chaque élément* $a_{i,j}$ *de la ligne* **do**
        if *la position* (i, j) *n'appartient pas à la structure creuse de A* **then**
            Mettre $a_{i,j} \leftarrow 0$;
        end if
        else
            Calculer $l_{i,j}$ ou $u_{i,j}$ normalement, en utilisant uniquement les éléments disponibles dans la structure;
        end if
    end for
end for
return $L_{\text{ILU}}, U_{\text{ILU}}$;

L'application de LU à un système rectangulaire impose de passer par le calcul de ($A^\top A$ ou $AA^\top$), une transformation dont le coût peut devenir prohibitif.

2.3.5 L'algorithme CALU (Communication Avoiding LU)

La méthode CALU (Communication-Avoiding LU) est une variante optimisée de la factorisation LU avec pivot partiel, conçue pour minimiser les échanges de données en environnement parallèle tout en maintenant une stabilité numérique proche de celle du pivot partiel classique (Grigori

et al., 2011). Elle repose sur une stratégie de tournament pivoting, qui effectue une sélection hiérarchique des pivots à travers une réduction arborescente.

Principe de la factorisation CALU

On considère un système linéaire à résoudre :

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n} \tag{2.16}$$

où A est la matrice issue des équations du système. La factorisation CALU cherche une décomposition de la forme :

$$PA = LU \tag{2.17}$$

où P est une matrice de permutation, L est triangulaire inférieure et U triangulaire supérieure. Contrairement à la méthode LU classique, CALU construit P en plusieurs niveaux par une stratégie de tournoi :

- À chaque étape, les candidats pivots sont sélectionnés localement dans des blocs sous-matriciels.
- Ces candidats sont ensuite comparés dans un arbre de réduction (*tournament tree*) jusqu'à obtention du pivot global.

La sélection hiérarchique des pivots peut être formellement décrite par :

$$P^{(k)} A^{(k)} = L^{(k)} U^{(k)} \tag{2.18}$$

avec $P = P^{(1)} \cdots P^{(s)}$ pour s niveaux de réduction, et $A^{(k)}$ la matrice mise à jour à l'étape k.

Formulation matricielle par blocs

À chaque étape, la matrice A est décomposée en blocs :

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \tag{2.19}$$

CALU applique la factorisation LU au bloc A_{11} (pivoté via tournoi), puis utilise ce facteur pour mettre à jour les blocs suivants. On obtient la factorisation :

$$PA = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & S \end{bmatrix} \quad \text{où} \quad S = A_{22} - L_{21}U_{12}. \tag{2.20}$$

Ce terme S, appelé complément de Schur, correspond à la mise à jour de la partie restante de la matrice après l'élimination de la première ligne et colonne de blocs. Ainsi, à chaque étape, on "retire" une partie de la matrice pour réduire la taille du problème, en actualisant $A \leftarrow S$.

Parallélisation et performance

Cette structure en blocs est naturellement adaptée à une exécution parallèle. En effet :

- La sélection de pivots locaux dans les sous-blocs peut être faite en parallèle.
- Les produits blocs $L_{21}U_{12}$ peuvent être calculés indépendamment sur des cœurs différents.
- Le calcul du complément de Schur S est un produit matrice-bloc, donc massivement parallélisable. Ces blocs peuvent être traités localement en mémoire cache/GPU. Chaque bloc est factorisé indépendamment avant d'être combiné, ce qui permet d'exploiter plusieurs cœurs ou nœuds en parallèle.

Selon les résultats de (Grigori *et al.*, 2011), la méthode CALU réduit le volume de communications d'un facteur logarithmique par rapport à la méthode LU avec pivot partiel classique, tout en conservant une croissance modérée des erreurs d'arrondi. L'intégration de CALU a permis une

accélération jusqu'à 3 fois plus rapide sur architectures multi-cœurs pour des grilles de type 256^3.

Bien que l'algorithme CALU semble offrir des performances exceptionnelles en réduisant les communications inter-processeurs, son utilisation sur une matrice rectangulaire impose une transformation vers un système carré dont le surcoût de calcul risque de neutraliser les gains de parallélisation attendus.

2.3.6 Méthode GMRES

La méthode GMRES *(Generalized Minimal RESidual)* sert à résoudre des systèmes $Ax = b$ où A n'est pas forcément symétrique. Partant d'une approximation initiale x_0, on définit le résidu initial $r_0 = b - Ax_0$. On construit un espace de Krylov :

$$\mathcal{K}_m(A, r_0) = \{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}, \tag{2.21}$$

puis on recherche $x_m \in x_0 + \mathcal{K}_m(A, r_0)$ qui minimise la norme du résidu $\|b - Ax_m\|$. Cet espace de Krylov se construit grâce au théorème de Cayley–Hamilton, les puissances de A s'expriment comme combinaisons linéaires de $I, A, \ldots, A^{n-1}$. Tout vecteur de $x_0 + \mathcal{K}_m(A, r_0)$ s'écrit

$$x = x_0 + q(A)r_0, \tag{2.22}$$

où q est un polynôme de degré $\leq m - 1$. Le résidu correspondant est

$$r = b - Ax = (I - Aq(A))r_0 = p(A)r_0, \tag{2.23}$$

avec $p(X) = 1 - Xq(X)$, de degré $\leq m$. On a donc $p(0) = 1$ automatiquement. Ainsi, chercher la meilleure approximation dans l'espace de Krylov revient à trouver le polynôme p (avec $p(0) = 1$) qui minimise $\|p(A)r_0\|$.

Algorithme 2.6 Méthode GMRES (sans redémarrage)

Algorithme : Méthode GMRES

Input : Matrice $A \in \mathbb{R}^{n \times n}$, vecteur $b \in \mathbb{R}^n$, dimension maximale m

Output : Approximation x_m de la solution du système $Ax = b$

Choisir une initialisation x_0 (souvent $x_0 = 0$);
$r_0 \leftarrow b - Ax_0$;
$v_1 \leftarrow r_0 / \|r_0\|$;
for $j = 1$ **to** m **do**
    $w \leftarrow Av_j$;
    for $i = 1$ **to** j **do**
        $h_{i,j} \leftarrow v_i^\top w$;
        $w \leftarrow w - h_{i,j} \cdot v_i$;
    end for
    $h_{j+1,j} \leftarrow \|w\|$;
    $v_{j+1} \leftarrow w / h_{j+1,j}$;
end for
Construire la matrice $H \in \mathbb{R}^{(m+1) \times m}$ à partir des $h_{i,j}$;
Résoudre le problème aux moindres carrés : $y \leftarrow \arg\min_{y \in \mathbb{R}^m} \left\| Hy - \|r_0\| e_1 \right\|$;
Construire $V_m = [v_1, \ldots, v_m]$;
$x_m \leftarrow x_0 + V_m y$;
return x_m;

En pratique, GMRES est souvent combinée avec un préconditionneur $M \approx A^{-1}$ pour accélérer la convergence. De plus, pour éviter un coût mémoriel trop grand, on "redémarre" la méthode (GMRES(m)), ce qui permet de limiter la taille de l'espace de Krylov à m.

2.3.7 Décomposition en valeurs singulières (SVD)

La décomposition en valeurs singulières, ou SVD, consiste à écrire pour une matrice $A \in \mathbb{R}^{m\times n}$ quelconque :

$$A = U\Sigma V^\top, \tag{2.24}$$

avec

- $U \in \mathbb{R}^{m\times m}$ orthonormée,
- $\Sigma \in \mathbb{R}^{m\times n}$, contenant les valeurs singulières $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$ sur la diagonale,
- $V \in \mathbb{R}^{n\times n}$ orthonormée.

Les valeurs singulières d'une matrice A sont les racines carrées des valeurs propres de la matrice $A^\top A$ ou $AA^\top$. Elles jouent un rôle fondamental dans l'étude du rang de la matrice et dans des applications telles que l'approximation de matrices (réduction de rang) et la compression de données. En effet, le rang d'une matrice A s'obtient directement à partir du nombre de valeurs singulières strictement positives :

$$rang(A) = \#\{\sigma_i > 0\}. \tag{2.25}$$

Calcul mathématique de U, Σ et V

Soit $A \in \mathbb{R}^{m\times n}$ une matrice quelconque. La décomposition en valeurs singulières consiste à trouver des matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ orthogonales et $\Sigma \in \mathbb{R}^{m\times n}$ telles que :

$$A = U\Sigma V^\top \tag{2.26}$$

avec :

$$U^\top U = I_m, \quad V^\top V = I_n, \quad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0) \quad \text{avec } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0. \tag{2.27}$$

Étapes du calcul :

1. Calcul de $A^\top A \in \mathbb{R}^{n\times n}$, symétrique définie positive.
2. Diagonalisation :
$$A^\top A = V\Lambda V^\top, \quad \text{avec } \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n). \tag{2.28}$$
3. Calcul des valeurs singulières :
$$\sigma_i = \sqrt{\lambda_i}. \tag{2.29}$$
4. Les colonnes de V sont les vecteurs propres de $A^\top A$ associés aux valeurs propres λ_i. Leurs correspondants à gauche s'obtiennent par
$$u_i = \frac{1}{\sigma_i} A v_i. \tag{2.30}$$

On peut également utiliser $AA^\top$ pour construire directement U si $m < n$, car $AA^\top$ est plus petit à diagonaliser.

Décomposition en valeurs singulières aléatoire (RSVD)

La méthode de RSVD est une technique probabiliste conçue pour approximer rapidement de grandes matrices (Martinsson *et al.*, 2011). Son objectif principal est de générer une approximation de rang $k < r$ d'une matrice A, avec r le rang de la matrice ou d'identifier les k valeurs singulières les plus importantes et leurs vecteurs singuliers correspondants (Martinsson *et al.*, 2011). Cette approche permet donc de réduire la complexité du calcul tout en préservant les modes dominants de la matrice. La figure 2.7 illustre la différence entre la SVD complète et sa version randomisée, où seules les k composantes principales sont conservées.

