THESE

présentée à

L'UNIVERSITE DE LA MEDITERRANEE

(AIX-MARSEILLE II)

par **Estelle ONUPHRE**

pour obtenir le grade de

DOCTEUR

SPECIALITE : MECANIQUE DES FLUIDES

DEVELOPPEMENT D'UN SOLVEUR PARALLELE DANS LE CADRE DE LA METHODE DES ELEMENTS FINIS APPLICATION AUX EQUATIONS DE NAVIER-STOKES

EXEMPLAIRE PROVISOIRE

Commission d'examen :

D. ZEITOUN	Rapporteurs
A. BEN HADID	
C. KHARIF	Examinateurs
B. ROUX	
A. CHAMBAREL	

Travail effectué à l'Institut de Recherche sur les Phénomènes Hors Equilibre,
I.R.P.H.E., Université de la Méditerranée - Aix – Marseille II.

TABLE DES MATIERES

Chapitre 3 : Méthodes Itératives et Calcul Parallèle

Chapitre 4 : Validation du Solveur Parallèle

Notations

Les vecteurs ainsi que les matrices sont notés en caractère gras.

Le symbole < > représente un vecteur ligne et le symbole { } un vecteur colonne.

k_i Matrices d'inertie élémentaires

k_{cv} Matrices de convection élémentaires

k_g Matrices de gradient élémentaires

k_{ct} Matrices de continuité élémentaires

k_d Matrices de dissipation élémentaires

m Matrices masse élémentaires

K Matrice de rigidité

$\mathbf{K_T}$ Matrice tangente

$\tilde{\mathbf{K}}$ Matrice de l'itération courante

$\tilde{\mathbf{K}}_\varepsilon$ Matrice de pénalité

M Matrice masse

F Vecteur des sollicitations

u vecteur vitesse

C Constante

D Largeur de la cavité entraînée

u_1 Première composante du vecteur **u**

u_2 Deuxième composante du vecteur **u**

u_3 Troisième composante du vecteur **u**

p_g Pression Motrice

T Température du fluide

C_p Chaleur massique à pression constante [$J.kg^{-1}.K^{-1}$]

P_2-P_1 Fonctions d'interpolation quadratique pour la vitesse et la température et linéaire pour la pression

U Solution exacte ou vitesse de référence

p Pression adimensionnée $p=p_g/\rho U^2$

Re Nombre de Reynolds [Re=UD/v]

$\bar{\bar{I}}$	Tenseur identité
$\bar{\bar{\sigma}}$	Tenseur total des contraintes
ε	Paramètre de pénalité
Φ	Fonction de dissipation
λ	Conductivité thermique [$W.m^{-1}.K^{-1}$]
μ	Viscosité de cisaillement ou viscosité dynamique
ν	Viscosité cinématique [$m2.s^{-1}$]
ρ	Masse volumique [$kg.m^{-3}$]
ω	Paramètre de relaxation

Introduction

Introduction

Observer, rendre compte pour pouvoir prédire, telle est la finalité de la Mécanique des Fluides dont l'objet est d'étudier tout phénomène mettant en jeu un gaz ou un liquide en mouvement. Au quotidien, l'occasion est offerte à l'Homme d'observer d'innombrables phénomènes relevant de cette discipline. A cet égard, citons à titre d'exemple la fumée s'échappant d'une cheminée d'usine ou plus simplement d'une cigarette, un liquide contenu dans un récipient quelconque et porté à ébullition ou encore l'eau s'écoulant d'un robinet. La plupart de ces phénomènes souvent perçus comme étant simples car familiers, masquent en réalité des mécanismes physiques fort complexes dont la compréhension n'est pas toujours aisée et demeure parfois partielle. Or, le niveau de compréhension d'un phénomène conditionne la fidélité avec laquelle l'Homme est en mesure de le décrire et, lorsque cela est possible, de le prédire.

Pendant longtemps, l'évolution de ce niveau de compréhension s'est appuyée sur une utilisation conjointe de :

- l'approche théorique élaborée à partir d'un modèle (éventuellement semi-empirique) souvent très simplifié afin qu'il puisse aboutir à une solution analytique,
- l'approche expérimentale.

Ces deux approches ont montré leurs limitations, la première en raison d'hypothèses simplificatrices souvent trop sévères pour rendre correctement compte de la réalité et dont le champ d'application est, de ce fait, majoritairement confiné à des problèmes relativement académiques, quant à la seconde en raison d'une limitation technologique au niveau instrumental (coût d'exploitation élevé, dispositif de mesure intrusif, grandeurs étudiées d'ordre infinitésimal, zones de mesure inaccessibles,...).

Avec l'avènement des moyens informatiques, une nouvelle voie s'est ouverte : la simulation numérique. Cette dernière apporte un remède à certaines des restrictions précitées, en particulier à celles liées aux techniques de mesure. Elle a surtout permis d'appréhender des problèmes plus proches de la réalité que via l'approche théorique, en autorisant un degré de sophistication beaucoup plus élevé au niveau des modèles numériques. Elle a, de ce fait, marqué le début d'une véritable accélération de la progression des connaissances, tout particulièrement dans le domaine de la Dynamique des Fluides. En effet, les équations fondamentales de cette discipline ont été établies depuis fort longtemps sur la base de

principes de conservation. Toutefois, elles sont demeurées quelque peu lettre morte jusqu'à ce que les progrès réalisés en matière d'informatique aient fourni des outils permettant d'accéder à des solutions approchées.

Depuis, de concert avec l'augmentation de la puissance des calculateurs, la Mécanique des Fluides Numérique s'est orientée vers des problèmes physiques complexes caractérisés par des modèles mathématiques de plus en plus élaborés, prenant en compte le couplage de phénomènes non-linéaires dans des géométries compliquées. La mise en œuvre de telles simulations se heurte toutefois à des problèmes d'espace mémoire insuffisant, et nécessite souvent des temps de calcul prohibitifs (coût, temps de restitution irréaliste,...). Or, se basant sur l'évolution technologique actuelle, de nombreux auteurs pensent raisonnablement que les calculateurs séquentiels (monoprocesseurs) vont sous peu atteindre leur maximum de puissance et ne pourront plus répondre aux besoins des universitaires et des industriels. Face à ce constat, les scientifiques cherchent de nouvelles voies pour accroître leur puissance de calcul. Ceci explique l'émergence du *parallélisme* (possibilité d'effectuer plusieurs tâches simultanément) dans les architectures des ordinateurs modernes, comme un moyen de calculer plus vite (réduire les temps de restitution) et plus gros (disposer de plus d'espace mémoire).

Concrètement, les applications de cette récente technologie sont multiples :

* l'exploitation de ressources informatiques hétérogènes à travers des réseaux de communication,
* la génération de maillages à géométrie très complexe, et/ou hétérogènes (morcellement de grille),
* la prise en compte de la coexistence de différents modèles mathématiques et/ou numériques au sein d'un même domaine de calcul,
* le couplage de codes,...

Toutefois, force est de constater que le parallélisme est très peu exploité dans le domaine de l'industrie. Le principal obstacle à son expansion réside dans un manque de convivialité et de standards au niveau des plates-formes actuellement sur le marché, hérité d'un décalage entre l'évolution rapide de l'architecture de ces machines et celle beaucoup plus lente des environnements de développement. De ce fait, la mise en œuvre du calcul parallèle dans des domaines tels que la Mécanique des Fluides, demeure pour l'instant une entreprise ardue qui incombe, dans le cas général, à l'utilisateur. Le défi que de nombreux scientifiques ont

soulevé depuis quelques années est de minimiser au maximum, voire totalement supprimer, cette partie là.

Dans ce contexte, l'efficacité du parallélisme est subordonnée à la définition d'un champ précis d'application. En ce qui concerne le présent travail, ce dernier consiste en la résolution de problèmes relevant de la Dynamique des Fluides. La finalité de cette étude réside dans *l'élaboration d'un solveur itératif parallèle pour la résolution des équations de Navier-Stokes par la Méthode aux Éléments Finis* (M.E.F.). L'absence de travaux antérieurs au sein de l'équipe, implique un travail de mise en place du sujet comprenant :

* Le choix d'une approche du problème,
* La définition d'une procédure numérique concrétisant cette approche,
* La validation de cette procédure par une série d'essais numériques pour lesquels la littérature fournit des solutions de référence.

Le plan du présent mémoire s'articule autour des ces trois principaux axes. Un tour d'horizon rapide du contexte de l'étude est réalisé dans le chapitre 1, à l'issue duquel sont définis les objectifs et les orientations du présent travail. Les approches numériques ainsi que l'environnement informatique sont décrits au chapitre 2. Une présentation des détails de l'élaboration du solveur parallèle est effectuée au chapitre 3. Les résultats numériques issus de la mise en œuvre de ce solveur sont exposés et analysés dans le chapitre 4.

Chapitre 1

Contexte de l'étude

Chapitre 1 : Contexte de l'étude

1.1. Cadre général de l'étude

La présente étude concerne l'application du Calcul Parallèle à la simulation numérique par éléments finis d'écoulements de fluides visqueux newtoniens. Nous nous intéressons en particulier à la résolution des équations de Navier-Stokes en variables primitives.

La Mécanique des Fluides Numérique est un champ d'application pluridisciplinaire car elle est au carrefour de la physique, des mathématiques, des méthodes numériques et de l'informatique. En effet, la simulation numérique d'un problème physique s'articule autour de deux aspects : la modélisation numérique du problème et l'implémentation de ce modèle sur un calculateur.

La modélisation numérique d'un problème physique se décompose principalement en trois étapes. La première est une *description physique* complète du problème, basée sur l'identification de l'ensemble des phénomènes physiques mis en jeu (variables, propriétés physiques, lois, sollicitations, etc.). La seconde consiste à générer une *image mathématique* de cette description (mise en équations). La troisième étape est la transformation du modèle mathématique en un *modèle numérique*. Il s'agit d'effectuer le passage d'un espace mathématique continu à un espace discret via une discrétisation analytique et géométrique du modèle mathématique, puis de sélectionner la ou les méthodes permettant une résolution numérique du problème discret.

L'implémentation du modèle numérique sur un calculateur implique l'écriture d'un algorithme de résolution, le choix d'une machine hôte adaptée à la mise en œuvre de cet algorithme, ainsi que d'un paradigme et d'un langage de programmation, d'une stratégie de stockage des données en mémoire, etc.

Le principal intérêt de notre étude réside essentiellement dans l'application d'algorithmes parallèles à la résolution de divers problèmes de Mécanique des Fluides. Ce faisant, les étapes concernant la description physique et la modélisation mathématique de ces problèmes seront très peu traitées.

La description analytique utilisée se base sur la méthode des éléments finis.

1.1.1. Mécanique des Fluides et Eléments Finis

La Méthode des Eléments Finis a été développée dans les années 50 afin de résoudre des problèmes de mécanique des structures et du solide. Les nouveautés par rapport à la méthode des différences finies sont la possibilité de traiter des problèmes à géométrie complexe et à maillage non structuré, et celle d'introduire « naturellement » certaines conditions aux limites dans la formulation intégrale du problème. En outre, cette méthode bénéficie d'une assise mathématique solide. Ces avantages ont fait son succès aussi bien dans le domaine de la recherche que dans celui de l'ingénierie et de l'industrie, et motivent en partie notre choix. Une présentation de cette méthode est proposée un peu plus loin dans ce chapitre. Seules les grandes lignes sont abordées, de nombreux ouvrages étant disponibles sur le sujet [46, 7].

Au tout début des années 70, sous l'impulsion entre autres de Oden[26] et Hood[18], l'utilisation des éléments finis est peu à peu étendue à la Mécanique des Fluides. La méthode est appliquée à la résolution des équations de Navier-Stokes. Des astuces sont utilisées pour éliminer l'équation de continuité, et la pression est déterminée par résolution d'une équation de type Poisson. Lors de ces premières tentatives, les résultats obtenus se révèlent être décevants, en particulier dans le cas d'écoulements dominés par des phénomènes convectifs. Il faut attendre 1978 pour que des auteurs tels que Hughes et al[19] mettent en cause le caractère non symétrique des opérateurs dans de tels cas et suggèrent le développement de nouvelles techniques adaptées à leur traitement.

Aujourd'hui, toutes les procédures de simulation numérique d'écoulements de fluides s'articulent autour de deux choix. Le premier s'effectue au niveau de la modélisation mathématique du problème traité entre un schéma couplé ou un schéma découplé. Le second intervient au niveau du modèle numérique entre un solveur direct ou itératif. Potentiellement, il existe donc quatre voies possibles que nous résumons par : couplé – direct, couplé – itératif, découplé – direct et découplé – itératif.

Afin de mieux expliciter les raisons qui nous ont conduites à choisir un schéma de type couplé – itératif, nous présentons tout d'abord succinctement les principes de base de ces différentes approches.

1. Résolution en couplé – direct

Dès 1973, Taylor et Hood[35] introduisent la formulation en variables primitives basée sur un schéma couplé pour des écoulements de fluides incompressibles, et l'associent à une résolution par méthode directe. De nombreux travaux ont révélé que, dans le cadre de la simulation

d'écoulements laminaires bidimensionnels de taille modérée, une telle association conduit à des algorithmes à convergence rapide et résistant bien à des variations des paramètres physiques du fluide[40]. Malheureusement, les méthodes dites directes nécessitent le stockage en mémoire de la matrice du système à résoudre. Il en résulte une limitation très sévère au niveau de la taille des problèmes potentiellement traitables, qui prohibe l'application de telles méthodes à la plupart des problèmes de Mécanique des Fluides réels.

Face à ce constat, des études ont été menées pour tenter de réduire l'espace mémoire requis tout en maintenant l'emploi de solveurs directs. Dans l'une d'elles menée en différences finies, Braaten et Patankar[4] (1989) ont montré que l'utilisation d'une fonction de pénalité* permettait de réduire d'environ un tiers l'encombrement mémoire grâce à l'élimination de la pression dans les équations du mouvement, tout en préservant les qualités de l'algorithme. Malgré tout, ceci reste insuffisant pour des problèmes de grande taille et a fortiori lorsque la matrice est issue d'une discrétisation par éléments finis.

2. *Résolution en couplé – itératif*

La modélisation de l'écoulement d'un fluide conduit très souvent à des systèmes algébriques linéaires, soit de par la nature même des équations aux dérivées partielles régissant le problème (dans le cas, par exemple, des écoulements de Stokes), soit suite à une linéarisation (traditionnellement Picard ou Newton†). Lorsque la taille du système issu d'un schéma couplé interdit la mise en œuvre d'une méthode de résolution directe, une solution réside dans l'utilisation d'un *méthode de résolution itérative*.

La méthode itérative de base (Richardson) est connue pour son faible taux de convergence. Pour améliorer ce taux, l'emploi de préconditionneurs (Jacobi, Gauss-Seidel, SSOR, ILU,...) est courant. Si cela ne suffit pas, il existe aujourd'hui des techniques d'accélération[21] qui, appliquées à la méthode itérative de base, aboutissent à des méthodes de type gradient conjugué (CG) ou résidu conjugué (CR). Ces méthodes sont parfois rencontrées dans la littérature sous l'appellation anglo-saxonne *matrix-free solvers* car, contrairement aux méthodes directes, elles n'impliquent pas obligatoirement le stockage complet de la matrice globale.

Ces méthodes sont particulièrement efficaces pour la résolution de systèmes linéaires bien conditionnés (matrice définie positive, diagonale principale dominante). Par exemple, on peut montrer que pour des problèmes à matrice symétrique définie positive, la méthode du gradient conjugué introduite en 1971 est la technique optimale pour accélérer la méthode de Richardson.

---

* Cf. paragraphe 2.1.2.4

Malheureusement, les performances de ce type de méthode sont très affectées par la perte des propriétés susdites. Or, Engelman et Hasbani[6] (1988) ont démontré que la discrétisation par éléments finis de Galerkin des équations de Navier-Stokes produit des systèmes à matrice non symétrique particulièrement mal conditionnée, que la formulation soit mixte ou pénalisée. De ce fait, le choix d'un bon préconditionneur afin d'améliorer le conditionnement de la matrice devient déterminant. C'est pourquoi l'élaboration de bons préconditionneurs pour de telles méthodes est un secteur de recherche actuellement très actif.

3. Schémas découplés

L'utilisation de schémas découplés est aujourd'hui une approche classique pour appréhender les problèmes rencontrés dans le monde réel (3D, turbulents, instationnaires) en raison du très grand nombre de degrés de liberté à traiter. Il s'agit de manipuler les équations du mouvement afin de pouvoir résoudre le champ de pression séparément des autres champs. On obtient des procédures de type prédiction - correction (variantes de l'algorithme SIMPLE de Patankar en volumes finis) qui diffèrent suivant la manière dont la pression a été extraite des équations couplées.

En pratique, ces schémas se traduisent typiquement par la résolution séquentielle, à chaque étape de prédiction – correction, de deux systèmes linéaires issus d'une équation de type Poisson pour la pression et d'une équation de type advection – diffusion pour la vitesse. Une résolution très efficace peut être réalisée en utilisant des méthodes de sous-espace de Krylov (introduisant des itérations internes au processus de prédiction – correction). C'est pourquoi l'association d'un schéma découplé et d'un solveur itératif est très fréquente.

En revanche, la nature séquentielle de ce type de procédures conduit à une convergence plus lente que les schémas couplés du fait que le couplage vitesse / pression n'est plus véritablement traité.

4. Comparaison

La simulation numérique a besoin d'algorithmes répondant aux deux critères suivants : une convergence rapide (temps de restitution court) et un stockage mémoire limité. Dans cette optique une certaine hiérarchie s'est instaurée entre les différentes approches précitées sur la base de la taille mémoire du problème à traiter. Si cette taille est faible, la meilleure stratégie demeure l'utilisation d'un schéma de type couplé – direct (factorisation LU, par exemple). Pour

---

[†] Cf. paragraphe 1.3.2.3

une taille supérieure pour laquelle cette stratégie n'est plus envisageable, une résolution de type couplé – itératif préconditionné conduit en principe à une convergence plus rapide qu'un schéma découplé. Par contre, à l'heure actuelle, seuls les schémas découplés permettent d'appréhender la résolution des systèmes de très grande taille tels que ceux issus de la modélisation de phénomènes physiques complexes (turbulence, combustion, ...).

Nous avons souligné le handicap causé par la limitation en capacité mémoire imposée par l'implémentation des algorithmes sur machine et nous avons indiqué deux stratégies traditionnellement employées pour contourner ce problème (schémas découplés ou méthodes itératives). Depuis quelques années, une nouvelle stratégie s'est dessinée : l'utilisation de calculateurs parallèles.

1.1.2. Calcul Parallèle en Mécanique des Fluides

Pour pouvoir utiliser des méthodes directes dans le cadre de schémas couplés, Vanka[39] (1983) a eu l'idée de décomposer le domaine de calcul en tranches avec zones de recouvrement (« *overlapping subdomains* ») et de résoudre les équations tour à tour sur chacun des sous-domaines ainsi définis en appliquant exactement la même méthode directe que pour le problème global. Il a testé sa technique nommée TSDA (*Telescoping SubDomain Analysis*) sur un écoulement turbulent. Le taux de convergence annoncé est comparable à celui de la méthode directe appliquée au problème global alors que l'encombrement mémoire est réduit d'une manière très significative. L'inconvénient d'une telle méthode réside dans le traitement séquentiel des sous – problèmes qui peut résulter en un temps de restitution important.

Vanka a en fait exploité une idée très ancienne redevable à Schwarz (1865) et utilisée à l'origine pour résoudre les équations de type Poisson. Cette idée a ouvert la voie *aux méthodes dites de sous-domaines* ou de *décomposition de domaine*. Depuis l'apparition des machines parallèles qui permettent le traitement simultané de plusieurs opérations grâce à la présence de plusieurs unités de traitement dans une même machine, ces méthodes connaissent un regain d'intérêt.

Nous ne ferons pas ici un exposé sur le Calcul Parallèle dans un contexte général. Étant donnée la richesse de cette discipline et surtout la diversité de ses applications, il ne nous semble pas approprié d'en faire une description exhaustive, autant que faire se peut, dans le cadre de ce mémoire. En outre, cet état de l'art est fait périodiquement par de nombreux auteurs[10]. On s'y référera donc pour de plus amples détails. Néanmoins, le Calcul Parallèle étant une discipline

relativement peu répandue, nous allons présenter quelques notions nécessaires à la bonne compréhension des prochains chapitres.

1.1.2.1. Quelques généralités sur le Calcul Parallèle

1. Principe

Actuellement, nul numéricien n'ignore les problèmes posés par le besoin croissant de puissance de calcul émanant du domaine de l'industrie et de la recherche, en termes de temps d'exécution et d'espace mémoire[44]. Se basant sur l'évolution technologique actuelle, de nombreux auteurs pensent raisonnablement que les calculateurs séquentiels (monoprocesseurs) vont sous peu atteindre leur maximum de puissance et ne pourront plus répondre à cette demande. En effet, la rapidité de communication de l'information est un facteur clef de la conception d'un ordinateur. Or, une limitation majeure dans ce domaine réside en l'impossibilité de dépasser la vitesse de la lumière. Face à ce constat, les scientifiques cherchent de nouvelles voies pour accroître leurs moyens de calcul. Ceci explique l'émergence du ***parallélisme*** dans l'architecture des ordinateurs d'aujourd'hui, comme un moyen de calculer plus c'est à dire plus vite (réduire le délai de restitution) et plus gros (disposer de plus de mémoire).

Le principe du parallélisme s'apparente à l'organisation du travail au sein d'une équipe. Si un seul travailleur ne suffit plus pour mener à bien une tâche, on la confie à une équipe de travailleurs. Charge est alors au chef de cette équipe de distribuer et d'organiser le travail correctement de manière à obtenir le meilleur rendement possible. Pour cela, les deux conditions suivantes doivent être remplies :

A. il faut pouvoir décomposer le travail en un certain nombre de tâches pouvant être effectuées au même moment par les différents membres de l'équipe,

B. il faut assurer un échange d'informations entre ces membres, de manière à ce que chacun d'eux puisse accéder à celles dont il a besoin pour mener à bien la fonction qui lui a été assignée puis, une fois cette fonction remplie, transmettre le résultat obtenu.

De manière similaire, si un seul processeur ne permet pas de réaliser une tâche donnée, l'idée qui vient naturellement est de distribuer le travail inhérent à cette tâche entre plusieurs processeurs accomplissant simultanément la fonction qui leur a été confiée, ce qui implique :

(I) la détermination du degré de parallélisme de cette tâche ($\Leftrightarrow$ condition A),

(II) la conception d'un réseau de communication adapté ($\Leftrightarrow$ condition B),

(III) l'implémentation des tâches parallèles définies en (I) sur ce réseau.

Ainsi sont nées les ***machines parallèles***. Si le principe de base de telles machines tombe sous le sens, leur réalisation n'est elle pas chose aisée autant du point de vue conception de l'architecture de la machine (*hardware*) que du point de vue développement de son environnement d'utilisation (*software*). Examinons de plus près ce problème qui constitue le point de départ du présent travail.

2. Mise en œuvre

Environ cinquante années d'exploitation des calculateurs séquentiels ont permis d'enrichir leur interface machine – utilisateur, offrant à ce dernier un environnement de travail et de nombreux outils de développement qui rendent très confortable l'utilisation de ces machines. Ces améliorations sont allées de concert avec les progrès techniques effectués au niveau hardware. A l'heure actuelle, on ne peut en dire autant des machines parallèles. Ne bénéficiant pas d'autant d'années d'exploitation, il est normal qu'elles n'aient pas atteint le même degré de convivialité. Par contre, plus inquiétant est de constater que les outils d'aide aux développement d'applications parallèles n'ont pas connu la même vitesse de développement que l'architecture de ces machines. Ce fossé est très certainement responsable du fait que le Calcul Parallèle est actuellement très peu répandu dans le domaine industriel. Il reste une affaire de spécialistes.

a) Les différents niveaux de parallélisme

Pour inverser ce processus et mettre le parallélisme à la disposition de tous, l'idéal serait de le rendre transparent pour l'utilisateur en cachant les détails de son implémentation, par exemple dans les sous-couches de la machine. En d'autres termes, il s'agit d'automatiser sa mise en œuvre. Malheureusement ceci soulève un problème de taille. En effet, nous avons vu que « paralléliser » une tache implique en premier lieu de déterminer son degré de parallélisme (I). Or, suivant la tache à accomplir, ce dernier pourra se situer à différents niveaux :

(1) Au niveau du job

→ lancer concurremment plusieurs unités de programme

(2) Au niveau du programme

→ exécuter plusieurs blocs d'instructions en même temps

ou

→ exécuter simultanément entrée/sortie et calcul

(3) Au niveau des instructions (boucles)

→ décomposer une instruction en sous opérations (comme pour le pipelining)

(4) Au niveau arithmétique

→ effectuer des opérations arithmétiques sur des bits en parallèle

Chacun de ces niveaux correspond à une ***granularité*** différente, la granularité d'un programme étant définie par le poids relatif des instructions qui le constituent (en temps CPU ou en temps de restitution). Plus le niveau de parallélisme est élevé, plus le grain est grossier. La granularité est dite forte (cas (1)), moyenne (cas(2)) ou faible (cas (3) et (4)).

En pratique, la prise en charge du parallélisme est pondérée entre le concepteur de l'architecture de la machine (hardware), les concepteurs des logiciels associés tels que le système d'exploitation, les compilateurs ou les langages (software), et l'utilisateur (algorithmes). De la granularité du parallélisme désiré va dépendre cette pondération. En particulier, elle va déterminer le degré d'intervention de l'utilisateur. Par exemple, pour un grain faible tel que (4), le parallélisme est du ressort des deux premiers intervenants exclusivement et peut donc être rendu totalement transparent pour l'utilisateur. Une certaine automatisation est donc possible dans ce cas. Par contre, à des niveaux plus élevés tels que (2) où le parallélisme pourrait être beaucoup plus efficace, l'intervention de l'utilisateur semble incontournable.

En clair, une mise en œuvre vraiment efficace du Calcul Parallèle dépend entièrement de l'application visée et ne peut donc être réalisée sans une connaissance préalable de cette application. Ceci explique la difficulté d'élaborer pour ces machines un environnement de développement standard et convivial indispensable à une expansion dans le domaine industriel et qui, quoiqu'il en soit, ne pourra se faire qu'au prix d'un sacrifice au niveau de l'efficacité du calcul. C'est également la raison pour laquelle il existe une grande variété de calculateurs parallèles dont nous allons à présent donner une classification.

b) Les architectures parallèles

En 1972, M. J. Flynn[13] a établi une taxonomie des architectures parallèles fondée sur les choix faits au niveau du traitement des opérations (les instructions à exécuter) et des données. Parmi elles, on retiendra plus particulièrement les deux catégories suivantes :

- S.I.M.D. (Single Instruction stream / Multiple Data stream)
  La même instruction est exécutée par plusieurs unités de traitement, chaque unité ne sachant traiter qu'une seule instruction à la fois.
- M.I.M.D. (Multiple Instruction stream / Multiple Data stream)

Plusieurs unités de traitement savent traiter des paquets d'instructions pas forcément identiques.

Cette classification macroscopique en cache une autre plus fine basée sur la gestion de la mémoire. Au niveau physique, cette mémoire est soit partagée (S.M. : Shared Memory), soit distribuée (D.M. : Distributed Memory). Dans le premier cas, le parallélisme se fait au travers des instructions. Il ne nécessite donc pas ou peu de gestion de données. Par contre, il implique de reconnaître les blocs d'instructions indépendants et de synchroniser leur traitement. Dans le second cas, la distribution de la mémoire vise à éviter les problèmes d'étranglement au niveau de l'accès mémoire rencontrés dans les machines à mémoire partagée, lorsque le nombre d'unités de traitement devient important. Le parallélisme est alors un parallélisme de données ce qui induit des communications entre unités de traitement.

En combinant ces différents éléments, on obtient une grande variété de machines. Nous citons entre autres :

* Les architectures en pipeline et multipipeline[24]
* Les architectures massivement parallèles
* Les architectures multiprocesseur à mémoire partagée
* Les architectures multiprocesseur à mémoire distribuée

c) Efficacité versus portabilité

Lorsque l'utilisation du Calcul Parallèle est requise, le numéricien doit choisir entre les deux approches opposées suivantes :

- Utiliser au maximum les propriétés spécifiques à un problème et à un calculateur donnés
- Développer des programmes portables pouvant traiter une large classe d'applications

En d'autres termes, cela revient à choisir entre l'efficacité optimale et la portabilité.

La recherche d'une efficacité optimale passe par l'écriture d'un programme informatique spécifique à la simulation d'un problème physique donné sur une machine donnée. Dans cette optique, tout réemploi du programme dans un autre contexte (problème ou machine) que celui pour lequel il a été conçu est exclu. Sa réécriture est systématiquement nécessaire, ce qui se traduit en général en un coût en termes de développements informatiques beaucoup trop important par rapport au profits éventuels. C'est pourquoi, nous ne nous engagerons pas dans cette voie.

Entre l'efficacité maximale et la portabilité, nous choisissons la portabilité.

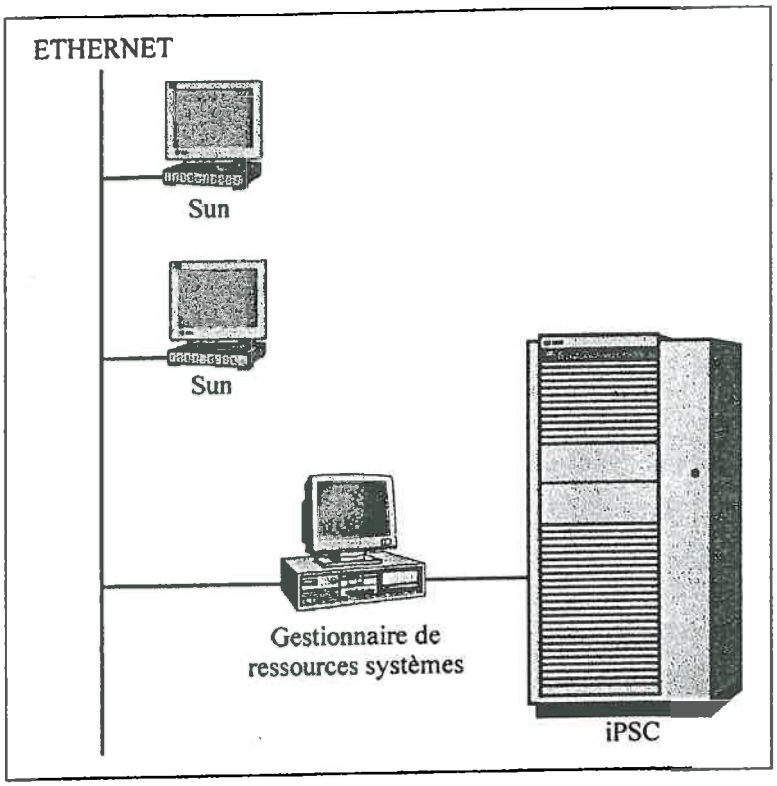


Figure 1.1 : L'iPSC-860

3. Évaluation de performance

La granularité est un premier indicateur d'efficacité. Pour mieux caractériser les performances d'un Calcul Parallèle et établir des comparaisons avec d'autres calculateurs, on introduit les notions d'accélération S (speed-up), et « d'évolutivité » (scalability), traditionnellement définies comme ci-après. En reprenant la nomenclature anglo-saxonne, on désigne par :

- « speed-up » la comparaison entre le temps de calcul sur n processeurs et le temps de calcul sur un seul processeur de la même machine,
- « reference speed-up » la comparaison entre le temps de calcul sur n processeurs d'une machine parallèle et le temps de calcul sur un seul processeur d'une machine de puissance comparable, typiquement un CRAY (nous n'aurons pas la possibilité de le faire ici),
- « scaled speed-up » la comparaison des temps de calcul obtenus en augmentant la taille du problème traité proportionnellement au nombre de processeurs utilisés,

- « scalability » l'influence sur le temps de calcul de l'augmentation de la taille du problème pour un nombre de processeurs fixé, ou de l'augmentation du nombre de processeurs pour un problème de taille fixée.

Étant donnée f la fraction de calcul réalisé en parallèle, la loi d'Amdahl évalue le temps de calcul sur p processeur par la relation

$$t_p = \underbrace{(1-f)\,t_1}_{\substack{travail \\ séquentiel}} + \underbrace{f\,(t_1/p)}_{\substack{travail \\ parallèle}},$$

d'où l'estimation du speed-up suivante :

$$S = \frac{t_1}{t_p} = \frac{p}{f + (1-f)\,p}$$

En résumé, la mise en œuvre du Calcul Parallèle est une tache ardue qui revêt à la fois un aspect hardware et un aspect software; ce dernier incombant souvent en partie à l'utilisateur. Le défi que de nombreux scientifiques ont depuis quelques années soulevé est de minimiser au maximum, voire totalement supprimer, cette partie là. Elaborer une machine parallèle permettant de traiter le plus efficacement possible n'importe quelle application telle une boîte noire est à l'évidence irréaliste. Notons néanmoins que l'on peut s'en approcher si l'on se restreint à des champs d'application particuliers tels que le traitement d'images. Malheureusement, tous les domaines d'application ne présentent pas un terrain aussi propice et la mécanique des fluides numériques compte parmi ces domaines moins favorables.

1.1.2.2. Finalité du Calcul Parallèle

Comme nous l'avons déjà expliqué, les utilisateurs de simulations numériques ont souvent besoin de moyens informatiques puissants pour effectuer rapidement des calculs de taille importante. Le Calcul Parallèle est un moyen potentiel de satisfaire cette exigence. Malheureusement, à l'heure actuelle, son utilisation est loin d'être répandue. Nous avons précédemment abordé les différents éléments concourant à ce manque d'enthousiasme, principalement l'absence de standards au niveau des plates-formes (hardware et software). De ce fait, le passage du séquentiel au parallèle implique à l'heure actuelle un investissement initial important du point de vue financier (achat d'une machine parallèle, formation des utilisateurs) et humain (période d'apprentissage, portage de codes, développement de nouvelles applications).

Or, d'une manière générale, il est difficile de prédire si l'accélération du calcul (speed-up) et/ou la diminution de l'encombrement mémoire finalement réalisées justifieront un tel investissement. Cette incertitude représente un risque que peu d'entreprises peuvent se permettre.

Pourtant, dans de nombreuses situations, le Calcul Parallèle pourrait être un outil numérique formidable si l'on parvenait à faciliter sa mise en œuvre. Parmi les applications intéressantes, citons pêle-mêle :

* L'exploitation de ressources informatiques hétérogènes à travers des réseaux de communication
* La génération de maillage à géométrie très complexe et/ou hétérogènes (morcellement de grille – « *partitioning* »)
* La prise en compte de la coexistence de différents modèles mathématiques et/ou numériques au sein d'un même domaine de calcul
* La résolution de systèmes algébriques de grande taille

De nombreux travaux sont en cours dans ces diverses voies, qui suivent des cheminements assez variés car souvent fortement liés au problème traité, à la machine ciblée ou à la méthode numérique employée. Malgré tout, deux motivations principales transparaissent : prendre le relais des supercalculateurs vectoriels ou bien augmenter à faible coût la puissance d'une machine.

Dans le premier cas, la tendance actuelle s'oriente vers des machines multiprocesseurs comportant peu de processeurs mais des processeurs très puissants, donc très chers. Dès lors, le prix de ces calculateurs est prohibitif pour la grande majorité des utilisateurs potentiels. Néanmoins, le but n'est pas de démocratiser le Calcul Parallèle mais de se donner les moyens de traiter des problèmes jusqu'à présent insolubles numériquement par manque de puissance de calcul.

La seconde motivation est de banaliser l'emploi du Calcul Parallèle en élaborant des machines puissantes financièrement abordables. Cette approche prend le contre-pied de la précédente dans la mesure où l'idée de base est de doter une machine de nombreux processeurs mais des processeurs peu puissants, donc peu chers.

On comprend aisément que l'existence de machines de conceptions si différentes va se refléter, en pratique, dans des stratégies de parallélisation également très différentes.

1.1.2.3. Principales approches actuelles

Nous classons de manière arbitraire les stratégies de parallélisation suivant deux catégories : l'exploitation d'un parallélisme « naturel » ou d'un parallélisme « acquis ».

1. Parallélisme « naturel »

Par parallélisme naturel, nous entendons le parallélisme préexistant dans une simulation numérique traditionnelle (séquentielle). L'idée la plus simple est d'exploiter le parallélisme inhérent à un programme déjà construit. Celui-ci se situe en général au niveau des opérations arithmétiques ou des instructions (boucles), rarement plus haut. Cette option offre l'avantage de permettre la mise en parallèle d'un programme à faible coût (rapidité du portage, peu de pré-requis donc personnel spécialisé non obligatoire) du fait que la parallélisation peut être effectuée par un compilateur, par certains langages de programmation tels le FORTRAN 90, ou bien par l'insertion d'appels à des bibliothèques standards telles que BLAS, LINPACK, LAPACK ... En revanche, le degré de parallélisme atteint est généralement faible. Le programme n'est parallèle que localement. Il reste globalement séquentiel ce qui implique un gain limité en performances et justifie une réserve quant à l'utilité d'une telle approche suivant le type d'application visé.

Pour augmenter la granularité du programme tout en conservant une relative transparence du Calcul Parallèle vis-à-vis de l'utilisateur, une autre voie consiste à rechercher le parallélisme inhérent au modèle numérique. Par exemple, depuis la fin des années 80, des travaux sont menés mettant à profit le parallélisme hérité d'une discrétisation de type éléments finis[22,45,2] ou éléments spectraux[11]. Le principe est d'associer un élément (ou un groupe d'éléments) et un processeur (réel ou virtuel si le nombre d'éléments dépasse le nombre de processeurs disponibles). En pratique, cela se traduit par une *distribution des données* du calcul entre les processeurs *(« data-parallel distribution »)* qui peut être rendue transparente pour l'utilisateur, comme cela a été réalisé par Fischer[9] dans le cadre du code aux éléments spectraux « Nekton ». Les unités de traitement réalisent de manière synchrone les mêmes opérations mais sur des données différentes (SIMD). Ce type d'application s'effectue de préférence sur des machines massivement parallèles, et est généralement caractérisé par une granularité moyenne. Les performances obtenues sont donc potentiellement supérieures à celles escomptées via les procédures de parallélisation automatique proposées par certains compilateurs.