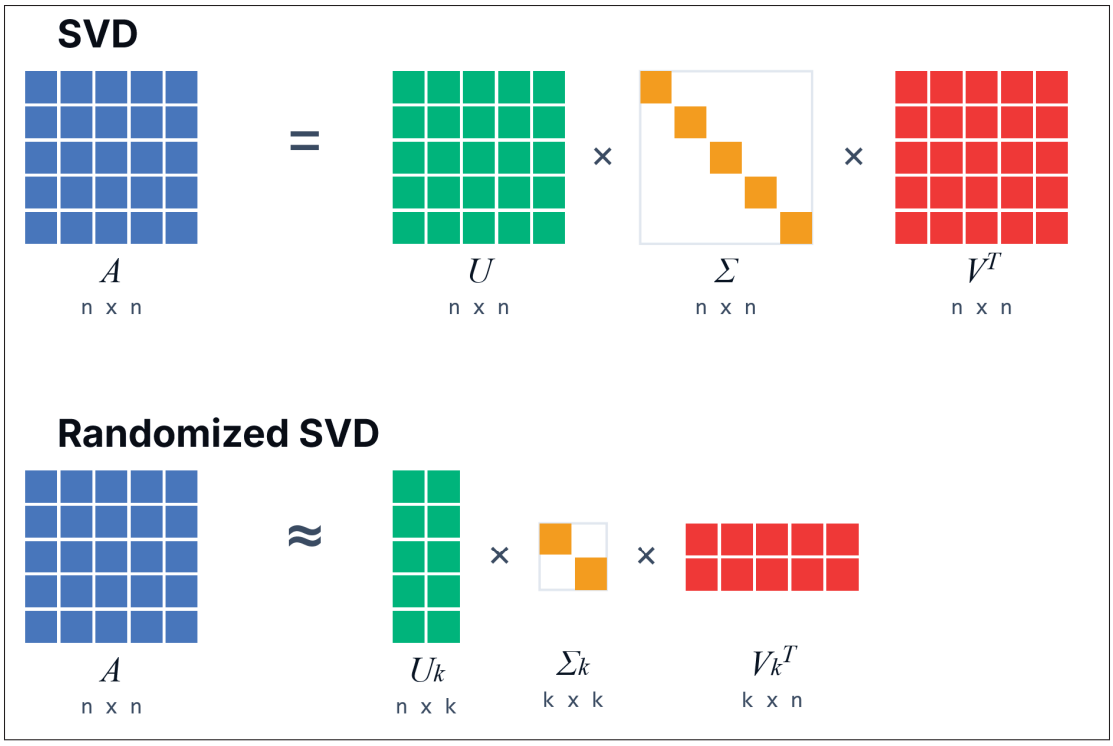


Figure 2.7 Comparaison entre la SVD complète et la SVD randomisée (RSVD). Alors que la SVD calcule l'ensemble des valeurs et vecteurs singuliers, la RSVD se limite à une approximation de rang k, ne conservant que les composantes principales dominantes. Cette approche permet de réduire le coût de calcul et l'utilisation mémoire, tout en préservant les informations essentielles

Algorithme 2.7 SVD aléatoire (version de base)

Algorithme : SVD aléatoire (version de base)

Input : Matrice $A \in \mathbb{R}^{m \times n}$, rang cible k, suréchantillonnage p avec $k + p < \min(m, n)$

Output : Approximation $A \approx U_k \Sigma_k V_k^\top$

Générer une matrice aléatoire $\Omega \in \mathbb{R}^{n \times (k+p)}$ (les entrées sont des variables indépendantes et identiquement distribuées suivant une loi normale centrée réduite $\mathcal{N}(0, 1)$);

Calculer la matrice échantillonnée : $Y = A\Omega \in \mathbb{R}^{m \times (k+p)}$;

Orthonormaliser les colonnes de Y par décomposition QR : $Q \in \mathbb{R}^{m \times (k+p)}$;

Projeter A sur le sous-espace colonne de Q : $B = Q^\top A \in \mathbb{R}^{(k+p) \times n}$;

Calculer la SVD de B : $B = \widehat{U} \Sigma V^\top$;

Calculer $U = Q\widehat{U}$;

Tronquer : ne conserver que les k premières colonnes de U et V, et les k premières valeurs singulières dans Σ;

return U_k, Σ_k, V_k;

Le fonctionnement de cette méthode s'articule autour de deux phases principales, l'idée étant de sonder un sous-espace réduit de la matrice pour en extraire l'information essentielle sans avoir à traiter la matrice entière (Martinsson *et al.*, 2011).

Tout d'abord, on procède à la construction d'une base de l'image de la matrice transposée A. Pour cela, un ensemble de $k + p$ vecteurs aléatoires est généré et multiplié par A, capturant ainsi l'information dominante dans un espace de dimension réduite (Martinsson *et al.*, 2011). À partir de ces vecteurs, une base orthonormée est ensuite construite.

Dans un second temps, la SVD est calculée sur une matrice projetée de taille réduite. Plus précisément, la matrice A est projetée sur la base obtenue, réduisant fortement sa taille. Une décomposition en valeurs singulières classique peut alors être effectuée sur cette petite matrice, ce qui permet de retrouver les k plus grandes valeurs singulières et leurs vecteurs associés pour la matrice d'origine (Martinsson *et al.*, 2011). Voir exemple dans l'Annexe II.

La SVD randomisée est particulièrement efficace lorsque les produits Av et $A^T v$ peuvent être calculés rapidement (Martinsson *et al.*, 2011). Elle est également robuste, fonctionnant bien indépendamment de la structure de la matrice A. En termes de précision, l'erreur d'approximation $\|A - Z\|$ est proportionnelle à $\sigma_{k+1}\sqrt{l \ \max(m, n)}$ avec une forte probabilité, où :

- Z est l'approximation de rang k de A obtenue par la SVD aléatoire,
- $l = k + p$ avec p le paramètre de suréchantillonnage,
- m est le nombre de lignes de A et n le nombre de colonnes,
- σ_{k+1} est la $(k + 1)$-ième valeur singulière de A.

(Martinsson *et al.*, 2011). L'algorithme est donc conçu pour minimiser le coût de calcul tout en maintenant une approximation quasi-optimale. Cette méthode est également pertinente dans un cadre plus général, notamment pour les problèmes de valeurs propres associés à de grandes matrices creuses (Golub & Van Loan, 2013).

Tableau 2.1 Comparaison des solveurs pour la résolution d'un système linéaire avec une matrice creuse rectangulaire de taille $m \times n$

Algorithme	**Gestion matrice rectangulaire**	**Compatibilité matrices creuses**	**Compatibilité parallélisation**	**Particularité**
QR	Factorisation $A = QR$	Faible (remplissage)	Moyenne	Très coûteux pour de grandes dimensions.
LU / CALU	Via Reformulation Carrée	Très faible (remplissage)	Élevée	Risque de saturation mémoire par remplissage massif.
SOR / MC-SOR	Via Reformulation Carrée	Élevée	Très élevée	Calculs par voisins directs et sans stockage additionnel.
BiCGStab	Via Reformulation Carrée	Élevée	Moyenne	Présente des risques d'instabilité numérique.
GMRES	Via méthode des moindres carrés	Bonne	Moyenne	Robuste mais limité par le stockage de la base de Krylov.
RSVD	Réduction de rang directe	Bonne	Élevée	Précision dépendante du rang cible k (approximation).

2.4 Choix des algorithmes

À la suite de l'analyse comparative présentée ci-dessus, l'algorithme *GMRES* (*Generalized Minimal Residual method*) a été sélectionné pour la première phase d'implémentation. Ce choix est motivé par la grande polyvalence et la robustesse de cette méthode de Krylov face aux systèmes linéaires complexes générés par les maillages non structurés.

Cependant, dans le cadre de NSMP, la méthode GMRES a conduit à des résultats non concluants. Malgré sa robustesse théorique, l'algorithme a présenté des défauts de précision significatifs

et une convergence instable sur les configurations de maillages complexes, ne permettant pas d'atteindre les seuils de tolérance requis pour la validation physique de l'écoulement.

Face à ces limitations, une approche alternative hybride combinant la décomposition en valeurs singulières aléatoire (*Randomized SVD* - RSVD) et un lissage par SOR (*Succesive Over-Relaxation*) a été développée. Cette stratégie repose sur une séparation spectrale de la résolution : l'algorithme RSVD traite les composantes de basse fréquence de l'erreur en projetant le problème sur un sous-espace de rang faible $k \ll N$, permettant de capturer rapidement la structure globale de la pression.

L'efficacité de cette approche hybride repose sur la complémentarité spectrale entre la RSVD et le lisseur SOR. Par construction, l'algorithme RSVD fournit une approximation de rang réduit qui capture les modes dominants du système via un seuil d'énergie (voir partie 3.2.2), traitant ainsi efficacement les composantes de basse fréquence spatiale (erreurs globales). Cependant, cette troncature laisse subsister des résidus entre les mailles, à une plus petite échelle.

Pour corriger ces imperfections, la solution issue de la RSVD est injectée comme condition initiale dans un algorithme de *Successive Over-Relaxation* (SOR). Grâce à sa formulation itérative locale, il dissipe rapidement les erreurs locales (haute fréquence). Ce couplage permet ainsi d'obtenir une solution précise sur l'ensemble du spectre d'erreur en un nombre réduit d'itérations.

Le couplage entre la méthode RSVD et le lisseur SOR repose sur un transfert de solution. Le processus mathématique se décompose en trois étapes clés :

1. Initialisation globale (RSVD) : On calcule une première approximation $\mathbf{x}_{\text{RSVD}}$ en projetant le système sur la base réduite Q de rang $k \ll N$:

$$\mathbf{x}_{\text{RSVD}} = Q(Q^T\mathbf{b}) \tag{2.31}$$

   Ici, $Q^T\mathbf{b}$ représente la projection du second membre sur la base, et la multiplication par Q redonne un vecteur de pression de taille N.

2. Transfert de solution : La solution RSVD est injectée comme point de départ (*initial guess*) pour la phase itérative :

$$\mathbf{x}^{(0)} = \mathbf{x}_{\text{RSVD}} \tag{2.32}$$

3. Lissage local (SOR) : L'algorithme SOR est appliqué pour corriger les erreurs résiduelles de haute fréquence. Pour chaque cellule i, la mise à jour s'écrit :

$$\mathbf{x}_i^{(k+1)} = (1-\omega)\mathbf{x}_i^{(k)} + \frac{\omega}{A_{ii}}\left(\mathbf{b}_i - \sum_{j<i} A_{ij}\mathbf{x}_j^{(k+1)} - \sum_{j>i} A_{ij}\mathbf{x}_j^{(k)}\right) \tag{2.33}$$

Cette initialisation par RSVD fournit un résidu initial $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ déjà très faible. Le lisseur SOR n'a alors besoin que de quelques itérations pour ajuster les valeurs de chaque cellule et ainsi stabiliser la solution finale.