2. Parallélisme « acquis »

Au lieu de rechercher le parallélisme (éventuellement) préexistant dans une application donnée, une autre stratégie consiste à le provoquer. Pour cela, il faut concevoir un algorithme parallèle[37] et l'appliquer au problème traité. L'avantage de cette stratégie par rapport à la précédente est que l'on peut, par modification de l'algorithme, agir sur le niveau de parallélisme de l'application et ainsi espérer améliorer les performances du calcul. Par contre, le degré d'intervention de l'utilisateur est ici très élevé puisqu'il englobe l'élaboration de l'algorithme, le choix de la plate-forme parallèle et l'écriture d'un nouveau programme. En pratique, cette approche exige un personnel spécialement formé et de consacrer une période au développement du nouveau code.

On ne peut pas évoquer les algorithmes parallèles en Mécanique des Fluides sans mentionner la *Décomposition de Domaine*. Il s'agit d'un terme générique englobant un ensemble vaste de méthodes pour la résolution de systèmes d'équations linéaires ou non linéaires résultant de la discrétisation d'équations aux dérivées partielles. Le succès de ces méthodes dans le domaine du Calcul Parallèle est motivé par la forte granularité des programmes qui en découlent. Leur principe se base sur les deux points suivants :

- travailler sur des sous domaines en parallèle, ces sous domaines étant obtenus par morcellement géométrique du domaine de calcul,
- raccorder la solution au niveau de l'interface de ces sous domaines.

Les nombreux algorithmes de décomposition de domaines que l'on rencontre dans la littérature se distinguent essentiellement par :

- le type de morcellement du domaine initial (découpage en bandes, en boîtes ..., avec ou non recouvrement des sous domaines),
- la construction du problème à l'interface.

Ces méthodes sont généralement associées à des méthodes itératives telles que la méthode du gradient conjugué (CG) ou du résidu minimum généralisé (GMRES) dans le cas de matrices symétriques définies positives, la méthode du gradient conjugué carré (CGS) ou du gradient biconjugué stabilisé (BiCGSTAB) dans le cas de matrices quelconques.

L'utilisation de la décomposition de domaine dans le cadre du Calcul Parallèle est un secteur de recherche très actif. Cet intérêt s'est traduit par un grand nombre de travaux sur le sujet. Pourtant, force est de constater que, à l'heure où nous écrivons, ces méthodes ne sont toujours pas passées du stade de la recherche au stade de l'exploitation. Là encore, la faute en incombe en

partie au manque de librairies et d'outils d'aide au développement. Un autre handicap réside dans le fait que ces algorithmes sont souvent perçus comme complexes. Or, à l'instar de Smith et al[33], nous pensons que la grande majorité d'entre eux deviennent aisément compréhensibles pour peu qu'une notation matricielle soit utilisée. C'est pourquoi, dès que cela sera possible, nous privilégierons une écriture sous forme matricielle.

1.2. Objectifs de l'étude

1.2.1. L'idée de départ

Le sujet de cette étude consiste à appliquer le Calcul Parallèle à la résolution des équations de Navier-Stokes dans le cadre de la Méthode des Eléments Finis[‡]. Celle-ci supporte, de par sa conception même, une programmation parallèle par distribution de données[§] au sens où elle fournit les facilités qui rendent cette technique pratique (aisée et efficace). En effet, cette méthode manipule trois niveaux de données : un niveau nodal (degré de liberté), un niveau élémentaire (matrice et vecteur élémentaires) et un niveau global (matrice et vecteur globaux). Chaque niveau résulte d'un certain assemblage de données de niveau directement inférieur. L'ensemble de ces données s'organise autour d'une structure arborescente dont le sommet est le système assemblé et la base les degrés de liberté du problème (*fig. 1.3*). La gestion des données de haut niveau peut, de ce fait, être prise en charge à un niveau inférieur, en particulier au niveau des degrés de liberté. Or, chaque degré de liberté est associé à une certaine localisation dans le plan géométrique. La répartition des données de bas niveau entre plusieurs unités de traitement peut donc être effectuée par le biais de simples tests géométriques, entraînant par cascade une répartition de l'ensemble des données. Cette procédure est très intéressante car elle ne dépend ni du problème traité, ni de sa géométrie. Par conséquent, elle autorise la dissimulation, vis à vis de l'utilisateur, de l'architecture parallèle de la plate-forme d'accueil derrière l'interface machine – utilisateur.

Il n'en est pas de même, a priori, pour la méthode de résolution du système final car l'étape de résolution, qui s'effectue après l'assemblage, ne manipule plus que des données de niveau global (produits matrice – vecteur, produits scalaires, ...) ce qui implique, à première vue, un modèle de programmation parallèle différent. Il faudrait pouvoir effectuer la résolution du

[‡] présentée paragraphe 1.3.1
[§] présentée paragraphe 1.1.2.3

système global à partir de structures de données de niveau inférieur afin d'inscrire cette étape dans un modèle de programmation par distribution de données.

A cette fin, notre idée de départ est d'effectuer un découpage algébrique (« *splitting* ») de la matrice associée au problème. En effet, à chaque élément constitutif d'un maillage de type éléments finis, est associé un ensemble de matrices et de vecteurs élémentaires qui contribuent à la construction des matrices et vecteurs globaux. Par conséquent, tout sous-bloc d'une matrice ou d'un vecteur global est lié à un certain ensemble d'éléments qui, lui-même, correspond à une région connexe ou non connexe du maillage. Ainsi, un découpage de la matrice du problème équivaut à un certain regroupement des éléments par sous-ensembles ou, en d'autres termes, à un morcellement du maillage initial. La réciproque est également vraie. Cette bijection créée entre le plan algébrique et le plan géométrique est la clef de voûte de notre étude (*fig. 1.2*).

La finalité du présent travail est d'élaborer un environnement pour le développement de solveurs parallèles appliqués à la résolution des équations de Navier-Stokes. Cette résolution fait appel à des méthodes itératives, du fait de leur caractère non linéaire. Or, une partie de ces méthodes peut être déduite à partir de découpages matriciels. Par conséquent, à la lumière des précédentes remarques, nous pouvons dire que de telles méthodes supportent un modèle de programmation par distribution de données.

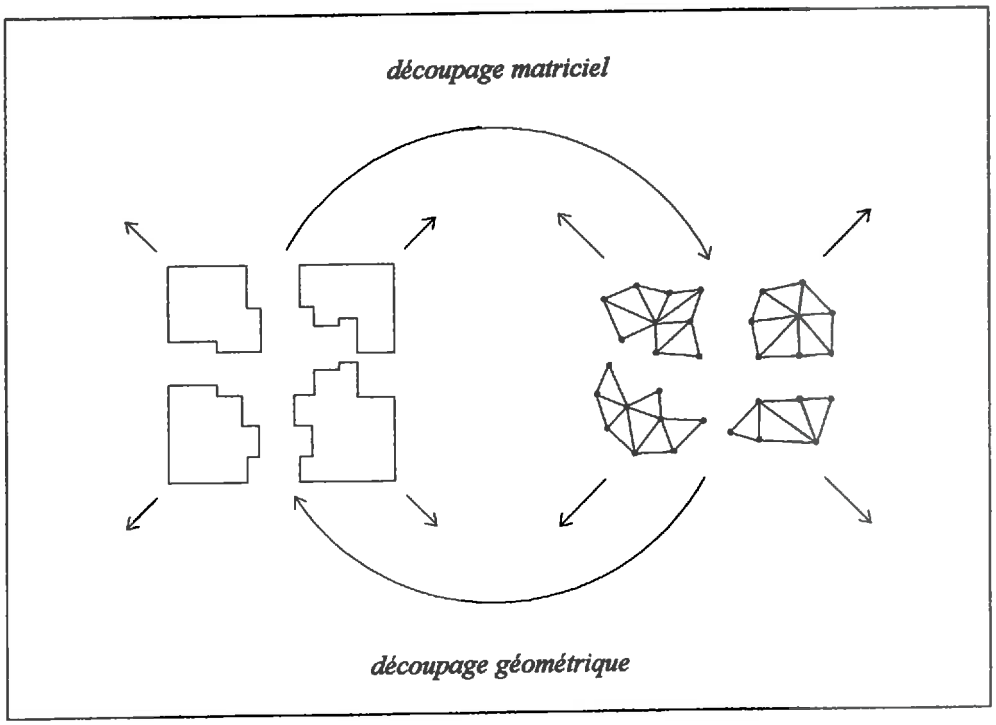


Figure 1.2 : Interaction entre un découpage algébrique et un découpage géométrique

1.2.2. L'orientation de travail

L'axe de recherche choisi par l'équipe s'oriente vers l'élaboration d'un ensemble d'unités de programme pour le développement de solveurs aux éléments finis[6]. La finalité de ce projet est de mettre à la disposition de l'utilisateur une bibliothèque d'objets qui lui permette de construire le solveur dont il a besoin en combinant plusieurs objets de la bibliothèque.

Avec l'avènement du Calcul Parallèle, la volonté s'est faite jour de proposer cet outil à l'utilisateur de telle sorte que sa mise en œuvre soit la plus transparente possible pour des raisons de convivialité. Un autre point important est que les objets de cette bibliothèque soient le moins affectés possible par des modifications spécifiques au Calcul Parallèle afin que leur utilisation, dans un contexte de Calcul Parallèle, ne soit, à aucun moment, remise en cause. La présente thèse s'inscrit dans cette optique.

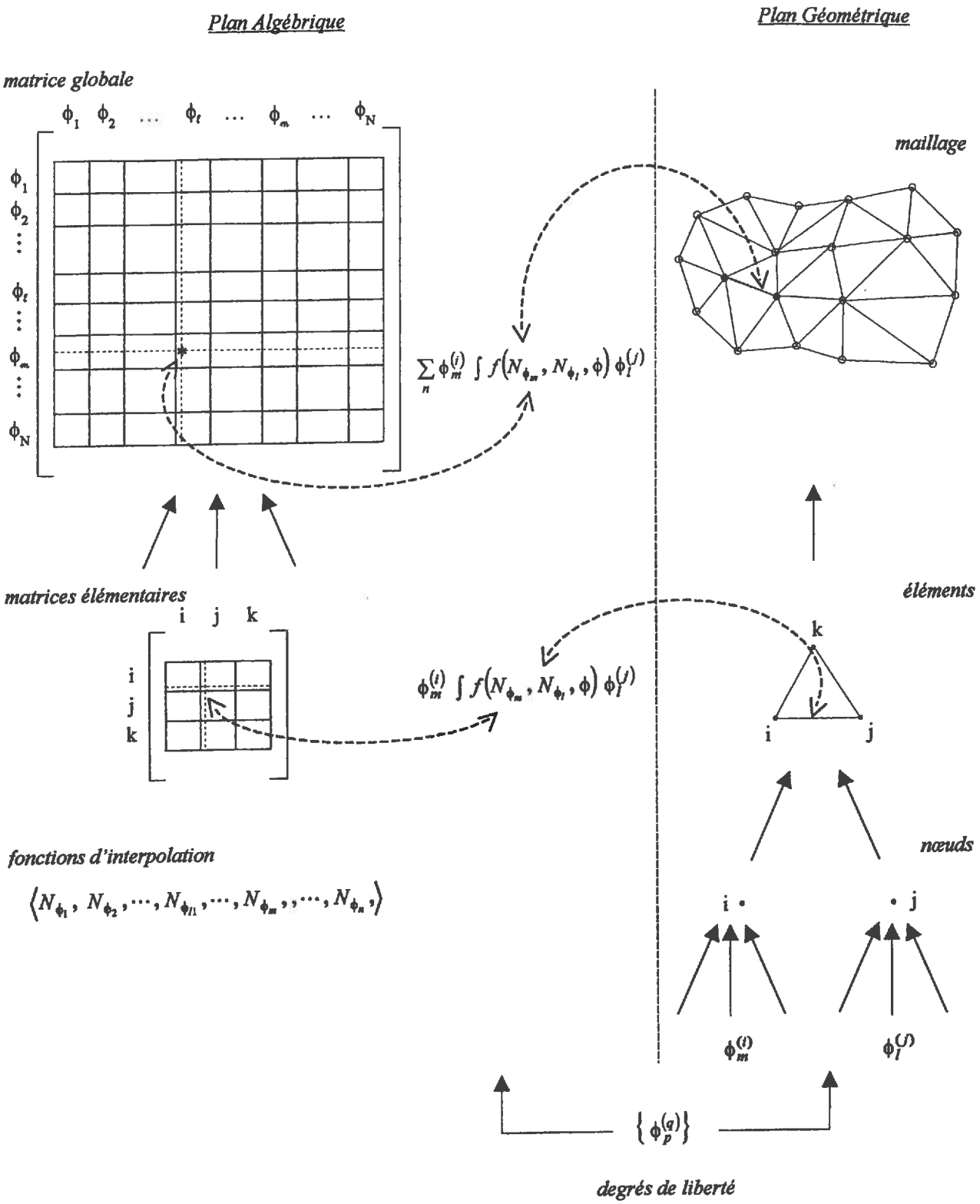
Ce thème étant entièrement nouveau au sein de l'équipe, notre rôle consiste à asseoir une base pour les développements à venir. A partir de l'idée de départ précédemment exposée, nous avons dégagé quatre points autour desquels devra s'articuler la mise en place de cette plate-forme de travail, à savoir :

(1) définition d'une approche qui permette de résoudre les équations de Navier-Stokes par une procédure numérique basée sur un découpage algébrique de la matrice associée au problème et pouvant être mise en œuvre par la combinaison d'unités de programmation

(2) détermination des outils nécessaires,

(3) élaboration de ces outils,

(4) validation de l'ensemble de la procédure.



n : *nombre de variable du problème*

n_n : *nombre de nœuds du maillage*

ϕ_p : *p-ième variable du problème*

$\phi_p^{(q)}$: *valeur de ϕ_p au nœud numéro q*

N_{ϕ_p} : *fonctions d'interpolation du champ de la variable ϕ_p*

Figure 1.3 : Discrétisation par éléments finis

Chapitre 2
Outils de l'étude

Chapitre 2 : Outils de l'Etude

2.1. Outils numériques

Le cadre numérique de ce travail est basé sur la méthode éléments finis dont nous allons, à présent, rappeler brièvement les grandes lignes. Pour plus de détails, on se référera par exemple aux ouvrages de Zienkiewicz[46] ou de Dhatt et Touzot[7]. Les problèmes physiques sont, en général, régis par la loi universelle de conservation et par les lois de comportement. Afin d'illustrer nos propos dans un contexte simple, nous nous restreindrons à la présentation de la discrétisation par éléments finis de Galerkin des équations de conservation de la masse, de la quantité du mouvement et de l'énergie dans le cas d'un fluide visqueux newtonien et incompressible.

2.1.1. Rappels sur la méthode des éléments finis

L'objectif de la méthode des éléments finis est de réduire le problème continu traité (nombre infini de degrés de liberté) à un problème discret (nombre fini de degrés de liberté) décrit par un système d'équations algébriques. Autrement dit, il s'agit d'obtenir une solution approchée d'un système d'équations aux dérivées partielles à partir d'une *formulation faible* de ce dernier, en le transformant en un système d'équations algébriques.

Les principales étapes de cette transformation sont:

– la discrétisation géométrique du domaine : l'espace continu est remplacé par un espace discret résultant d'une subdivision du domaine de calcul initial en un nombre fini de domaines élémentaires à géométrie simple (triangles, quadrilatères, tétraèdres...) appelés éléments, comme le montre la *figure 2.1* .

– la discrétisation analytique : elle comprend le choix des fonctions d'interpolation des variables u_i, p et T à l'intérieur de chaque élément, la discrétisation spatiale et temporelle des équations, la formulation sous forme de matrices élémentaires et l'assemblage.

2.1.1.1. Modèle variationnel

Contrairement à la méthode des différences finies qui traite directement la formulation forte du problème étudié, les éléments finis se basent sur un modèle variationnel (faible) de ce problème. La première étape consiste donc à écrire ce dernier. Pour cela nous formons le produit scalaire des équations aux dérivées partielles régissant le problème traité et d'un vecteur générique de fonctions test.

Dans ce but, nous introduisons les notations suivantes. Soit $H^1(\Omega)$ l'espace de Sobolev des fonctions scalaires définies sur Ω,

$$H^1(\Omega) = \{u | u \in L^2(\Omega), u' \in L^2(\Omega)\}, \tag{2.1}$$

où le prime représente la dérivée première par rapport à la variable d'espace. De manière similaire, notons $H^1(\Omega)^d$ l'espace de Sobolev des fonctions vectorielles,

$$H^1(\Omega)^d = \{u | u \in L^2(\Omega)^d, u' \in L^2(\Omega)^d\}. \tag{2.2}$$

Dans les espaces $L^2(\Omega)$ et $L^2(\Omega)^d$ des fonctions scalaires et vectorielles de carré intégrable sur Ω relativement à la mesure de Lebesgue

$$d\Omega = \prod_{i=1}^{d} dx_i,$$

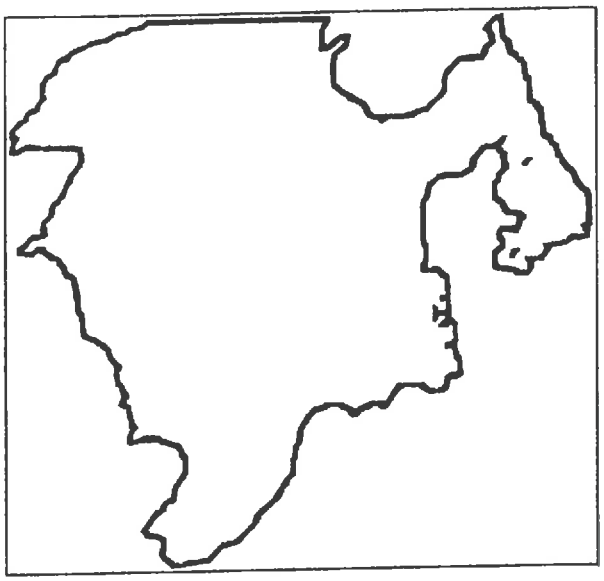
le produit scalaire usuel est respectivement défini par:

$$(u, v) = \int uv d\Omega \tag{2.3}$$

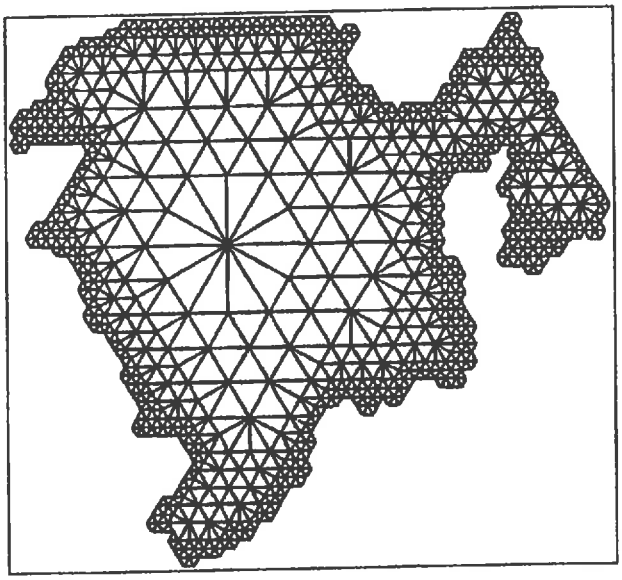
et

$$(u, v) = \int d\Omega uv = \sum_{i=1}^{d} (u_i, v_i). \tag{2.4}$$

Considérons, à présent, un problème de mécanique des fluides régi par les équations de conservation ci-après, écrites ici en formulation vitesse - pression. Ces équations sont généralement complétées par certaines conditions aux limites et conditions initiales que nous n'expliciterons pas ici, mais que nous supposons choisies de telle sorte que l'ensemble représente un problème bien posé.



Domaine continu



Domaine discret

Figure 2.1 : Exemple de discrétisation géométrique à base d'éléments triangulaires tiré de [43]

- Formulation Forte -

* Conservation de la masse

$$\nabla.u=0\ ,\quad \text{sur } \Omega\times(0,T) \tag{2.5}$$

* Conservation de la quantité de mouvement

$$\rho\frac{\partial \mathbf{u}}{\partial t}+\rho(\mathbf{u}\cdot\nabla)\,\mathbf{u}=\nabla\cdot\bar{\bar{\sigma}}\ ,\quad \text{sur } \Omega\times(0,T) \tag{2.6}$$

* Conservation de l'énergie

$$\rho C_p\frac{\partial T}{\partial t}+\rho C_p(\mathbf{u}\cdot\nabla)T=\nabla\cdot(\lambda\nabla T)+\Phi\ ,\quad \text{sur } \Omega\times(0,T) \tag{2.7}$$

avec :

$$\bar{\bar{\sigma}}=-p\bar{\bar{I}}+2\mu\overline{\overline{\varepsilon(u)}}, \tag{2.8}$$

$$\overline{\overline{\varepsilon(u)}}=\tfrac{1}{2}\left[\nabla u+(\nabla u)^T\right], \tag{2.9}$$

$$\Phi=\mu\cdot\left[\nabla u+(\nabla u)^T\right]\cdot\nabla u, \tag{2.10}$$

Une formulation faible de Galerkin du problème (2.5)- (2.7) s'écrit:

- Formulation Faible -

Etant donnés Q, V et T les espaces admissibles de fonctions pour la pression, la vitesse et la température respectivement, trouver $(\mathbf{u}, p, T)\in V\times Q\times T$ tels que

$$(\nabla.\mathbf{u},\delta p)=0 \tag{2.11}$$

$$\left(\rho\frac{\partial \mathbf{u}}{\partial t},\delta\mathbf{u}\right)+\left(\rho(\mathbf{u}\cdot\nabla)\mathbf{u},\delta\mathbf{u}\right)=-\left(\nabla\cdot\left(p\bar{\bar{I}}\right),\delta\mathbf{u}\right)+\left(\nabla\cdot\left(\mu\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\right),\delta\mathbf{u}\right) \tag{2.12}$$

$$\left(\rho C_p\frac{\partial T}{\partial t},\delta T\right)+\left(\rho C_p(\mathbf{u}\cdot\nabla)T,\delta T\right)=\left(\nabla\cdot(\lambda\nabla T),\delta T\right)+\left(\Phi,\delta T\right) \tag{2.13}$$

pour tout $\delta p\in Q$, $\delta\mathbf{u}\in V$ et $\delta T\in T$.

La nature des espaces Q, V et T dépend entre autres des conditions aux limites du problème.

Les dérivées secondes sont traitées comme suit. Nous avons :

$$(\nabla.(\lambda\nabla T),\delta T)=\int \nabla\cdot(\lambda\nabla T)\cdot\delta T\ d\Omega=\int \nabla\cdot(\lambda\nabla T\cdot\delta T)\ d\Omega-\int \lambda\nabla T\cdot\nabla\delta T\ d\Omega$$

En appliquant le théorème d'Ostrogradsky, nous pouvons donc écrire:

$$\nabla.(\lambda\nabla T),\delta T)=\int \lambda\nabla T\cdot\delta T\cdot n d\Gamma-(\lambda\nabla T,\nabla\delta T)$$

De même, nous obtenons:

$$(\nabla.(\nu(\nabla\mathbf{u}+(\nabla\mathbf{u})^T),\delta\mathbf{u})=\int \upsilon\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\cdot\delta\mathbf{u}\cdot n d\Gamma-(\nu(\nabla\mathbf{u}+(\nabla\mathbf{u})^T),\nabla\delta\mathbf{u})$$

Les intégrales de contour permettent d'introduire « naturellement » des conditions aux limites du problème.

Après ces transformations, il en résulte que : $(\delta p, \delta\mathbf{u}, \delta T)\in H^1(\Omega)\times H^1(\Omega)^d\times H^1(\Omega)$.

Les équations aux éléments finis sont obtenues en considérant une approximation variationnelle $(\mathbf{u}_h, p_h, T_h)$ de la solution $(\mathbf{u}, p, T)$ de (2.11)-(2.13):

- Approximation de la Formulation Faible -

Etant donnés V^h, Q^h et T^h des sous-espaces de dimension finie des espaces V, Q et T respectivement, nous cherchons $(\mathbf{u}_h, p_h, T_h)\in V^h\times Q^h\times T^h$ tel que:

$$(\nabla_h.\mathbf{u}_h,\delta p_h)=0 \tag{2.14}$$

$$\left(\rho\frac{\partial\mathbf{u}_h}{\partial t},\delta\mathbf{u}_h\right)+(\rho(\mathbf{u}_h\cdot\nabla_h)\mathbf{u}_h,\delta\mathbf{u}_h)=-\left(\nabla_h\cdot\left(p_h\bar{\bar{\mathbf{I}}}\right),\delta\mathbf{u}_h\right)$$

$$-\left(\mu\left(\nabla_h\mathbf{u}_h+(\nabla_h\mathbf{u}_h)^T\right),\nabla\delta\mathbf{u}_h\right)+\int\mu\left(\nabla_h\mathbf{u}_h+(\nabla_h\mathbf{u}_h)^T\right)\cdot\delta\mathbf{u}_h\cdot\mathbf{n}\,d\Gamma \tag{2.15}$$

$$\left(\rho C_p\frac{\partial T_h}{\partial t},\delta T_h\right)+\left(\rho C_p(\mathbf{u}_h\cdot\nabla_h)T_h,\delta T_h\right)=-(\nabla_h T_h,\nabla_h\delta T_h)+(\Phi_h,\delta T_h)$$

$$+\int\nabla_h T_h\cdot\delta T_h\cdot\mathbf{n}\,d\Gamma \tag{2.16}$$

pour tout $\delta p_h\in Q^h$, $\delta\mathbf{u}_h\in V^h$et $\delta T_h\in T^h$.

∇_h est l'opérateur linéaire défini par:

$$\nabla_h: V^h \rightarrow Q^h;\ (\nabla.v_h,q_h)=(\nabla_h.v_h,q_h),\ \forall(v_h,q_h)\in V^h\times Q^h.$$

2.1.1.2. Discrétisation géométrique

Pour obtenir les sous-espaces V^h, Q^h et T^h, la méthode des éléments finis décompose, par triangulation, le domaine continu Ω en un domaine discrétisé résultant de l'union de n_e domaines élémentaires Ω_e tels que:

$$\bigcup_{n_e}\Omega_e=\Omega \text{ et } \Omega_e\cap\Omega_{e'}=\varnothing \text{ si } e\neq e'.$$

Dans ces conditions, la mesure de Lebesgue s'écrit:

$$\text{Mesure}\bigcup_{n_e}\Omega_e=\sum_{n_e}\text{Mesure}\Omega_e\,. \qquad (2.17)$$

Ces domaines, appelés éléments, sont caractérisés par un certain nombre de nœuds.

La formulation en variables primitives du problème rend plus délicat le choix de l'élément. En effet, Brezzi (1974) et Babuska (1973) ont établi une théorie selon laquelle, dans pareil cas, le choix des fonctions d'interpolation pour la vitesse et pour la pression doit se plier à une condition dite LBB. Cette condition impose une interpolation du champ de pression d'au moins un degré moindre par rapport à celle adoptée pour les autres champs (ici vitesse et température). L'utilisation d'éléments ne respectant pas cette condition n'est pas impossible mais risque de déboucher sur des problèmes de stabilité lors d'un raffinement ou d'une distorsion du maillage.

Dans le présent travail, nous favorisons l'utilisation d'éléments triangulaires pour leur aptitude à mailler des géométries complexes. Concernant des problèmes bidimensionnels du type (11)-(13), la condition LBB nous oriente vers les éléments de Taylor-Hood (P_2-P_1, i.e. fonctions d'interpolation quadratiques pour la vitesse et la température et linéaires pour la pression) présentés sur la *figure 2.2*. Dans d'autres cas, nous opterons pour des triangles à trois nœuds (fonctions d'interpolation linéaires pour toutes les variables). Pour une description détaillée de ces éléments on se référera, par exemple, à Dhatt et Touzot[7].

2.1.1.3. Discrétisation analytique

Notons $\{n_u^{(i)};\ i=1,n\}$ (resp. $\{ n_p^{(i)};\ i=1,m\}$ et $\{n_T^{(i)};\ i=1,s\}$) une base de fonctions de l'espace de dimension finie V^h (resp. Q^h et T^h). Les équations aux éléments finis globales s'obtiennent en appliquant les approximations suivantes:

$$u_i(x,t)=\sum_{k=1}^{n} n_u^{(k)} u_i^{(k)} = <n_u>.\{u_i^{(k)}\} \tag{2.18}$$

$$p(x,t)=\sum_{k=1}^{n} n_p^{(k)} p^{(k)} = <n_p>.\{p^{(k)}\} \tag{2.19}$$

$$T(x,t)=\sum_{k=1}^{n} n_T^{(k)} T^{(k)} = <n_T>.\{T^{(k)}\} \tag{2.20}$$

où $u_i^{(k)}$, $p^{(k)}$ et $T^{(k)}$ sont les valeurs des inconnues aux noeuds de l'élément considéré. Ici, la même base de fonctions est employée pour toutes les composantes de la vitesse.

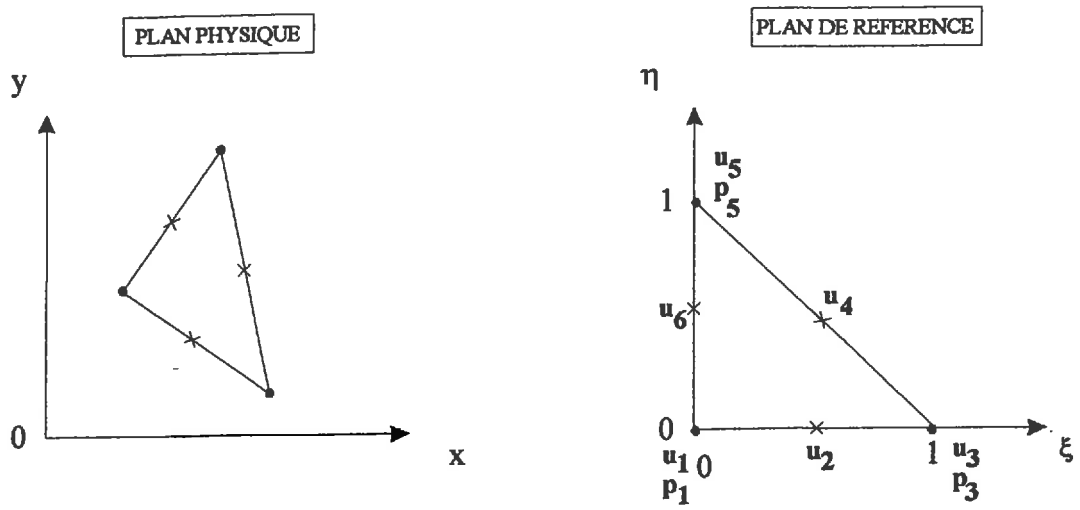
Pour un élément Ω_e donné, en remplaçant chaque fonction par son approximation polynomiale (18)-(20), les équations (14)-(16) s'écrivent sous forme intégrale:

$$\sum_{n_e} \left\langle \delta p^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_p\} \cdot \left\langle \frac{\partial n}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ u_j^{(k)} \right\} = 0 \tag{2.21}$$

$$\sum_{n_e} \left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \rho \cdot \{n_u\} \cdot \langle n_u \rangle \cdot d\Omega_e \cdot \left\{ \dot{u}_i^{(k)} \right\} \tag{2.22}$$

$$+\sum_{n_e} \left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \rho \cdot \{n_u\} \cdot u_j \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ u_i^{(k)} \right\} =$$

$$-\sum_{n_e} \left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_u\} \cdot \left\langle \frac{\partial n_p}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ p^{(k)} \right\}$$



- *Vitesse*

Base polynomiale :

$$<P>=<1\ \xi\ \eta\ \xi^2\ \xi\eta\ \eta^2>$$

Fonctions d'interpolations :

	{N}	{∂N/∂ξ}	{∂N/∂η}
1	$-\lambda(1-2\lambda)$	$1-4\lambda$	$1-4\lambda$
2	$4\xi\lambda$	$4(\lambda-\xi)$	-4ξ
3	$-\xi(1-2\xi)$	$-1+4\xi$	0
4	$4\xi\eta$	4η	4ξ
5	$-\eta(1-\eta)$	0	$-1+4\eta$
6	$4\eta\lambda$	-4η	$4(\lambda-\eta)$

Avec : $\lambda = 1 - \xi - \eta$

- *Pression*

Base polynomiale :

$$<P>=<1\ \xi\ \eta>$$

Fonctions d'interpolations :

	{N}	{∂N/∂ξ}	{∂N/∂η}
1	$1-\xi-\eta$	−1	−1
2	ξ	1	0
3	η	0	1

Figure 2.2 : Eléments de Taylor-Hood (t6-3)

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \mu \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ u_i^{(k)} \right\}$$

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \mu \cdot \left\langle \frac{\partial n_u}{\partial x_i} \right\rangle \cdot d\Omega_e \cdot \left\{ u_j^{(k)} \right\}$$

$$\sum_{n_e}\left\langle \delta T^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_T\} \cdot \langle n_T \rangle \cdot d\Omega_e \cdot \left\{ \dot{T}^{(k)} \right\} \tag{2.23}$$

$$+\sum_{n_e}\left\langle \delta T^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_T\} \cdot u_j \cdot \left\langle \frac{\partial n_T}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ T^{(k)} \right\} =$$

$$-\sum_{n_e}\left\langle \delta T^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_T}{\partial x_j} \right\} \cdot \frac{\lambda}{\rho C_p} \cdot \left\langle \frac{\partial n_T}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ T^{(k)} \right\}$$

Cette formulation peut être organisée sous forme matricielle. Notons n_dim la dimension du problème traité (n_dim=1, 2 ou 3). En rangeant les degrés de liberté dans l'ordre ($u_i^{(k)}$, $p^{(k)}$, $T^{(k)}$) nous voyons apparaître des matrices élémentaires par blocs notés (i_bloc, j_bloc), i=1,..,n_dim et j=1,..,n_dim. Notons, entre autres:

* Les matrices masse élémentaires **m**:

$$m_{uu} = \int_{\Omega_e} \rho \cdot \{n_u\} \cdot \langle n_u \rangle \cdot d\Omega_e$$

i_bloc=j_bloc=1,..,n_dim

$$m_{TT} = \int_{\Omega_e} \rho C_p \cdot \{n_T\} \cdot \langle n_T \rangle \cdot d\Omega_e$$

i_bloc=j_bloc=n_dim+1

* Les matrices d'inertie élémentaires $\mathbf{k_i}$:

$$k_{i,uu} = \int_{\Omega_e} \rho \cdot \{n_u\} \cdot u_j \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e$$

i_bloc=j_bloc=1,..,n_dim

* Les matrices de convection élémentaires $\mathbf{k_{cv}}$:

$$k_{cv,TT} = \int_{\Omega_e} \rho \cdot \{n_T\} \cdot u_j \cdot \left\langle \frac{\partial n_T}{\partial x_j} \right\rangle \cdot d\Omega_e$$

i_bloc=j_bloc=n_dim+1

* Les matrices de gradient élémentaires k_g :

$$k_{g,up} = \int_{\Omega_e} \{n_u\} \cdot \left\langle \frac{\partial n_p}{\partial x_j} \right\rangle \cdot d\Omega_e$$

i_bloc=1,..,n_dim , j_bloc=n_dim+2

* Les matrices de continuité élémentaires k_{ct} :

$$k_{ct,pu} = \int_{\Omega_e} \{n_p\} \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e$$

i_bloc= n_dim+2 , j_bloc= 1,..,n_dim

* Les matrices de dissipation élémentaires k_d :

$$k^1_{d,uu} = \int_{\Omega_e} \mu \cdot \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e$$

i_bloc= j_bloc= 1,..,n_dim

$$k^2_{d,uu} = \int_{\Omega_e} \mu \cdot \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \left\langle \frac{\partial n_u}{\partial x_i} \right\rangle \cdot d\Omega_e$$

i_bloc=i=1,..,n_dim , j_bloc=j=1,..,n_dim

$$k_{d,TT} = \int_{\Omega_e} \lambda \cdot \left\{ \frac{\partial n_T}{\partial x_j} \right\} \cdot \left\langle \frac{\partial n_T}{\partial x_i} \right\rangle \cdot d\Omega_e$$

i_bloc=j_bloc=n_dim+1

La présentation matricielle des intégrales (21)-(23) peut se mettre sous la forme d'un produit scalaire, comme suit :

$$\sum_{n_e}\left\langle \delta u_i^{(k)}\delta T^{(k)}\delta p^{(k)}\right\rangle\cdot\left(\left\{\begin{array}{c}\\ f_e\\ \\ \end{array}\right\}-\left[K_e\right]\cdot\left\{\begin{array}{c}u_i^{(k)}\\ T^{(k)}\\ p^{(k)}\end{array}\right\}-\left[M_e\right]\cdot\left\{\begin{array}{c}\dot{u}_i^{(k)}\\ \dot{T}^{(k)}\\ \dot{p}^{(k)}\end{array}\right\}\right)=0 \qquad (2.24)$$

Compte-tenu de l'élément choisi, ce produit scalaire est calculé dans $\Re^k$ (k=21 pour des éléments triangulaires de Taylor-Hood).

Après une opération d'assemblage par extension des vecteurs élémentaires, désormais devenue classique, nous obtenons:

$$\langle 0|\quad \delta U_i\quad\rangle\cdot\left(\left[\begin{array}{c|c}\mathbf{m_{11}} & \mathbf{m_{12}}\\ \hline \mathbf{m_{21}} & \mathbf{M}\end{array}\right]\cdot\left\{\begin{array}{c}\dot{U}_0\\ \hline \dot{U}_i\end{array}\right\}\right.$$

$$\left.+\left[\begin{array}{c|c}\mathbf{k_{11}} & \mathbf{k_{12}}\\ \hline \mathbf{k_{21}} & \mathbf{K}\end{array}\right]\cdot\left\{\begin{array}{c}U_0\\ \hline U_i\end{array}\right\}\right)=\langle 0|\quad \delta U_i\quad\rangle\cdot\left\{\begin{array}{c}f_0\\ \hline f_i\end{array}\right\} \qquad (2.25)$$

Le vecteur $\langle U_0|U_i\rangle$ représente l'ensemble des degrés de liberté aux noeuds. Les composantes (U_0) correspondent aux degrés de liberté imposés (Dirichlet) tandis que les (U_i) sont les inconnues du système. Nous avons supposé que les valeurs imposées sont regroupées dans les premières composantes du vecteur des valeurs nodales. Cette numérotation particulière des noeuds, effectuée pour la clarté des explications, n'enlève rien à la généralité de la formulation. Dans ces conditions, cette égalité est réalisée quelque soient les (δU_i). Il en résulte un système algébrique à résoudre qui, en posant

$$\mathbf{U}=\left\{U_i\right\}\quad\text{et}\quad\mathbf{F}=\left\{f_i\right\}-\left[\quad\mathbf{k_{21}}\quad\right]\cdot\left\{U_0\right\}, \qquad (2.26)$$

s'écrit ici:

$$\mathbf{M}\,\dot{\mathbf{U}}+\mathbf{K(U)}\,\mathbf{U}=\mathbf{F(U)} \qquad (2.27)$$

La matrice K(U) est une matrice bande, a priori non symétrique compte tenu des propriétés des matrices élémentaires et de l'opération d'assemblage. Cette structure de bande est une caractéristique très intéressante de la méthode des éléments finis car elle permet des économies au niveau du stockage en mémoire de K(U) et accélère la résolution du système d'équations algébriques final. Il est donc important de minimiser au mieux la largeur de cette bande. Cette dernière dépend de l'ordre de numérotation des nœuds du maillage.

Les matrices résultant d'une discrétisation par éléments finis sont généralement plus remplies que celles générées par différences finies. De ce fait, les équations algébriques constitutives du système matriciel sont d'avantage couplées. Or, l'indépendance des données est un facteur essentiel au parallélisme. Ceci explique pourquoi le calcul parallèle trouve dans la méthode des éléments finis un terrain moins favorable que dans celle des différences finies. Néanmoins, nous avons choisi la méthode des éléments finis pour, outre les avantages précités, les raisons suivantes :

* Les codes aux éléments finis sont très répandus dans le milieu industriel. Cette méthode numérique nous semble donc être un bon vecteur pour l'expansion du calcul parallèle dans ce milieu.
* Cette méthode présente un bon compromis entre précision et niveau de remplissage de la matrice.
* Enfin, nous ne nions pas l'influence exercée lors de ce choix par l'existence dans l'équipe d'un certain savoir faire concernant cette méthode.

2.1.2. Méthodes numériques et techniques associées

Comme nous venons de le voir, la méthode des éléments finis aboutit à la résolution d'un système d'équations algébriques généralement non linéaires. La formulation de ce système sous forme matricielle est classique car elle permet d'aborder plus simplement la mise en œuvre effective de cette méthode. Il est nécessaire pour cela d'associer à la méthode des éléments finis des méthodes numériques et des techniques permettant de calculer les éléments des diverses matrices et de résoudre le système algébrique final. De nombreuses méthodes sont disponibles. Nous nous limitons ici à un rapide descriptif de celles utilisées dans le cadre du présent travail.

2.1.2.1. Intégration numérique

Les matrices élémentaires constitutives du système matriciel final s'expriment sous la forme d'intégrales à une, deux ou trois dimensions. Il s'agit donc d'évaluer ces intégrales par le biais d'une intégration numérique. Les calculs présentés dans ce mémoire nécessitent des intégrations à deux dimensions. Celles-ci ont été effectuées sur l'élément de référence via une méthode directe. Plus précisément, nous avons opté pour la formule de Hammer d'ordre m=2, celle-ci intégrant exactement des monômes $\xi^i\eta^j$ pour lesquels $i+j \leq m$:

$$\int_0^1 \int_0^{1-\xi} y(\xi,\eta)\, d\eta\, d\xi \approx \sum_{i=1}^{r} w_i\, y(\xi,\eta) \tag{2.28}$$

2.1.2.2. Stockage matriciel

Nous rappelons que la structure de la matrice globale est de type bande. Le stockage est effectué suivant la technique de la ligne de ciel c'est à dire que seuls les termes inclus dans la bande sont stockés. Le stockage s'effectue par lignes et colonnes de longueurs variables décrivant des « L » comme le montre la *figure 2.3* . Pour plus de détails, on se référera à des ouvrages tels que *[7]*.

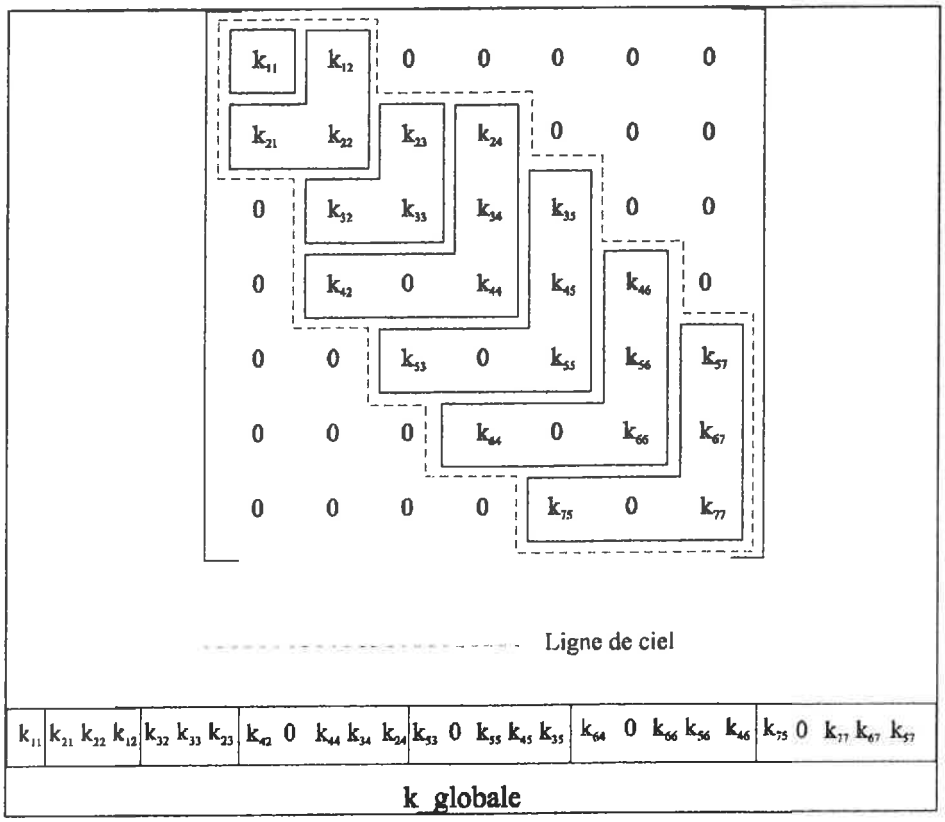


Figure 2.3 : Principe du stockage en ligne de ciel.

2.1.2.3. Méthodes numériques de linéarisation

Pour linéariser si nécessaire le système algébrique final, nous avons programmé la méthode de substitution (itérations de Picard) et la méthode de Newton-Raphson.

Les itérations de substitution, appelées également approximations successives, s'écrivent :

$$\mathbf{K}(\mathbf{U}^{(k)}).\mathbf{U}^{(k+1)}=\mathbf{F}(\mathbf{U}^{(k)}) \tag{2.29}$$

$$\Leftrightarrow \mathbf{K}(\mathbf{U}^{(k)}).(\mathbf{U}^{(k+1)}-\mathbf{U}^{(k)})=\mathbf{F}(\mathbf{U}^{(k)})-\mathbf{K}(\mathbf{U}^{(k)}).\mathbf{U}^{(k)} \tag{2.30}$$

Les itérations de Newton-Raphson impliquent le calcul de la matrice tangente $\mathbf{K_T}$ par ajout dans $\mathbf{K}$ de termes supplémentaires :

$$\mathbf{K}^{T}(\mathbf{U}^{(k)}).(\mathbf{U}^{(k+1)}-\mathbf{U}^{(k)})=\mathbf{F}(\mathbf{U}^{(k)})-\mathbf{K}(\mathbf{U}^{(k)}).\mathbf{U}^{(k)} \tag{2.31}$$

avec

$$\mathbf{K}^{T}(\mathbf{U})=\frac{\partial \mathbf{F}(\mathbf{U})}{\partial \mathbf{U}}+\mathbf{K}(\mathbf{U})+\frac{\partial \mathbf{K}(\mathbf{U})}{\partial \mathbf{U}} \tag{2.32}$$

Le taux de convergence de la méthode de substitution est asymptotiquement linéaire avec :

$$\left\|\mathbf{U}^{(k+1)}-\mathbf{U}\right\| \leq C\left\|\mathbf{U}^{(k)}-\mathbf{U}\right\|, \tag{1.33}$$

C désignant une constante et U la solution exacte, tandis que les itérations de Newton-Raphson sont caractérisées par une convergence quadratique :

$$\left\|\mathbf{U}^{(k+1)}-\mathbf{U}\right\| \leq C\left\|\mathbf{U}^{(k)}-\mathbf{U}\right\|^{2}. \tag{2.34}$$

En revanche, le rayon de convergence de la méthode de Newton-Raphson peut parfois être très petit ce qui nécessite une solution initiale suffisamment proche de la solution exacte. La méthode de substitution est quant à elle très peu sensible aux conditions initiales du système à résoudre. Il est de ce fait parfois utile d'effectuer quelques itérations de substitution avant d'appliquer la méthode de Newton-Raphson. C'est pourquoi il est utile de prévoir une méthode globale en complément d'une méthode de type Newton.

2.1.2.4. Fonction de Pénalité

L'absence de termes de pression dans l'équation de continuité des équations de Navier-Stokes incompressibles se traduit par l'apparition de zéros sur la diagonale principale de la matrice finale $\mathbf{K}$ ce qui soulève un problème lorsque l'utilisation d'une méthode de résolution

directe est envisagée. Vanka et Leaf[40] en ont déduit la nécessité de réordonner les équations et les inconnues avant résolution de telle sorte que tous les éléments de la diagonale principale soient non nuls.

D'autres approches ont été explorées par Braaten[3]. L'une d'elles, dite de la fonction de pénalité, consiste à substituer à l'équation de continuité originelle la relation :

$$\frac{\partial u_i}{\partial x_i} + \varepsilon p = 0 \tag{2.35}$$

où ε est un paramètre de pénalité. La pression est ensuite éliminée par condensation. Le champ de pression peut être déduit ultérieurement par l'équation (2.35). Il a été montré que la perte de précision engendrée par cette substitution n'est pas significative pourvu que ε soit suffisamment petit. L'ajout du terme εp équivaut à introduire un terme de légère compressibilité du fluide puisque, dans cette approche, le champ de vitesse n'est plus à divergence nulle. D'autre part, Braaten et Patankar[4] soulignent que l'élimination de la pression se révèle être un sévère désavantage pour le développement de procédures de correction par blocs efficaces, du fait que le champ de pression ne peut être corrigé indépendamment du champ de vitesse. Lorsque ce dernier est corrigé, le champ de pression est modifié d'une manière prescrite par la relation de pénalité sans qu'aucun contrôle ne soit possible.

Dans une autre approche, la relation de pénalité (2.35) est utilisée mais la pression est conservée en tant que variable du problème au même titre que la vitesse. En ce qui nous concerne, la mise en œuvre de méthodes itératives nous conduit à résoudre des systèmes du type :

$$\tilde{\mathbf{K}}\left(\mathbf{u} - \mathbf{u}^*\right) = \mathbf{f} - \mathbf{K}\mathbf{u}^* \tag{2.36}$$

où $\mathbf{u}=<u_i, p>$ est le vecteur des inconnues nodales, $\mathbf{u}^*$ une approximation antérieure de $\mathbf{u}$ et $\tilde{\mathbf{K}}$ la matrice de l'itération. L'équation de continuité (sans le terme de pénalité) est utilisée pour évaluer le membre de droite de l'équation (2.36). Seule la matrice $\tilde{\mathbf{K}}$ est remplacée par la matrice pénalisée $\tilde{\mathbf{K}}_\varepsilon$. Ce faisant, si le processus itératif converge, le champ de vitesse obtenu vérifie bien la contrainte d'incompressibilité.

La matrice de pénalité $\tilde{\mathbf{K}}_\varepsilon$ se distingue de la matrice $\tilde{\mathbf{K}}$ par des termes supplémentaires sur la diagonale principale. Ces termes sont issus de la discrétisation par éléments finis de la relation de pénalité (2.35) qui introduit des matrices élémentaires du type :

$$P_{pp} = \int_{\Omega_e} \{n_p\}\, \varepsilon\, \{n_p\}\, d\Omega_e \qquad (2.37)$$

i_bloc=j_bloc=n_deg_lib_noeud

Par exemple, les relations (2.30) et (2.31) sont en fait traitées respectivement sous la forme :

$$\mathbf{K}_\varepsilon(\mathbf{U}^{(k)})(\mathbf{U}^{(k+1)}-\mathbf{U}^{(k)})=\mathbf{F}(\mathbf{U}^{(k)})-\mathbf{K}(\mathbf{U}^{(k)})\mathbf{U}^{(k)} \qquad (2.38)$$

$$\mathbf{K}^T_\varepsilon(\mathbf{U}^{(k)})(\mathbf{U}^{(k+1)}-\mathbf{U}^{(k)})=\mathbf{F}(\mathbf{U}^{(k)})-\mathbf{K}(\mathbf{U}^{(k)})\mathbf{U}^{(k)} \qquad (2.39)$$

De ce fait, nous n'utilisons pas à proprement parler la méthode de substitution ou celle de Newton-Raphson. Néanmoins, nous omettrons de rappeler à chaque fois la présence de ce terme de pénalité dans les matrices $\tilde{\mathbf{K}}$ considérées dans la suite de ce mémoire et nous nous permettrons quelques abus de langage au profit d'une plus grande clarté de rédaction.

2.2. Outils informatiques

2.2.1. Généralités

Un code aux éléments finis est généralement constitué de trois parties :

- un mailleur,
- un solveur
- un post-traitement.

Le tout peut éventuellement être précédé d'un logiciel de C.A.O. permettant de saisir des géométries complexes (*fig. 2.4*). Dans notre cas, le mailleur est soit « fait maison » lorsqu'il s'agit de géométries élémentaires telles que des cavités carrées ou rectangulaires à maillage structuré, soit commercial dans le cas de géométries plus complexes à maillage non structuré. En effet, lors de la phase de développement d'un code, il est pratique de concevoir ce qu'on pourrait appeler un « mailleur d'étude », c'est à dire un programme relativement court permettant de mailler une géométrie simple (dans notre cas, un carré ou un rectangle) afin de procéder à des tests rapides dans l'optique, par exemple, d'un débogage ou de la mise à l'essai d'un nouvel algorithme. Les résultats numériques, présentés au chapitre 4, ont été obtenus ainsi.

Le mailleur effectue la discrétisation géométrique du domaine de calcul. Il fournit au solveur un fichier d'entrée comportant :

* la géométrie du maillage (coordonnées des nœuds),
* les connectivités entre éléments (la liste des numéros des nœuds composant chacun des éléments),
* les conditions aux limites du problème.

Le solveur est notre principal objet d'intérêt. Un solveur aux éléments finis doit, à partir du fichier d'entrée, accomplir les tâches suivantes :

1 - Calcul des matrices élémentaires

2 - Assemblage de la matrice globale

3 - Résolution du système final

2.2.2. Technique de programmation

Le présent travail s'oriente vers la création d'une bibliothèque d'objets permettant de construire des solveurs aux éléments finis*. L'iPSC-860 propose à l'heure actuelle un compilateur Fortran 77 et un compilateur C. Or, ni le langage C, ni le langage 77 ne *supporte* cette dernière, au sens où ces deux langages ne fournissent pas de mécanismes particuliers qui facilitent sa mise en œuvre. Le langage C^{++}, par contre, a été conçu pour être un langage naturel pour ce type de programmation. Ce langage a été développé à partir du C et, à peu d'exceptions, retient ce dernier comme un sous ensemble. C'est pourquoi, le langage C a été adopté, dans la mesure où il est structuré (contrairement au Fortran 77) et facilitera, à terme, le portage des programmes en C^{++}.

Au fil des ans, le point primordial de la conception des programmes s'est écarté de la conception des procédures pour se rapprocher de l'organisation des données. Cela reflète, entre autres choses, l'augmentation de la taille des programmes. Dans le cadre de ce travail, nous disposons initialement d'une base d'unités de programmes qui ont été conçue suivant ce précepte. Si l'on désigne par *module* un ensemble de procédures dépendantes et les données qu'elles manipulent, le paradigme de programmation adopté est :

* choisir le module dont on a besoin.
* morceler le programme de telle sorte que les données soient cachées dans les modules.

Ce paradigme est aussi connu comme le principe d' « encapsulation des données ».

2.2.3. Solveur de base

En pratique, un problème aux éléments finis donné se distingue d'un autre par :

1. l'élément choisi,
2. les équations discrétisées traitées,
3. la méthode numérique sélectionnée pour résoudre le système d'équations algébriques résultant.

* Cf. paragraphe 1.2.2

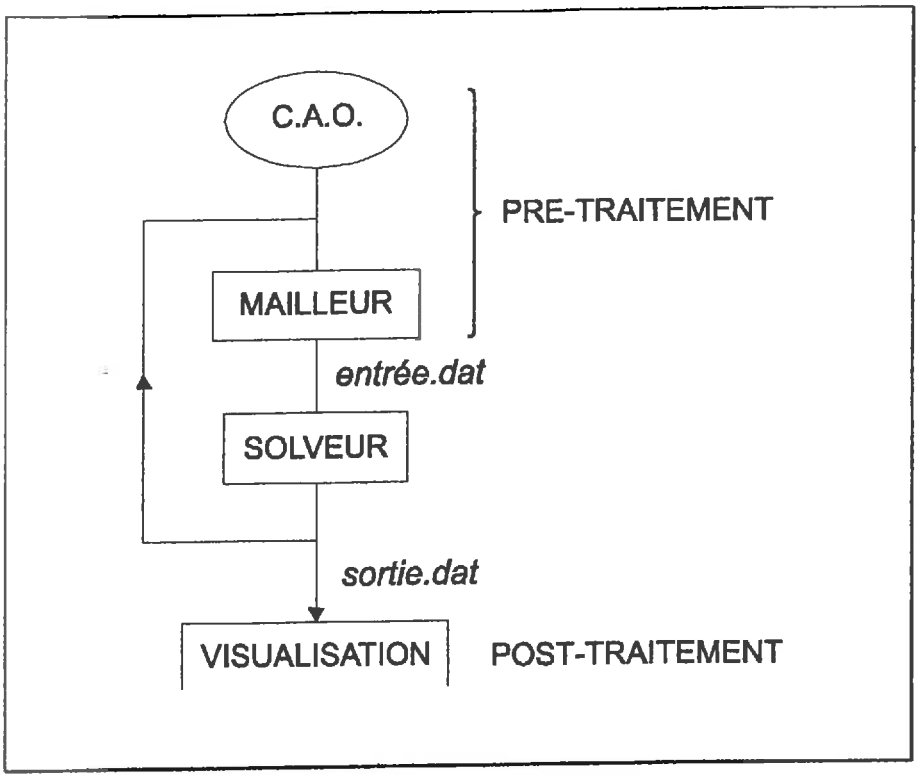


Figure 2.4 : Architecture type d'un code aux éléments finis

C'est pourquoi le solveur a été décomposé en trois modules, comme l'indique le tableau suivant :

1	Calcul des fonctions d'interpolation	*Module « élément »*
2	Calcul des paramètres nécessaires au calcul des intégrales	
3	Calcul des matrices élémentaires - matrice rigidité élémentaire k_e - matrice masse élémentaire m_e et des vecteurs élémentaires - vecteur sollicitation élémentaire f_e - vecteur résidu élémentaire r_e	*Module « matrices et vecteurs élémentaires »*
4	Assemblage des matrices élémentaires (dans K_globale ou M_globale) et des vecteurs résidu élémentaire (dans résidu)	*Module « assemblage et Résolution »*
5	Factorisation L.U. (résultat dans K_globale)	
6	Résolution	

Les étapes (1), (2), (3) et (4) mentionnées dans ce tableau, sont effectuées pour chaque élément du maillage considéré.

L'un des intérêts de la programmation par encapsulation de données est de limiter, de par son principe même, le nombre de passages de paramètres entre les différentes parties du programme. Ceci se limite ici :

- au numéro de l'élément courant *i_elem* (boucle sur le nombre d'éléments),
- aux matrices et vecteurs élémentaires correspondants,
  - la matrice de rigidité élémentaire k_e,
  - la matrice masse élémentaire m_e si nécessaire,
  - le vecteur sollicitation élémentaire f_e,
- aux caractéristiques nécessaires au calcul des intégrales.

Ce modèle de programmation présente deux autres avantages. D'une part, les risques de propagation des éventuelles erreurs de programmation se limitent à un seul module, ce qui simplifie et donc accélère les rectifications ou, en d'autres termes, le « débogage ». D'autre part, cela permet le réemploi d'un module dans un autre solveur. Ainsi, à partir de quelques modules, nous pouvons obtenir plusieurs solveurs traitant des problèmes très différents, sans réécrire un programme aux éléments finis : il suffit de combiner différemment les modules adéquats déjà créés.

2.2.4. validation du code séquentiel

Il ne s'agit pas à proprement parler d'une validation car celle-ci nécessiterait des résultats beaucoup plus complets. Nous nous contentons ici de tester le programme séquentiel à travers les deux problèmes de mécanique des fluides suivants : l'écoulement de Couette généralisé, et la cavité carrée entraînée. Il s'agit d'écoulements laminaires et stationnaires d'un fluide visqueux newtonien incompressible.

Les calculs ont été effectués sur un processeur i-860. La discrétisation spatiale se base sur des éléments triangulaires de Taylor-Hood (*fig. 2.2*). La méthode de résolution du système matriciel obtenu est soit la méthode des approximations successives, soit la méthode de Newton-Raphson. Les systèmes linéaires obtenus après linéarisation sont résolus par une factorisation LU de la matrice (décomposition de Doolittle). L'utilisation d'une méthode directe permet une résolution

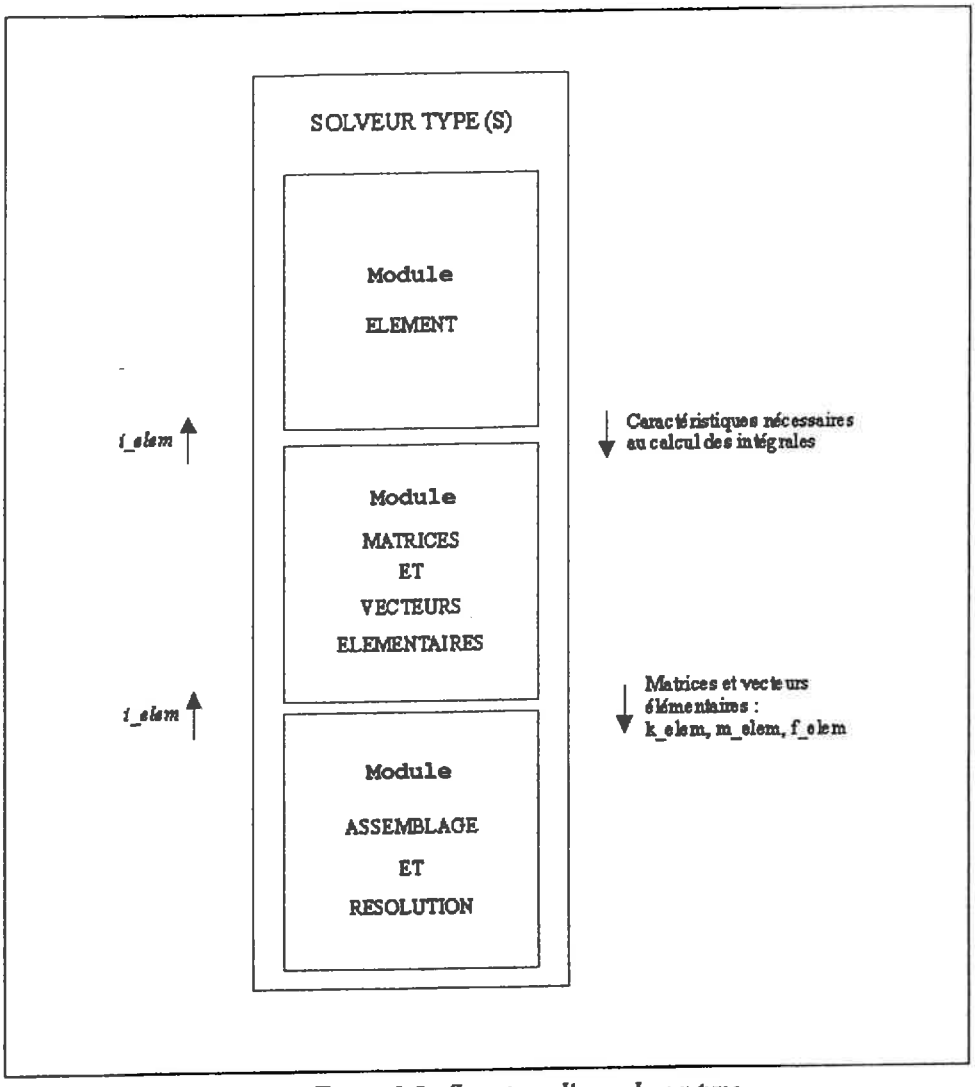


Figure 2.5 : Structure d'un solveur type

rapide et précise de ces systèmes. Par contre, le stockage mémoire étant important et la mémoire limitée à 16 Mbytes, nous subissons une limitation assez sévère au niveau de la finesse du maillage tolérée (41×41 nœuds maximum).

2.2.4.1. L'écoulement de Couette généralisé

Ce problème a été choisi car il admet une solution analytique, ce qui constitue une base idéale pour tester un programme.

1. Configuration du problème

On considère l'écoulement plan, isotherme et totalement développé d'un fluide newtonien incompressible entre deux plaques parallèles supposées de dimensions infinies. La plaque

supérieure subit un mouvement de translation de vitesse $\vec{U} = U \cdot \vec{x}$. La plaque inférieure est immobile.

2. *Solution analytique*

➢ équations aux dimensions

Pour le problème décrit ci-dessus, l'écriture de la conservation de la masse et de la quantité de mouvement du système se réduit à :

$$\frac{dp_g}{dx} = \mu \frac{d^2u}{dy^2}$$

En posant $-a = \frac{dp_g}{dx}$, où a est une constante positive ou négative, il vient :

$$u(y) = \frac{U}{2} \cdot \left(1 + \frac{y}{y_0}\right) + \frac{a{y_0}^2}{2\mu} \cdot \left(1 - \frac{y^2}{{y_0}^2}\right)$$

Par unité de largeur, le débit s'écrit :

$$q_v = Uy_0 + \frac{2ay_0^3}{3\mu}$$

ce qui impose à la constante a de vérifier la relation suivante :

$$\frac{-2ay_0^2}{\mu U} \leq 3$$

Posons $\frac{-2ay_0^2}{\mu U} = \alpha$ avec $\alpha \leq 3$, il vient :

$$\frac{dp_g}{dx} = \alpha \cdot \frac{\mu U}{2y_0^2}$$

➢ équations sans dimension

En introduisant les grandeurs sans dimension suivantes :

$$x = \frac{x}{2y_0},\ y = \frac{y + y_0}{2y_0},\ u = \frac{u}{U} \text{ et } p = \frac{p_g}{\rho U^2},$$

Le nombre de Reynolds s'écrit $Re = 2\rho U y_0/\mu$, et l'on obtient :

* Si $\alpha = 0$ $\quad u(y) = y \quad$ et $\quad \frac{dp}{dx} = 0 \quad$ ($u_{max}=1$ en $y_{max}=1$)

* Si $\alpha \neq 0$ $\quad u(y)=\alpha y^2+(1-\alpha)y$ et $\dfrac{dp}{dx}=\dfrac{2\alpha}{Re}$ $\quad (u_{max}=\dfrac{(1-\alpha)^2}{4\alpha}$ en $y_{max}=\dfrac{1-\alpha}{2\alpha})$

où C est une constante.

3. Simulation numérique

En utilisant les grandeurs de référence mentionnées précédemment, l'écriture de la conservation de la masse et de la quantité de mouvement conduit au système au dérivées partielles suivant :

$$\begin{cases} \dfrac{\partial u}{\partial x}+\dfrac{\partial v}{\partial y}=0 \\ u\dfrac{\partial u}{\partial x}+v\dfrac{\partial u}{\partial y}=-\dfrac{\partial p}{\partial x}+\dfrac{1}{Re}\left(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}\right) \\ u\dfrac{\partial v}{\partial x}+v\dfrac{\partial v}{\partial y}=-\dfrac{\partial p}{\partial y}+\dfrac{1}{Re}\left(\dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2}\right) \end{cases}$$

avec les conditions aux limites :

$$u(0,y)=\alpha y^2+(1-\alpha)y,\ v(0,y)=0$$
$$u(x,0)=v(x,0)=0$$
$$u(1,y)=\alpha y^2+(1-\alpha)y,\ v(1,y)=0$$
$$u(x,1)=1,\ v(x,1)=0$$
$$p(0,1)=1$$

Le traitement de ce problème par la méthode des éléments finis conduit au système suivant :

$$\sum_{n_e}\left\langle \delta p^{(k)}\right\rangle \cdot \int_{\Omega_e}\{n_p\}\cdot\left\langle \frac{\partial n}{\partial x_j}\right\rangle \cdot d\Omega_e \cdot \left\{u_j^{(k)}\right\}=0 \qquad (2.40)$$

$$\sum_{n_e}\left\langle \delta u_i^{(k)}\right\rangle \cdot \int_{\Omega_e}\{n_u\}\cdot\langle n_u\rangle \cdot d\Omega_e \cdot \left\{u_i^{(k)}\right\} \qquad (2.41)$$

$$+\sum_{n_e}\left\langle \delta u_i^{(k)}\right\rangle \cdot \int_{\Omega_e}\{n_u\}\cdot u_j \cdot\left\langle \frac{\partial n_u}{\partial x_j}\right\rangle \cdot d\Omega_e \cdot \left\{u_i^{(k)}\right\}=$$

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_u\} \cdot \left\langle \frac{\partial n_p}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ p^{(k)} \right\}$$

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \frac{1}{Re} \cdot \left\langle \frac{\partial n_u}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ u_i^{(k)} \right\}$$

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \frac{1}{Re} \cdot \left\langle \frac{\partial n_u}{\partial x_i} \right\rangle \cdot d\Omega_e \cdot \left\{ u_j^{(k)} \right\}$$

Le profil de vitesse est imposé en entrée et en sortie du domaine pour une valeur donnée de alpha (α=–3, α=–2, α=–1, α=0, α=1, α=2 et α=3). La valeur de la pression est fixée à 1 dans le coin supérieur gauche du domaine.

Un maillage régulier a été réalisé sur une portion de conduite de longueur unité et des tests ont été effectués pour différentes grilles ((5×5), (11×11) et (31×31) nœuds). La valeur du nombre de Reynolds a été successivement fixée à 100, 500 et 1000. Les calculs ont été réalisés en double précision sur un processeur de l'iPSC i860. Le critère d'arrêt est défini par :

$$\frac{\left\| \vec{u}_{analytique} - \vec{u}_{numérique} \right\|}{\left\| \vec{u}_{analytique} \right\|} \leq \varepsilon$$

avec $\varepsilon=10^{-13}$. À titre d'exemple, nous avons réuni dans le *tableau 2.1* le nombre d'itérations nécessaires pour atteindre ce critère suivant la valeur du nombre de Reynolds, dans le cas de la grille (31×31).

Nous comparons les gradients de pression longitudinaux et transversaux ainsi que les profils de vitesse intermédiaires obtenus à la solution analytique du problème. Sur la *figure 2.7*, nous avons reporté pour toutes les valeurs de l'abscisse x de la grille (31×31) et pour α=3, α=2, α=1, α=0, α=–1, α=–2 et α=–3, les approximations numériques de la composante u de la vitesse en fonction de l'ordonnée y. On constate que tous les profils de vitesse u(y) se superposent parfaitement avec le profil analytique.

A partir de la même grille et pour les mêmes valeurs de la constante α, ont été portées sur la *figure 2.8* les valeurs numériques de la pression en fonction de l'abscisse x, pour toutes les valeurs de l'ordonnée y. L'ensemble de ces points décrit une série de segments de droites de pente $2\alpha/Re$, concourant en (x=1,p=1).

En conclusion, la solution numérique recouvre parfaitement la solution analytique du problème. Ceci s'explique du fait que cette dernière, exposée précédemment, est de type polynomial. Pour $\alpha \neq 0$, la solution exacte est un polynôme du deuxième degré pour la vitesse et un polynôme du premier degré pour la pression. Ainsi, en choisissant les éléments de Taylor-Hood, le calcul numérique fournit la solution exacte du problème aux erreurs d'arrondi près et ceci quelque soit le maillage. On bénéficie, par ailleurs, d'une convergence quadratique caractéristique de la méthode de Newton-Raphson *(figure 2.6)*.

Tableau 2.1 :
Nombre d'itérations pour la grille (31x31)

α	Méthode des approximations successives			Méthode de Newton-Raphson		
	Re=100	Re=500	Re=1000	Re=100	Re=500	Re=1000
3	14	20	19	4	6	7
2	12	21	22	4	6	7
1	7	12	13	4	6	7
0	9	12	13	4	5	– / 4*
–1	9	13	14	4	6	8 / 4*
–2	10	13	16	4	6	– / 5*
–3	10	14	15	4	7	– / 4*

Le symbole "–" signifie que la méthode considérée n'a pas convergé.
* Méthode de Newton-Raphson précédée d'une itération de la méthode des approximations successives.