2.5 Analyse des performances et validation numérique

Afin d'évaluer la robustesse et l'efficacité de la méthode hybride RSVD+SOR, une campagne de tests a été menée sur différentes matrices. L'objectif de cette étude paramétrique est d'isoler l'impact des propriétés algébriques du système sur la vitesse de convergence et la précision de l'algorithme.

Les matrices générées aléatoirement reproduisent la topologie creuse issue de la discrétisation des équations de Navier–Stokes sur nos maillages. Les tests ont été effectués pour deux tailles de matrices NxN caractéristiques ($N = 200$ et $N = 400$). Pour chaque configuration, les paramètres physiques suivants ont été modifiés afin de balayer un large spectre de difficultés :

- La densité de la matrice : le niveau de remplissage a varié entre 0 et 80%.
- Le conditionnement cible (κ_{cible}) : cinq ordres de grandeur ont été imposés lors de la génération, allant de systèmes triviaux ($\kappa \approx 10^0$) à des systèmes numériquement raides ($\kappa \approx 10^5$).

La Figure 2.8 présente l'erreur relative de la solution en fonction du conditionnement réel du système. Les performances ont été tracées pour deux valeurs distinctes du paramètre de sur-relaxation ($\omega = 1.2$ et $\omega = 1.7$) et pour un nombre d'itérations SOR fixé (ici 20).

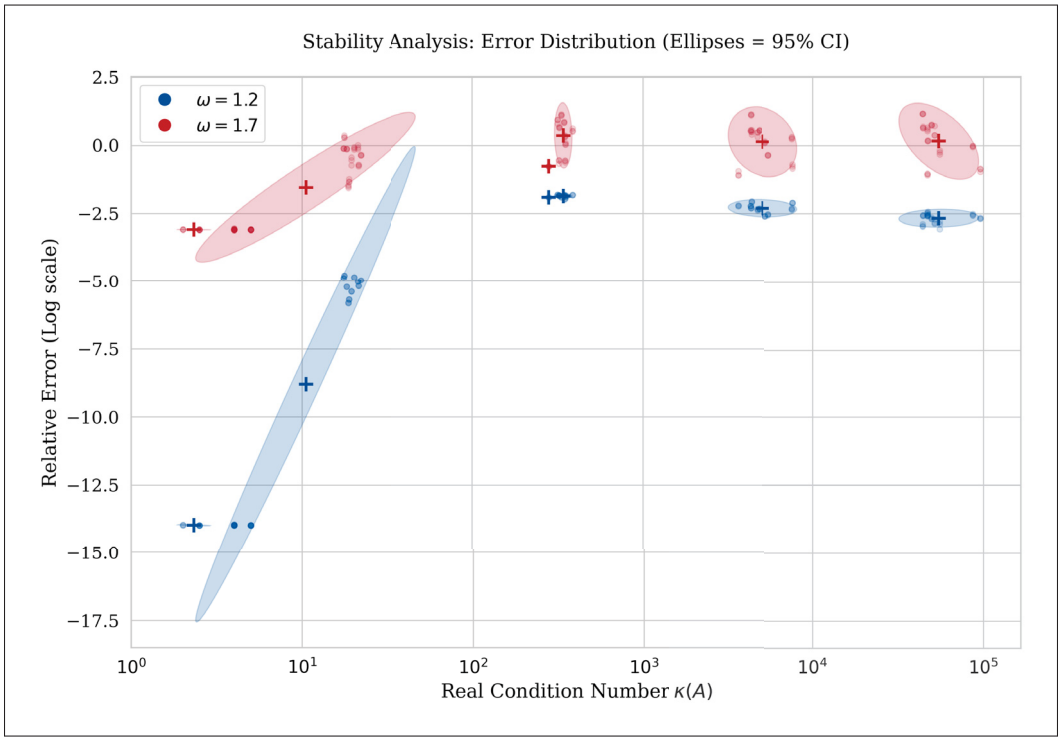


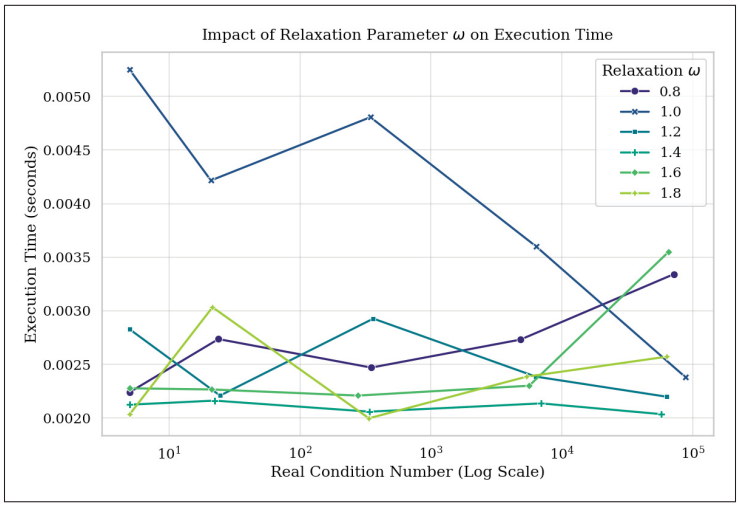
Figure 2.8 Erreur relative de l'algorithme hybride RSVD+SOR pour $\omega = 1.2$ et $\omega = 1.7$ avec un seuil d'énergie de 90%, en fonction du conditionnement de la matrice du système

Sur ce graphique, on observe que les points de mesure se regroupent naturellement en amas, mis en évidence par les ellipses. Chaque ellipse correspond à un unique conditionnement cible allant de 10^0 à 10^5. La dispersion des points à l'intérieur d'une même ellipse s'explique par la variation de la densité : cela engendre une déviation entre le conditionnement théorique imposé et le conditionnement réel (calculé par SVD exacte).

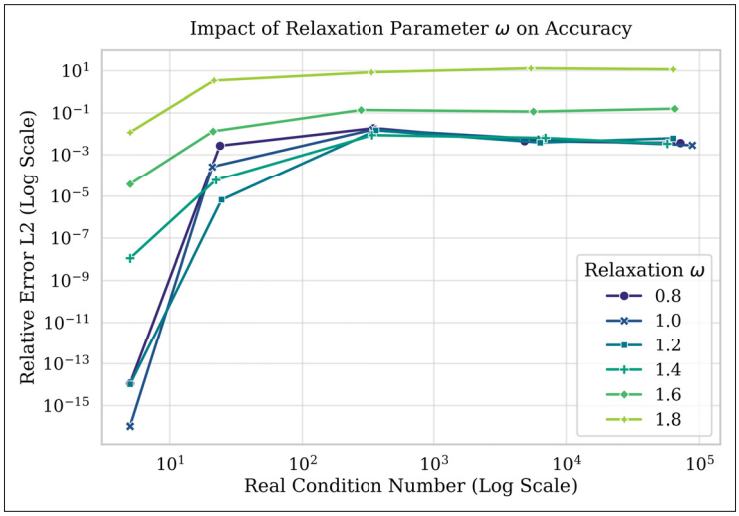
L'analyse des courbes confirme que globalement l'augmentation du conditionnement d'une matrice affecte la précision de l'algorithme pour un nombre d'itérations fixé.

Afin d'optimiser les performances du solveur hybride RSVD+SOR, une étude de sensibilité du paramètre de sur-relaxation successive ω a été réalisée. Un nombre d'itérations SOR était fixé (à 20), le temps d'exécution CPU et l'erreur résiduelle relative en sortie d'algorithme ont été mesurés.

La Figure 2.9 illustre cette double comparaison pour plusieurs valeurs du paramètre $\omega \in [0.8, 1.8]$, en fonction du conditionnement réel du système.



a) Temps d'exécution CPU (secondes)



b) Erreur relative résiduelle L_2

Figure 2.9 Impact du paramètre de relaxation ω sur les performances de la méthode RSVD+SOR à nombres d'itérations SOR fixé (taille de la matrice $N = 400$)

Grâce à ces figures 2.9a et 2.9b, on observe que la valeur $\omega = 1.4$ offre le meilleur compromis dynamique : elle permet de s'approcher le plus rapidement possible de la solution exacte sans

provoquer les instabilités numériques que l'on peut observer pour des valeurs trop proches de la limite théorique de divergence ($\omega \to 2$).

Cependant, l'application de l'algorithme à notre cas réel du pilier de pont sur un maillage non structuré impose des contraintes de stabilité supplémentaires. Les tests faits en calcul parallèle ont révélé une divergence systématique pour $\omega = 1.3$ et $\omega = 1.4$. Afin de garantir la robustesse de la simulation hydraulique tout en conservant une convergence rapide, nous retiendrons la valeur $\omega = 1.2$ pour la suite des travaux, celle-ci offrant le meilleur compromis stabilité-performance sur cette géométrie complexe.

CHAPITRE 3

APPLICATION SUR LE CAS TAYLOR GREEN VORTEX : RÉSULTATS, ANALYSE ET DISCUSSION

Afin de valider la précision de l'algorithme hybride RSVD+SOR, nous avons confronté nos résultats numériques à la solution analytique du vortex de Taylor-Green.

Le vortex de Taylor-Green (TGV) possède une solution analytique exacte des équations de Navier–Stokes, ce qui permet de mesurer rigoureusement l'erreur numérique.

Pour un domaine périodique $[0, 2\pi]^2$, les composantes de la vitesse u et v, ainsi que la pression p, évoluent selon les équations suivantes :

$$u(x, y, t) = U_0 \sin(x) \cos(y) e^{-2\nu t} \tag{3.1}$$

$$v(x, y, t) = -U_0 \cos(x) \sin(y) e^{-2\nu t} \tag{3.2}$$

$$p(x, y, t) = \frac{U_0^2}{4} \left(\cos(2x) + \cos(2y)\right) e^{-4\nu t} \tag{3.3}$$

où U_0 est l'amplitude initiale de la vitesse et ν la viscosité cinématique du fluide. L'étude de cet écoulement permet de confirmer que le solveur hybride RSVD+SOR capture correctement la structure spectrale du champ de pression tout en éliminant les erreurs de haute fréquence induites par le maillage.