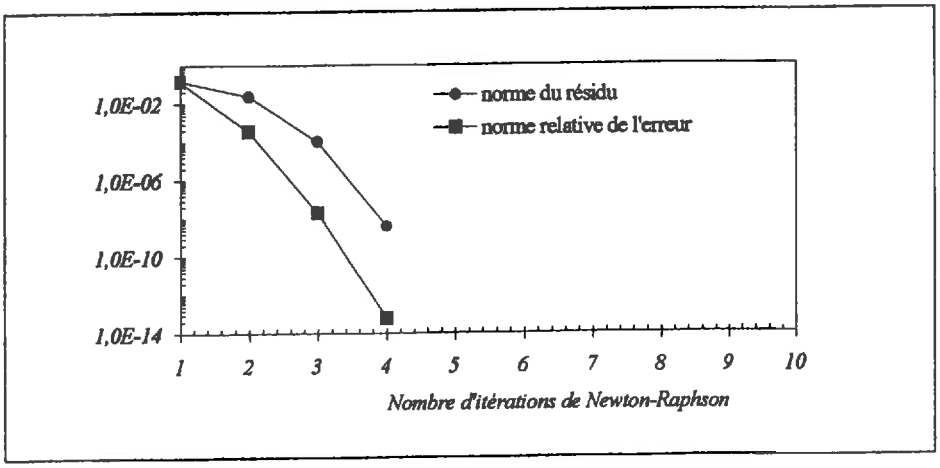


Figure 2.6 : Convergence quadratique de la méthode de Newton-Raphson

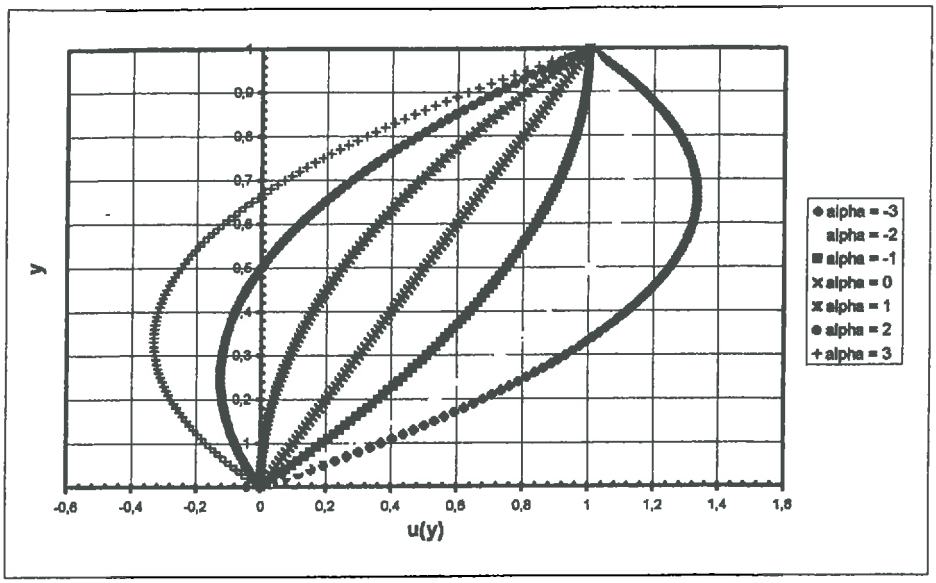


Figure 2.7 : profil transversal de vitesse u(y) en fonction de α

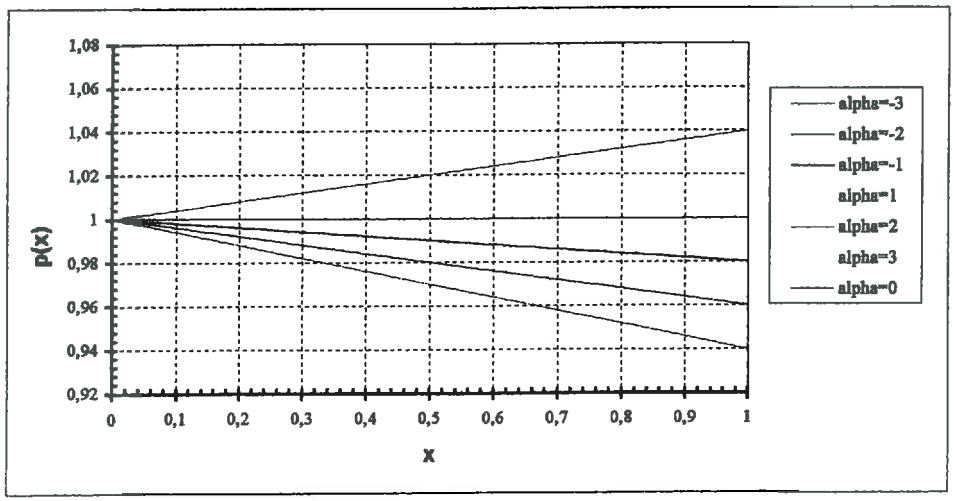
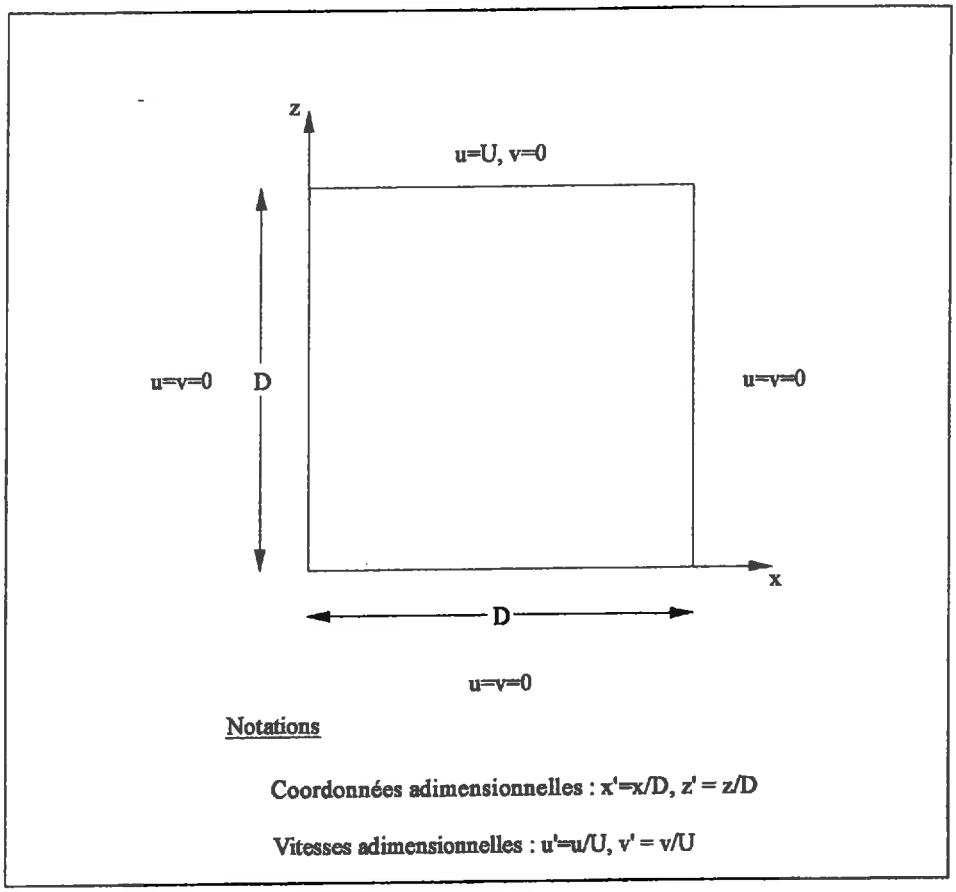


Figure 2.8 : répartition de la pression en fonction de l'abscisse

2.2.4.2. La cavité entraînée

1. Configuration du problème



2. Mise en équations

➢ équations aux dimensions

- conservation de la masse :

$$\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}=0$$

- conservation de la quantité de mouvement :

$$\begin{cases} \rho u\dfrac{\partial u}{\partial x}+\rho v\dfrac{\partial u}{\partial y}=-\dfrac{\partial p_g}{\partial x}+\mu\left(\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}\right) \\ \rho u\dfrac{\partial v}{\partial x}+\rho v\dfrac{\partial v}{\partial y}=-\dfrac{\partial p_g}{\partial y}+\mu\left(\dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2}\right) \end{cases}$$

➢ équations sans dimensions

Nous choisissons la largeur de la cavité D comme longueur de référence et la vitesse d'entraînement U comme vitesse de référence. En posant $p=p_g/\rho U^2$, on obtient :

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y} = 0 \\ u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} = -\dfrac{\partial p}{\partial x} + \dfrac{1}{Re}\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) \\ u\dfrac{\partial v}{\partial x} + v\dfrac{\partial v}{\partial y} = -\dfrac{\partial p}{\partial y} + \dfrac{1}{Re}\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right) \end{cases}$$

avec les conditions aux limites :

$$u(0,y) = u(1,y) = u(x,0) = 0,$$
$$v(0,y) = v(1,y) = v(x,0) = v(x,1) = 0,$$
$$u(x,1) = 1$$
$$p(0,1) = 1$$

3. Simulation numérique

Les conditions aux limites étant de type Dirichlet, on obtient à nouveau les équations aux éléments finis (2.40) et (2.41) utilisées dans le cadre de l'écoulement de Couette généralisé.

Dans le cadre de la validation du code séquentiel, notre choix s'est porté sur le cas de la cavité entraînée, ceci en raison de l'abondante littérature y afférant. Pour que la confrontation des résultats soit rendue possible, un profil uniforme a été imposé malgré la discontinuité des conditions aux limites qui en résulte dans le voisinage des coins supérieurs.

L'exploitation des résultats porte sur :

* Les profils d'une composante du vecteur vitesse sur les sections centrales *(fig. 2.9 et)*.

* Les propriétés des zones tourbillonnaires en terme de localisation spatiale et d'intensité *(fig. 2.11 et tab. 2.3)*.

La confrontation avec les résultats numériques de divers auteurs nous conduit aux observations suivantes :

* Les profils de vitesse sont satisfaisants pour des valeurs du nombre Reynolds inférieures à 10^3 (*tab. 2.2*). Au delà, il faudrait affiner le maillage, notamment dans les zones proches pariétales.

* L'intensité de la zone tourbillonnaire principale est comparable à celle obtenue par Ghia & al, pour des nombre de Reynolds allant jusqu'à 10^3. Pour les valeurs supérieures, on obtient une légère surestimation, vraisemblablement due à une densité de maillage pas assez élevée.

* Le centre de la zone tourbillonnaire se décale, conformément aux autres auteurs, vers le centre de la cavité (fig. 2.12).

* Le calcul met en évidence des zones tourbillonnaires de faible étendue au niveau des coins gauche et droit, dans le bas de la cavité. Toutefois, la gamme de valeurs de la fonction de courant n'est pas comparable à celle obtenue par Ghia et al. Ceci nous amène à penser qu'il s'agit là d'un effet de maillage (pas assez resserré).

En conclusion, nous estimons que les prédictions sont globalement satisfaisantes. Par contre, nous constatons que leur qualité se détériore quelque peu avec l'augmentation des non-linéarités. Nous en attribuons la cause, a priori, à l'utilisation d'un maillage beaucoup moins dense (41×41) que les auteurs pris comme référence (129×129 et 256×256). Toutefois, la mémoire disponible par processeur ne nous permet pas d'envisager un affinement afin de vérifier cette hypothèse.

Tableau 2.2 :
Extrema des profils de vitesse le long des sections centrales

Re = 100

Auteurs	u_{min}	y_{min}	v_{max}	x_{max}	v_{min}	x_{min}
Ghia & al (†)	-0.2109	0.4531	0.1753	0.2344	-0.2453	0.8047
Bruneau & Jouron (‡)	-0.2106	0.4531	0.1786	0.2344	-0.2521	0.8125
Présent travail (§)	-0.214	0.461	0.179	0.239	-0.251	0.794

Re = 400

Auteurs	u_{min}	y_{min}	v_{max}	x_{max}	v_{min}	x_{min}
Ghia & al	-0.3273	0.2813	0.302	0.2266	-0.4499	0.8594
Présent travail	-0.327	0.273	0.302	0.239	-0.436	0.854

Re = 1000

Auteurs	u_{min}	y_{min}	v_{max}	x_{max}	v_{min}	x_{min}
Ghia & al	-0.3829	0.1719	0.3709	0.1563	-0.5155	0.9063
Vanka (**)	-0.3798	0.1680	0.3669	0.1563	-0.5186	0.9102
Zhang (††)	-0.3901	0.1699	0.3785	0.1582	-0.5284	0.9082
Bruneau & Jouron	-0.3764	0.1602	0.3665	0.1523	-0.5208	0.9102
Présent travail	-0.389	0.175	0.376	0.146	-0.536	0.905

Re = 3200

Auteurs	u_{min}	y_{min}	v_{max}	x_{max}	v_{min}	x_{min}
Ghia & al	-0.4193	0.1016	0.4277	0.0938	-0.5405	0.9453
Présent travail	-0.499	0.120	0.452	0.0955	-0.590	0.946

Re = 5000

Auteurs	u_{min}	y_{min}	v_{max}	x_{max}	v_{min}	x_{min}
Ghia & al	-0.4364	0.0703	0.4365	0.0781	-0.5541	0.9531
Bruneau & Jouron	-0.4359	0.0664	0.4259	0.0762	-0.5675	0.9590
Présent travail	-0.474	0.120	0.469	0.0955	-0.574	0.94

(†) Grille 129x129 pour Re ≤ 3200 et 257x257 pour Re = 5000.
(‡) Grille 41x41.
(§) Grille 41x41.
(**) et (††) Grille 256x256.

Tableau 2.3 :

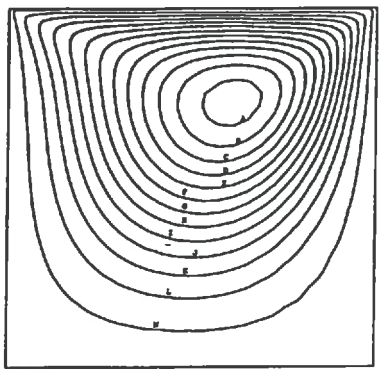
Intensité et localisation des tourbillons primaires

		Re				
		100	400	1000	3200	5000
Ψ_{min}	Ghia & al	-0.103423	-0.113909	-0.117929	-0.120577	-0.118966
	Bruneau & J.	-0.1026	——	-0.1163	——	-0.1142
	Ramasnamy-	-0.1030	-0.1140	-0.1180	——	-0.1190
	Présent travail	-0.1036	-0.11405	-0.11957	-0.12737	-0.12947
Position Des centres	Ghia & al	0.6172 ; 0.7344	0.5547 ; 0.6055	0.5313 ; 0.5625	0.5165 ; 0.5469	0.5117 ; 0.5352
	Bruneau & J.	0.6172 ; 0.7344	——	0.5313 ; 0.5586	——	0.5156 ; 0.5352
	Ramasnamy	——	——	——	——	——
	Présent travail	0.6171 ; 0.7357	0.5545 ; 0.6140	0.5314 ; 0.5754	0.50 ; 0.5439	0.50 ; 0.5403

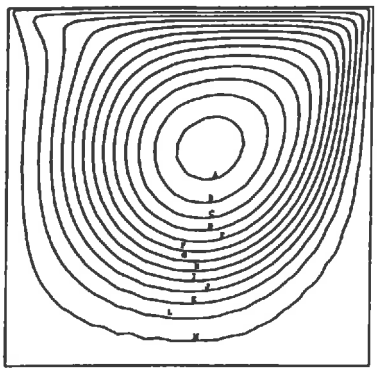
Tableau 2.4 :

Valeurs des niveaux de la fonction de courant

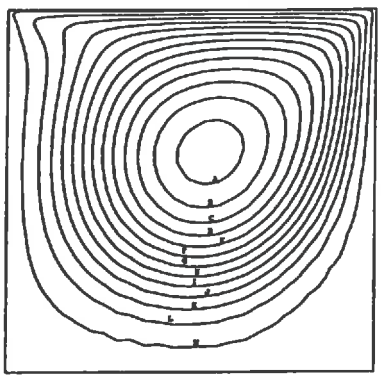
	Re					
	100	200	400	1000	3200	5000
A	-0.09962	-0.1096	-0.1096	-0.1149	-0.1220	-0.1234
B	-0.09164	-0.01008	-0.1008	-0.1056	-0.1112	-0.1114
C	-0.08366	-0.09205	-0.09205	-0.09631	-0.1004	-0.09936
D	-0.07569	-0.08325	-0.08325	-0.08700	-0.08956	-0.08732
E	-0.06771	-0.07445	-0.07445	-0.07770	-0.07875	-0.07528
F	-0.05973	-0.06565	-0.06565	-0.06839	-0.06795	-0.06323
G	-0.05175	-0.05685	-0.05685	-0.05909	-0.05715	-0.05119
H	-0.04377	-0.04805	-0.04805	-0.04978	-0.04634	-0.03914
I	-0.03580	-0.03925	-0.03925	-0.04048	-0.03554	-0.02710
J	-0.02782	-0.03045	-0.03045	-0.03117	-0.02474	-0.01506
K	-0.01984	-0.02165	-0.02165	-0.02187	-0.01394	-0.003014
L	-0.01186	-0.01285	-0.01285	-0.01256	-0.003133	——
M	- 0.003885	-0.004054	-0.004054	-0.003255	——	——



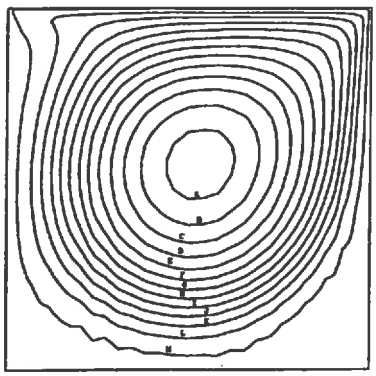
$Re = 100$



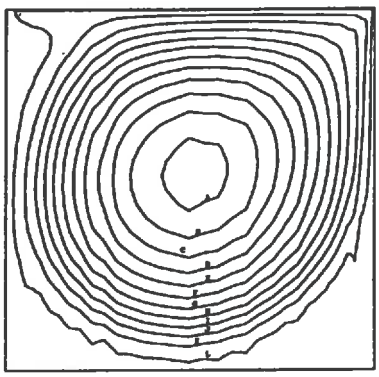
$Re = 200$



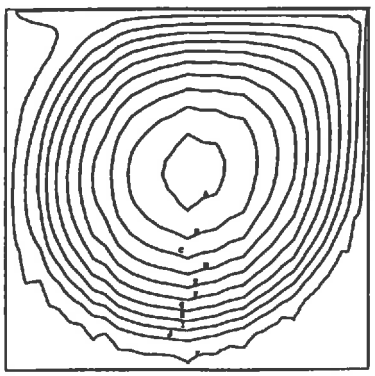
$Re = 400$



$Re = 1000$

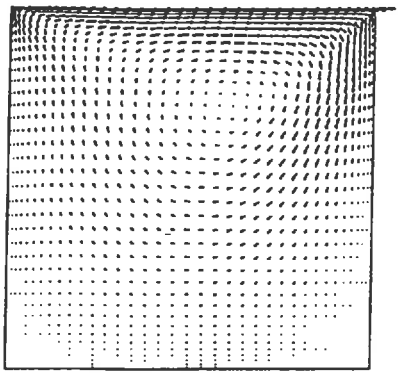


$Re = 3500$

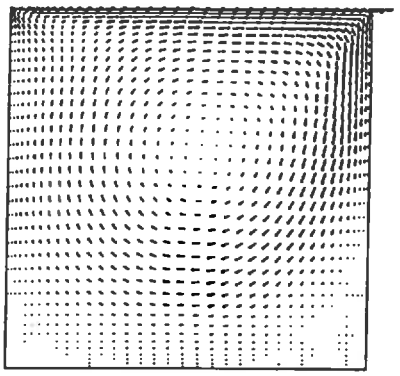


$Re = 5000$

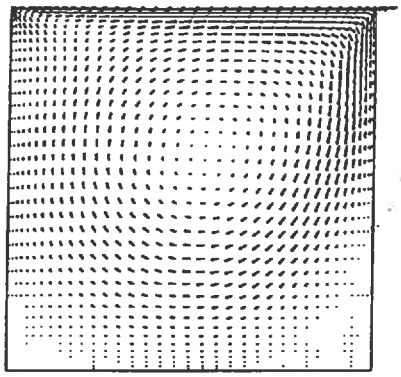
Fig. 2.11: Représentation des lignes de courant dans le cas de la cavité entraînée



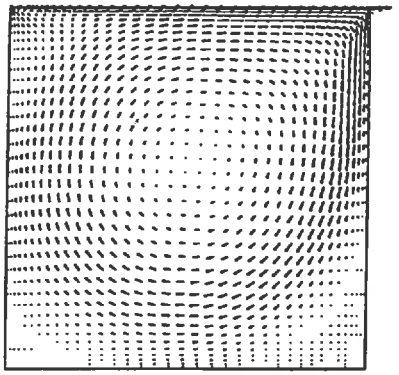
$Re = 100$



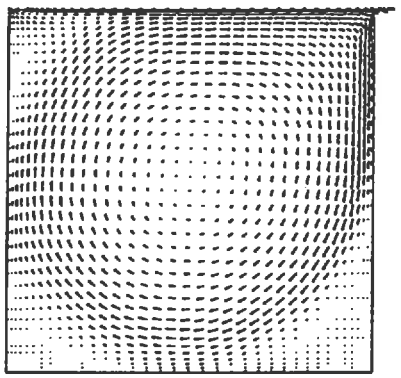
$Re = 200$



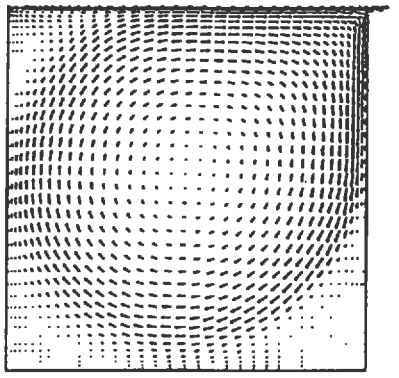
$Re = 400$



$Re = 1000$



$Re = 3500$



$Re = 5000$

Fig. 2.12 : Représentation du champ dynamique dans le cas de la cavité entraînée

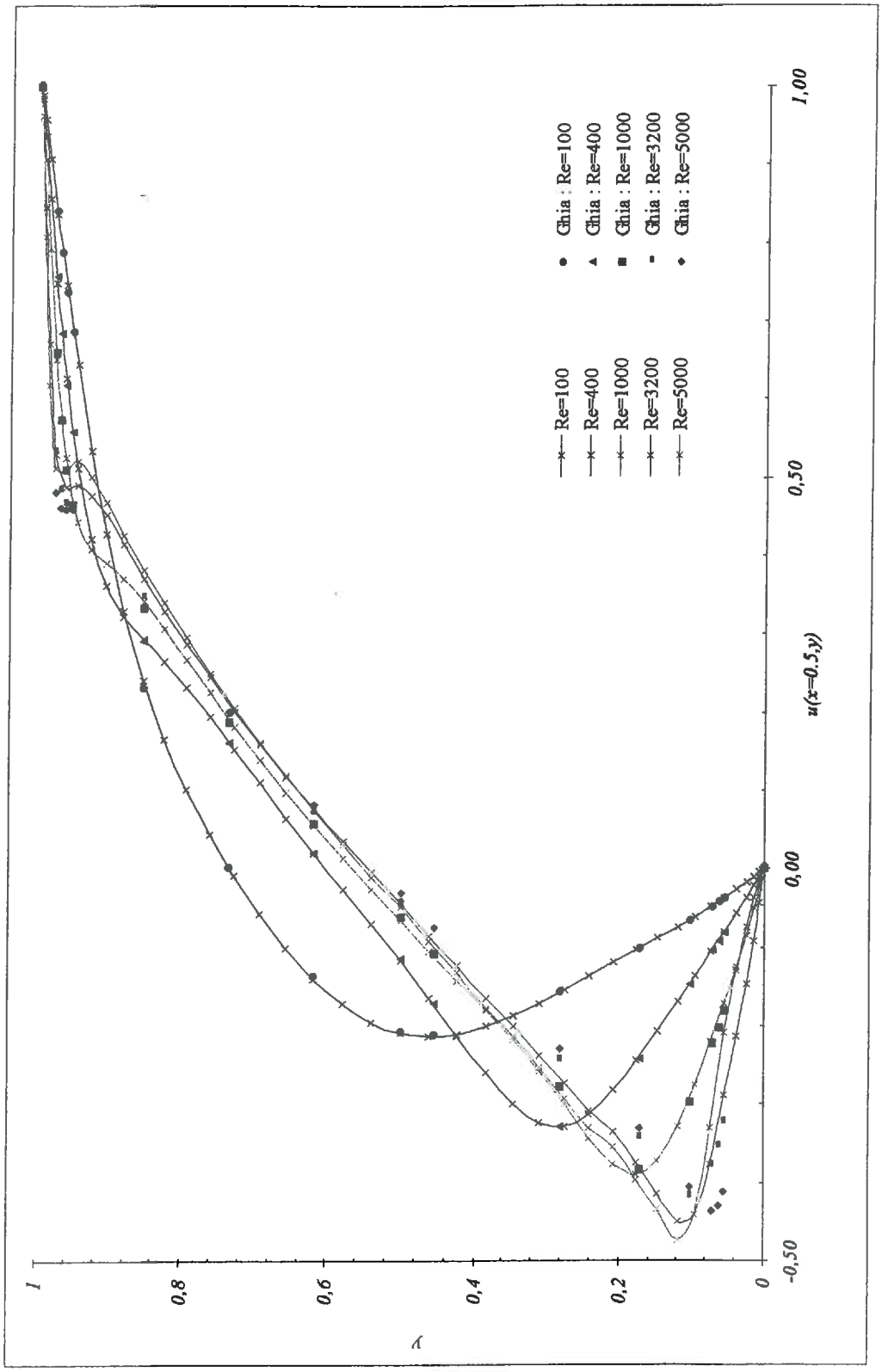


Figure 2.9 : Profil transversal de vitesse en x=0,5 pour la cavité entraînée

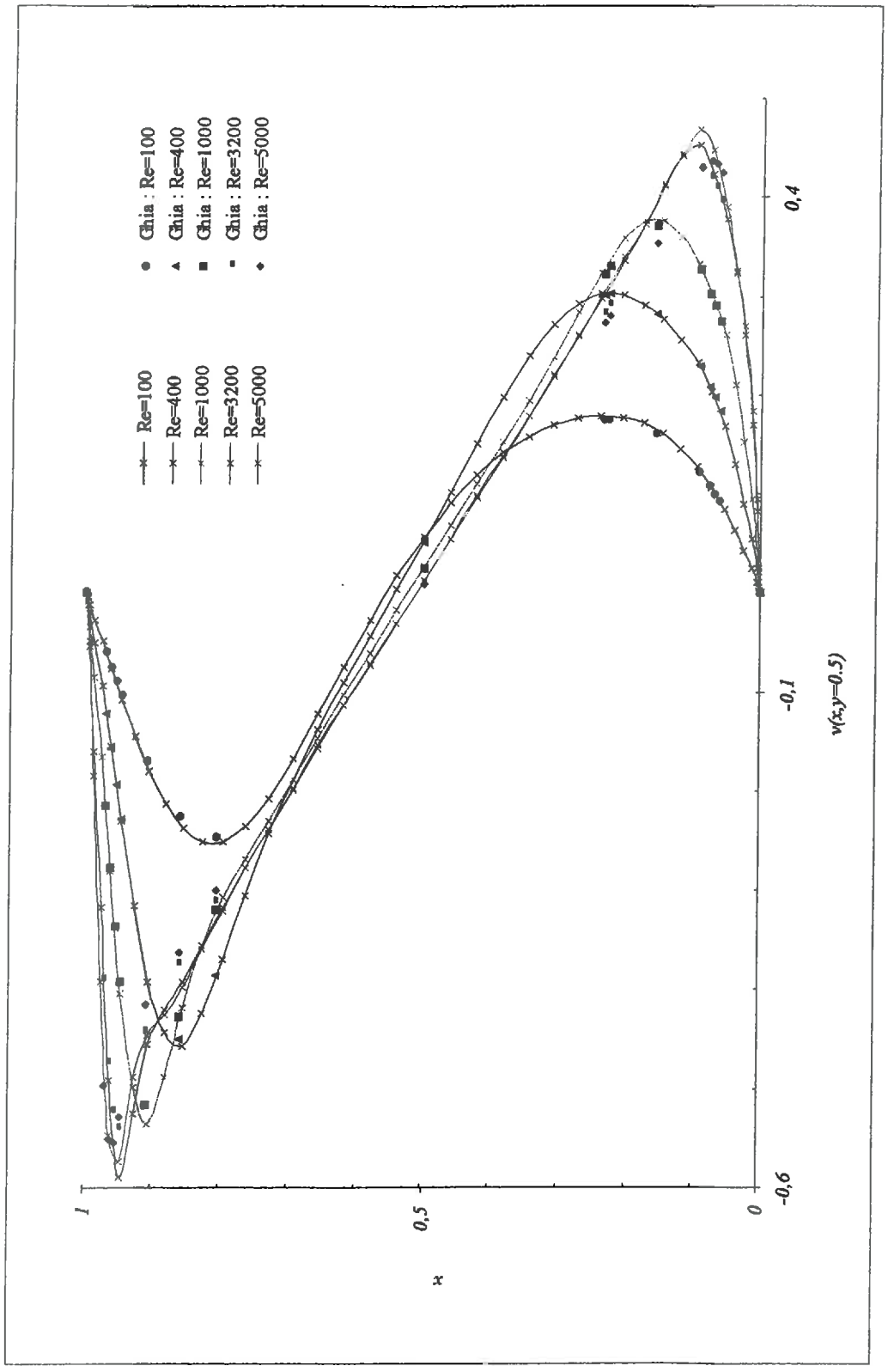


Figure 2.10 : Profil longitudinal de vitesse en y=0,5 pour la cavité entraînée

Chapitre 3

Méthodes itératives et calcul parallèle

Chapitre 3 :

Méthodes itératives et calcul parallèle

Nous avons vu au chapitre précédent que la méthode des éléments finis de Galerkin appliquée aux équations de Navier-Stokes stationnaires, ou instationnaires lorsqu'un schéma d'intégration temporel est utilisé, aboutit à la résolution d'un système d'équations algébriques non linéaires que nous représentons sous forme matricielle par:

$$\mathbf{A(x).x=b(x).} \tag{3.1}$$

La matrice globale du système $A_{N\times N}=(a_{ij})$ appartient à l'espace vectoriel des matrices réelles carrées régulières symétriques ou non, $\mathbf{x}_{N\times 1}$ est le vecteur inconnu et $\mathbf{b}_{N\times 1}$ le vecteur des sollicitations éventuellement dépendant de **x**. Résoudre le système (3.1) équivaut à chercher le vecteur **x** annulant le résidu $\mathbf{r(x)=b(x)-A(x).x}$.

Notre but est de résoudre ce système en parallèle.

Le système (3.1) étant a priori non linéaire, nous optons pour l'utilisation de méthodes itératives écrites sous forme incrémentale :

$$\Delta\mathbf{x}=\mathbf{H}.\Phi(\mathbf{r}^m,\ldots,\mathbf{r}^n) \tag{3.2}$$

avec $$\mathbf{x}^{k+1}=\mathbf{x}^k+\Delta\mathbf{x}$$

où le symbole Φ désigne une fonction linéaire et homogène du résidu, **H** un opérateur et $\mathbf{x}^k$ l'approximation à l'itération (k) de la solution exacte **u** du système (3.1). Choisir une méthode, c'est choisir **H** et Φ *(tableau 3.1)*. La formulation sous forme incrémentale telle que ci-dessus, nous paraît être intéressante car elle permet de mettre à jour une formulation générique de l'ensemble des méthodes itératives englobant à la fois des méthodes stationnaires (résolution de systèmes d'équations algébriques) et des méthodes instationaires ou pseudo-temporelles (résolution de systèmes d'équations différentielles). De ce fait, tous les programmes que nous développons dans le cadre du présent travail sont basés sur cette formulation. Ainsi, certaines routines du module « assemblage et résolution » ne nécessiteront aucune modification quelque soit la méthode numérique employée.

Tableau 3.1 :

Quelques expressions possibles de la fonction Φ et de l'opérateur H

Méthode itérative	H	Φ
Jacobi par points	$(A_d(x^k))^{-1}$	$r^k=b(x^k)-A(x^k).x^k$
Jacobi par blocs	$(A_b(x^k))^{-1}$	$r^k=b(x^k)-A(x^k).x^k$
Newton-Raphson	$\left(A_t\left(x^k\right)\right)^{-1} = \left(A\left(x^k\right) + \left(\frac{dA}{dx} \cdot x^k\right)\right)^{-1}$	$r^k=b(x^k)-A(x^k).x^k$
Prédiction-correction	$\Delta t.M^{-1}$	$\sum_{j=1}^{n} \lambda_j \cdot r_j$ λ_j: coefficients dépendants de l'ordre n de la méthode $r_j=b(x_{t+\Delta t-j\Delta t})-A(x_{t+\Delta t-j\Delta t}).\ x_{t+\Delta t-j\Delta t}$

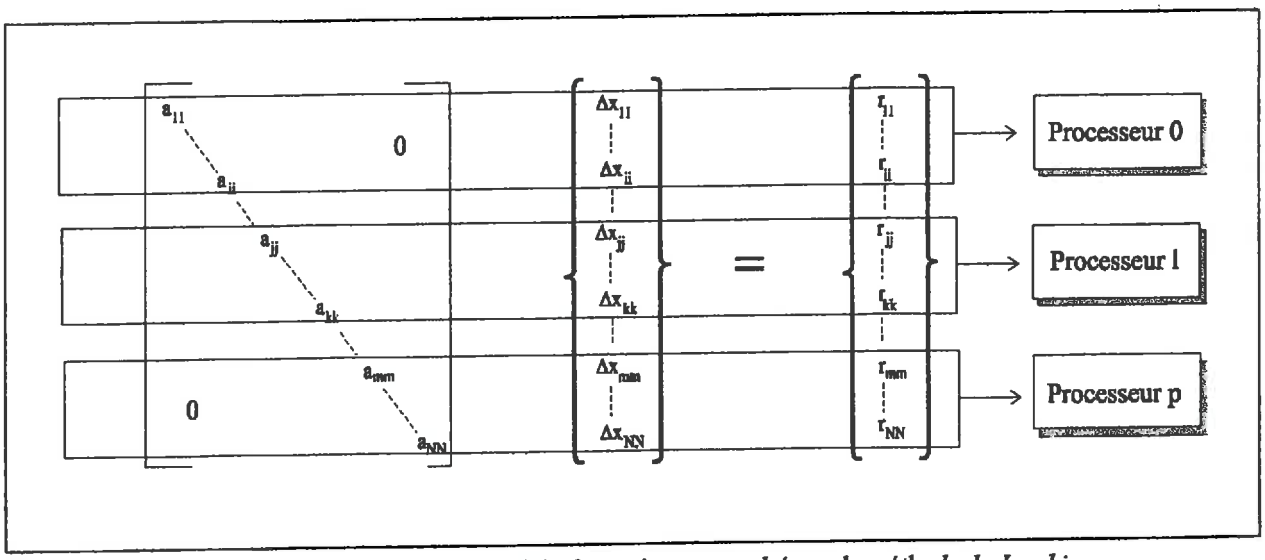


Figure 3.1 : Résolution en parallèle du système engendré par la méthode de Jacobi

Parmi les méthodes itératives, certaines s'adaptent plus naturellement au calcul parallèle que d'autres. En effet, rappelons que pour que deux algorithmes (a) et (b) puissent se dérouler en parallèle, ils doivent vérifier ***la condition de Bernstein***:

$$inp(a) \cap out(b) = \varnothing$$

et

$$out(a) \cap inp(b) = \varnothing$$

et

$$out(a) \cap out(b) = \varnothing$$

Cette condition est automatiquement vérifiée si, une fois la solution de rang (k) connue, il est possible de décomposer la charge de calculs sur p processeurs pour obtenir la solution de rang (k+1). Ceci est typiquement le cas des méthodes explicites pour lesquelles le calcul de la solution de rang (k+1) ne fait intervenir, par définition, que les solutions de rang inférieur ou égal à (k).

3.1. Méthodes itératives de base

De nombreuses méthodes itératives pour la résolution de systèmes algébriques (3.1) peuvent être obtenues par découpage de la matrice **A** (« matrix splitting »). Nous ne traiterons pas ici l'aspect théorique du découpage matriciel ainsi que l'analyse de la convergence des méthodes itératives correspondantes, ceci ayant été présenté en détails par Varga[41] ou Golub et Van Loan[16].

3.1.1. La méthode de Richardson

Etant donnée **I** la matrice identité, le découpage le plus simple de **A** consiste à écrire :

$$\mathbf{A}=\mathbf{I}-(\mathbf{I}-\mathbf{A}) \tag{3.3}$$

Par application du théorème du point fixe, ce découpage conduit à la méthode de Richardson, bien connue :

$$\mathbf{x}^{k+1}=\mathbf{x}^{k}+(\mathbf{b}-\mathbf{A}\cdot\mathbf{x}^{k})=\mathbf{x}^{k}+\mathbf{r}^{k} \tag{3.4}$$

En posant $\mathbf{H}=\mathbf{I}$ et $\Phi=\mathbf{r}^{k}$, on retrouve la formulation (3.2) .

Tout découpage matriciel $\mathbf{B=P-S}$ peut se ramener au découpage précédent en écrivant que $\mathbf{S=P-B}$ et en posant $\mathbf{A=(P^{-1}.B)}$. En d'autres termes, toute méthode itérative pouvant s'interpréter comme un découpage matriciel, peut également être vue comme étant la méthode de Richardson appliquée au système algébrique (3.1) *préconditionné* par une certaine matrice $\mathbf{P^{-1}}$, soit :

$$\mathbf{P^{-1}.A\ x = P^{-1}.b} \tag{3.5}$$

$$\mathbf{x^{k+1}=x^k+P^{-1}.(b-A\cdot x^k)=x^k+P^{-1}.r^k} \tag{3.6}$$

Posons : $$\mathbf{e^k=x^k-x^{k-1}}. \tag{3.7}$$

Nous avons : $$\mathbf{e^k=(I-P^{-1}.A)\,e^{k-1}}. \tag{3.8}$$

Ceci implique que :

$$\left\| \mathbf{e}^k \right\|_2 \le \max\left\{ \left| \lambda_{\min}^k \left(\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}\right) \right|, \left| \lambda_{\max}^k \left(\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}\right) \right| \right\} \left\| \mathbf{e}^0 \right\|_2 . \tag{3.9}$$

Nous désignons par $\rho(\mathbf{P^{-1}.A})$ le nombre de conditionnement de $\mathbf{P^{-1}.A}$:

$$\rho\left(\mathbf{P}^{-1}\mathbf{A}\right)=\frac{\lambda_{\max}\left(\mathbf{P}^{-1}\mathbf{A}\right)}{\lambda_{\min}\left(\mathbf{P}^{-1}\mathbf{A}\right)} \tag{3.10}$$

et $$\left\| \mathbf{e}^k \right\|_2 \le \left[\frac{\rho\left(\mathbf{P}^{-1}\mathbf{A}\right)-1}{\rho\left(\mathbf{P}^{-1}\mathbf{A}\right)+1} \right]^k \left\| \mathbf{e}^0 \right\|_2 \tag{3.11}$$

Le taux de convergence dépend de la valeur du nombre de conditionnement $\rho(\mathbf{P^{-1}.A})$ de la matrice de l'itération. Plus celle-ci sera faible, plus la convergence sera rapide. **Le préconditionneur P est donc un moyen d'accélérer la convergence** si il est choisi tel que :

$$\rho(\mathbf{P^{-1}.A}) \le \rho(\mathbf{A}) \tag{3.12}$$

En outre, une matrice de préconditionnement **P** doit avoir les propriétés suivantes :

1) être une bonne approximation de **A**

2) être facilement calculable
3) être raisonnablement creuse
4) les équations du type $\mathbf{P}\,\mathbf{x} = \mathbf{c}$ sont faciles à résoudre

Les préconditionneurs de Jacobi

Parmi les matrices de préconditionnement couramment employées, l'une des plus classiques et des plus élémentaires est ***le préconditionneur de Jacobi*** qui émane de la méthode du même nom. Il s'agit d'une matrice diagonale par points ou par blocs, dont l'utilisation génère des algorithmes très faciles à programmer et d'un niveau de parallélisme élevé.