Une étude de convergence de grille a été menée en testant l'algorithme sur une large gamme de résolutions : 4×4, 8×8, 16×16, 20×20, 40×40, 64×64, 128×128 et 240×240.

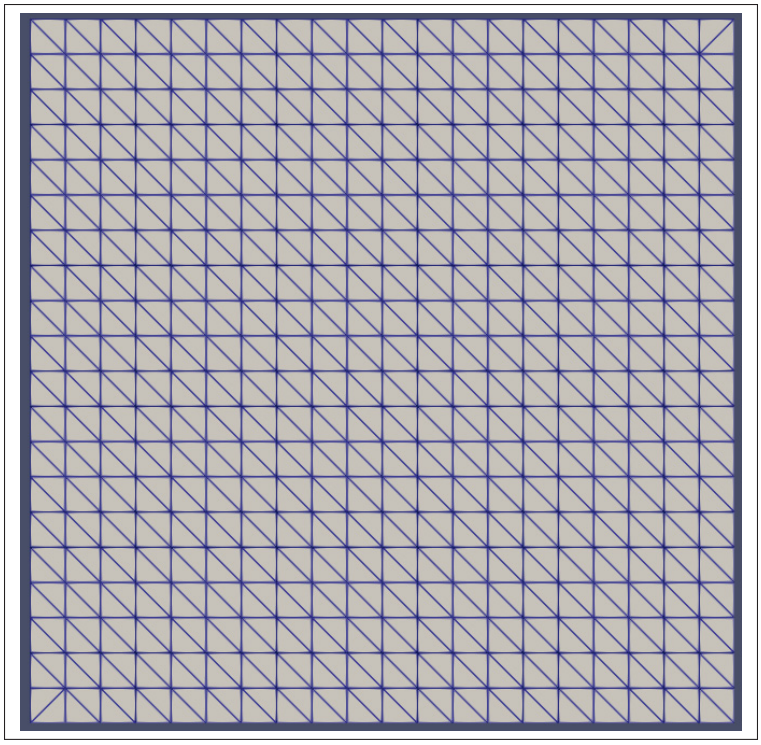


Figure 3.1 Exemple de maillage structuré utilisé pour la validation du vortex de Taylor-Green (ici résolution 20×20)

L'analyse détaillée des erreurs est présentée dans le chapitre suivant. Nous y utiliserons notamment une représentation logarithmique de l'erreur en norme L_2 en fonction du pas d'espace Δx pour déterminer l'ordre de convergence effectif de notre méthode.

3.1 Comparaison de l'erreur numérique

Cette section présente les performances numériques de l'algorithme hybride RSVD+SOR appliqué au cas de validation du vortex de Taylor-Green en 2D. L'objectif est de connaître l'ordre de convergence spatiale du solveur. La figure 3.2 montre les résultats obtenus sur le cas TGV avec l'algorithme RSVD+SOR, visualisés avec le logiciel Paraview.

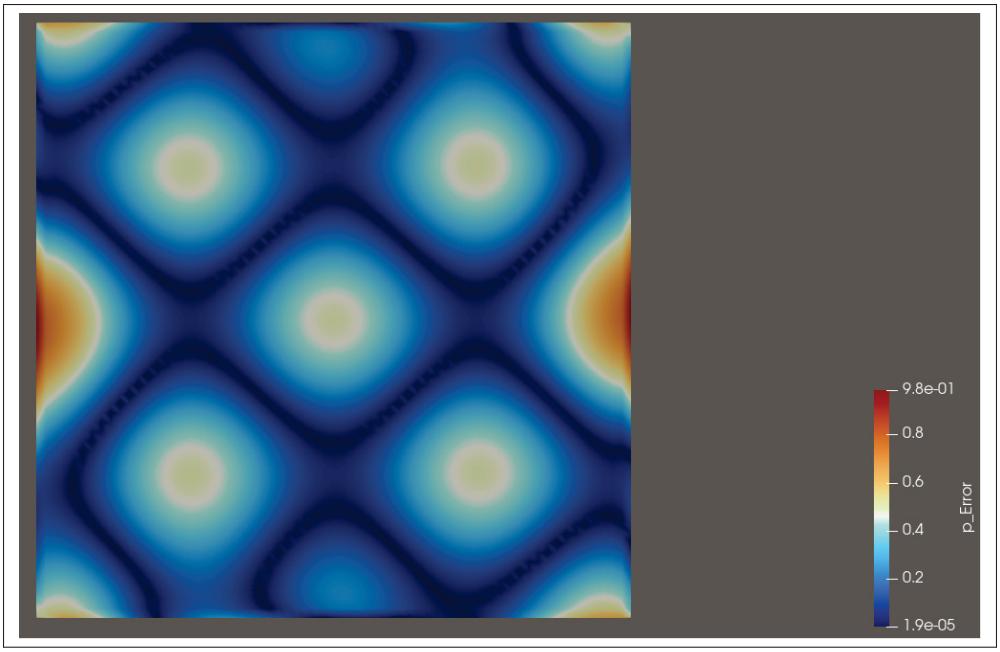


Figure 3.2 Erreur entre la pression calculée par RSVD+SOR et la pression exacte déterminée par les équations du vortex de Taylor-Green, sur une grille de résolution 20x20

La Figure 3.3 présente l'évolution du logarithme de la norme L_2 de l'erreur en fonction du logarithme du pas d'espace Δx.

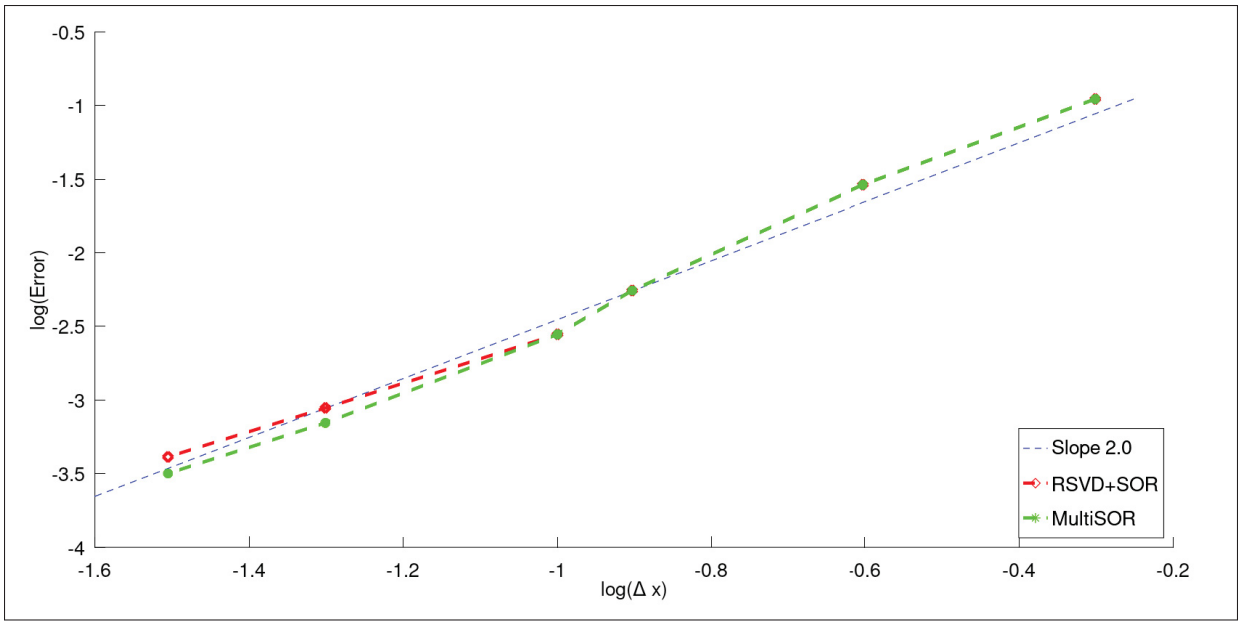


Figure 3.3 Étude de convergence spatiale pour le vortex de Taylor-Green en 2D. Comparaison de l'erreur en norme L_2 de la pression entre celle obtenue avec le solveur RSVD+SOR (losanges rouges) et les résultats avec l'algorithme MultiSOR (étoiles vertes). La droite en pointillés bleue représente la pente théorique d'ordre 2

L'analyse de la Figure 3.3 révèle que les points de données obtenus suivent la droite de référence de pente d'ordre 2, comme avec les résultats de MultiSOR (Zhang *et al.*, 2020). Cette tendance montre que le solveur hybride RSVD+SOR respecte un ordre de convergence global de second ordre pour la pression, garantissant ainsi une précision optimale même sur des maillages de résolution intermédiaire. La stabilité de l'algorithme est maintenue sur l'ensemble du spectre de résolutions testées, sans apparition d'oscillations numériques ou de divergences, validant ainsi l'utilisation du paramètre de relaxation $\omega = 1,2$ établi précédemment.

Ainsi, il est important de souligner que l'étape d'approximation, faite par la méthode RSVD, bien qu'elle se limite à capturer les composantes principales dominantes de la matrice, ne détériore pas la précision globale du solveur.

3.2 Comparaison du temps d'exécution avec passage à l'échelle

Afin d'évaluer l'efficacité de la parallélisation, une étude de performance (*strong scaling*) a été menée sur le cas du vortex de Taylor-Green avec une grille de 640×640 cellules. Les simulations ont été exécutées sur le cluster Narval, pour un nombre de processus MPI allant de 1 à 256.

La décomposition de domaine utilisée repose sur une répartition horizontale stricte entre les processeurs. La figure 3.4 présente l'évolution du temps de calcul CPU total en fonction du nombre de cœurs. La figure 3.5 expose l'accélération parallèle obtenue pour les deux solveurs et la figure 3.6 présente l'évolution de l'efficacité parallèle en fonction du nombre de processeurs.

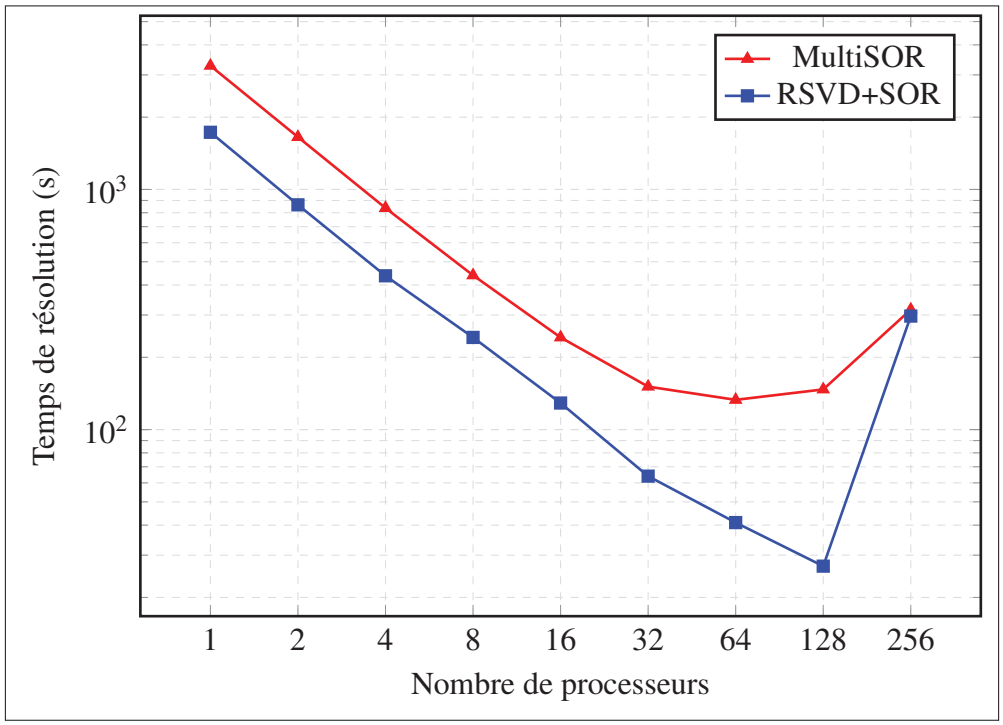


Figure 3.4 Temps d'exécution CPU en fonction du nombre de processeurs pour le cas TGV, résolution 640×640. Les tests couvrent une plage de 1 à 256 cœurs

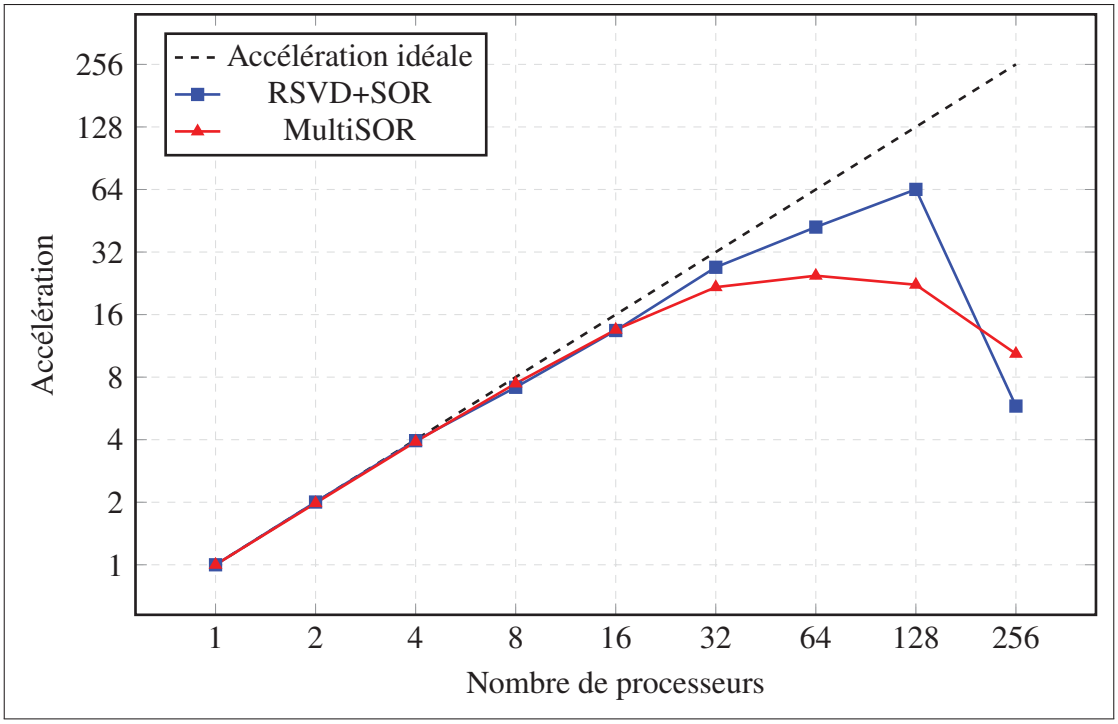


Figure 3.5 Accélération parallèle pour le cas TGV en fonction du nombre de processeurs, résolution 640×640

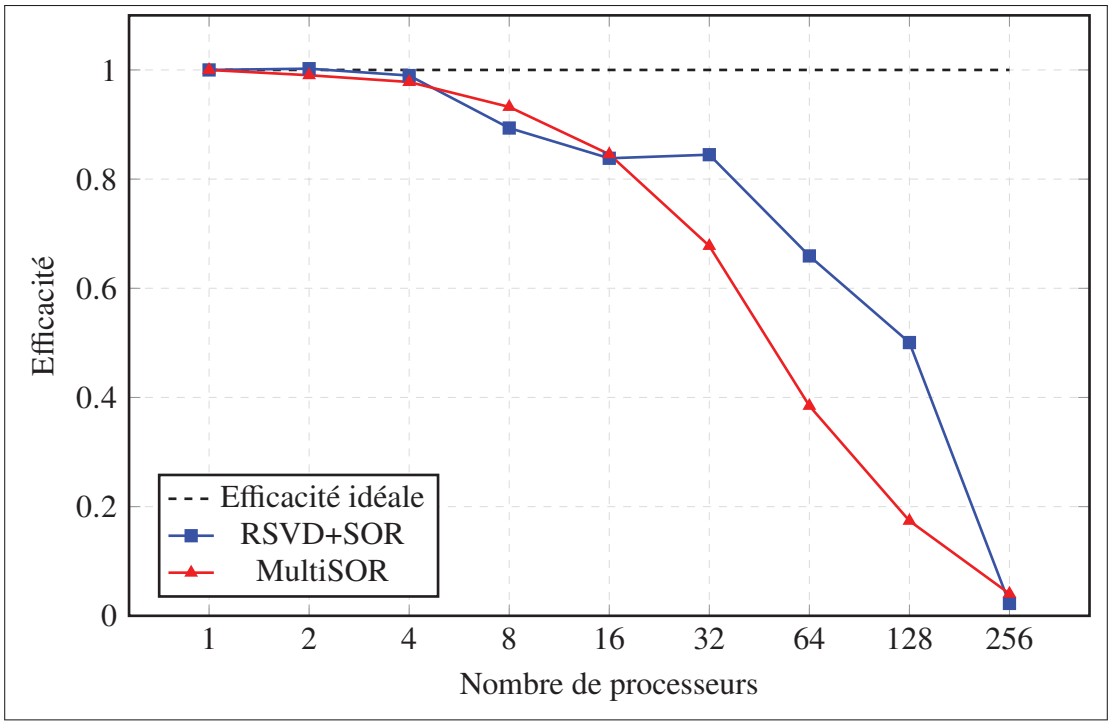


Figure 3.6 Efficacité parallèle pour le cas TGV en fonction du nombre de processeurs, résolution 640×640

Pour cette série de tests, le calcul a été configuré avec un seuil d'énergie de 75% (paramètre dont les mécanismes de contrôle sont détaillés à la section 3.2.2), contre 95% pour les précédents calculs. Ce réglage permet au solveur hybride d'obtenir des performances nettement supérieures à l'algorithme MultiSOR classique en termes de rapidité d'exécution, bien que les deux approches subissent une forte augmentation du temps d'exécution à 256 processeurs. Ce phénomène s'explique par la prédominance des coûts de communication entre processus (MPI) sur le gain de calcul brut. L'accélération, définie comme le rapport entre le temps de calcul séquentiel et le temps en parallèle, mesure la réduction du temps de traitement résultant de l'augmentation du nombre d'unités de calcul allouées, montrant ici une supériorité marquée jusqu'à 128 cœurs. Lors du passage à 256 processeurs, il y a une décomposition excessive du domaine de calcul avec des blocs de largeur de 2,5 mailles ($640/256 = 2,5$). Avec de si petits sous-domaines, le coût des échanges aux interfaces explose. De son côté, l'efficacité mesure le taux d'utilisation réelle des processeurs ; les résultats obtenus soulignent la robustesse de RSVD+SOR qui maintient

un rendement élevé bien plus longtemps que l'approche MultiSOR avant que les latences de communication ne deviennent prédominantes au-delà de ce seuil. L'efficacité est définie comme le rapport entre l'accélération obtenue et le nombre de processeurs.

Quant à la fiabilité de cette approche, bien que l'étude de convergence spatiale (Figure 3.3) ait été validée avec un seuil de 95% pour confirmer rigoureusement l'ordre de convergence 2 , il est important de souligner que l'étape d'approximation par RSVD ne détériore pas la précision globale du solveur final. L'initialisation par réduction de rang fournit une solution globale robuste que le lisseur SOR affine ensuite pour éliminer les résidus locaux et atteindre la précision requise sur l'ensemble du spectre d'erreur.

3.2.1 Validation sur un cas 3D réel

Enfin, pour éprouver la robustesse de l'approche hybride RSVD+SOR, le solveur a été testé sur un cas réel complexe : l'écoulement autour d'un pilier de pont. La figure 3.7 et 3.8 illustrent la distribution de la pression adimensionnelle p sur l'ensemble du domaine au même instant t, respectivement avec le solveur RSVD+SOR et MultiSOR.