3.1.1.1. La méthode de Jacobi par points

Rappelons brièvement le principe de cette méthode. Le passage d'un vecteur $\mathbf{x}^k$ de la suite au suivant se fait en corrigeant successivement une composante de ce vecteur. Ces corrections s'effectuent en annulant une composante du résidu $\mathbf{r}^k$.
A partir d'un vecteur $\mathbf{x}^0$, on construit un processus itératif de la manière suivante:

pour i=1 à N, calculer :

$$x_i^{k+1} = \frac{1}{a_{ii}(\mathbf{x}^k)}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{N} a_{ij}(\mathbf{x}^k)x_j^k\right) \tag{3.13}$$

Il faut, bien sûr, pour cela que $a_{ii}(\mathbf{x}^k)\neq 0$ pour i=1 à N. En supposant cette condition remplie et en notant $a_{ij}(\mathbf{x}^k)=a_{ij}^k$ pour alléger l'écriture, nous pouvons nous ramener à la méthode de Richardson en posant:

$$\mathbf{P}^{-1}=[\mathrm{diag}(\mathbf{A})]^{-1}=\begin{bmatrix} \frac{1}{a_{11}^k} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{a_{NN}^k} \end{bmatrix}. \tag{3.14}$$

En clair, à chaque itération le système à résoudre est le suivant:

$$\underbrace{\begin{bmatrix} a_{11}^k & & & 0 \\ & \ddots & & \\ & & a_{ii}^k & \\ & & & \ddots \\ 0 & & & a_{NN}^k \end{bmatrix}}_{A_d^k} \cdot \underbrace{\begin{Bmatrix} \Delta x_1^k \\ \vdots \\ \Delta x_i^k \\ \vdots \\ \Delta x_N^k \end{Bmatrix}}_{\cdot \Delta x^k} = \underbrace{\begin{Bmatrix} r_1^k \\ \vdots \\ r_i^k \\ \vdots \\ r_N^k \end{Bmatrix}}_{= r^k} \tag{3.15}$$

avec $$\Delta x_i^k = x_i^{k+1} - x_i^k$$

c'est à dire $$\Delta \mathbf{x} = \mathbf{x}^{k+1} - \mathbf{x}^k.$$

Présentée sous cette forme, cette méthode révèle ***une évidente prédisposition à la mise en œuvre en parallèle*** des calculs qu'elle nécessite. De part le caractère diagonale de la matrice $\mathbf{A_d}$, la parallélisation devient triviale. En effet, chaque ligne i du système

$$a_{ii}^k . \Delta x_i^k = r_i^k \qquad i=1,..,N \tag{3.16}$$

constitue une équation totalement découplée des autres. Sa résolution est, de ce fait, réalisable indépendamment de celle du reste du système. Autrement dit, la résolution de ces équations peut être effectuée en parallèle à chaque itération, en affectant à chaque processeur une partie seulement du système (3.15) *(fig. 3.1)*, voir une seule équation.

3.1.1.2. La méthode de Jacobi par blocs

Le préconditionneur précédent peut être amélioré en utilisant une décomposition par blocs de la matrice A :

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix} \tag{3.17}$$

où les $\mathbf{A_{ii}}$ sont des matrices carrées. La matrice **A** est alors décomposée en la somme des deux matrices $\mathbf{A_b}$ et $\mathbf{A_r}$ représentées sur la *figure 3.2*. Le système (3.15) devient :

$$\underbrace{\begin{bmatrix} A_{11}^k & & & 0 \\ & \ddots & & \\ & & A_{ii}^k & \\ & & & \ddots \\ 0 & & & A_{pp}^k \end{bmatrix}}_{A_b^k} \cdot \underbrace{\begin{Bmatrix} \Delta x_1^k \\ \vdots \\ \Delta x_i^k \\ \vdots \\ \Delta x_p^k \end{Bmatrix}}_{\cdot \Delta x^k} = \underbrace{\begin{Bmatrix} r_1^k \\ \vdots \\ r_i^k \\ \vdots \\ r_p^k \end{Bmatrix}}_{= r^k} \tag{3.18}$$

avec $\mathbf{A_b}^k=\mathbf{A_b}(\mathbf{x}^k)=(\mathbf{A_{ii}}(\mathbf{x}^k))_{i=1,..,\,p}$. Les $\Delta\mathbf{x_i}$ et $\mathbf{r_i}$ sont des vecteurs dont la dimension est celle de l'ordre de la sous_matrice carrée $\mathbf{A_{ii}}$ correspondant. Ceci revient à poser $\mathbf{P}=\mathbf{A_b}^k$.

Le système ainsi obtenu se décompose en autant de sous-systèmes indépendants deux à deux que la matrice $\mathbf{A_b}$ comprend de sous-matrices $\mathbf{A_{ii}}$ de **A**. Pour obtenir la solution au rang (k+1), il faut résoudre:

$$\mathbf{A_{ii}}(\mathbf{x}^k).\Delta\mathbf{x_i}^k=\mathbf{r_i}^k \qquad \text{pour } i=1,\ldots,p. \tag{3.19}$$
$$\mathbf{x}^{k+1}=\mathbf{x}^k+\Delta\mathbf{x}^k$$

Grâce à cette indépendance, la résolution du problème peut être aisément effectuée en parallèle sur autant de processeurs qu'il y en a de disponibles, à concurrence de p d'entre eux, et ceci d'une manière totalement similaire au cas précédent en remplaçant les a_{ii} par les A_{ii} (fig. 3.3).

La convergence des méthodes de type Jacobi

Notons B la matrice de l'itération et $\mathbf{e}^k$ l'erreur à l'itération (k). Nous avons:

$$\mathbf{e}^k=\mathbf{B}^k\mathbf{e}^0. \tag{3.20}$$

Pour que la suite $(\mathbf{x}^k)$ converge vers la solution exacte **x** quelque soit $\mathbf{x}^0$, il faut et il suffit que la suite $(\mathbf{e}^k)$ converge vers zéro quelque soit $\mathbf{e}^0$, i.e. il faut et il suffit que:

$$\rho(\mathbf{B}) < 1 \tag{3.21}$$

Dans le cas de la méthode de Jacobi par points $\mathbf{B}=\mathbf{I}-(\mathbf{A_d})^{-1}.\mathbf{A}$.

En ce qui concerne la méthode de Jacobi par blocs $\mathbf{B}=\mathbf{I}-(\mathbf{A_b})^{-1}.\mathbf{A}$.

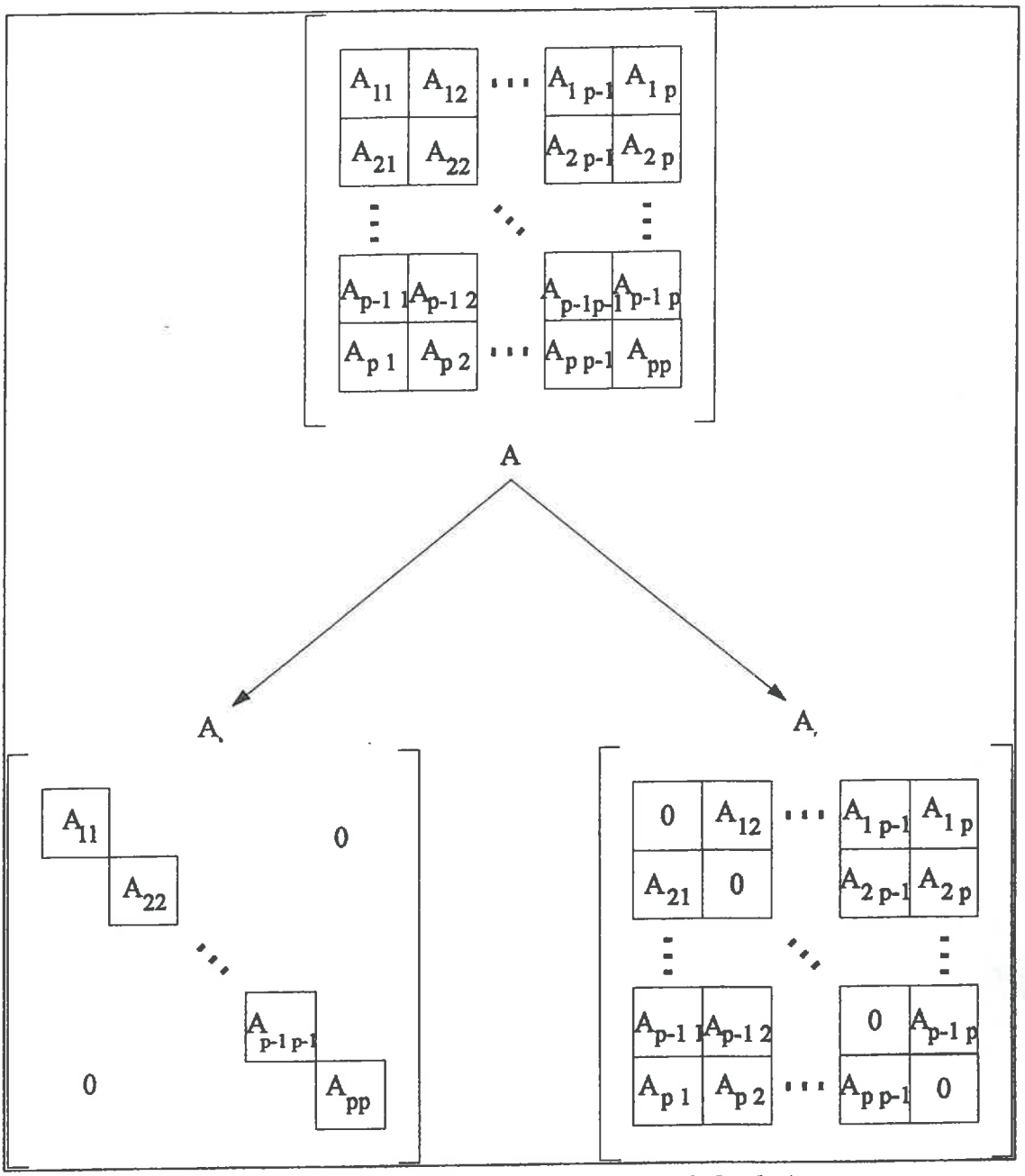


Figure 3.2 : extraction du préconditionneur de Jacobi A_b

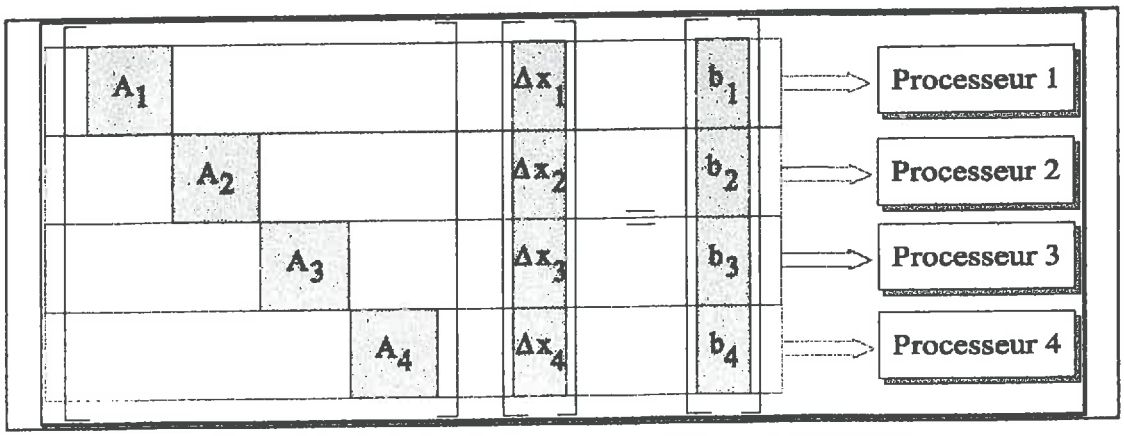


Figure 3.3: résolution du système en parallèle sur 4 processeurs

3.1.2. Découpage algébrique et découpage géométrique

Souvenons nous que, dans le cas qui nous préoccupe, la matrice **A** provient de l'écriture aux éléments finis des équations de Navier-Stokes. Elle est donc liée à un certain domaine de calcul (Ω) du plan physique.

Nous allons mettre en évidence que le découpage matriciel n'est pas sans connexion avec la décomposition de domaine. En particulier, le découpage matriciel de type Jacobi par blocs peut, dans certains cas, correspondre à un découpage géométrique de (Ω) - c'est à dire à la décomposition du domaine de calcul en sous-domaines (Ω_i). Ceci se produit lorsque ***la localisation dans le plan physique de l'ensemble des inconnues d'un sous-système*** ***($A_{ii}(x^k).\Delta x_i^k = r_i^k$) coïncide avec un sous-domaine connexe de (Ω). Cette coïncidence ne signifie néanmoins pas une équivalence***, dans la mesure où les solutions intermédiaires x^k d'un algorithme issu d'un découpage de la matrice **A**, quelqu'il soit, ne constituent pas des solutions physiques du problème traité mais uniquement les solutions algébriques des systèmes (3.21) successifs.

Dans ce qui suit, nous nous plaçons dans le cas où notre découpage algébrique coïncide avec un découpage géométrique que nous supposons être, pour la clarté de la rédaction, la décomposition d'un domaine (Ω) de frontière ($\Gamma(\Omega)$) en deux sous-domaines (Ω_1) et (Ω_2), séparés par une frontière interne (γ) *(fig..3.4)*.

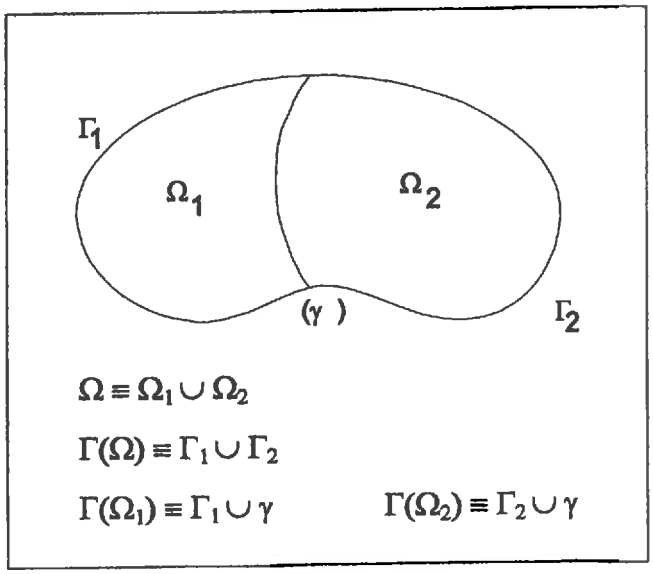


Figure 3.4 : Découpage géométrique du domaine

1. Écriture des systèmes matriciels

- Pour chacun de ces sous-domaines, et en imposant des conditions de Dirichlet sur la frontière (γ), nous avons:

$$\begin{bmatrix} A_{11} & A_{1\gamma} \\ A_{\gamma 1} & A_{\gamma\gamma}^{(1)} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_\gamma \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_\gamma^{(1)} \end{Bmatrix} \qquad \text{dans } (\Omega_1)$$

$$\begin{bmatrix} A_{\gamma\gamma}^{(2)} & A_{\gamma 2} \\ A_{2\gamma} & A_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_\gamma \end{Bmatrix} = \begin{Bmatrix} b_\gamma^{(2)} \\ b_2 \end{Bmatrix} \qquad \text{dans } (\Omega_2) \qquad (3.22)$$

Le problème classique monodomaine est constitué de l'assemblage des deux précédents:

$$\begin{bmatrix} A_{11} & A_{1\gamma} & 0 \\ A_{\gamma 1} & A_{\gamma\gamma}^{(1)} + A_{\gamma\gamma}^{(2)} & A_{\gamma 2} \\ 0 & A_{2\gamma} & A_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_\gamma \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_\gamma^{(1)} + b_\gamma^{(2)} \\ b_2 \end{Bmatrix}$$

soit

$$\begin{bmatrix} A_{11} & A_{1\gamma} & 0 \\ A_{\gamma 1} & A_{\gamma\gamma} & A_{\gamma 2} \\ 0 & A_{2\gamma} & A_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_\gamma \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_\gamma \\ b_2 \end{Bmatrix} \qquad \text{dans } (\Omega). \qquad (3.23)$$

Dans ces conditions, il est possible, par exemple, d'extraire de ce système le complément de Shur qui s'écrit ici:

$$S = A_{\gamma\gamma} - A_{\gamma 1} \cdot (A_{11})^{-1} \cdot A_{1\gamma} - A_{\gamma 2} \cdot (A_{22})^{-1} \cdot A_{2\gamma}.$$

Le système à résoudre devient alors:

$$\begin{bmatrix} A_{11} & A_{1\gamma} & 0 \\ 0 & 0 & S \\ 0 & A_{2\gamma} & A_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_\gamma \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_\gamma' \\ b_2 \end{Bmatrix} \qquad (3.24)$$

avec $b_\gamma' = b_\gamma - A_{\gamma 1} \cdot (A_{11})^{-1} \cdot b_1 - A_{\gamma 2} \cdot (A_{22})^{-1} \cdot b_2$.

- Par ailleurs, nous appliquons au même problème monodomaine un certain découpage algébrique de la matrice globale A suivant le modèle de Jacobi par blocs avec p=2:

$$\begin{bmatrix} A_{11}^* & 0 \\ 0 & A_{22}^* \end{bmatrix} \cdot \begin{Bmatrix} x_1^* \\ x_2^* \end{Bmatrix} = \begin{Bmatrix} b_1^* - A_{12}^* \cdot x_2^* \\ b_2^* - A_{21}^* \cdot x_1^* \end{Bmatrix} \qquad (3.25)$$

Pour que le système (3.23) issu d'une décomposition de domaine coïncide avec le système (3.25) issu d'un découpage algébrique, il faut ***affecter les conditions d'interface (γ) à l'un des deux domaines***, par exemple à (Ω_2). En effet, si nous posons

$$\mathbf{A_{22}}'=\begin{bmatrix} A_{\gamma\gamma} & A_{\gamma 2} \\ A_{2\gamma} & A_{22} \end{bmatrix}, \mathbf{A_{1\gamma}}'=\begin{bmatrix} A_{1\gamma} & 0 \end{bmatrix}, \mathbf{A_{\gamma 1}}'=\begin{bmatrix} A_{\gamma 1} \\ 0 \end{bmatrix}$$

$$\mathbf{x_2}'=\begin{Bmatrix} x_\gamma \\ x_2 \end{Bmatrix} \text{ et } \mathbf{b_2}'=\begin{Bmatrix} b_\gamma \\ b_2 \end{Bmatrix}, \tag{3.26}$$

le système (3.23) devient :

$$\begin{bmatrix} A_{11} & 0 \\ 0 & A'_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x'_2 \end{Bmatrix} = \begin{Bmatrix} b_1 - A'_{1\gamma} \cdot x'_2 \\ b'_2 - A'_{\gamma 1} \cdot x_1 \end{Bmatrix} \tag{3.27}$$

et nous pouvons aisément identifier (3.27) à (3.25).

Nous avons: $A^*_{11} \equiv A_{11}$, $A^*_{22} \equiv A'_{22}$,

$$x^*_1 \equiv x_1,\ x^*_2 \equiv x'_2$$

$$b^*_1 \equiv b_1 - A'_{1\gamma} \cdot x'_2,\ b^*_2 \equiv b'_2 - A'_{\gamma 1} \cdot x_1$$

2. *Couplage*

A présent, examinons le couplage entre les deux sous-systèmes ainsi constitués (*fig. 3.5*). Nous constatons que le premier sous-système (domaine (Ω_1)) est couplé au second (domaine (Ω_2)) à travers le terme

$\mathbf{A_{1\gamma}}'.\mathbf{x_2}'=\mathbf{A_{1\gamma}}.\mathbf{x_\gamma}$,

c'est à dire à travers les degrés de liberté du domaine (Ω_2) situés sur la frontière (γ).

En ce qui concerne le second sous-système (domaine (Ω_2)), le couplage réside dans le terme

$$\mathbf{A_{\gamma 1}}'.\mathbf{x_1}=\begin{bmatrix} A_{\gamma 1} \cdot x_1 \\ 0 \end{bmatrix}.$$

Or, la sous-matrice $A_{\gamma 1}$ traduit l'interaction entre la frontière (γ) et le domaine (Ω_1). Les matrices élémentaires qui la constituent sont donc nulles, excepté si l'élément considéré possède au moins un noeud sur (γ) et au moins un noeud dans (Ω_1).

a) Éléments linéaires

Dans le cas d'éléments triangulaires à trois nœuds, les matrices élémentaires non nulles correspondent aux éléments hachurés en noir sur la *figure 3.6 -a*. Nous voyons sur cette même figure que ce couplage ne fait intervenir en fait, qu'un sous-vecteur de x_1 dont les composantes sont les degrés de liberté des noeuds de (Ω_1) situés sur une pseudo-frontière ou frontière artificielle (γ').

En définitive, lorsque la matrice **A** subit un découpage algébrique du type Jacobi par blocs où la matrice A_b est constituée de deux sous-matrices A_{11} et A_{22} telles que les inconnues du premier sous-système ainsi obtenu se localisent dans le domaine (Ω_1) et les inconnues du second sous-système dans le domaine (Ω_2) affecté de la frontière interne (γ), alors le premier sous-système est couplé au second à travers (γ) tandis que le second est couplé au premier à travers (γ'). Dans ce cas, le système $\mathbf{A}(\mathbf{x}).\mathbf{x}=\mathbf{b}$ peut s'écrire:

$$\left[\begin{array}{cc|cc} A_{11} & A_{1\gamma'} & 0 & 0 \\ A_{\gamma'1} & A_{\gamma'\gamma'} & A_{\gamma'\gamma} & 0 \\ \hline 0 & A_{\gamma\gamma'} & A_{\gamma\gamma} & A_{\gamma 2} \\ 0 & 0 & A_{2\gamma} & A_{22} \end{array}\right] \cdot \left\{\begin{array}{c} x_1 \\ x_{\gamma'} \\ x_{\gamma} \\ x_2 \end{array}\right\} = \left\{\begin{array}{c} b_1 \\ b_{\gamma'} \\ b_{\gamma} \\ b_2 \end{array}\right\}$$

Par découpage algébrique nous obtenons :

$$\begin{bmatrix} A_{11} & A_{1\gamma'} \\ A_{\gamma'1} & A_{\gamma'\gamma'} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_{\gamma'} \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_{\gamma'} - A_{\gamma\gamma'} \cdot x_{\gamma} \end{Bmatrix} \qquad \text{(I)}$$

$$\begin{bmatrix} A_{\gamma\gamma} & A_{\gamma 2} \\ A_{2\gamma} & A_{22} \end{bmatrix} \cdot \begin{Bmatrix} x_{\gamma} \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_{\gamma} - A_{\gamma\gamma'} \cdot x_{\gamma'} \\ b_2 \end{Bmatrix} \qquad \text{(II)}$$

Le système (I) coïncide avec le système que nous aurions écrit si nous avions considéré an tant que domaine de calcul le domaine (Ω_1) de frontière $(\Gamma_1)\cup(\gamma)$. De même, le système (II) coïncide avec celui que nous aurions obtenu en considérant le domaine (Ω_2) de frontière $(\Gamma_2)\cup(\gamma')$ (*fig.* 3.6).

Ceci n'est autre que ***la méthode de Schwarz additive avec un recouvrement de 1***. Il s'agit du plus simple des algorithmes de décomposition de domaines qui, comme nous venons de le voir, peut être considéré comme une généralisation de la méthode de Jacobi par blocs[33]. Nous y reviendrons ultérieurement.

b) Éléments quadratiques

Lorsque des éléments triangulaires à six nœuds sont utilisés, le couplage est légèrement différent comme l'illustre la *figure 3.6* . En effet, on voit sur cette représentation que le second sous-système (domaine Ω_2) demeure couplé au premier via la frontière artificielle (γ') (en bleu sur le schéma). Par contre, le premier sous-système (domaine Ω_1) est à présent couplé au second non plus au travers de la frontière (γ) mais au travers d'une bande que nous avons noté Ω_{12} (hachurée en rouge sur le schéma). Le domaine Ω_1 dispose donc de plus d'information sur le domaine Ω_2 que ce dernier n'en a sur lui. On ne peut plus parler d'équivalence avec la méthode de Schwarz additif.

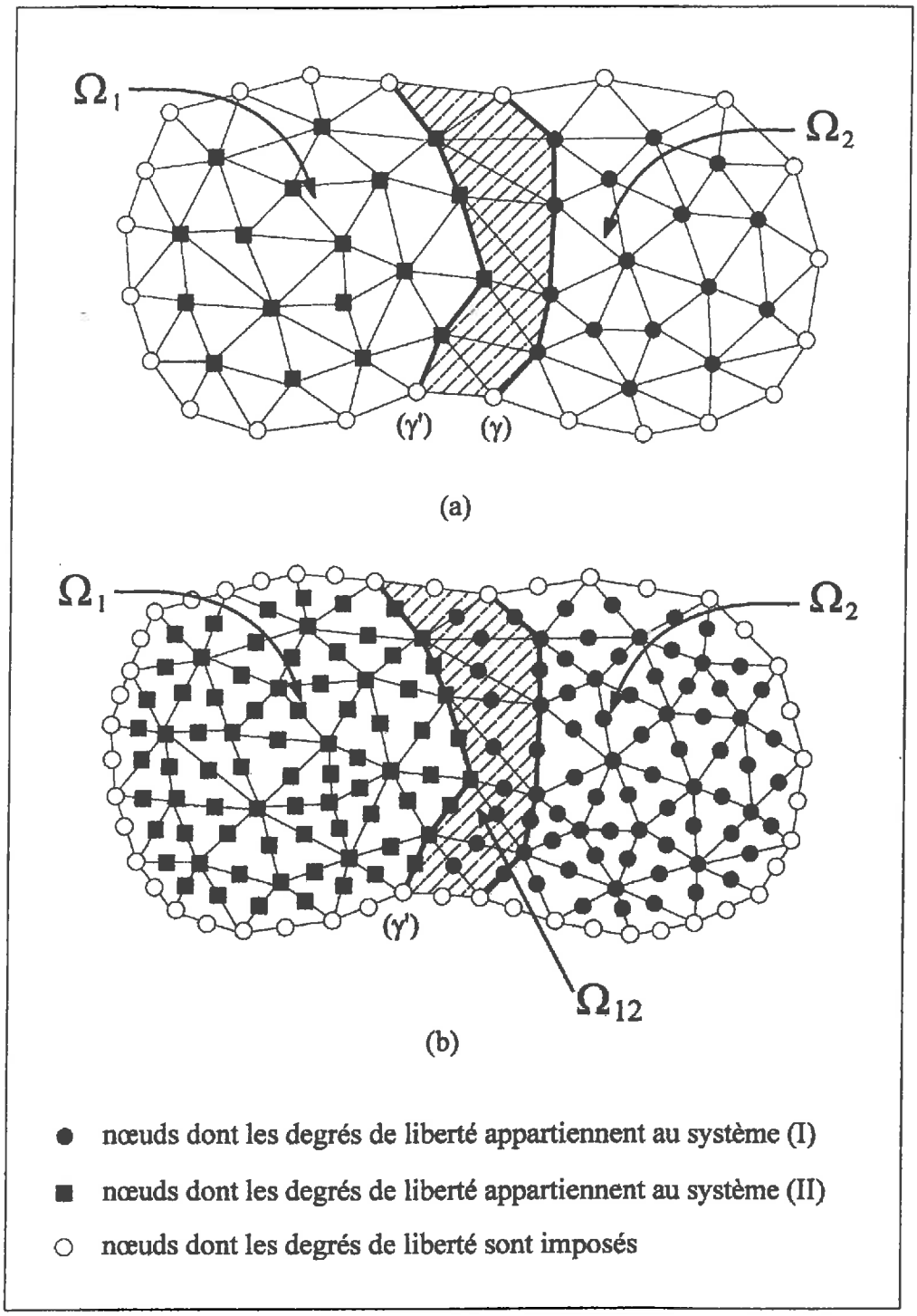


Figure 3.6 : Couplage entre les sous-systèmes générés par un découpage matriciel coïncidant avec un découpage géométrique du domaine initial

(a) dans le cas d'éléments triangulaires à trois nœuds

(b) dans le cas d'éléments triangulaires à six nœuds

Avant d'aller plus loin, nous désirons mettre en relief les raisons pour lesquelles nous nous attardons autant sur la méthode de Jacobi par blocs malgré le manque de robustesse et la lenteur de convergence de celle-ci. Ces raisons sont liées à son implémentation sur machines parallèles.

3.2. implémentation sur machines parallèles

Soit à résoudre un problème de mécanique des fluides dont la discrétisation par éléments finis s'écrit :

$$\mathbf{K}.\mathbf{u} = \mathbf{f} \tag{3.28}$$

Nous avons $\mathbf{x}\equiv\mathbf{u}$, $\mathbf{A}\equiv\mathbf{K}$ et $\mathbf{b}\equiv\mathbf{f}$. Nous introduisons les notations suivantes :

$$K = \begin{bmatrix} K_1 \\ \vdots \\ K_i \\ \vdots \\ K_p \end{bmatrix} = [K_i]_{i=1\ldots p} \quad \text{et} \quad K_b = \begin{bmatrix} K_{ii} & & 0 \\ & \ddots & \\ 0 & & K_{pp} \end{bmatrix} = [K_{ii}]_{i=1\ldots p}$$

Nous avons précédemment décrit comment la résolution des différents sous-systèmes de Jacobi pouvait être distribuée sur plusieurs processeurs, voyons à présent comment mettre cela en pratique. Dans ce but, nous devons déterminer le meilleur algorithme en prenant comme critères son niveau de parallélisme, sa facilité de programmation - nous rappelons que la structure type de nos programmes détaillée au chapitre précédent sert de support à tous nos développements - et l'architecture parallèle requise.

3.2.1. niveau de parallélisme

Nous cherchons à déterminer le degré de parallélisme de la méthode de Jacobi par blocs appliquée au problème (3.28). L'itération (k+1) s'écrit :

$$\mathbf{K_b}^k.\Delta\mathbf{u}^k = \mathbf{f} - \mathbf{K}^k.\mathbf{u}^k$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \Delta\mathbf{u}^k$$

Nous avons précédemment noté que les calculs pouvaient être exécutés en parallèle suivant la *figure 3.3* ce qui conduit à **l'algorithme 1** :

choisir $\mathbf{u}^0$

répéter pour k=0, 1, 2, ...

calculer sur le processeur (i), i=1,..p

- $\mathbf{r}_i(\mathbf{u}^k)=\mathbf{f}_i-\mathbf{K}_i\cdot\mathbf{u}^k$
- $\Delta\mathbf{u}_i^k=(\mathbf{K}_{ii}(\mathbf{x}^k))^{-1}.\mathbf{r}_i(\mathbf{u}^k)$
- $\mathbf{u}_i^{k+1}=\mathbf{u}_i^k+\Delta\mathbf{u}_i^k$

tant que $\dfrac{\left\|u^{k+1}-u^k\right\|_2}{\left\|u^k\right\|_2}<\varepsilon$

En détaillant les matrices et les vecteurs de l'itération comme suit :

$$\begin{bmatrix} K_{11}^k & 0 & \cdots & 0 \\ & \ddots & & \\ \cdots & 0 & K_{ii}^k & 0 \cdots \\ & & \ddots & \\ 0 & \cdots & 0 & K_{pp}^k \end{bmatrix} \cdot \begin{Bmatrix} \Delta u_1^k \\ \vdots \\ \Delta u_i^k \\ \vdots \\ \Delta u_p^k \end{Bmatrix} = \begin{Bmatrix} f_1 \\ \vdots \\ f_i \\ \vdots \\ f_p \end{Bmatrix} - \begin{bmatrix} K_1 \\ \vdots \\ K_i \\ \vdots \\ K_p \end{bmatrix} \cdot \begin{Bmatrix} u_1^k \\ \vdots \\ u_i^k \\ \vdots \\ u_p^k \end{Bmatrix}$$

$$\begin{Bmatrix} u_1^{k+1} \\ \vdots \\ u_i^{k+1} \\ \vdots \\ u_p^{k+1} \end{Bmatrix} = \begin{Bmatrix} u_1^k \\ \vdots \\ u_i^k \\ \vdots \\ u_p^k \end{Bmatrix} + \begin{Bmatrix} \Delta u_1^k \\ \vdots \\ \Delta u_i^k \\ \vdots \\ \Delta u_p^k \end{Bmatrix}$$

on constate que les données nécessaires au calcul du nouveau vecteur solution sont locales, hormis celles concernant le vecteur solution $\mathbf{u}^k$ de l'itération précédente qui doit être connu dans sa totalité par l'ensemble des processeurs. Ainsi, en se basant sur l'algorithme 1, la méthode de Jacobi par blocs se traduit à chaque itération par les quatre actions suivantes:

I - distribuer le vecteur solution de l'itération précédente sur les p processeurs,

II - fournir à chaque processeur les blocs matriciels et vectoriels qui le concernent,

III - calculer sur chaque processeur un des blocs constitutifs du nouveau vecteur solution,

IV - reconstruire le nouveau vecteur solution.

Les actions (I) et (IV) concernent exclusivement l'implémentation de l'algorithme sur la machine parallèle hôte. Nous discuterons cet aspect ultérieurement. Pour le moment, nous nous

concentrons sur les étapes (II) et (III) qui sont fortement liées à la méthode numérique des éléments finis ainsi qu'à la structure du programme informatique.

Plaçons nous dans le contexte de la méthode des éléments finis. Cette dernière peut être schématiquement décomposée en trois étapes :

A - Calcul des matrices et vecteurs élémentaires

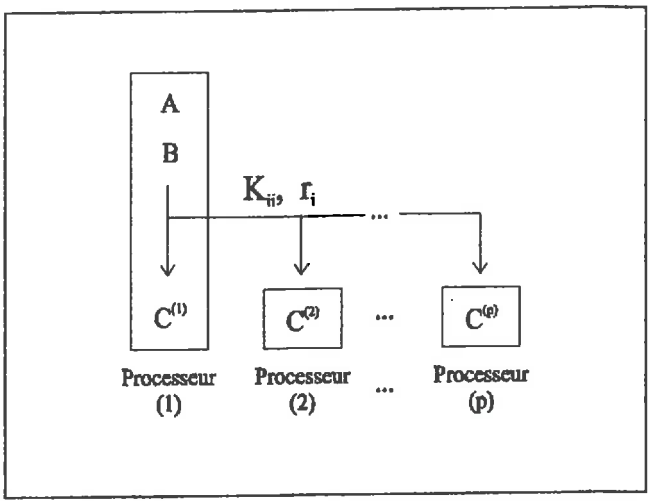
B - Assemblage

C - Résolution

L'action (II) peut alors se concevoir de trois manières différentes :

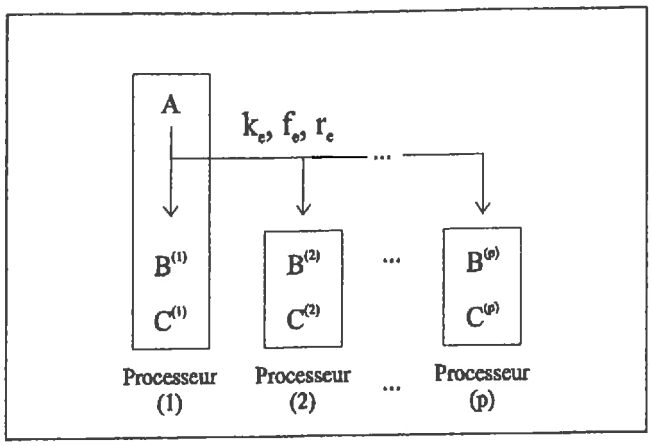
- option 1 -

1. Calcul des matrices de rigidité élémentaires k_e, des vecteurs des sollicitations élémentaires f_e et du résidu élémentaire r_e
2. Assemblage du résidu global **r** et du préconditionneur $\mathbf{K_b}$
3. **Distribution des blocs constitutifs du vecteur résidu et de la matrice sur les p processeurs**



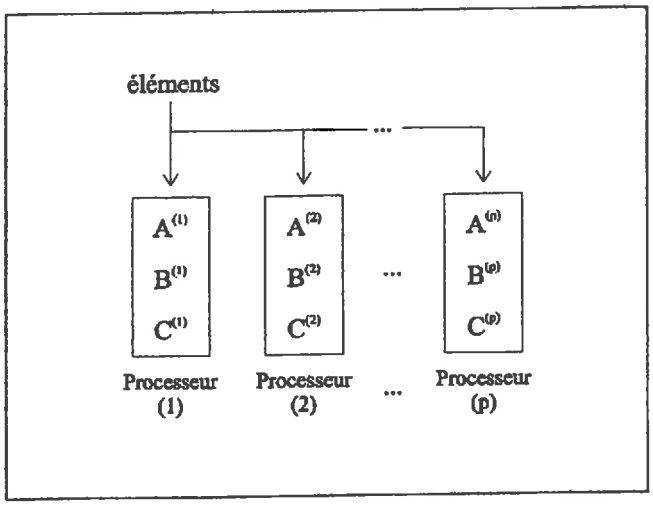
- option 2 -

1. Calcul des matrices de rigidité élémentaires k_e, des vecteurs des sollicitations élémentaires f_e et du résidu élémentaire r_e
2. **Distribution des matrices et des vecteurs élémentaires sur les p processeurs**
3. Assemblage sur chaque processeur des blocs r_i et K_{ii} concernés



- option 3 -

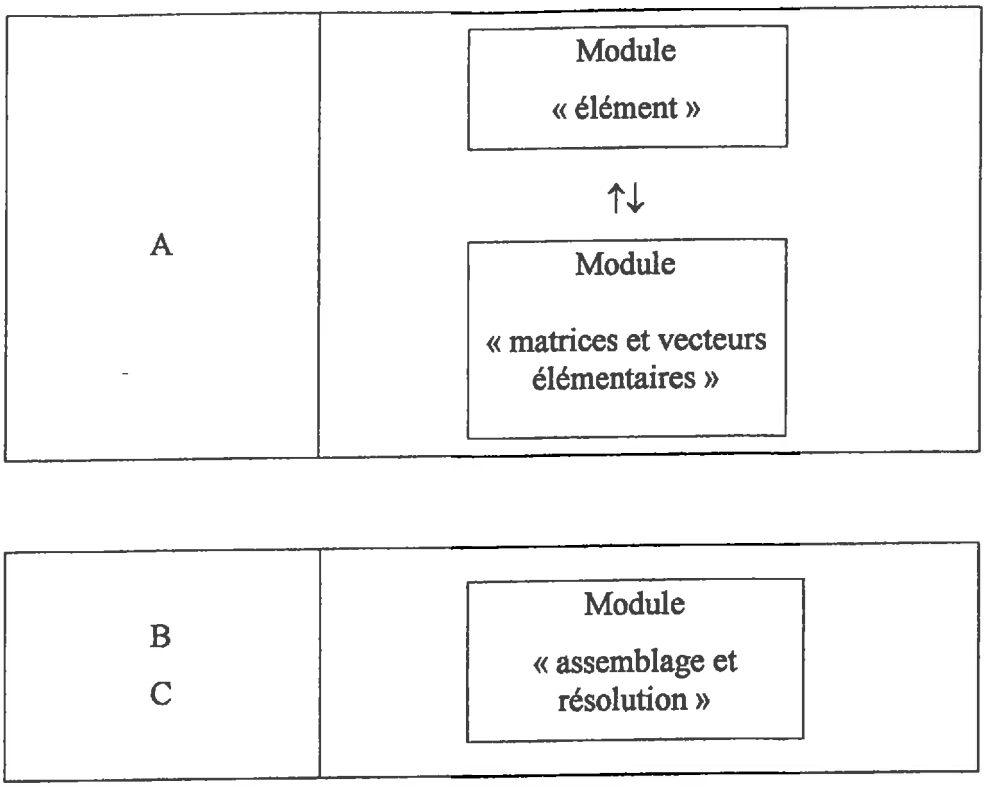
1. **Distribution des éléments sur les p processeurs**
2. Calcul sur chaque processeur des matrices de rigidité élémentaires k_e, des vecteurs des sollicitations élémentaires f_e et des résidus élémentaires r_e concernés
3. Assemblage sur chaque processeur des blocs r_i et K_{ii}



Du point de vue niveau de parallélisme, l'option 3 constitue le meilleur choix car elle présente à l'évidence le grain le plus fort donc le meilleur équilibrage des taches.