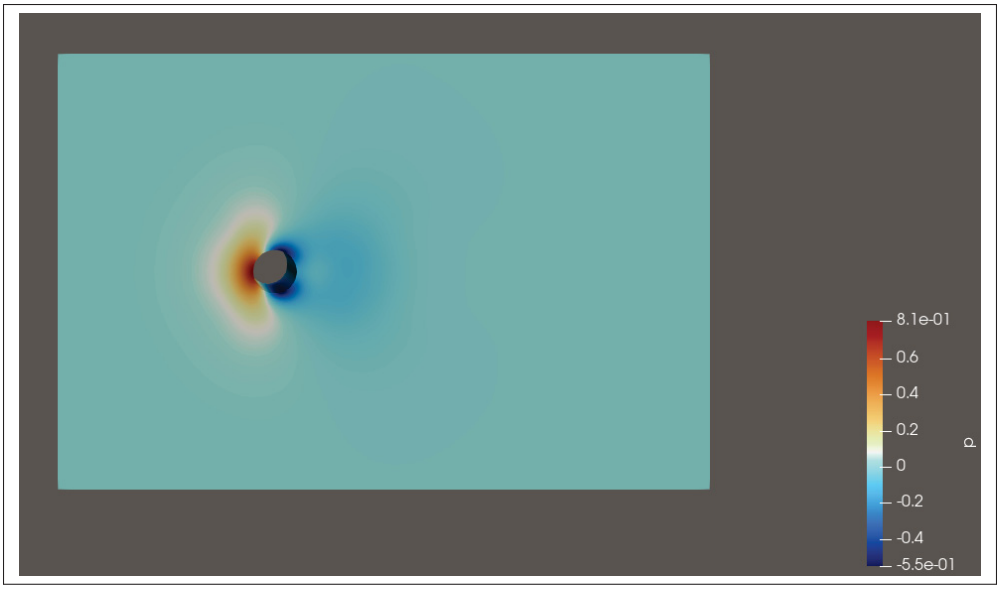


Figure 3.7 Distribution de la pression adimensionnelle autour d'un pilier de pont. La zone rouge en amont indique le point d'arrêt (surpression), tandis que la zone bleue foncée en aval illustre le sillage et la dépression caractéristique derrière l'obstacle. Le domaine est ici traversé par un courant allant de l'extrémité gauche à l'extrémité droite

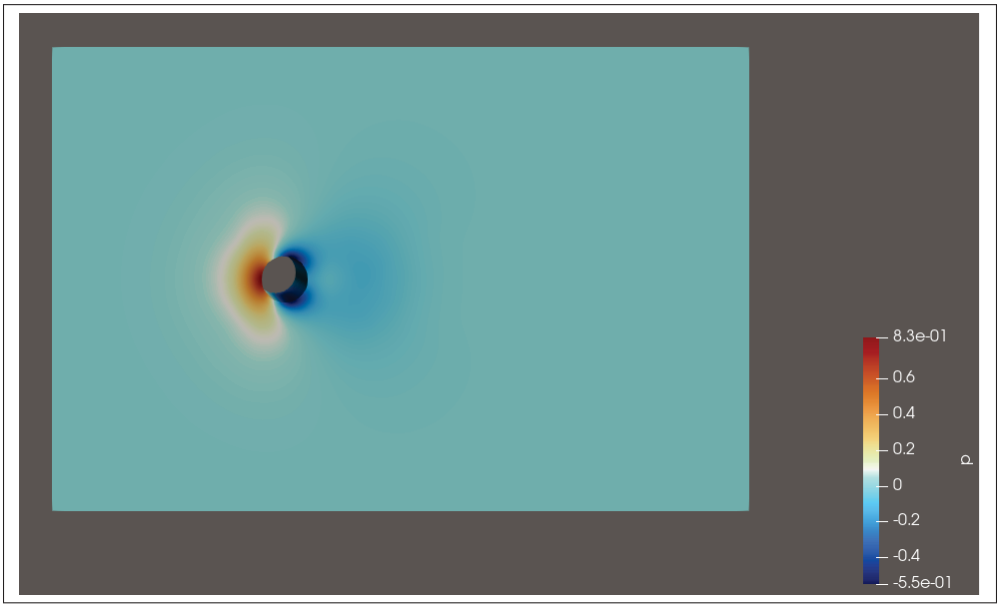


Figure 3.8 Résultat obtenu en exécutant le code NSMP dans les mêmes conditions mais en utilisant l'algorithme MultiSOR

L'analyse du champ de pression montre une parfaite symétrie et une capture précise des phénomènes physiques attendus. On observe distinctement la zone de haute pression frontale, correspondant au ralentissement du fluide au contact du pilier, ainsi que la zone de dépression dans le sillage de l'obstacle. La capacité du solveur à converger sur ce type de maillage non structuré valide l'utilisation de la réduction de rang pour initialiser les calculs de pression dans le code NSMP.

3.2.2 Flexibilité et compromis précision-vitesse du solveur RSVD+SOR

Au-delà des performances de *strong scaling*, la méthode RSVD se distingue par une grande variété de paramètres internes permettant d'ajuster finement l'efficacité de la convergence selon les besoins de la simulation. Contrairement à l'algorithme MultiSOR, dont la performance dépend essentiellement du facteur de relaxation et du nombre d'itérations, le solveur RSVD introduit plusieurs leviers de contrôle permettant d'optimiser la qualité de l'approximation initiale sans altérer la solution finale du système $Ax = b$:

- **Paramètre de sur-échantillonnage (p)** : Ce paramètre définit la dimension de la matrice de projection aléatoire $\Omega \in \mathbb{R}^{n\times(k+p)}$. Il garantit, avec une probabilité extrêmement élevée, que l'espace image de la matrice A est correctement capturé, sécurisant ainsi la robustesse de la base réduite nécessaire à l'accélération du calcul.
- **Seuil d'énergie (α)** : Ce paramètre permet de déterminer dynamiquement le rang effectif k pour capturer un pourcentage défini de l'information (ou variance) de la matrice de pression. Le rang k est choisi comme le plus petit entier satisfaisant le ratio de l'énergie accumulée sur l'énergie totale (basée sur la norme de Frobenius) :

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{j=1}^{\min(m,n)} \sigma_j^2} \geq \alpha \tag{3.4}$$

  Où σ_i représentent les valeurs singulières classées par ordre décroissant. Un seuil de 75% (utilisé pour les tests de performance) accélère la phase de projection en réduisant significativement k, fournissant un *initial guess* global qui sera ensuite affiné. Un seuil plus

élevé (95%) produit une approximation initiale plus proche de la solution réelle, au prix d'un investissement computationnel supérieur lors de la décomposition en valeurs singulières.

- **Itérations de lissage** : Spécifique à l'approche hybride, ce paramètre contrôle le nombre de cycles SOR appliqués après la résolution RSVD. Puisque la RSVD traite prioritairement les erreurs de basse fréquence spatiale, le lisseur utilise la matrice A originale pour éliminer les résidus de haute fréquence restants. Cette étape assure la convergence vers la solution exacte du système initial :

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{A_{ii}}\left(b_i - \sum_{j<i} A_{ij}x_j^{(k+1)} - \sum_{j>i} A_{ij}x_j^{(k)}\right) \tag{3.5}$$

Ces paramètres offrent une flexibilité supérieure pour traiter des configurations variées. Ils permettent soit de minimiser le temps de calcul global par une réduction de rang agressive (seuil α modéré), soit de limiter la charge de travail du lisseur en investissant davantage dans la fidélité de l'approximation initiale.

L'efficacité du solveur hybride observée lors des tests de performance peut s'expliquer par la complémentarité spectrale des deux méthodes employées. D'un point de vue théorique, cette approche revient à effectuer une réduction de dimensionnalité du problème par projection dans un espace de vecteurs singuliers, avant d'opérer un lissage dans l'espace physique.

À l'instar d'une transformée de Fourier où l'on ne conserverait que les modes de basse fréquence, la RSVD projette le système de Poisson sur ses vecteurs singuliers dominants. En tronquant la base selon un critère d'énergie (typiquement 75 %), on capture l'essentiel de la structure globale du champ de pression. La reprojection de cette solution approchée dans l'espace physique fournit alors un point de départ informé (*initial guess*), représentant fidèlement la structure globale de l'écoulement mais dépourvu des détails de haute fréquence.

Le rôle des itérations de SOR qui suivent est alors de lisser les résidus locaux subsistant après la troncature. Dans cette configuration, le SOR n'agit plus comme un solveur global, mais comme

un lisseur spatial qui élimine rapidement les composantes haute fréquence de l'erreur. C'est cette succession d'étapes qui fait l'intérêt de cette approche.

CONCLUSION ET RECOMMANDATIONS

Ce travail de recherche visait à lever le principal verrou numérique du code NSMP : le coût prohibitif de la résolution de l'équation de Poisson pour la pression. Pour y répondre, un solveur hybride combinant la décomposition en valeurs singulières aléatoire (RSVD) et un lisseur SOR a été développé et implémenté.

Les résultats obtenus sur le cas de référence du vortex de Taylor-Green confirment la validité de cette approche. L'étude de convergence a prouvé que l'utilisation d'une approximation de rang faible en début de résolution ne détériore pas la précision finale du calcul, le solveur ayant un ordre de convergence spatiale de 2, comme MultiSOR. Les tests de performance réalisés sur le cas TGV et sur un cas 3D réel avec la plateforme Narval ont démontré qu'un seuil d'énergie de 75% offrait de très bonnes performances, avec une accélération significative par rapport à l'algorithme MultiSOR classique en fournissant une approximation robuste de la solution, ce qui réduit drastiquement l'effort requis par SOR. Par ailleurs, l'analyse de l'efficacité montre que le solveur hybride maintient un rendement des ressources nettement supérieur à celui du MultiSOR sur la quasi-totalité de la plage de test.

En somme, l'apport de la RSVD réside dans sa capacité à fournir une approximation initiale globalement proche de la solution, déchargeant ainsi le solveur SOR d'une part importante de l'effort de calcul. Cette synergie permet d'exploiter la rapidité de la réduction de rang pour traiter la structure dominante du champ de pression, tout en garantissant la convergence vers le système exact grâce au raffinement itératif final.