3.2.2. passage d'un programme séquentiel à un programme parallèle

Nous allons déterminer à présent, pour chaque option, les modifications à apporter au niveau du programme séquentiel. La structure de ce dernier se présente comme suit :



En ce qui concerne l'*option 1*, l'opération (C) est effectuée en paralléle. Or, au niveau du programme séquentiel, les opérations (B) et (C) sont regroupées dans le même module. La réécriture au moins partielle de celui-ci est donc nécessaire, c'est à dire le développement d'modules spécifiques au calcul parallèle. Ceci est typiquement ce que nous voudrions éviter.

Grâce à la répartition de l'opération d'assemblage sur l'ensemble des processeurs, l'*option 2* évite cet écueil en préservant le module « assemblage et résolution ». En effet, les paramètres dont ce dernier a besoin, sont exclusivement les matrices et les vecteurs élémentaires à assembler (chapitre I). À partir de ces données, le module construit automatiquement le système matriciel correspondant et le stocke à l'aide de deux tableaux : « K_globale » et « residu ». Il applique ensuite la méthode de résolution programmée. En clair, si on fournit à cet module la totalité des matrices élémentaires du problème alors, en sortie, le tableau « K_globale » contiendra la matrice de rigidité globale **K**. Par contre, si on fournit à cet module les matrices élémentaires correspondant aux blocs diagonaux de la matrice **K** alors, en sortie, le tableau « K_globale » contiendra la matrice de préconditionnement $\mathbf{K_b}$. De même, si on fournit à cet module les matrices élémentaires correspondant à un seul block diagonal de $\mathbf{K_b}$ alors, en sortie, le tableau « K_globale » contiendra une des sous matrices $\mathbf{K_{ii}}$. Il suffit donc que chaque processeur (i) active le module « assemblage et résolution » tel qu'il est écrit dans le programme

séquentiel, en limitant les paramètres de passage à ceux nécessaires à la construction de $\mathbf{K}_{ii}$ et de $\mathbf{r}_i$. Ainsi, le module du solveur séquentiel peut être réutilisé dans le cadre du solveur parallèle.

Les remarques faites dans le cadre de l'option précédente restent vraies pour l'*option 3*. La différence ici est qu'on désire effectuer également en parallèle le calcul des matrices et des vecteurs élémentaires. Si l'on se réfère à la schématisation de la structure du programme séquentiel présentée ci-dessus, on constate que cette opération implique les deux autres modules du solveur séquentiel. Or, comme le montre le tableau ci-après, l'activation des trois modules se fait en cascade suivant un processus ascendant qui part du module « assemblage et résolution » pour remonter jusqu'au module « élément ».

Module « élément »	* Reçoit le numéro de l'élément courant (*i_elem*) * Calcule les grandeurs caractéristiques utiles relatives à *i_elem* (fonctions d'interpolation, jacobien, élément de volume ...)

↑ ↓

Module « matrices et vecteurs élémentaires »	* Reçoit le numéro de l'élément courant (i_elem) * **Active le module « élément »** * Calcule les matrices et les vecteurs élémentaires pour i_elem

↑ ↓

Module « assemblage et résolution »	* Pour *i_elem*=1 à n → **Active le module « matrices et vecteurs élémentaires »** → Insère le(s) matrice(s) (récip. vecteur) élémentaire(s) dans les matrice(s) (récip. vecteur) globale(s) * Résout le système assemblé

C'est l'option qui respecte le plus la structure du programme séquentiel. Dans ce cas, les actions (II) et (III) correspondent à la ventilation des trois modules sur les p processeurs. Les paramètres à fournir au processeur (i) se réduisent alors à la connaissance des numéros des éléments concernés par le sous-système correspondant. L'équilibrage des taches entre processeurs n'est subordonné qu'à la constitution de sous-systèmes de taille équivalente.

3.2.3. Une application de type S.P.M.D.

Soit à résoudre un problème stationnaire de mécanique des fluides dont la discrétisation par éléments finis s'écrit:

$$\langle 0 \mid \quad \delta U_i \quad \rangle \cdot \left[\begin{array}{c|c} k_{11} & k_{12} \\ \hline k_{21} & K(U) \end{array}\right] \cdot \left\{\begin{array}{c} U_0 \\ \hline U_i \end{array}\right\} = \langle 0 \mid \quad \delta U_i \quad \rangle \left\{\begin{array}{c} f_0 \\ \hline f_i \end{array}\right\} \qquad (3.29)$$

La **résolution de ce problème en séquentiel** (monoprocesseur) traite directement le système ci-dessus, soit:

$$[K(U)].\{U_i\} = \{f_i\} - [k_{21}].\{U_0\}$$

Dans ce cas, nous avons
$\mathbf{x} \equiv \mathbf{U}$
$\mathbf{A} \equiv \mathbf{K}$
$\mathbf{b} \equiv \{f_i\} - [k_{21}].\{U_0\}$

Nous appliquons la méthode de Jacobi par blocs. Pour simplifier l'écriture nous choisissons **p=2**, mais ceci n'est nullement restrictif. Posons:

$$\{U_i\} = \left\{\begin{array}{c} U_1 \\ \hline U_2 \end{array}\right\} \text{ et } \{f_i\} = \left\{\begin{array}{c} f_1 \\ \hline f_2 \end{array}\right\}$$

(29) devient:

$$\langle 0 | \delta U_1 \quad \delta U_2 \rangle \cdot \left[\begin{array}{c|cc} k_{11} & k_{12}^* & k_{13}^* \\ \hline k_{21}^* & K_{11} & K_{12} \\ k_{31}^* & K_{21} & K_{22} \end{array}\right] \cdot \left\{\begin{array}{c} U_0 \\ \hline U_1 \\ U_2 \end{array}\right\} = \langle 0 | \delta U_1 \quad \delta U_2 \rangle \cdot \left\{\begin{array}{c} f_0 \\ \hline f_1 \\ f_2 \end{array}\right\}$$

soit

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \cdot \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} - \begin{bmatrix} k_{21}^* \\ k_{31}^* \end{bmatrix} \cdot \{U_0\} = \begin{Bmatrix} f_1' \\ f_2' \end{Bmatrix}$$

$$\boxed{\begin{array}{l} \left\{\begin{bmatrix} K_{11} & 0 \\ 0 & K_{22} \end{bmatrix}^k \cdot \begin{Bmatrix} \Delta U_1 \\ \Delta U_2 \end{Bmatrix} = \begin{Bmatrix} f_1' \\ f_2' \end{Bmatrix} - \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}^k \cdot \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix}^k \right. \\ \left. \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix}^{k+1} = \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix}^k + \begin{Bmatrix} \Delta U_1 \\ \Delta U_2 \end{Bmatrix} \right. \end{array}} \qquad (3.30)$$

Simulons à présent la **résolution en parallèle du même problème.** Pour la clarté de l'exposé, nous nous replaçons dans le cas où le découpage de la matrice **A** coïncide avec une décomposition du domaine de calcul (Ω) (voir II.1.3.2) et nous considérons deux processeurs

(P1) et (P2). Nous traitons sur chacun d'eux le même problème (mêmes équations discrétisées) mais dans un domaine de calcul ($\overline{\Omega}$) différent, de frontière $\overline{\Gamma}(\overline{\Omega})$ *(fig. 3.1)*.

Pour (P1), $\overline{\Omega} \equiv \Omega_1, \overline{\Gamma}(\overline{\Omega}) \equiv \Gamma_1 \cup \gamma$.

Pour (P2), $\overline{\Omega} \equiv \Omega_2, \overline{\Gamma}(\overline{\Omega}) \equiv \Gamma_2 \cup \gamma'$.

Pour (P1), les inconnues du problème aux éléments finis se limitent alors aux composantes du vecteur $\mathbf{U}_1$ tel que nous l'avons défini précédemment. Tout se passe pour lui, comme si il traitait un problème dont l'écriture éléments finis conduirait à la résolution du système matriciel ci-après.

$$\langle\, 0 \quad 0 \mid \delta U_1 \rangle \cdot \left[\begin{array}{cc|c} k_{11} & k_{13}^* & k_{12}^* \\ k_{31}^* & K_{22} & K_{21} \\ \hline k_{21}^* & K_{12} & K_{11} \end{array}\right] \cdot \left\{\begin{array}{c} U_0 \\ U_2^0 \\ \hline U_1 \end{array}\right\} = \langle\, 0 \quad 0 \mid \delta U_1 \rangle \cdot \left\{\begin{array}{c} f_0 \\ f_2 \\ \hline f_1 \end{array}\right\}$$

soit $$[K_{11}]\cdot\{U_1\} = \{f_1\} - [k_{21}^*]\cdot\{U_0\} - [K_{12}]\cdot\{U_2^0\} = \{f_1'\} - [K_{12}]\cdot\{U_2^0\}$$

De manière totalement similaire, le processeur (P2) résout le système:

$$\langle\, 0 \quad 0 \mid \delta U_2 \rangle \cdot \left[\begin{array}{cc|c} k_{11} & k_{12}^* & k_{13}^* \\ k_{21}^* & K_{11} & K_{12} \\ \hline k_{31}^* & K_{21} & K_{22} \end{array}\right] \cdot \left\{\begin{array}{c} U_0 \\ U_1^0 \\ \hline U_2 \end{array}\right\} = \langle\, 0 \quad 0 \mid \delta U_2 \rangle \cdot \left\{\begin{array}{c} f_0 \\ f_1 \\ \hline f_2 \end{array}\right\}$$

$$[K_{22}]\cdot\{U_2\} = \{f_2\} - [k_{31}^*]\cdot\{U_0\} - [K_{21}]\cdot\{U_1^0\} = \{f_2'\} - [K_{21}]\cdot\{U_1^0\}$$

La résolution sous forme incrémentale de ces systèmes nous mène à:

$$\left\{\begin{array}{l} \left[\begin{matrix} K_{11} & 0 \\ 0 & K_{22} \end{matrix}\right]^i \cdot \left\{\begin{matrix} \Delta U_1 \\ \Delta U_2 \end{matrix}\right\} = \left\{\begin{matrix} f_1' \\ f_2' \end{matrix}\right\} - \left[\begin{matrix} K_{11} & 0 \\ 0 & K_{22} \end{matrix}\right]^i \cdot \left\{\begin{matrix} U_1 \\ U_2 \end{matrix}\right\}^i - \left[\begin{matrix} 0 & K_{12} \\ K_{21} & 0 \end{matrix}\right]^i \cdot \left\{\begin{matrix} U_1^0 \\ U_2^0 \end{matrix}\right\} \\ \left\{\begin{matrix} U_1 \\ U_2 \end{matrix}\right\}^{i+1} = \left\{\begin{matrix} U_1 \\ U_2 \end{matrix}\right\}^i + \left\{\begin{matrix} \Delta U_1 \\ \Delta U_2 \end{matrix}\right\} \end{array}\right. \qquad (3.31)$$

Nous constatons que les systèmes (30) et (31) sont identiques si et seulement si, à chaque pas (i), les vecteurs $\mathbf{U}_1^0$ et $\mathbf{U}_2^0$ sont réactualisés par les vecteurs $\mathbf{U}_1^{i+1}$ et $\mathbf{U}_2^{i+1}$ respectivement. Autrement dit, à la fin de chaque étape (i) du processus itératif (31), chaque processeur doit transmettre à son homologue son vecteur solution et recevoir de celui-ci un nouveau vecteur dont les composantes remplaceront une partie des données du problème qu'il traite. Il faut donc prévoir des échanges de données entre processeurs.

Cette condition remplie, nous pouvons résoudre le problème (30) en lançant simultanément le même solveur aux éléments finis (S) sur deux processeurs distincts (P1) et (P2) c'est à dire

développer une application de type S.P.M.D. Ce faisant, la parallélisation ne concerne plus uniquement l'étape de résolution: le calcul des matrices et des vecteurs élémentaires est également réparti sur les différents processeurs, accélérant sur chacun d'eux l'opération d'assemblage.

3.2.4. Mise en oeuvre

Le gros intérêt d'un découpage algébrique de la matrice par rapport à une décomposition géométrique du domaine de calcul, est l'absence d'interfaces à gérer. Ceci procure une grande souplesse et facilité de mise en œuvre et une grande liberté au niveau du découpage.
Choisir la matrice A_b c'est, en pratique, choisir le nombre p de processeurs alloué au calcul ainsi qu'une certaine numérotation des inconnues du système. Pour une numérotation donnée, faire varier p équivaut à modifier le nombre de sous-matrices A_{ii}. Par contre, changer la numérotation des inconnues, le nombre p de processeurs étant fixé, revient à permuter des lignes ou des colonnes dans la matrice A avant d'en extraire la matrice A_b. En jouant sur ces deux paramètres, nous pouvons découper A de multiples façons.

D'un point de vue technique, l'extension à un nombre quelconque de sous-systèmes ne pose pas de problèmes particuliers, ce qui permet même d'envisager son automatisation en fonction du nombre de processeurs utilisés. Néanmoins, d'un point de vue numérique, une telle automatisation n'est guère envisageable car le découpage algébrique « en aveugle » qu'elle induit pourrait, dans certains cas, mener à la non convergence de l'algorithme I, sans que celui-ci ne soit en cause.

Nous avons vu en détails combien la matrice de préconditionnement de Jacobi par blocs présentait d'avantages d'un point de vue parallélisme :

- *la possibilité d'utiliser une méthode directe comme algorithme subsidiaire d'où un gain en temps de calcul,*
- *une diminution significative de la mémoire par processeur nécessaire au stockage de la matrice de l'itération, d'où une factorisation LU (si nécessaire) et un assemblage plus rapides,*
- *un équilibrage des taches entre processeurs très simple à effectuer,*

- *une mise en œuvre possible sous forme d'une application de type SPMD (forte granularité),*
- *un portage sur machine parallèle non polluant au niveau programmatoire.*

Malheureusement, la méthode de Richardson est une méthode qui bien souvent ne converge pas ou converge très lentement. En utilisant un préconditionneur, il est possible d'améliorer son taux de convergence. Néanmoins, il est fréquent que ce dernier demeure encore trop faible. Aussi, nous allons voir au paragraphe suivant des technique algébriques d'accélération des méthodes itératives préconditionnées standards.

3.3. Méthodes de sous espaces de krylov

Toute méthode produisant une suite $(x^{(k)})$ est une méthode de sous-espace de Krylov (ou méthode polynomiale) si le k-ième résidu $r^{(k)}= b-Ax^{(k)}$ peut être écrit sous la forme

$$r^{(k)}= q_k(\tilde{A})\, r^{(0)}$$

où q_k est un polynôme de degré k vérifiant $q_k(0)=1$ et $\tilde{A}$ est l'image de A par une certaine application linéaire.

3.3.1. Passage des méthodes itératives de base aux méthodes de sous-espace de Krylov

Reprenons l'itération de Richardson :

$$x^{(k+1)}=x^{(k)}+(b-Ax^{(k)})=x^{(k)}+r^{(k)}$$

En composant par $-A$ et en ajoutant b, on obtient :

$$b-Ax^{(k+1)}=b-Ax^{(k)}-Ar^{(k)}$$

c'est à dire $$r^{(k+1)}=(I-A)\, r^{(k)}=(I-A)^{k+1}r^{(0)}$$

d'où $$x^{(k+1)}= \sum_{j=0}^{k} r^{(j)} = \sum_{j=0}^{k} (I - A)^j r^{(0)}$$

On en déduit que la méthode de Richardson génère une suite de vecteurs $x^{(k+1)}$ tels que :

$$x^{(k+1)} \in \{ r^{(0)}, Ar^{(0)}, \ldots, A^k r^{(0)} \} = K^{k+1}(A; r^{(0)})$$

Ces vecteurs construisent donc des sous espaces de Krylov $K^{k+1}(A; r^{(0)})$ et la méthode de Richardson peut être caractérisée par le polynôme $P_{k+1}(A)$ tel que :

$$P_{k+1}(A) =(I-A)^{k+1}$$

et $$P_{k+1}(0)=1.$$

Dès lors, on peut déduire des méthodes beaucoup plus robustes en agissant sur la façon d'extraire un meilleur vecteur $x^{(k+1)}$ du sous-espace de Krylov correspondant (généré par la méthode itérative de base). Il ne nous semble pas pertinent de détailler ici ces méthodes, car nous n'apporterions rien de nouveau par rapport aux nombreuses publications et ouvrages disponibles sur le sujet[1, 16, 32, 41, 42]. Nous allons seulement situer brièvement les principales

d'entre elles par rapport à la manière utilisée pour obtenir $x^{(k+1)}$ et donner quelques références supplémentaires.

3.3.2. Les principales classes de méthodes

Le point de départ est en général de choisir le vecteur $x^{(k+1)}$ tel qu'une certaine norme de l'erreur $e^{(k+1)}=(x^{(k+1)}-x)$ ou du résidu $r^{(k+1)}=b-Ax^{(k+1)}$ soit minimale. A partir de là, on trouve dans la littérature deux grandes classes de méthodes : se dessinent, suivant les propriétés de la matrice A.

a) A est symétrique définie positive

$(x, y)_A \equiv (x, Ay)$ définit un produit interne et $\left\|x^{(k+1)} - x\right\|_A$ minimal pour $x^{(k+1)} \in K^{k+1}\left(A; r^{(0)}\right)$ implique :

$$x^{(k+1)}-x \perp_A K^{k+1}\left(A; r^{(0)}\right) \quad \Rightarrow \quad r^{(k+1)} \perp K^{k+1}\left(A; r^{(0)}\right)$$

Dès lors que la solution exacte n'a pas été obtenue à l'itération (k+1), $\{r^{(0)}, r^{(1)}, \ldots, r^{(k+1)}\}$ constitue donc une base orthogonale de $K^{k+1}\left(A; r^{(0)}\right)$. L'idée est alors de construire une base orthogonale de $K^{k+1}\left(A; r^{(0)}\right)$, puis de projeter $x^{(k+1)}$-x sur ce sous-espace et d'en déduire $x^{(k+1)}$. L'exploitation de cette idée conduit à des méthodes telles que la méthode de Lanczos pour des systèmes symétriques[25] et la méthode du gradient conjugué[17] (CG).

b) A symétrique non définie positive

Lorsque la matrice A n'est plus définie positive mais demeure symétrique, $(\ ,\ )_A$ ne définit plus un produit interne. C'est pourquoi des méthodes telles que MINRES[28] ou SYMMLQ[28] se basent non plus sur la minimisation de l'erreur mais sur celle du résidu.

c) A est non symétrique (matrice quelconque)

Il existe trois principales manières de résoudre un système linéaire non symétrique en conservant une certaine orthogonalité entre les résidus :

1) résoudre $A^TAx = A^Tb$ à l'aide des gradients conjugués : la méthode de l'équation normale[29].

2) rendre tous les résidus explicitement orthogonaux : méthodes FOM, GMRES[31], GMRES(m) ou des méthode mathématiquement équivalentes telles que Orthores (...) pour FOM et Orthomin, Orthodir (...) pour GMRES.
3) Essayer de construire un ensemble adéquat de vecteurs de base du sous-espace de Krylov à partir d'une relation de récurrence du type

$$\alpha_{i+1}\, r^{(i+1)} = Ar^{(i)} - \beta_i\, r^{(i)} - \gamma_i\, r^{(i-1)}.$$

Il s'agit des méthodes de gradients bi-conjugués telles que Bi-Lanczos[25], Bi-CG[12], QMR[14], CGS[34], Bi-CGSTAB[38], BiCGstab(l)[32], ...

3.3.3. Niveau de parallélisme

Nous allons déterminer le degré de parallélisme d'une méthode de type gradient conjugué préconditionnée par Jacobi par blocs. Nous prenons comme exemple la méthode CGS dont l'algorithme est donné ci-après *(fig. 3.2)*. Celui-ci est constitué des opérations suivantes :

(1) produits scalaires (×2)
(2) produits matrice par vecteur (×2)
(3) résolutions d'un système de type $A_b x=b$ (×2)
(4) divisions entre scalaires
(5) produits scalaire par vecteur
(6) sommes entre vecteurs

Exception faite de la (4), toutes les opérations peuvent être décomposées en sous-tâches exécutables en parallèle. En effet, considérons deux vecteurs x et y de composantes respectives $(x_i)_{i=1,\dots,n}$ et $(y_i)_{i=1,\dots,n}$, (p−1) scalaires $(\alpha_k)_{k=1,\dots,p-1}$ tels que $\alpha_k < n$, un scalaire λ et une matrice carrée A, nous avons :

- Produit scalaire :

$$(x, y) = \sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{\alpha_1} x_i y_i + \sum_{i=\alpha_1+1}^{\alpha_2} x_i y_i + \sum_{i=\alpha_2+1}^{\alpha_3} x_i y_i + \cdots + \sum_{i=\alpha_{p-1}+1}^{n} x_i y_i$$

soit
$$(x, y) = \underbrace{(x^{(1)}, y^{(1)})}_{\textit{processeur } 1} + \underbrace{(x^{(2)}, y^{(2)})}_{\textit{processeur } 2} + \underbrace{(x^{(3)}, y^{(3)})}_{\textit{processeur } 3} + \cdots + \underbrace{(x^{(p)}, y^{(p)})}_{\textit{processeur } p} \qquad (4.32)$$

x_0 estimation initiale

$r_0=A_b^{-1}(b-Ax_0)$; $\tilde{r}_0 = r_0$

$\rho_{-1}=1$; $p_{-1} = q_0 = 0$

i=0

Faire

$\rho_i = (\tilde{r}_0, r_i)$

- $\beta_i = \rho_i/\rho_{i-1}$

$u_i=r_i+\beta_{i-1}q_i$

$p_i=u_i+\beta_{i-1}(q_i+\beta_{i-1}p_{i-1})$

- $v_i = A_b^{-1}(Ap_i)$

$\sigma_i = (\tilde{r}_0, v_i)$

- $\alpha_i = \rho_i/\sigma_i$

$q_{i+1} = u_i - \alpha_i v_i$

$v_i = \alpha_i (u_i + q_{i+1})$

$x_{i+1} = x_i + v_i$

- $w_i = A_b^{-1}(Av_i)$

$r_{i+1} = r_i - w_i$

i=i+1

tant que le critère de convergence n'est pas satisfait

Figure 2 : algorithme CGS

avec $$x = \begin{Bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ \vdots \\ x^{(p)} \end{Bmatrix} \quad \text{et} \quad y = \begin{Bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(p)} \end{Bmatrix}$$

Les calculs relatifs aux produits scalaires peuvent donc être distribués entre p processeurs suivant (4.4.32).

- Produit matrice par vecteur

$$A \cdot x = \begin{bmatrix} A_1 \\ \vdots \\ A_i \\ \vdots \\ A_p \end{bmatrix} \cdot \{x\} = \begin{Bmatrix} A_1 \cdot x \\ \vdots \\ A_i \cdot x \\ \vdots \\ A_p \cdot x \end{Bmatrix} \begin{array}{l} \longrightarrow \textit{processeur } 1 \\ \\ \longrightarrow \textit{processeur } i \\ \\ \longrightarrow \textit{processeur } p \end{array} \qquad (3.33)$$

- Résolution d'un système de type $\mathbf{A_b x = b}$

$\mathbf{A_b}$ étant la matrice de Jacobi par blocs, chaque sous-système peut être résolu par un processeur différent conformément à la *figure 3.3.*

- Produit scalaire par vecteur

$$\lambda x = \begin{Bmatrix} \lambda x_1 \\ \lambda x_2 \\ \lambda x_3 \\ \vdots \\ \lambda x_p \end{Bmatrix} \begin{array}{l} \longrightarrow \textit{processeur } 1 \\ \longrightarrow \textit{processeur } 2 \\ \longrightarrow \textit{processeur } 3 \\ \vdots \\ \longrightarrow \textit{processeur } p \end{array} \qquad (3.34)$$

- Somme entre vecteurs

$$x + y = \begin{Bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \\ \vdots \\ x_p + y_p \end{Bmatrix} \begin{array}{l} \longrightarrow \textit{processeur } 1 \\ \longrightarrow \textit{processeur } 2 \\ \longrightarrow \textit{processeur } 3 \\ \vdots \\ \longrightarrow \textit{processeur } p \end{array} \qquad (3.35)$$

Nous détectons quatre points de synchronisation (démarqués par un point noir dans l'algorithme CGS), à savoir des étapes qui ne peuvent être démarrées qu'une fois accomplies les précédentes par l'ensemble des processeurs. Ceci signifie que certains processeurs devront éventuellement attendre que les autres aient fini, d'où l'importance de correctement équilibrer les taches entre eux. D'autre part, dès lors qu'une machine à mémoire distribuée est utilisée, ces 4 points de synchronisation se traduisent par 4 communications entre processeurs par itération CGS.

Nous pouvons à présent proposer une implémentation de l'algorithme CGS sur une architecture multiprocesseur à mémoire distribuée (voir page suivante). Pour la clarté du schéma, nous nous cantonnons à deux processeurs mais il est clair, au vu de ce qui précède, que cela n'a rien de restrictif.

Processeur 1	Communication	Processeur 2
$x_0^{(1)}$ estimation initiale	$\leftrightarrow$	$x_0^{(1)}$ estimation initiale
$r_0^{(1)} = A_{11}^{-1}\left(b^{(1)} - A_{11}x_0^{(1)} - A_{12}x_0^{(2)}\right)$		$r_0^{(2)} = A_{22}^{-1}\left(b^{(2)} - A_{21}x_0^{(1)} - A_{22}x_0^{(2)}\right)$
$\tilde{r}_0^{(1)} = r_0^{(1)}$		$\tilde{r}_0^{(1)} = r_0^{(1)}$
$p_{-1}^{(1)} = q_0^{(1)} = 0$		$p_{-1}^{(1)} = q_0^{(1)} = 0$
$\rho_{-1}=1$; $i=0$		$\rho_{-1}=1$; $i=0$
Faire		
$\rho_i^{(1)} = \left(\tilde{r}_0^{(1)}, r_i^{(1)}\right)$		$\rho_i^{(2)} = \left(\tilde{r}_0^{(2)}, r_i^{(2)}\right)$
	$\rho_i = \rho_i^{(1)} + \rho_i^{(2)}$	
$\beta_i = \rho_i / \rho_{i-1}$		$\beta_i = \rho_i / \rho_{i-1}$
$u_i^{(1)} = r_i^{(1)} + \beta_i q_i^{(1)}$		$u_i^{(2)} = r_i^{(2)} + \beta_i q_i^{(2)}$
$p_i^{(1)} = u_i^{(1)} + \beta_i\left(q_i^{(1)} + \beta_i p_{i-1}^{(1)}\right)$		$p_i^{(2)} = u_i^{(2)} + \beta_i\left(q_i^{(2)} + \beta_i p_{i-1}^{(2)}\right)$
	$p_i = \begin{Bmatrix} p_i^{(1)} \\ p_i^{(2)} \end{Bmatrix}$	
$v_i^{(1)} = A_{11}^{-1}\left(A_{11}p_i^{(1)} + A_{12}p_i^{(2)}\right)$		$v_i^{(2)} = A_{22}^{-1}\left(A_{21}p_i^{(1)} + A_{22}p_i^{(2)}\right)$
$\sigma_i^{(1)} = \left(\tilde{r}_0^{(1)}, v_i^{(1)}\right)$		$\sigma_i^{(2)} = \left(\tilde{r}_0^{(2)}, v_i^{(2)}\right)$
	$\sigma_i = \sigma_i^{(1)} + \sigma_i^{(2)}$	
$\alpha_i = \rho_i / \sigma_i$		$\alpha_i = \rho_i / \sigma_i$
$q_i^{(1)} = u_i^{(1)} - \alpha_i v_i^{(1)}$		$q_i^{(2)} = u_i^{(2)} - \alpha_i v_i^{(2)}$
$v_i^{(1)} = \alpha_i\left(u_i^{(1)} + \beta_i q_{i+1}^{(1)}\right)$		$v_i^{(2)} = \alpha_i\left(u_i^{(2)} + \beta_i q_{i+1}^{(2)}\right)$
$x_{i+1}^{(1)} = x_i^{(1)} + v_i^{(1)}$		$x_{i+1}^{(2)} = x_i^{(2)} + v_i^{(2)}$
	$v_i = \begin{Bmatrix} v_i^{(1)} \\ v_i^{(2)} \end{Bmatrix}$	
$w_i^{(1)} = A_{11}^{-1}\left(A_{11}v_i^{(1)} + A_{12}v_i^{(2)}\right)$		$w_i^{(2)} = A_{22}^{-1}\left(A_{21}v_i^{(1)} + A_{22}v_i^{(2)}\right)$
$r_{i+1}^{(1)} = r_i^{(1)} - w_i^{(1)}$		$r_{i+1}^{(2)} = r_i^{(2)} - w_i^{(2)}$
$i=i+1$		$i=i+1$
tant que le critère de convergence n'est pas satisfait		

Implémentation parallèle de l'algorithme CGS préconditionné par Jacobi par blocs

3.4. Conclusion

Nous avons établi, au fil de ce chapitre, les points suivants :

➢ Le préconditionnement à gauche du système matriciel issu d'une discrétisation par éléments finis, par une matrice de type Jacobi par blocs, équivaut à une décomposition algébrique de ce système qui permet la répartition de la résolution des sous-systèmes entre plusieurs processeurs.

➢ Lorsque la définition des blocs coïncide avec un morcellement du maillage en régions connexes, le couplage entre les différents sous-systèmes se matérialise au niveau des pseudo-frontières (internes) du domaine.

➢ En ce qui concerne la mise en œuvre, les modules servant à la construction du solveur peuvent être utilisés en l'état dans le cadre de la procédure parallèle. Ils seront appelés par chaque processeur. Le découpage algébrique de la matrice globale est réalisé par répartition des éléments entre les processeurs, ce qui permet un contrôle direct sur l'équilibrage des tâches.

➢ Du point de vue algébrique, chaque sous-système obtenu peut être considéré comme un problème aux éléments finis et être traité comme tel c'est à dire par l'intermédiaire d'un solveur séquentiel (application SPMD). Il n'est donc pas nécessaire de modifier ce dernier.

Nous disposons donc d'une procédure numérique de résolution supportant la distribution de données. Grâce à la correspondance entre le plan algébrique et le plan géométrique, cette dernière est effectuée à l'aide d'un morcellement du maillage basé sur des tests géométriques simples ne dépendant ni de la géométrie du problème, ni du type de maillage (structuré ou non structuré).

La gestion de l'architecture parallèle fait l'objet d'un module indépendant. Le passage d'un solveur séquentiel à un solveur parallèle est réalisé par l'adjonction de ce module et se concrétise par des passages de messages entre les processeurs. Ceux-ci interviennent

- En fin d'itération externe, dans le cas de la méthode itérative de base, pour actualiser le vecteur solution
- Plusieurs fois par itération lorsqu'un accélérateur algébrique est utilisé (quatre fois pour CGS) pour construire des vecteurs intermédiaires ou sommer des contributions à des produits scalaires.

CHAPITRE 4

Validation du solveur parallèle

CHAPITRE 4 : VALIDATION DU SOLVEUR PARALLÈLE

L'objet du présent chapitre est double. Il s'agit de valider le code parallèle développé dans le cadre de cette thèse à partir des éléments exposés précédemment, et d'évaluer ses capacités à traiter les problèmes aux dérivées partielles rencontrés en Mécanique des Fluides.

Nous avons choisi comme support de validation, le problème de la cavité entraînée tel que nous l'avons formulé dans le premier chapitre de ce mémoire. Nous pensons qu'il constitue un bon cas test car il est caractérisé à la fois par une résolution numérique délicate (de par la non linéarité des équations de Navier-Stokes et la présence de forts gradients locaux) , et à la fois par une géométrie très simple et des conditions aux limites bien définies. Pour la configuration du problème, sa modélisation mathématique et sa modélisation numérique, on se référera donc au paragraphe 2.2.2.2. Le maillage de la cavité est réalisé à l'aide d'éléments de Taylor-Hood* (t6 - t3) en effectuant une numérotation séquentielle des nœuds par balayage linéaire de la cavité.

Nous rappelons que le système matriciel à résoudre est de la forme :

$$\left[\begin{array}{c|c} \mathbf{k}_{11} & \mathbf{k}_{12} \\ \hline \mathbf{k}_{21} & \mathbf{K} \end{array}\right] \cdot \left\{\begin{array}{c} \underline{U_0} \\ \hline U_i \end{array}\right\} = \left\{\begin{array}{c} \underline{f_0} \\ \hline f_i \end{array}\right\} \tag{4.1}$$

soit

$$\mathbf{K(U)\,U = F(U)} \tag{4.2}$$

avec

$$\mathbf{U}=\left\{U_i\right\} \quad \text{et} \quad \mathbf{F}=\left\{f_i\right\}-\left[\begin{array}{c} \\ \mathbf{k}_{21} \\ \\ \end{array}\right]\cdot\left\{U_0\right\}, \tag{4.3}$$

Par soucis de clarté, une représentation des différents découpages matriciels effectués sera proposée. La taille des matrices associées au problème global rendant difficile leur visualisation, cette représentation sera basée sur un problème simplifié (grille 9×9 soit 32 éléments

* Voir paragraphe 2.1

triangulaires à 6 nœuds et un seul degré de liberté par nœud). Celle-ci n'a pour but que de présenter le principe de ces découpages et d'étayer certains de nos commentaires.

Dans un premier temps nous allons asseoir notre étude à l'aide de la méthode itérative de base (Richardson), puis nous évaluerons les performances de deux accélérateurs algébriques : les méthodes CGS et BiCGSTAB.

4.1. Préliminaires

4.1.1. Définition des Critères d'arrêt

Notre critère d'arrêt se base sur la norme n_r du résidu global :

$$C_1 : n_r = \|\mathbf{r}\| = \left\|\mathbf{F} - \mathbf{K}\left(\mathbf{U}^{(k)}\right)\mathbf{U}^{(k)}\right\| \tag{4.4}$$

Nous considérerons la solution numérique du système (4.2) obtenue lorsque cette norme vérifiera :

$$n_r < \alpha_r \tag{4.5}$$

Le traitement du couplage entre le champ de vitesse et le champ de pression est connu pour être une source de difficultés[40] dans le cas de fluides incompressibles. L'absence d'équation permettant de déterminer explicitement le champ de pression en est la cause. En effet, la pression intervenant uniquement au travers de son gradient dans les équations du mouvement, l'approximation de ce champ ne peut être effectuée qu'à une constante additive p_0 près. En pratique, la pression joue le rôle d'un multiplicateur de Lagrange : la constante p_0 s'ajuste, pendant le calcul, de sorte que le champ de vitesse vérifie la contrainte d'incompressibilité $\mathrm{div}\,\vec{U} = 0$.

De ce fait, nous prenons soin de définir deux autres critères de convergence que nous notons C_2 pour le champ de vitesse **u** et C_3 pour le champ de pression p:

$$C_2 : n_u = \frac{\left\|u^{(k+1)} - u^{(k)}\right\|}{\left\|u^{(k+1)}\right\|} < \alpha_u \tag{4.6}$$

$$C_3 : n_p = \left|\frac{p^{(k+1)} - p^{(k)}}{p^{(k+1)}}\right| < \alpha_p \tag{4.7}$$

4.1.2. Vérification de l'implémentation sur l'ipsc-860

Avant toute chose, nous effectuons un test simple visant à vérifier l'implémentation du programme parallèle sur l'iPSC-860. Pour cela, nous résolvons le système matriciel (4.2) à l'aide de la méthode itérative de base pour une valeur du nombre de Reynolds fixée à 10^2. Le maillage utilisé est régulier et constitué de 200 éléments de Taylor-Hood.

Dans un premier temps, ce calcul est effectué sur un seul processeur. La matrice $\mathbf{A_b}$ est obtenue à partir de la matrice de rigidité globale $\mathbf{K}(\mathbf{U}^{(k)})$ en remplissant directement de zéros les deux blocs non diagonaux. Une réactualisation de ses composantes est réalisée à chaque itération.

Le calcul est ensuite lancé sur deux processeurs de la manière décrite au *chapitre 2*. Le découpage est équivalent au calcul monoprocesseur, c'est à dire les N/2 premières inconnues traitées par le premier processeur et les autres par le second (si N est pair, l'équilibrage des tâches est parfait).

Le programme séquentiel ayant été testé au *chapitre 1*, l'obtention de la même suite d'itérés dans les deux cas nous a assuré d'un calcul en parallèle correct. Cette vérification a été également effectuée pour quatre blocs diagonaux (i.e. sur un seul processeur puis sur quatre).

Nous allons à présent tester différents préconditionneurs issus de découpages matriciels par blocs. Ces préconditionneurs sont ici appliqués à la méthode de Richardson. Notons que lorsqu'un calcul monoprocesseur est accompli, ceci équivaut à la méthode des approximations successives (AS). Dans le cas d'un calcul multiprocesseur, il s'agit de la méthode de Jacobi par blocs précédée d'une éventuelle renumérotation des inconnues par rapport au calcul précédent, laquelle dépendant du découpage (« *splitting* ») matriciel choisi. Cette dernière méthode est soit appliquée directement au système non linéaire, dans ce cas nous parlerons de la méthode de Jacobi non linéaire (JNL), soit précédée par une étape de linéarisation (Picard ou Newton-Raphson).

4.1.3. Choix du modèle numérique

Avant d'entreprendre notre étude numérique, nous avons deux points à préciser. Le premier a trait à la formulation intégrale du problème. En effet, cette formulation diffère suivant que l'on intègre ou non le terme issu des forces de pression :

$$\underbrace{-\int_{\Omega} \delta u_i \frac{\partial p}{\partial x_i}\, d\Omega}_{I} = \underbrace{\int_{\Omega} \frac{\partial \delta u_i}{\partial x_i}\, p\, d\Omega}_{II} - \underbrace{\int_{\Gamma} \delta u_i\, p\, n_i\, d\Gamma}_{III} \tag{4.8}$$

Dans notre cas, l'intégrale de contour *(III)* disparaît car la vitesse est imposée sur la frontière (Γ) du domaine de calcul (Ω). La discrétisation par éléments finis des deux autres termes aboutit à :

$$I \rightarrow -\sum_{n_e} \left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_u\} \cdot \left\langle \frac{\partial n_p}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ p^{(k)} \right\} \tag{4.9}$$

$$II \rightarrow -\sum_{n_e} \left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \left\{ \frac{\partial n_u}{\partial x_j} \right\} \cdot \langle n_p \rangle \cdot d\Omega_e \cdot \left\{ p^{(k)} \right\} \tag{4.10}$$

D'un point de vue résultat numérique, les deux formulations sont totalement équivalentes. Par contre, la *II* nécessiterait des calculs supplémentaires si les conditions aux limites du problème n'induisaient pas la nullité de l'intégrale de contour. Aussi, nous préférons conserver la formulation intégrale initiale *(I)* qui ne dépend pas des conditions aux limites.

Le second point que nous désirons détailler, concerne le champ de pression. Comme nous l'avons précédemment souligné, ce champ n'est déterminé qu'à une constante additive p_0 près. Certains auteurs imposent cette constante en fixant la valeur de la pression en un nœud du maillage, tandis que d'autres laissent le calcul la déterminer. Nous nous sommes donc interrogés sur la démarche à suivre. Nous avons décidé, de manière arbitraire, d'effectuer une première série de calculs en n'imposant aucune condition sur la pression. Le paragraphe suivant présente les résultats ainsi obtenus.

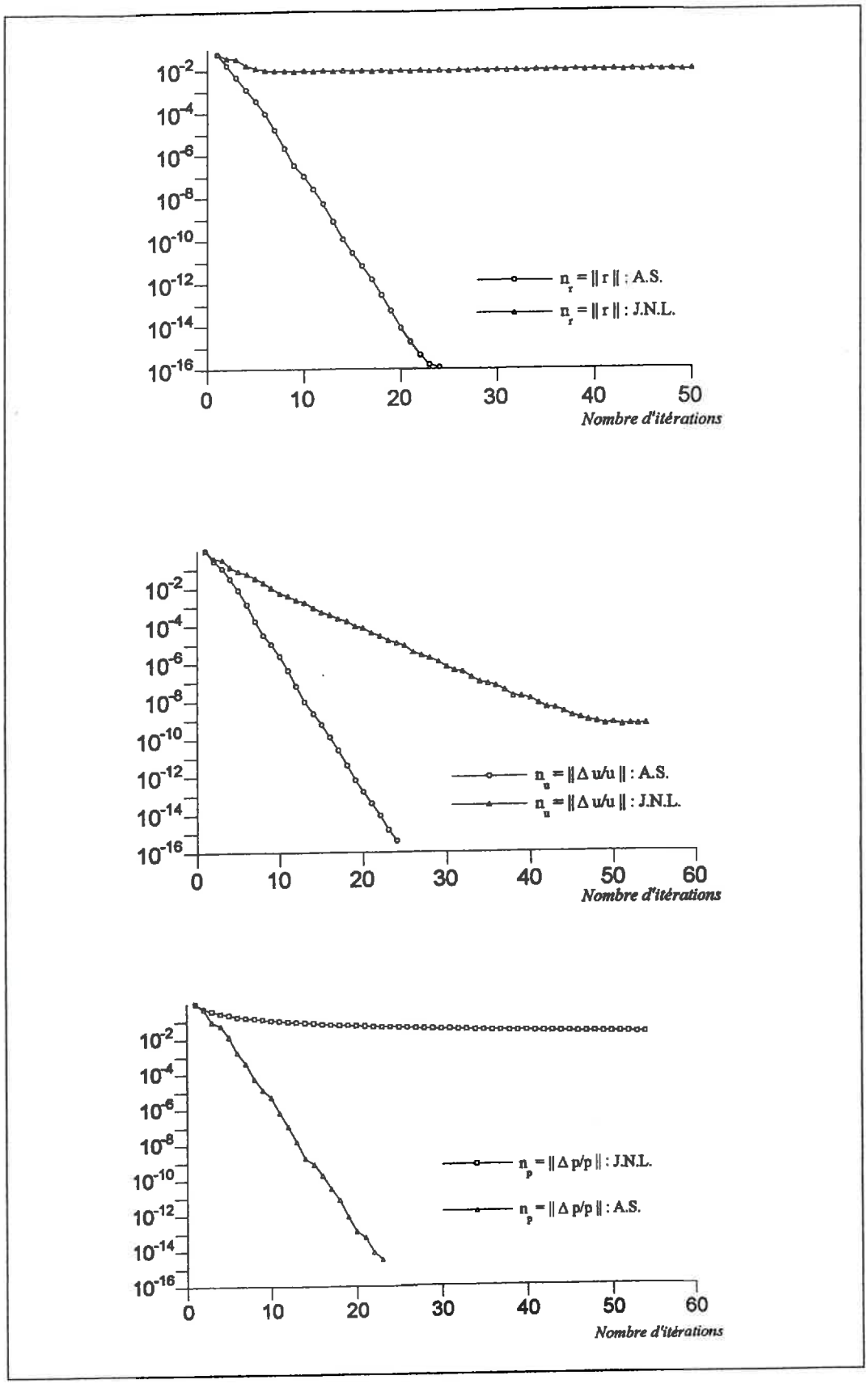


Figure 4.1 : Modèle sans référence de pression – Évolution des différents critères de convergence en fonction du nombre d'itérations de AS ou JNL (2 blocs)

4.2. Modèle sans référence de pression

4.2.1. Point de départ

La première matrice de préconditionnement $\mathbf{A_b}$ testée est issue du découpage de la matrice de rigidité **K** en deux blocs diagonaux. Nous appliquons la méthode de Richardson au système non linéaire (4.2) préconditionné par la matrice $\mathbf{A_b}$ que nous réactualisons à chaque itération. Nous rappelons que ceci équivaut à la méthode JNL. Les calculs sont effectués en parallèle sur deux processeurs i-860. La résolution du système sur un seul processeur (AS) nous sert de référence.

La mémoire disponible par processeur étant limitée à 16Mo, la matrice stockée sur un processeur ne peut contenir guère plus que 1,3 10^6 réels. De ce fait, pour pouvoir comparer les résultats obtenus sur plusieurs processeurs (JNL) à ceux obtenus sur un seul processeur (AS), nous sommes obligés de limiter notre maillage à une grille constituées de (41×41) nœuds.

Nous conservons pour l'instant la grille régulière constituée de 200 éléments de Taylor-Hood (soit 21×21 nœuds) et nous fixons le nombre de Reynolds à 10^2. La *figure 4.1* montre l'évolution des critères C_1, C_2 et C_3, précédemment définis, en fonction du nombre d'itérations. En ce qui concerne le calcul monoprocesseur (AS), on constate que la norme du résidu n_r (critère C_1) décroît rapidement (6 itérations pour $\alpha_r=10^{-4}$, 9 itérations pour $\alpha_r=10^{-6}$). Ce comportement se reflète dans celui de n_u (critère C_2) et de n_p (critère C_3). Par contre, dans le cas du calcul sur deux processeurs (JNL)., la norme n_r stagne aux alentours de 10^{-2}. En fait, si l'on poursuivait les calculs, on constaterait une décroissance extrêmement lente jusqu'à 10^{-4} puis une stabilisation à cette valeur.

L'introduction des critères de convergence C_2 et C_3, caractérisant respectivement le champ de vitesse et le champ de pression, révèle que la stagnation de C_1 reflète celle du critère C_3 tandis que le critère C_2 décroît jusqu'à une valeur de l'ordre de 10^{-9} (14 itérations pour $\alpha_u=10^{-4}$, 30 itérations pour $\alpha_u=10^{-6}$). Le champ de vitesse précédent (AS) est alors obtenu à deux chiffres après la virgule près (l'erreur relative en chaque point et en projection sur chaque axe est majorée par 10^{-2}). L'exploitation du champ de pression a permis de constater les faits suivants:

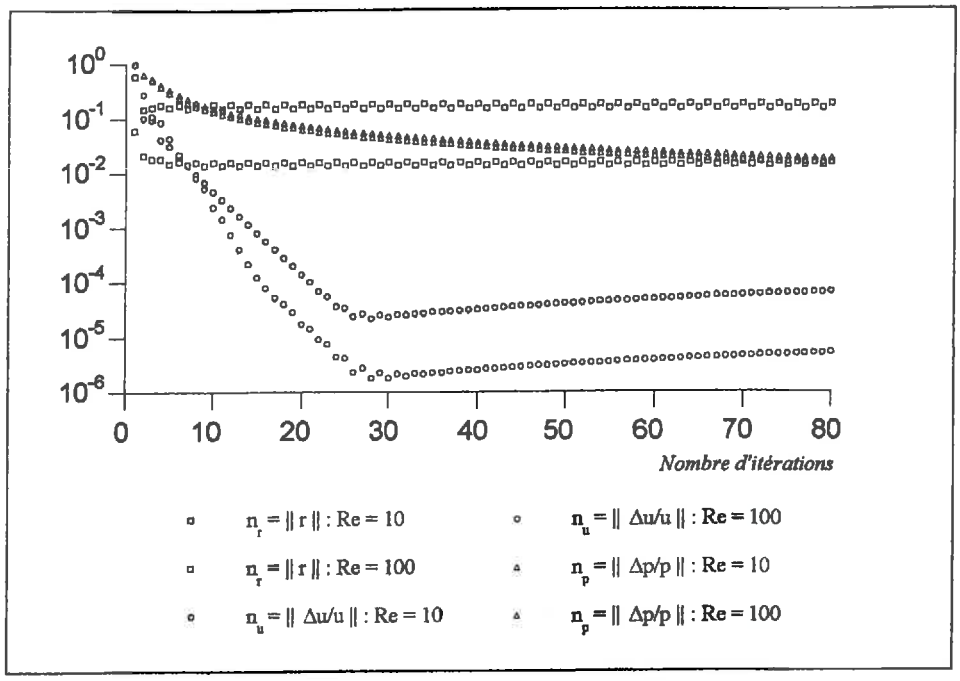


Figure 4.2 : Modèle sans référence de pression – Effet des non linéarités sur la convergence de JNL (2 blocs)

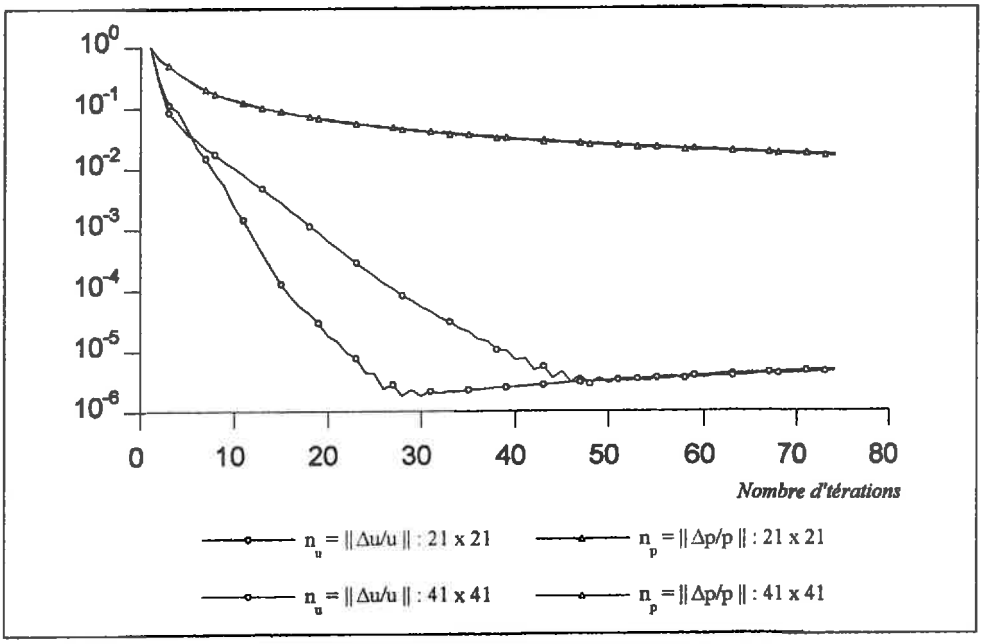


Figure 4.3 : Modèle sans référence de pression – Effet de la taille du problème sur la convergence de JNL (2 blocs)

- absence de zone de dépression,
- des valeurs trop élevées si l'on se réfère à Ramaswamy[30] ou Bruneau et Jouron[5],
- la relative uniformité du champ, y compris près des coins supérieurs, en particulier aucun pic n'est observé.

La poursuite des calculs ne permet pas de meilleure approximation. Nous allons tenter de comprendre ce phénomène. Pour cela, dressons la liste des paramètres susceptibles d'influencer la solution :

- Importance des termes non linéaires
- Effet du maillage
- Valeur du coefficient de pénalité ε
- Type de découpe matricielle (= définition des blocs)
- Linéarisation des équations

4.2.1.1. Effet des non linéarités

La présence de termes non linéaires constitue une des principales difficultés soulevées par la résolution des équations de Navier-Stokes en particulier lorsque des méthodes de base telle Jacobi par blocs sont employées. À partir d'un nombre de Reynolds de 10^2, l'effet de ces non linéarités n'est plus négligeable. C'est pourquoi nous avons effectué le même calcul que précédemment mais pour un nombre de Reynolds faible (fixé à 10). Sur la *figure 4.2*, on peut voir que le comportement de n_p est similaire. Ceci montre que la non linéarité des équations n'est pas la source du problème.

4.2.1.2. Effet du maillage

Un autre élément déterminant dans la stabilité d'un schéma numérique est le pas de discrétisation spatiale h. Dans les calculs précédents, ce pas était $h = 1/21$. Nous le divisons à présent par deux (grille 41×41) ce qui implique quatre fois plus de valeurs nodales. La *figure 4.3* montre que l'évolution de la norme n_p est quasiment inchangée.

4.2.1.3. Effet du coefficient de pénalité

Le coefficient et la matrice de pénalité ont été présentés au chapitre 2. Nous rappelons que le terme de pénalité n'est pas introduit dans le résidu mais uniquement dans la matrice associée au problème global, au niveau de la diagonale principale. De ce fait, il est possible que

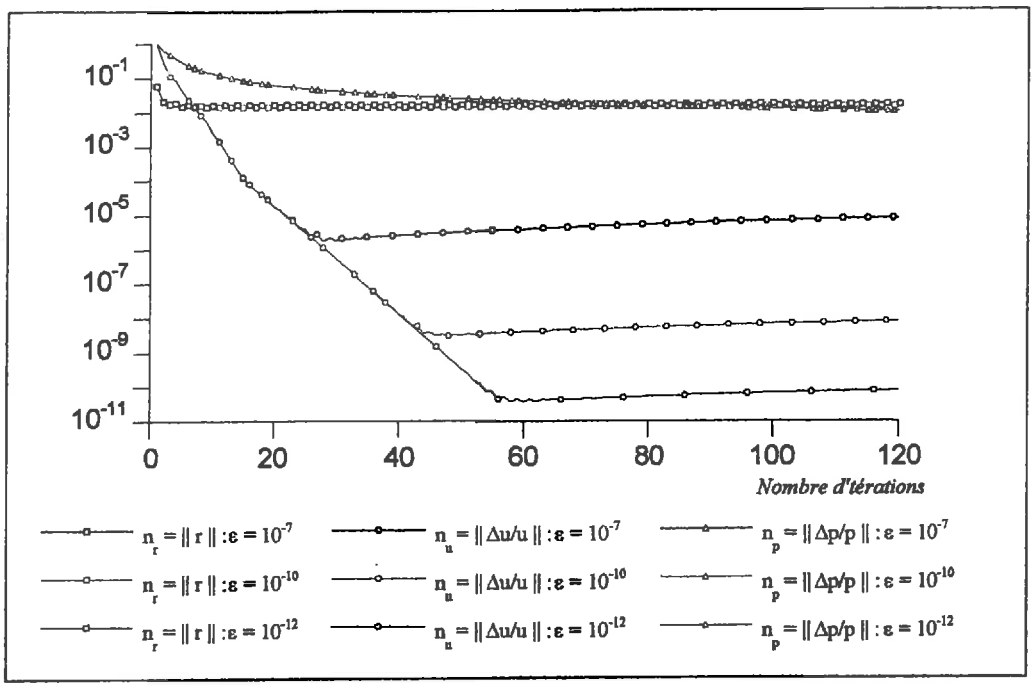


Figure 3.4 : Modèle sans référence de pression – Effet de la pénalité sur la convergence de JNL (2 blocs)

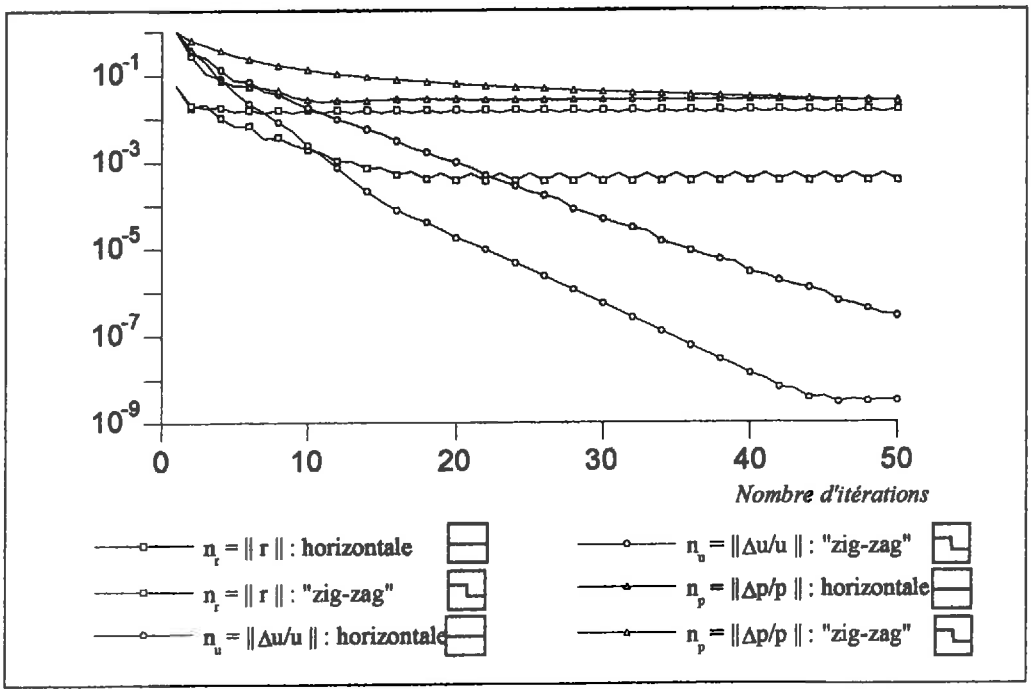


Figure 4.5 : Modèle sans référence de pression – Effet du découpage matriciel sur la convergence de JNL (2 blocs)

une variation de ce terme s'accompagne d'une modification du conditionnement du bloc pression – pression et, par conséquence, affecte le taux de convergence de la méthode de résolution itérative.

Nous avons fait varier le coefficient de pénalité de 10^{-2} à 10^{-12} . Certains résultats ont été reportés sur la *figure.3.4*. On remarque à nouveau que le comportement de la norme n_p demeure inchangé.

4.2.1.4. Effet de la linéarisation

Pour le test suivant, nous avons ajouté une étape de linéarisation basée sur des itérations de Picard†. Les systèmes linéaires successifs ainsi obtenus sont ensuite résolus par la méthode de Jacobi par blocs qui intervient donc sous la forme d'itérations internes. Ceci équivaut à résoudre le système non linéaire par JNL en réactualisant le préconditionneur toutes les m itérations (ici m=10). Cette tentative n'a pas été plus fructueuse que les précédentes. En effet, la *figure.4.6* montre que la norme du résidu n_r stagne aux environs de 5.10^{-3}.

4.2.1.5. Effet du découpage matriciel

La matrice initiale étant la matrice de rigidité K de dimension (N×N), nous avons jusqu'ici utilisé un découpage matriciel consistant à assigner les N/2 premières inconnues au premier processeur et les (N-N/2) autres au second (schéma *4.15.a*). Nous extrayons à présent la matrice de préconditionnement A_b suivant le *schémas 4.14.b*. Une visualisation de la localisation spatiale des inconnues traitées par chaque processeur est proposée respectivement par les *schémas 4.15.a* et *4.15.b*. On constate que cette localisation définit deux régions connexes de frontière horizontale. La *figure 4.5* montre qu'aucune amélioration notable n'est apportée.

4.2.2. Analyse du problème

Nous avons éliminé toutes les possibilités que nous avions envisagé pour améliorer les prédictions, notamment sur le champ de pression. Par conséquent, il nous faut réaliser une analyse plus fine de nos résultats.

Nous considérons les normes n_r, n_u et n_p calculées non plus pour le système global reconstitué mais pour chaque sous-système dans le but de déterminer si le comportement observé

† Voir chapitre 1.3

au niveau global est imputable à l'un des sous-systèmes ou aux deux. Un examen de la *figure 4.7* révèle des oscillations au niveau des résidus locaux (i.e. pour chaque processeur) qui émane de la pression.

Pour avoir une vue globale de l'évolution de la pression, nous construisons deux grandeurs vectorielles qui, nous le soulignons, n'ont aucune signification physique. Nous formons tout d'abord le vecteur constitué par les valeurs nodales de la pression calculées à chaque itération et nous calculons sa norme n_1. Comme nous l'avons précédemment précisé, la pression intervient dans les calculs sous la forme de son gradient dans le terme :

$$-\sum_{n_e}\left\langle \delta u_i^{(k)} \right\rangle \cdot \int_{\Omega_e} \{n_u\} \cdot \left\langle \frac{\partial n_p}{\partial x_j} \right\rangle \cdot d\Omega_e \cdot \left\{ p^{(k)} \right\}$$

Par intégration numérique, ce terme est évalué aux points de Gauss (voir *chapitre2*). C'est pourquoi nous construisons un second vecteur à partir de la valeur interpolée du gradient de pression aux points de Gauss, et nous calculons sa norme n_2.

La *figure 4.8* présente l'évolution de ces normes pour des calculs effectués sur un processeur (AS : *fig. 8.a*) puis sur deux (JNL. : *fig. 8.b*). On constate que dans le cas des A.S., les normes n_1 et n_2 se stabilisent très rapidement (2 ou 3 itérations) ce qui traduit la stabilisation du champ de pression.

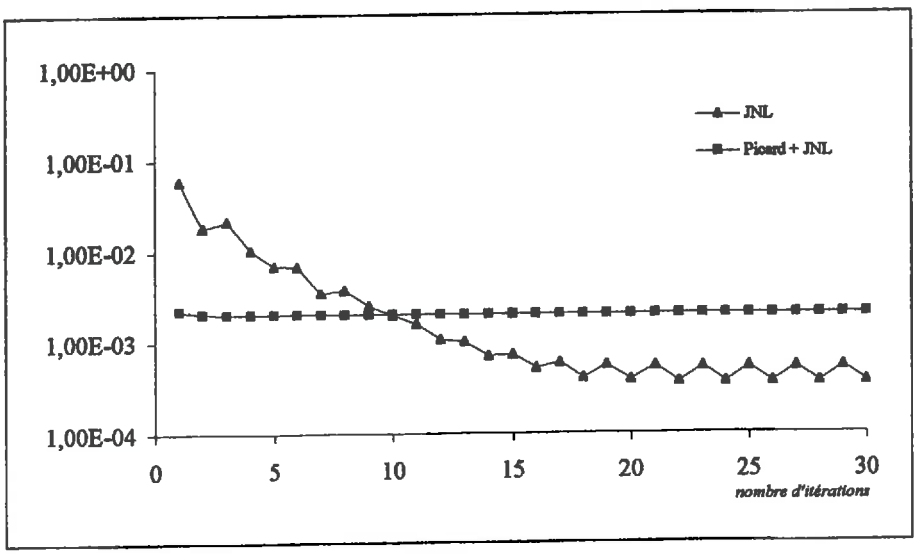
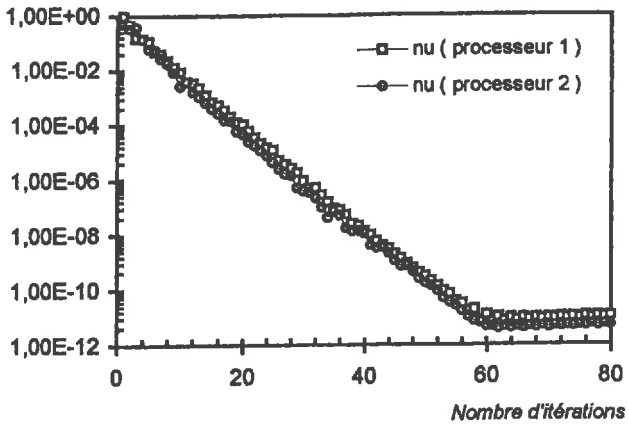
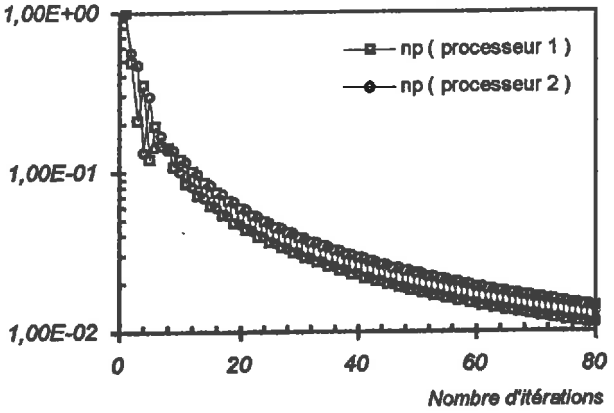


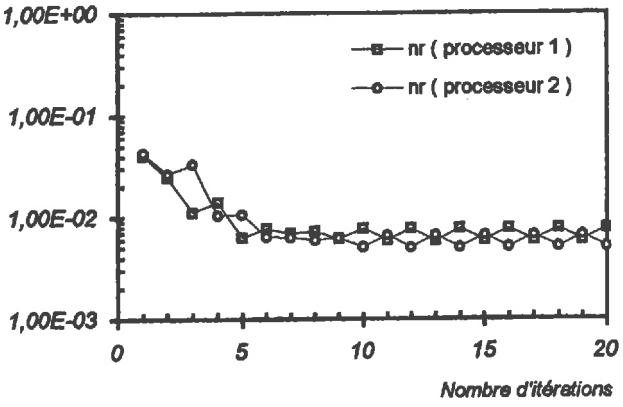
Figure 4.6 : Modèle sans référence de pression – Influence de l'ajout d'une étape de linéarisation de Picard sur le comportement de la norme du résidu n_r



(a) Champ de vitesse



(b) champ de pression



(c) résidu global

Figure 4.7 : Modèle sans référence de pression – Convergence de JNL (2 blocs) sur chaque processeur

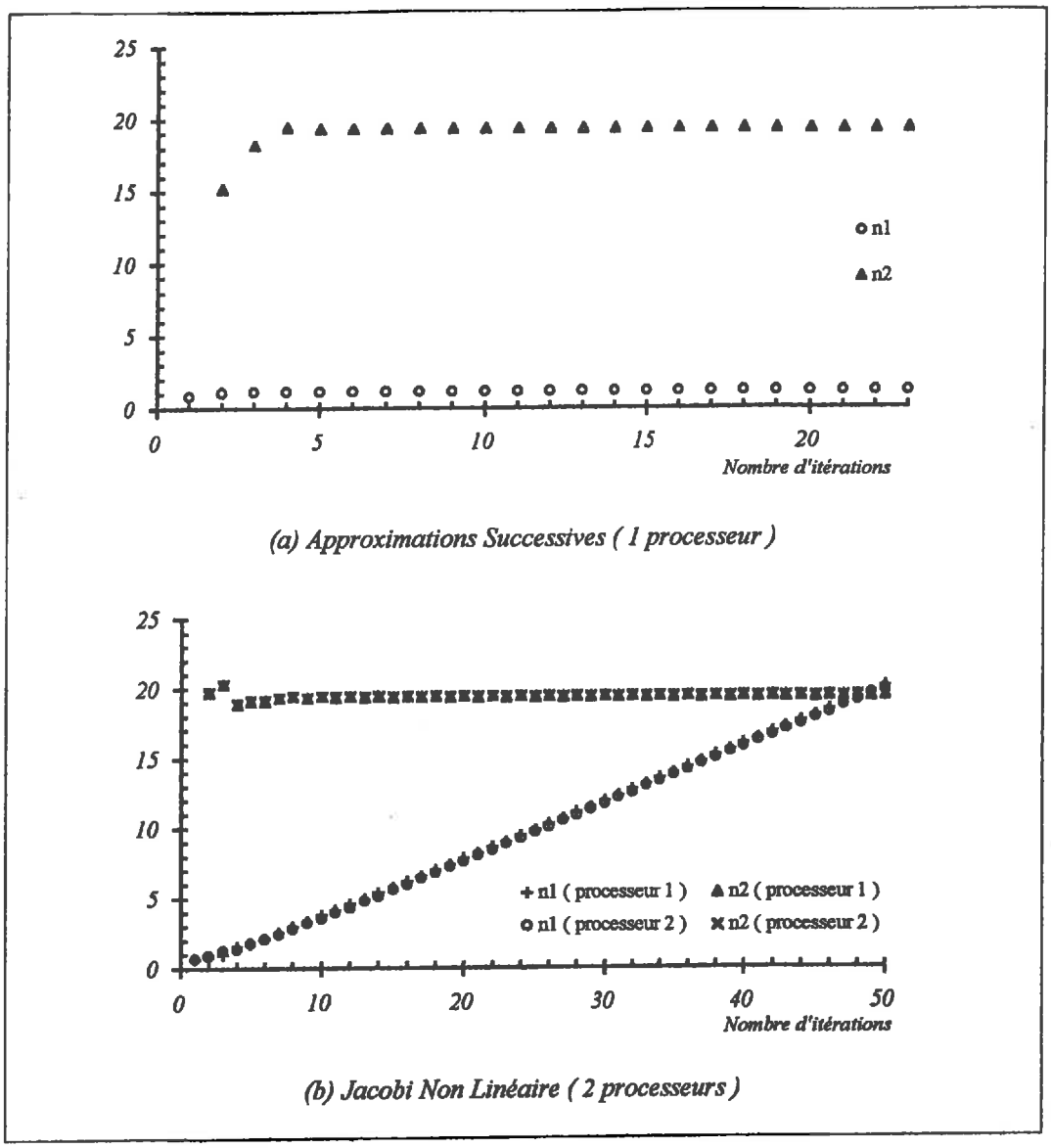


Figure 4.8 : Modèle sans référence de pression – Caractérisation du champ de pression par les normes n_1 et n_2

Lorsque le calcul est effectué sur deux processeurs, la norme n_2 se stabilise également (en une dizaine d'itérations) à la même valeur que précédemment (19,4). Par contre, n_1 croît quasi linéairement.

Conclusion

Ayant vérifié que le champ de pression ne présentait pas de pic localement, nous pensons que l'évolution de la norme n_1 est due à une augmentation de la valeur de la constante additive p_0 à chaque itération du calcul. *Nous suggérons, pour y remédier, l'adoption d'un modèle numérique comportant une référence de pression.*

4.2. MODÈLE AVEC RÉFÉRENCE DE PRESSION

4.2.3. étape de validation des solutions

Les résultats obtenus jusqu'ici semblent indiquer la nécessité d'ajouter une référence de pression. Pour contrôler le bien-fondé de cette hypothèse, nous imposons une valeur nodale de pression (p_0=10^{-2} au centre de la cavité) et conservons la même configuration que précédemment (21×21 noeuds, Re=100, JNL, découpage de type *4.1.a* en 2 blocs).

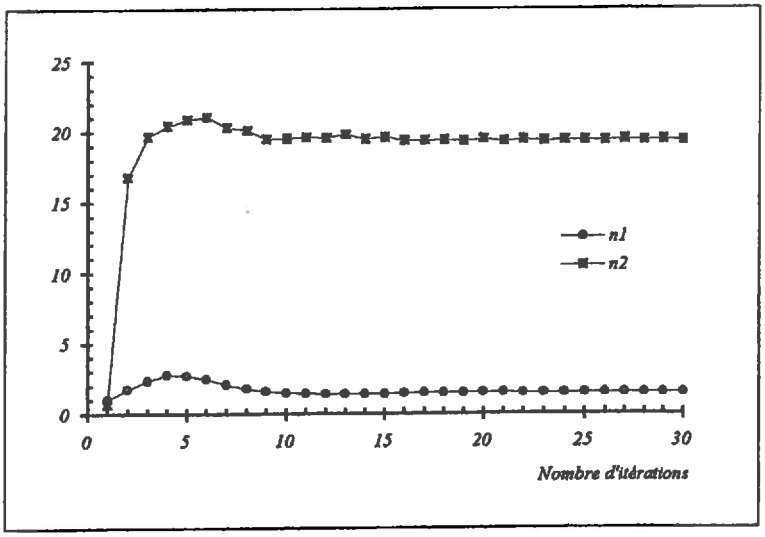


Figure 4.9 : Modèle avec référence de pression – Caractérisation du champ de pression par les normes n_1 et n_2

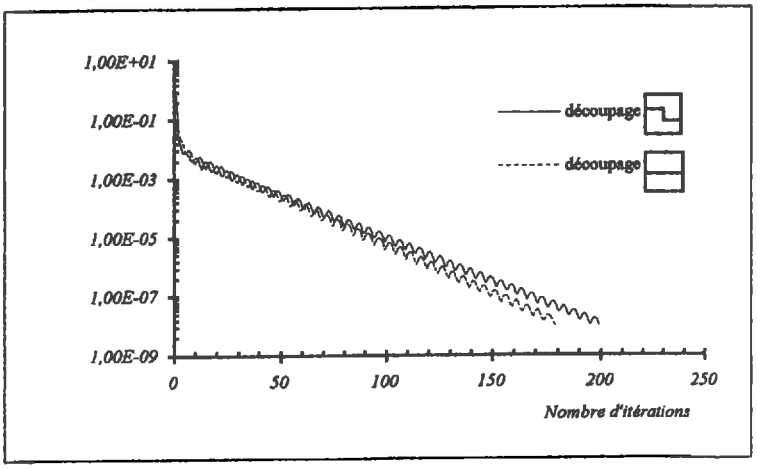


Figure 4.10 : Modèle avec référence de pression – Evolution de la norme du résidu global

On constate que les normes n_1 et n_2 se stabilisent après une dizaine d'itérations (*fig. 4.9*) à des valeurs respectives qui coïncident avec celles obtenues précédemment dans le cadre du calcul monoprocesseur (AS). La *figure 4.10* montre que le résidu global du calcul décroît jusqu'au critère d'arrêt imposé ici à $\alpha_r=10^{-8}$. Le comportement du champ global (vitesse et pression) paraît donc correct, d'autant plus qu'un calcul de l'erreur relative de ce champ par rapport à celui obtenu sur un seul processeur révèle un écart proche de 10^{-16} (c'est à dire le zéro de la machine puisque nous sommes en double précision) après une quarantaine d'itérations de linéarisation de Picard (*fig. 4.11*). Une vérification du champ au niveau local sera effectuée un peu plus loin dans ce chapitre, à partir d'une grille plus fine (81×81).