Bien que les performances soient prometteuses, l'exploitation du plein potentiel de ce solveur hybride appelle à des investigations approfondies, notamment au sujet du niveau d'énergie, il serait intéressant de mener une campagne de tests complémentaires visant à quantifier précisément l'impact de ce paramètre sur une plus grande diversité de configurations. Une telle étude permettrait d'établir des réglages optimaux en fonction des caractéristiques de l'écoulement

et de la finesse du maillage assurant ainsi une robustesse maximale du code dans des contextes variés.

Sur le plan de l'architecture logicielle, une amélioration de l'efficacité parallèle pourrait être obtenue par une refonte ciblée du code NSMP. Actuellement conçu pour des méthodes itératives classiques, certaines de ses structures internes gagneraient à être adaptées spécifiquement aux opérations de projection et aux produits matrice-vecteur massifs propres à la RSVD.

Par ailleurs, l'optimisation des performances en environnement distribué pourrait bénéficier de l'intégration de techniques de coloration des cellules, à l'instar de la stratégie employée dans le solveur MultiSOR. L'application d'un tel schéma de coloration au solveur SOR de notre méthode hybride permettrait de découpler les dépendances de données entre cellules voisines, facilitant ainsi une mise à jour simultanée des inconnues sur l'ensemble des processeurs. Cette approche réduirait les temps d'attente synchrones et accélérerait la convergence locale, rendant le solveur plus compétitif face aux augmentations massives du nombre de cœurs de calcul.

Enfin, l'application de ce solveur hybride à des cas plus complexes, par exemple avec plusieurs piles de pont, représentera une étape cruciale. Tester la résilience de la RSVD face à de tels défis permettra de confirmer si le gain de performance observé se maintient face aux enjeux concrets de la modélisation environnementale à grande échelle.

ANNEXE I

DÉFINITION DU CONDITIONNEMENT

1. Conditionnement d'une Matrice

Avant de définir précisément le conditionnement, rappelons que la norme 2 d'un vecteur $x \in \mathbb{R}^n$ est donnée par :

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}, \tag{A I-1}$$

De même, la norme 2 d'une matrice $A \in \mathbb{R}^{m \times n}$ est définie par :

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}, \tag{A I-2}$$

De plus, la norme 2 d'une matrice est égale à sa plus grande valeur singulière :

$$\|A\|_2 = \sigma_{\max}(A), \tag{A I-3}$$

où $\sigma_{\max}(A)$ désigne la plus grande valeur singulière de A , c'est-à-dire la racine carrée de la plus grande valeur propre de $A^\top A$ (ou de $AA^\top$).

Le conditionnement d'une matrice inversible A est défini comme :

$$\kappa(A) = \|A\|_2 \cdot \|A^{-1}\|_2. \tag{A I-4}$$

Lorsque A n'est pas inversible, le conditionnement n'est pas fini.
On sait (par la définition et la décomposition en valeurs singulières – SVD (*Singular Value Decomposition*) que :

$$\|A\|_2 = \sigma_{\max}(A) \quad \text{et} \quad \|A^{-1}\|_2 = \frac{1}{\sigma_{\min}(A)}, \text{ si A inversible.} \tag{A I-5}$$

Ainsi,

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}. \tag{A I-6}$$

Maintenant, observons que :

- On a $A = U\Sigma V^{\top}$, la décomposition en valeurs singulières (SVD), pour tout A,
- Alors $A^{-1} = V\Sigma^{-1}U^{\top}$ si A est inversible (c'est-à-dire que toutes les valeurs singulières $\sigma_i > 0$),
- Et donc Σ^{-1} est la matrice diagonale contenant les inverses $1/\sigma_i$ sur sa diagonale.

La méthode de décomposition en valeurs singulières (SVD) est présentée plus en détail dans la sous-partie 2.3.7. Par conséquent :

$$\sigma_{\max}(A^{-1}) = \frac{1}{\sigma_{\min}(A)}. \tag{A I-7}$$

où $\sigma_{\min}(A)$ est la plus petite valeur singulière non nulle.

D'où :

$$\kappa(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}, \tag{A I-8}$$

Le conditionnement mesure la sensibilité de la solution x du système $Ax = b$ aux erreurs ou perturbations sur b. Si $\kappa(A)$ est élevé, de petites erreurs sur b peuvent provoquer de grandes erreurs sur x.

Exemples illustratifs

Premier exemple : matrice bien conditionnée

Considérons :

$$A = \begin{bmatrix} 1 & 2 \\ 3 & -2 \end{bmatrix} \quad \text{et} \quad b = \begin{bmatrix} 2 \\ 2 \end{bmatrix}. \tag{A I-9}$$

La solution exacte est $x_1 = 1,\ x_2 = 0.5$.

L'inverse de A est :

$$A^{-1} = \frac{1}{-8}\begin{bmatrix} -2 & -2 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.25 \\ 0.375 & -0.125 \end{bmatrix}. \tag{A I-10}$$

Nous calculons maintenant $A^\top A$:

$$A^\top A = \begin{bmatrix} 1 & 3 \\ 2 & -2 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 10 & -4 \\ -4 & 8 \end{bmatrix}. \tag{A I-11}$$

Le polynôme caractéristique est donné par :

$$\det(A^\top A - \lambda I) = \lambda^2 - 18\lambda + 64 = 0. \tag{A I-12}$$

Les racines sont :

$$\lambda_{1,2} = 9 \pm \sqrt{17}. \tag{A I-13}$$

Les valeurs singulières de A sont donc :

$$\sigma_1 = \sqrt{\lambda_1} \approx 3.623, \quad \sigma_2 = \sqrt{\lambda_2} \approx 2.208. \tag{A I-14}$$

Ainsi :

$$\|A\|_2 = \sigma_{\max} \approx 3.623, \quad \|A^{-1}\|_2 = \frac{1}{\sigma_{\min}} = \frac{1}{9 - \sqrt{17}} \approx 0.453. \tag{A I-15}$$

Le conditionnement de A est alors :

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \approx 1.641. \tag{A I-16}$$

Le système est donc bien conditionné : de petites perturbations de b induisent de faibles perturbations de x.

Deuxième exemple : matrice mal conditionnée

Considérons :

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1.001 \end{bmatrix} \quad \text{et} \quad b = \begin{bmatrix} 2 \\ 2 \end{bmatrix}. \tag{A I-17}$$

La solution exacte est $x_1 = 2$, $x_2 = 0$.

L'inverse de A est :

$$A^{-1} = \frac{1}{0.001} \begin{bmatrix} 1.001 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1001 & -1000 \\ -1000 & 1000 \end{bmatrix}. \tag{A I-18}$$

Nous calculons maintenant $A^\top A$:

$$A^\top A = \begin{bmatrix} 1 & 1 \\ 1 & 1.001 \end{bmatrix}^\top \begin{bmatrix} 1 & 1 \\ 1 & 1.001 \end{bmatrix} = \begin{bmatrix} 2 & 2.001 \\ 2.001 & 2.002001 \end{bmatrix}. \tag{A I-19}$$

Le polynôme caractéristique est donné par :

$$\det(A^\top A - \lambda I) = \lambda^2 - 4.002001\lambda + 0.000001 = 0. \tag{A I-20}$$

Les racines sont :

$$\lambda_{1,2} = 2.0010005 \pm \sqrt{(2.0010005)^2 - 0.000001}. \tag{A I-21}$$

Les valeurs singulières de A sont donc :

$$\sigma_1 = \sqrt{\lambda_1} \approx 2.0005, \quad \sigma_2 = \sqrt{\lambda_2} \approx 0.0005. \tag{A I-22}$$

Ainsi :

$$\|A\|_2 = \sigma_{\max} \approx 2.0005, \quad \|A^{-1}\|_2 = \frac{1}{\sigma_{\min}} = \frac{1}{\sqrt{\lambda_2}} \approx 2000.5 \tag{A I-23}$$

Le conditionnement de A est alors :

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \approx 1000 \tag{A I-24}$$

Le système est donc très mal conditionné : une perturbation minime sur b provoque une grande erreur sur la solution x.

2. Création de matrices avec un conditionnement donné

Pour tester la robustesse des méthodes numériques, on peut construire des matrices de conditionnement fixé $\kappa(A) = \gamma$:

1. Fixer $\sigma_{\max} = 1$ et $\sigma_{\min} = 1/\gamma$.
2. Générer des matrices orthonormées U et V aléatoirement.
3. Former :

$$A = U\Sigma V^\top. \tag{A I-25}$$

Ainsi, $\kappa(A) = \sigma_{\max}/\sigma_{\min} = \gamma$, quelle que soit la forme de U et V (tant qu'elles sont orthogonales).

ANNEXE II

EXEMPLE NUMÉRIQUE DE RSVD APPLIQUÉ À UNE MATRICE $m \times n$

On considère la matrice $A \in \mathbb{R}^{6\times4}$ et un rang cible $k = 2$. La matrice de test $\Omega \in \mathbb{R}^{4\times2}$ est gaussienne aléatoire, chaque coefficient étant tiré indépendamment suivant $\mathcal{N}(0, 1)$. Les valeurs seront arrondies à 10^{-3}.

$$A = \begin{bmatrix} 20.0 & 22.0 & 25.0 & 28.0 \\ 19.5 & 21.8 & 24.9 & 27.8 \\ 19.0 & 21.5 & 24.7 & 27.5 \\ 18.8 & 21.3 & 24.5 & 27.3 \\ 18.5 & 21.0 & 24.2 & 27.0 \\ 18.0 & 20.8 & 24.0 & 26.8 \end{bmatrix}, \quad \Omega = \begin{bmatrix} 0.497 & -0.138 \\ 0.648 & 1.523 \\ -0.234 & -0.234 \\ 1.579 & 0.767 \end{bmatrix}. \tag{A II-1}$$

Esquisse $Y = A\Omega$ et QR réduit $Y = QR$:

$$Y = \begin{bmatrix} 62.548 & 46.376 \\ 61.877 & 46.011 \\ 61.008 & 45.439 \\ 60.510 & 45.056 \\ 59.763 & 44.480 \\ 59.116 & 44.138 \end{bmatrix}, \quad Q = \begin{bmatrix} -0.420 & 0.734 \\ -0.415 & 0.159 \\ -0.410 & -0.168 \\ -0.406 & -0.112 \\ -0.401 & -0.027 \\ -0.397 & -0.628 \end{bmatrix}, \quad R = \begin{bmatrix} -148.965 & -110.856 \\ 0.000 & -0.232 \end{bmatrix}. \tag{A II-2}$$

Projection réduite $B = Q^\top A$ et SVD :

$$B = \begin{bmatrix} -46.481 & -52.429 & -60.141 & -67.123 \\ 0.685 & -0.007 & -0.303 & -0.257 \end{bmatrix}, \tag{A II-3}$$

$$\tilde{U} = \begin{bmatrix} -1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 114.156 & 0 \\ 0 & 0.791 \end{bmatrix}, \tag{A II-4}$$

$$V^\top = \begin{bmatrix} 0.407 & 0.459 & 0.527 & 0.588 \\ 0.884 & 0.012 & -0.360 & -0.299 \end{bmatrix}. \tag{A II-5}$$

Remontée $U = Q\tilde{U}$ et approximation finale :

$$U = \begin{bmatrix} 0.420 & 0.734 \\ 0.415 & 0.159 \\ 0.410 & -0.168 \\ 0.406 & -0.112 \\ 0.401 & -0.027 \\ 0.397 & -0.628 \end{bmatrix}, \qquad A_{\text{approx}} = \begin{bmatrix} 20.019 & 22.009 & 25.029 & 27.995 \\ 19.416 & 21.777 & 24.933 & 27.841 \\ 18.921 & 21.473 & 24.681 & 27.533 \\ 18.804 & 21.298 & 24.463 & 27.294 \\ 18.629 & 21.034 & 24.136 & 26.936 \\ 18.016 & 20.811 & 24.057 & 26.799 \end{bmatrix}. \tag{A II-6}$$

Erreur relative (norme de Frobenius) :

$$\frac{\|A - A_{\text{approx}}\|_F}{\|A\|_F} \approx 1.98 \times 10^{-3} \quad \text{(soit 0.198\%)}. \tag{A II-7}$$

BIBLIOGRAPHIE

Aanjaneya, M., Gao, M., Liu, H., Batty, C. & Sifakis, E. (2017). Power Diagrams and Connectivity-Aware Gauss-Seidel Smoothing for Accurate Hot-Start VOS Fluid Simulation. *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–11.

Boivin, S., Cayré, F. & Hérard, J.-M. (2000). A finite volume method to solve the Navier–Stokes equations for incompressible flows on unstructured meshes. *International Journal of Thermal Sciences*, 39, 806–825.

Bolis, A., Cantwell, C. D., Moxey, D., Serson, D. & Sherwin, S. J. (2016). An adaptable parallel algorithm for DNS of incompressible turbulent flows using a Fourier spectral/hp element method. *Computer Physics Communications*, 206, 17–25.

Busto, S., Dumbser, M. & Río-Martín, L. (2023). An Arbitrary-Lagrangian-Eulerian hybrid finite volume/finite element method on moving unstructured meshes for the Navier-Stokes equations. *Applied Mathematics and Computation*, 437, 127539.

Buttari, A., Dongarra, J., Langou, J., Luszczek, P. & Kurzak, J. (2007). Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. *International Journal of High Performance Computing Applications*, 21(4), 457–466.

Cheng, M., Wang, G. & Mian, H. H. (2015). Reordering of hybrid unstructured grids for an implicit Navier-Stokes solver based on OpenMP parallelization. *Computers & Fluids*, 110, 245–253.

Chorin, A. J. (1968). Numerical Solution of the Navier–Stokes Equations. *Mathematics of Computation*, 22(104), 745–762.

Chowdhury, I. & L'Excellent, J.-Y. (2010). *Some Experiments and Issues to Exploit Multicore Parallelism in a Distributed-Memory Parallel Sparse Direct Solver* (Rapport n°RR-7411).

Cui, H., Pietrzak, J. D. & Stelling, G. S. (2012). Improved efficiency of a non-hydrostatic, unstructured grid, finite volume model. *Ocean Modelling*, 54–55, 55–67.

Cui, S. & Zhu, J. (2022). A new finite volume multi-resolution central WENO scheme for Navier–Stokes equations on staggered meshes. *Computer Methods in Applied Mechanics and Engineering*, 393, 114822.

Demmel, J. W. (1997). *Applied Numerical Linear Algebra*. Philadelphia : Society for Industrial and Applied Mathematics.

Eichstädt, J., Vymazal, M., Moxey, D. & Peiró, J. (2020). A comparison of the shared-memory parallel programming models OpenMP, OpenACC and Kokkos in the context of implicit solvers for high-order FEM. *Computer Physics Communications*.

Eijkhout, V. (2016). *Parallel Programming in MPI and OpenMP*. Book.

Gander, M. J. & Vandewalle, S. (2007). Analysis of the Parareal Time-Parallel Time-Integration Method. *SIAM Journal on Scientific Computing*, 29(2), 646–607.

Gander, M. J., Magoulès, F. & Nataf, F. (2002). Optimized Schwarz Methods without Overlap for the Helmholtz Equation. *SIAM Journal on Scientific Computing*.

Gander, M. J., Halpern, L. & Magoulès, F. (2006). Optimized Schwarz Methods for the Helmholtz Equation. *Lecture Notes in Computational Science and Engineering*, 00, 1–6.

Gava, F. (2022). *Optimisation des performances parallèles d'un solveur CFD pour les plateformes de calcul émergentes*. (Thèse de doctorat, Thèse de doctorat).

Golub, G. H. & Van Loan, C. F. (2013). *Matrix Computations* (éd. 4). Johns Hopkins University Press.

Grigori, L., Demmel, J. W. & Xiang, H. (2011). *CALU : A Communication Optimal LU Factorization Algorithm* (Rapport n°UCB/EECS-2010-29).

Guermond, J.-L. (1996). Some implementations of projection methods for Navier-Stokes equations. *RAIRO Modélisation Mathématique et Analyse Numérique*, 30(5), 637–667.

Guermond, J.-L. & Shen, J. (2003). On the error estimates for the rotational pressure correction projection. *Mathematics of Computation*, 41(1), 1719–1741.

Guermond, J.-L. & Shen, J. (2006). An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*.

Hammouti. (2009). *Simulation numérique directe en différence finie de l'écoulement d'un fluide incompressible en présence d'interfaces rigides*. (Thèse de doctorat, École des Ponts ParisTech).

Hanna, B. N. et al. (2020). Machine-learning based error prediction approach for coarse-grid Computational Fluid Dynamics (CG-CFD). *Progress in Nuclear Energy*, 118, 103140.

Li, T., Zou, Y., Zou, S., Chang, X., Zhang, L. & Deng, X. (2025). Learning to solve PDEs with finite volume-informed neural networks in a data-free approach. *Journal of Computational Physics*, 530, 113919.

Liu, Y. Y., Shu, C., Zhang, H. W., Yang, L. M. & Lee, C. (2021). An efficient high-order least square-based finite difference-finite volume method for compressible Navier-Stokes equations. *Computers and Fluids*, 222, 104926.

Liu, Z., Miao, S. & Zhang, Z. (2023). A family of edge-centered finite volume schemes for heterogeneous and anisotropic diffusion problems on unstructured meshes. *Computers & Mathematics with Applications*, 146, 165–175.

Luckáč, M. et al. (1996). *Massively Parallel Poisson and QR*.

Ma, G., Shi, F. & Kirby, J. T. (2012). Shock-capturing non-hydrostatic model for fully dispersive surface wave processes. *Ocean Modelling*, 43–44, 22–35.

Margenberg, N., Jendersie, R., Lessig, C. & Richter, T. (2024). DNN-MG : A hybrid neural network/finite element method with applications to 3D simulations of the Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 420, 116692.

Martinsson, P.-G., Rokhlin, V. & Tygert, M. (2011). A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30, 47–68.

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J. & Sifakis, E. (2011). A high-performance multigrid strategy for next-generation cloth simulation. *ACM Transactions on Graphics (TOG)*, 30(4), 1–10.

Nguyen, D. H., Levy, F., Pham Van Bang, D., Guillou, S., Nguyen, K. D. & Chauchat, J. (2012). Simulation of dredged sediment releases into homogeneous water using a two-phase model. *Ocean Modelling*, 48, 102–112.

Notay, Y. & Napov, A. (2015). A massively parallel solver for discrete Poisson-like problems. *Journal of Computational Physics*, 281, 237–250.

Novati, G. & Koumoutsakos, P. (2021). Automating Turbulence Modeling by Multi-Agent Reinforcement Learning. *Nature Machine Intelligence*.

Perron, S., Boivin, S. & Hérard, J.-M. (2004). A finite volume method to solve the 3D Navier–Stokes equations on unstructured collocated meshes. *Computers & Fluids*, 33, 1305–1333.

Setaluri, R., Aanjaneya, M., Bauer, S. & Fedkiw, R. (2014). A SPGrid based Schur complement solver for real-time cloth simulation. *ACM Transactions on Graphics (TOG)*, 33(4), 1–10.

Shah, M., Cohen, J., Tarini, M. & Panozzo, D. (2018). Distributing and Load Balancing Sparse Fluid Simulations. *Computer Graphics Forum*, 37(2), 415–425.

Uh Zapata, M., Zhang, W., Pham Van Bang, D. & Nguyen, K. D. (2019). A parallel second-order unstructured finite volume method for 3D free-surface flows using a σ coordinate. *Computers and Fluids*, 190, 15–29.

Uh Zapata, M. A. (2012). 3D Program. *Saint-Venant Laboratory for Hydraulics, École des Ponts ParisTech*.

Uh Zapata, M. A., Pham Van Bang, D. & Nguyen, K. D. (2016). Parallel SOR methods with a parabolic-diffusion acceleration technique for solving an unstructured-grid Poisson equation on 3D arbitrary geometries. *International Journal of Computational Fluid Dynamics*.

Uh Zapata, M. A., Hernández-López, F. J. & Itzá Balam, R. (2023). A parallel unstructured multi-color SOR solver for 3D Navier–Stokes equations on graphics processing units. *Computers & Structures*, 286, 106976.

Zhang, W., Uh Zapata, M., Bai, X., Pham Van Bang, D. & Nguyen, K. D. (2020). An unstructured finite volume method based on the projection method combined momentum interpolation with a central scheme for three-dimensional nonhydrostatic turbulent flows. *European Journal of Mechanics B/Fluids*, 84, 164–185.