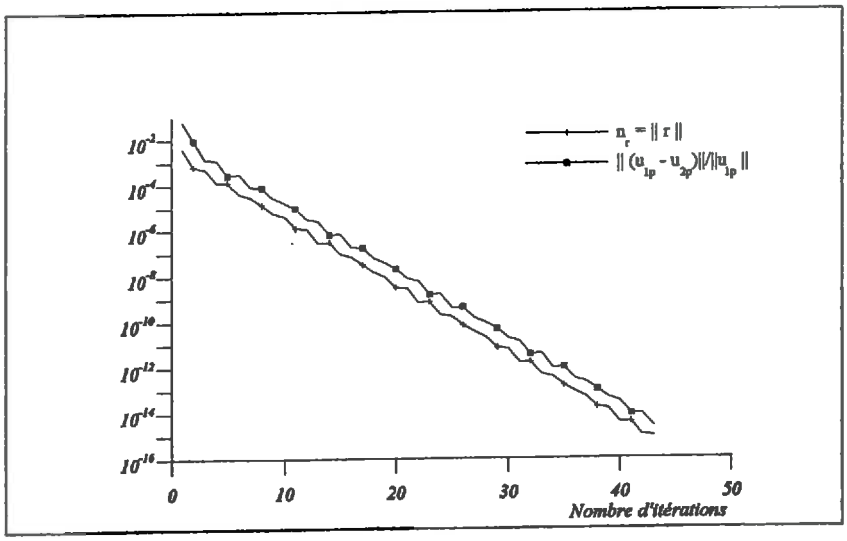
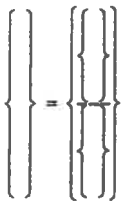


Figure 4.11 : Modèle avec référence de pression – Erreur relative entre les solutions obtenues sur un et sur deux processeurs

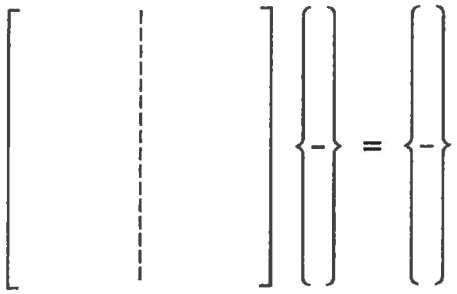
4.2.4. Choix du découpage algébrique

Il est établi que la Méthode des Eléments Finis supporte la distribution de données. Notre idée est d'utiliser le découpage matriciel comme un moyen de conserver ce modèle de programmation pour la résolution du système discrétisé. Ce modèle consiste à associer à chaque processeur un ensemble donné de degrés de liberté du problème, ce qui se traduit par un certain morcellement du vecteur des inconnues nodales. Pour fixer les idées, nous considérons une distribution des données sur deux processeurs P_1 et P_2, mais cela n'a rien de restrictif dans la mesure où l'ensemble des remarques à venir peut être étendu à l'utilisation de p processeurs. Quelle que soit cette distribution, on peut toujours ordonner les inconnues du système de sorte qu'elle équivaut à décomposer le vecteur des inconnues en deux sous-vecteurs comme ci-après :



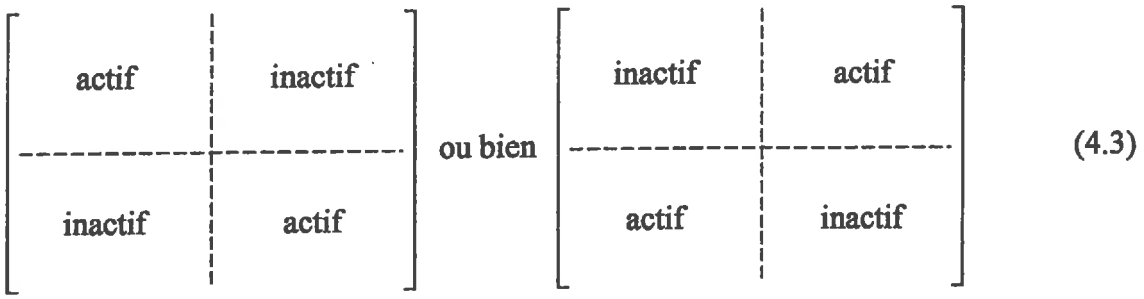
(4.1)

En vertu du produit ligne par colonne, on obtient alors deux sous-systèmes hyperparamétriques (nombre d'équations supérieur au nombre d'inconnues) :



(4.2)

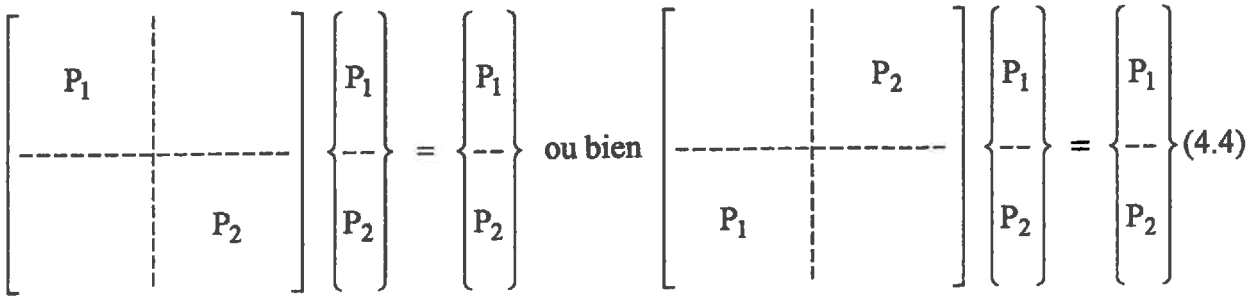
La fermeture de chacun des sous-systèmes implique un découpage matriciel par blocs :



(4.3)

où on qualifie d'*actif* un bloc constitutif de la matrice associée à un sous-système, et d'*inactif* un bloc contribuant au vecteur second membre. Nous nous référerons à ce découpage en tant que *découpage de base.*

Ceci se traduit au niveau machine par une des distributions de données suivantes :



(4.4)

Sur un plan purement algébrique, de nombreux découpages par blocs sont possibles. Voyons quels critères pourraient guider notre choix.

Une procédure d'optimisation numérique comporte deux principaux aspects. Le premier est relatif à la réduction de l'encombrement mémoire. Dans ce contexte, le découpage algébrique

de la matrice initiale doit conduire à *la diminution de la largeur de bande* des sous-matrices résultantes, afin de pouvoir traiter des systèmes de taille plus importante. Par exemple, lors de la résolution de problèmes fortement non linéaires, ceci permettrait d'affiner le maillage.

En ce qui concerne la diminution du temps de restitution, notons qu'une décomposition algébrique par blocs de la matrice s'accompagne obligatoirement d'une diminution du degré de couplage entre les variables nodales (laquelle est proportionnelle au taux de remplissage des blocs inactifs), d'où une dégradation des performances. Par conséquent, le second critère que nous retiendrons est *la minimisation de l'affaiblissement de ce couplage.*

Revenons à notre problème. La *figure 4.12* propose une représentation sous forme matricielle de la structure type des systèmes concernés. Le cas le plus défavorable a été envisagé, dans la mesure où le taux de remplissage de la matrice est maximal : toutes les valeurs nodales sont des inconnues (suite, par exemple, à des conditions aux limites de type Neumann). Ces dernières sont regroupées par variable (u, v, p) de sorte que la matrice globale est composée de neuf sous-matrices creuses, dont cinq sont carrées et quatre rectangulaires. Ces sous-matrices, prises une par une, présentent une structure par bandes. Il est clair, au vu de ce schéma, qu'appliquer le découpage de base (4.3) directement sur la matrice *(4.12)* provoquerait un découplage important. La structure par bandes des sous-matrices (détaillée en *annexe A*) suggère, pour amoindrir ce découplage, d'agir à leur niveau plutôt qu'au niveau global d'une part, et d'autre part d'activer des blocs suivant leur diagonale principale (*fig. 4.13*).

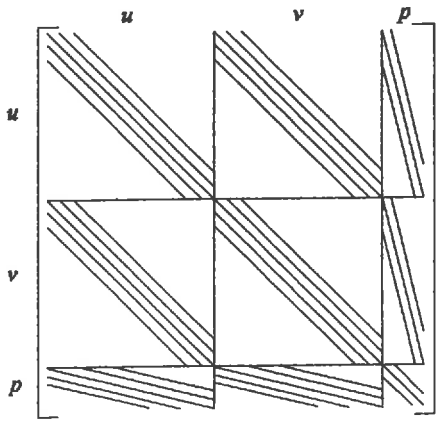


Figure 4.12 : Structure type d'une matrice globale issue d'une discrétisation par éléments finis basée sur des éléments de Taylor-Hood (t6-t3)

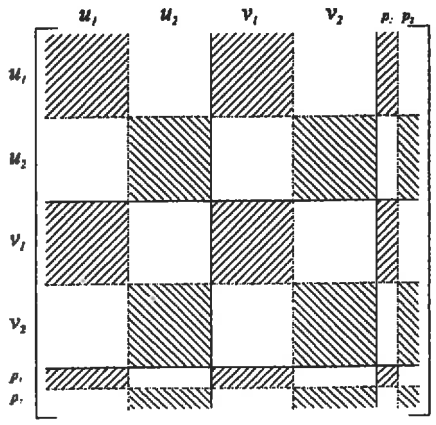


Figure 4.13 : Principe du découpage (de base) par blocs de la matrice globale au niveau de chaque sous-matrice

Intéressons-nous à présent à l'optimisation de la largeur de bande. A partir de maintenant, nous raisonnons sur une des sous-matrices (par exemple un des blocs « vitesse – vitesse ») issue d'une discrétisation basée sur un maillage constitué de 9×9 éléments de Taylor-Hood, balayés séquentiellement par lignes, et lorsque des conditions aux limites de type Dirichlet sont appliquées aux frontières du domaine *(figure 4.14)*. Le découpage de base (4.3) est appliqué : les blocs colorés en bleu ou en vert sont actifs pour un des deux processeurs. Ce schéma permet de visualiser la diminution résultante du couplage (qui porte ici sur 21 valeurs nodales). En ce qui concerne la largeur de bande de cette sous-matrice, on constate qu'elle est inchangée après découpage. Ceci se vérifie en se reportant au *tableau 4.1*. Ce dernier ne répond donc pas aux exigences que nous nous sommes imposées.

Pour remédier à cela, on doit réduire le nombre de zéros inclus dans les blocs actifs. Or, la bande de la sous-matrice est elle-même constituée de cinq bandes pour les blocs vitesse – vitesse et pression – vitesse, et de trois pour les blocs pression – pression et pression – vitesse (cf. *annexe A*). De ce fait, le principe sous-jacent à une réduction de la largeur de bande est l'application du découpage de base à des structures de plus en plus fines de la matrice, comme l'illustrent les *figures 4.14* et *4.14* . Toutefois, la prise en compte conjointe du critère relatif au couplage définit une limite à ce principe descendant.

Le principe de raffinement du découpage de base peut être appréhendé de manière plus simple en exploitant la correspondance qui existe entre le plan algébrique et le plan géométrique*, lequel permet d'agir sur la structure de la matrice associée au problème au travers du maillage. En effet, on peut dire que, dans le cas présent, la largeur de bande de la matrice globale dépend du nombre T de nœuds suivant la direction de balayage. Pour réduire la largeur de bande, il faut donc réduire T (*annexe A*). Ceci équivaut, dans le contexte présent, à morceler le maillage suivant la figure *4.15 (c et d)*, c'est à dire en découpant le maillage perpendiculairement au sens de balayage. Il s'ensuit, au vu du chapitre 3, que la réduction de la perte du couplage passe par la minimisation de l'étendue des pseudo-frontières internes.

* Voir chapitre 1

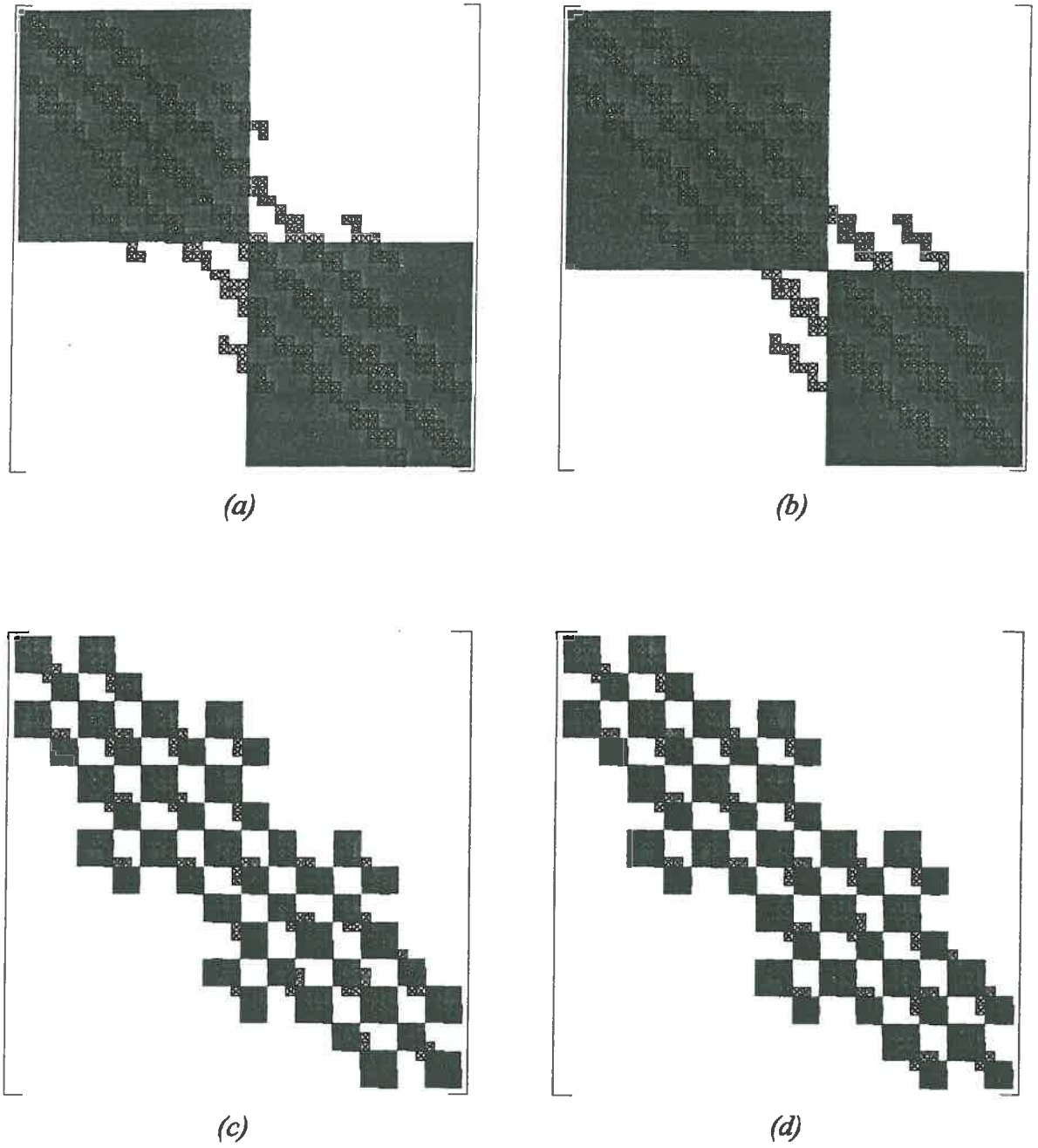


Figure 4.14 : Découpages algébriques générant des sous-matrices caractérisées par une largeur de bande identique *(cas a et b) ou inférieure (cas c et d) à celle de la matrice globale*

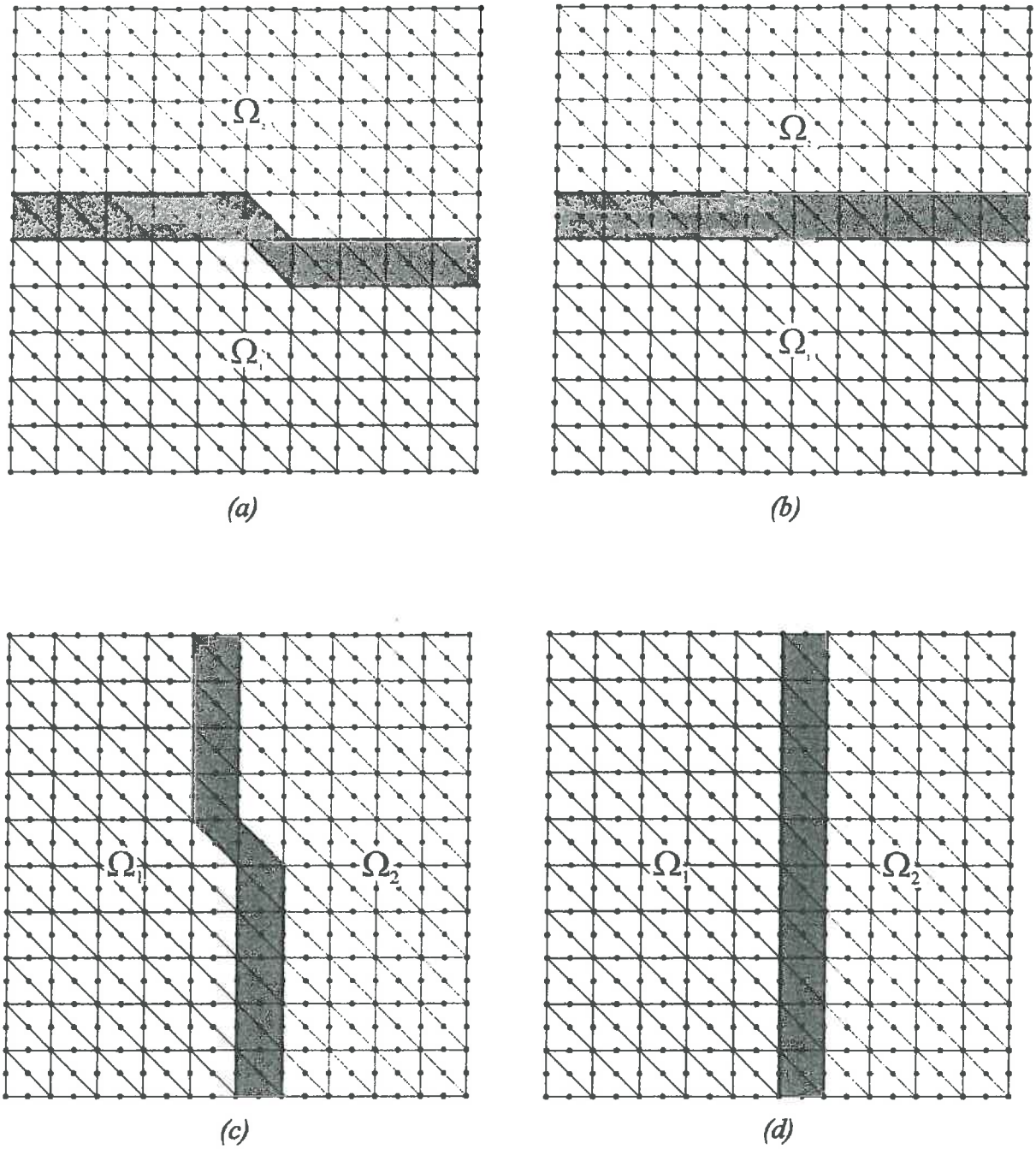


Figure 4.15 : Répercussions dans le plan géométrique de découpages algébriques induisant soit une conservation (a et b), soit une réduction (c et d) de la largeur de bande de la matrice associée à chaque sous-système

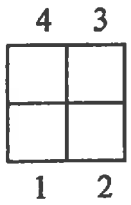
Nous avons établi que la distribution des degrés de liberté d'un système algébrique peut toujours se ramener à la décomposition par blocs de la matrice associée (par une renumérotation appropriée des noeuds). Concernant la cavité carrée, *si l'on désire une réduction significative de la largeur de bande, il faut d'une part que la numérotation des noeuds soit séquentielle et, d'autre part, que le découpage matriciel corresponde à un découpage géométrique orthogonal* à la direction de balayage des noeuds. Par ailleurs, la réduction de la perte du couplage existant entre les valeurs nodales s'effectue par une minimisation de l'étendue des pseudo-frontières*

Ces éléments amènent les réflexions suivantes :

1- Dans le cas de deux processeurs, le découpage optimal est celui soit de la *figure 4.15*, *soit de la figure 4.15*

2- Dans le cas de quatre processeurs, le découpage le plus intéressant pour la réduction de la largeur de bande est une découpe en bandes perpendiculaire au sens de balayage comme suit.

Par contre, la perte du couplage est ici relativement importante comparativement au découpage en damier représenté ci-après.



En effet, dans le premier cas on recense six unités de frontière interne tandis que le second en comprend quatre. Ce dernier devrait donc nécessiter moins d'itérations externes que le précédent. Une vérification est prévue un peu plus loin (*tableau 4.4*).

Nous allons, à présent, mettre en œuvre ces différents découpages. Afin de simplifier le texte, nous donnons un nom à chacun d'entre eux :

 découpage régulier horizontal

 découpage régulier vertical

 découpage en « zigzag » horizontal

 découpage en « zigzag » vertical

 découpage en damier

Tableau 4.1 : Taux de remplissage des matrices

(21 x 21) = 842 inconnues → 110 008 réels	
	397 inconnues → 48399 réels
	445 inconnues → 53743 réels
	397 inconnues → 24247 réels
	445 inconnues → 31459 réels
(41 x 41) = 3482 inconnues → 943 408 réels	
	1692 inconnues → 444 484 réels
	1790 inconnues → 465 268 réels
	1692 inconnues → 221 612 réels
	1790 inconnues → 252 434 réels
(61 x 61) = 7922 inconnues → 3 201 714 réels	
	3887 inconnues → 1 566 269 réels
	4035 inconnues → 1 612 493 réels
	3887 inconnues → 781 077 réels
	4035 inconnues → 851 909 réels
(81 x 81) = 14162 inconnues → 3 201 714 réels	
	3412 inconnues → 453 412réels
	3570 inconnues → 496 700 réels
	3569 inconnues → 496407 réels
	3611 inconnues → 516 991réels

* il ne s'agit pas d'un découpage orthogonal au sens géométrique du terme mais au sens de la numérotation, à savoir que l'on passe du noeud (i) au noeud (i+T), T étant le nombre de noeuds dans la direction de balayage (variable).

4.3.3. Etude paramétrique

On se place ici dans le cadre du modèle avec référence de pression.

4.3.3.1. Résultat des calculs sur deux processeurs

1. Effet du choix du découpage

Nous allons à présent comparer quatre découpages afin de déterminer lequel satisfait au mieux les deux critères* précédemment établis. Pour ce faire, nous introduisons les grandeurs suivantes :

$$\tau_I = \frac{\text{temps par itération}}{\text{temps de référence par itération}} \tag{4.1}$$

$$\tau_R = \frac{\text{nombre d'éléments stockés}}{N^2}, \qquad N : \text{ordre de la matrice} \tag{4.2}$$

$$\tau_T = \frac{\text{temps par itération} \times \text{nombre d'itérations}}{\text{temps de référence}} \tag{4.3}$$

Pour les besoins de ce calcul, sont définies les valeurs de référence :

* le temps de référence par itération vaut 7,6 secondes,
* le temps de référence vaut 1003,2 secondes.

Notons, tout d'abord, que le critère de convergence portant sur le résidu global a été fixé à $\alpha_r=10^{-8}$. Les principales caractéristiques de ce calcul ont été reportées dans le *tableau 4.2* . D'un examen de ce dernier, sont apparues les remarques que nous exposons ci-après.

1. Equilibrage des tâches

On constate d'emblée qu'un découpage en zigzag, qu'il soit horizontal ou vertical, équilibre les tâches entre les deux processeurs ($\tau_I(P1) = \tau_I(P2)$). Il n'en est pas de même pour un découpage dit régulier. En effet, les sous-systèmes à résoudre ont, dans ce cas, des tailles différentes. Aussi, en supposant que (P3) hérite le système de plus grande taille, nous avons :

$$\tau_I(P1) > \tau_I(P2) \Rightarrow \tau_I = \max(\tau_I(P1), \tau_I(P2))$$

* Minimisation de la largeur de bande et réduction de la perte de couplage.

2. Diminution de la largeur de bande et du temps de calcul par itération

* Les valeurs du paramètre τ_r montrent que le choix d'un découpage vertical en conjonction avec la numérotation séquentielle en ligne réduit la largeur de bande.

* D'autre part, l'effort de calcul dû à la factorisation LU de chacun des sous-systèmes est d'autant plus prononcé que la largeur de bande associée est grande, et accroît le temps par itération.

* Les données du tableau relatives au temps par itération (rapporté à une référence) et aux taux de remplissage des sous-matrices associées (valable également pour le système global) illustrent l'intérêt d'un découplage vertical comparativement à un découpage horizontal.

* Notons que du point de vue stockage requis (largeur de la bande), les deux types de découpages verticaux nécessitent des ressources machine quasiment équivalentes. Toutefois, pour ce qui est du temps par itération, le découpage vertical en «zigzag» est avantagé par rapport au découpage vertical régulier (du fait de l'équilibrage des tâches entre les processeurs dont il bénéficie).

3. Nombre d'itérations externes

* La figure 4.16 montre que les découpages de type régulier ) conduisent pour une taille donnée des systèmes, à un nombre équivalent d'itérations externes.
* Les découpages réguliers du fait qu'ils minimisent l'étendue des pseudo-frontières internes, réduisent par conséquent la perte de couplage (engendrée par la procédure de découplage algébrique par blocs) donc le nombre d'itérations externes, et ce, en comparaison avec des découpages en zigzag.

4. Temps total de restitution

* Il est proportionnel au temps par itération ainsi qu'au nombre d'itérations externes. Le découpage adopté doit bénéficier favorablement d'un compromis entre ces deux facteurs.

* C'est ainsi, que le découpage vertical régulier bien que pénalisé par le non-équilibre des tâches entre les processeurs, est, au vu des résultats présentés dans le *tableau 4.2* , le plus intéressant en terme de coût d'exploitation (temps total de restitution).

2. Comparaison entre calcul séquentiel et calcul parallèle

Le *tableau 4.2* propose une comparaison entre les calculs effectués sur un seul processeur (un bloc – AS) et ceux effectués sur deux processeurs à l'issue d'un découpage vertical régulier (2 blocs – JNL). On constate que le calcul parallèle ne se justifie pas d'un point de vue temps de restitution, dans une telle configuration. C'était prévisible. En effet, concernant les méthodes itératives de base, tant que la taille du système permettra un traitement sur un seul processeur (i.e. un seul bloc), celui-ci sera préférable car il préserve le couplage.

Tableau 4.2 :
Jacobi non linéaire, performances et stockage requis pour différents découpages ($\alpha_0=10^{-8}$, $\omega=1$, grille 21×21)*

						
τ_I	(P1)	1.0000	0,6316	1,2368	0,8421	1,6579
	(P2)	1.0000	0,6316	0,8132	0,4737	
Nombre d'itérations		132	119	64	70	12
τ_R	(P1)	0,2970	0,1570	0,2710	0,1590	0,155
	(P2)	0,2850	0,1600	0,3070	0,1540	
	Global	0,1455	0,0794	0,1441	0,0786	
τ_T	(P1)	1.0000	0,5694	0,5997	0,4466	0,151
	(P2)	1.0000	0,5694	0,3943	0,2512	

* N=842

Nous conservons les grandeurs de référence précédentes. Les relations (4.1)-(4.3) permettent donc d'accéder aux valeurs dimensionnées (temps et nombre de réels stockés).

Nous augmentons la densité du maillage (grille 41×41). Nous constatons (tableau. *4.3*) que l'augmentation de la taille du système s'accompagne d'une réduction de l'écart entre les temps de restitution entre le calcul monoprocesseur et biprocesseur. En effet, le rapport des speed-up vaut :

$$\frac{S_{p\,(41\times41)}}{S_{p\,(21\times21)}} = 2{,}2745$$

Ce résultat indique une amélioration du speed-up. Toutefois, le calcul parallèle ne se justifie toujours pas dans la mesure où l'on demeure en dessous de l'unité. Néanmoins, cela nous encourage à raffiner davantage la grille. Malheureusement, la mémoire par processeur de l'iPSC860 limite ici la taille du maillage à des grilles 41×41.

Tableau 4.3 :
Jacobi non linéaire, performances et stockage requis
Une grille 41×41 ($\alpha_r=10^{-8}$, $\omega=1$)

			
τ_I	(P1)	4.4404	1,0000
	(P2)	4.0944	
Nombre d'itérations		80	12
τ_R	(P1)	0,02082	0,07781
	(P2)	0,01828	
	Global	0,03099	
τ_T	(P1)	2.6912	2.0694
	(P2)	2.4814	

Tableau 4.4 :
Jacobi, performances et stockage requis
pour différents découpages (α_r=10^{-6}, ω=0,4 grille 81x81)*

				
τ_R	Global	0,00979	0,01919	0,03891
	(P1)	0,03965	0,07735	
	(P2)	0,03897	0,07667	
	(P3)	0,03897	0,07677	
	(P4)	0,03895	0,07606	
Nombre d'itérations		49	37	
τ_I	(P1)	0,79505	0,61254	
	(P2)	0,90717	0,66289	
	(P3)	0,91890	0,67730	
	(P4)	1,00000	0,70267	
τ_T	(P1)	0,79505	0,46253	
	(P2)	0,90717	0,50055	
	(P3)	0,91890	0,51143	
	(P4)	1,00000	0,53059	

temps de ref / iter : 698,89s

temps de ref : 34245,610s

4.3.3.2. Résultat des calculs sur quatre processeurs

Comme nous l'avions anticipé, le découpage en damier conduit à une convergence plus rapide en terme d'itérations externes (Tableau 4.4).

Les tests effectués sur quatre processeurs ont nécessité l'introduction d'un coefficient de sous-relaxation ω .

* N=14162

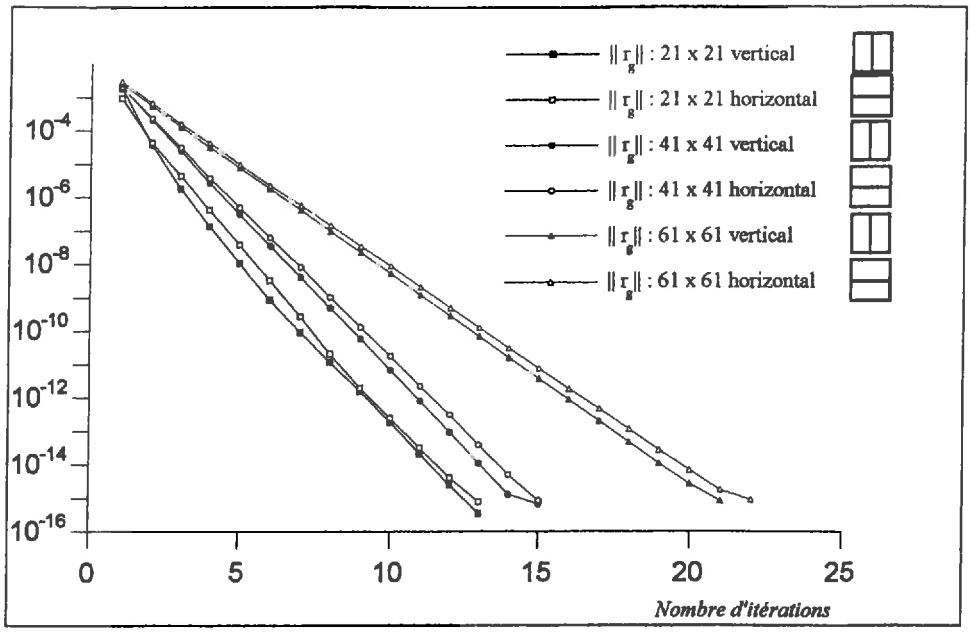


Figure 4.16 : Modèle avec référence de pression – Influence du maillage

* A partir du moment où certains termes changent au cours de la procédure de résolution (valeurs nodales sur les frontières artificielles i.e. correspondant à des blocs inactifs), il semble inutile de résoudre les équations précisément à chaque itération externe. Toutefois, à l'instar d'une approche découplée, une optimisation de ce type de méthode nécessite un soin tout particulier au choix du nombre d'itérations internes *(figure 4.19).*
* Un autre point important concerne la réactualisation du vecteur solution d'une itération externe à la suivante. Celle-ci doit parfois être limitée sous peine de non-convergence du processus itératif global. Un exemple est donné par les résultats des calculs sur quatre processeurs. La remise à jour complète du vecteur solution provoque des oscillations au niveau du résidu du calcul (voir *figure 4.20*). On voit sur la *figure 4.17* que l'adjonction d'un coefficient de sous-relaxation (ω =0,5 en l'occurrence) permet au processus global de converger.

A partir de ces deux remarques, une petite étude paramétrique a été menée dont les résultats ont été reporté sur les *figures 18* et *19*. Lorsqu'aucune relaxation est appliquée (ω=1), de fortes oscillations apparaissent au niveau du champ de pression qui viennent perturber le champ de vitesse. La *figure 4.20* montre que ces oscillations apparaissent d'abord au niveau des processeurs P2 et P3, ce qui va dans le sens de notre analyse précédente. En effet, ces processeurs sont associés à des sous-systèmes pour lesquels les blocs inactifs ont un taux de

remplissage supérieur aux processeurs 1 et 4, c'est à dire pour lesquels la perte de couplage est importante.

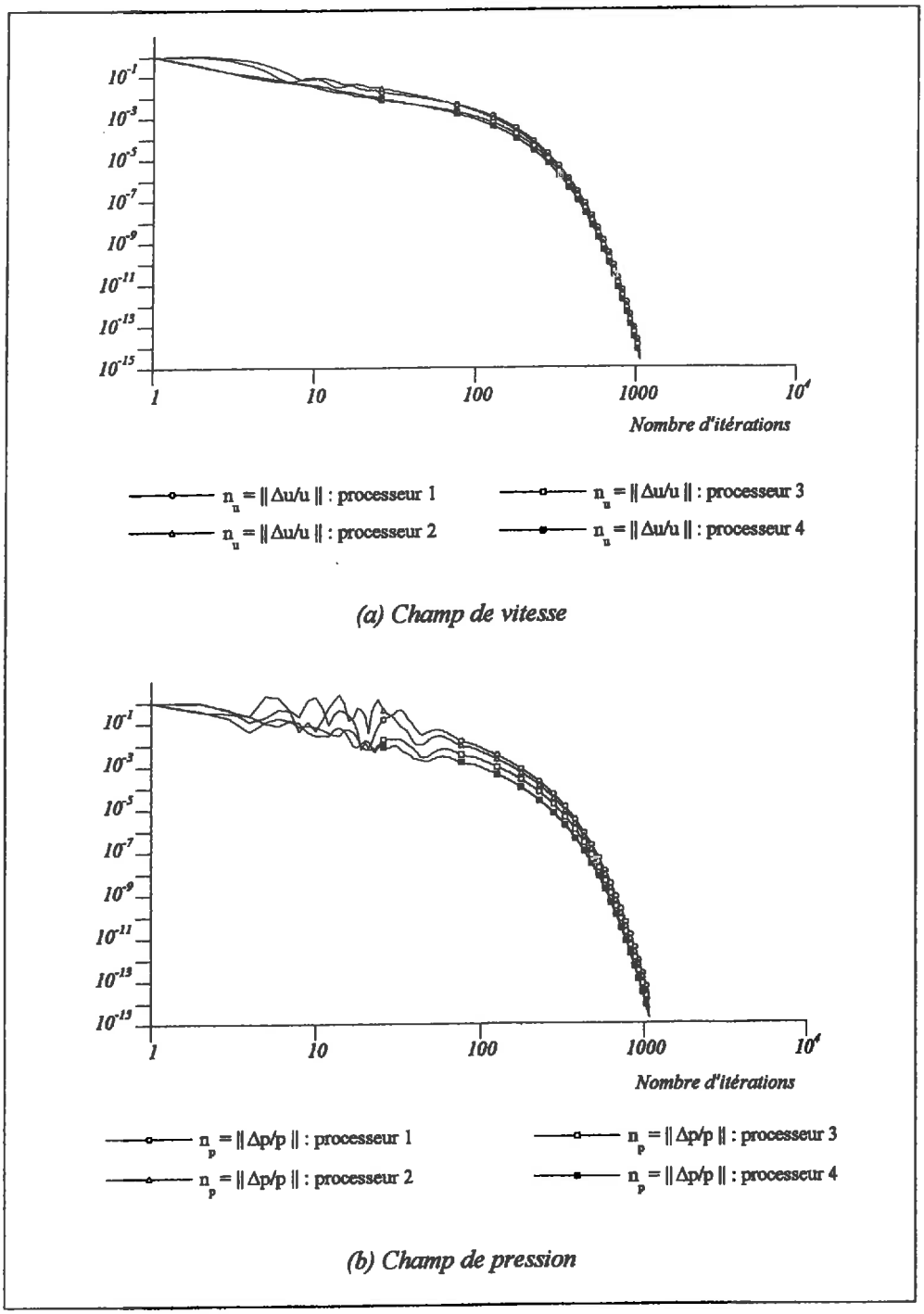


Figure 4.17 : Résolution par JNL – Influence d'un coefficient de sous-relaxation ω=0,5

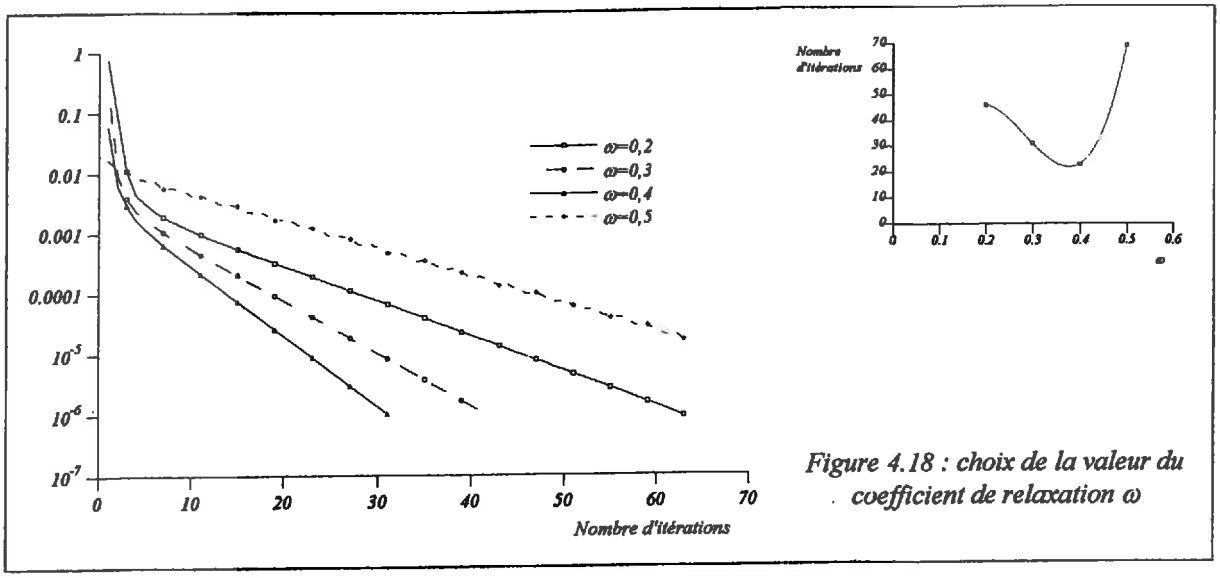


Figure 4.18 : choix de la valeur du coefficient de relaxation ω

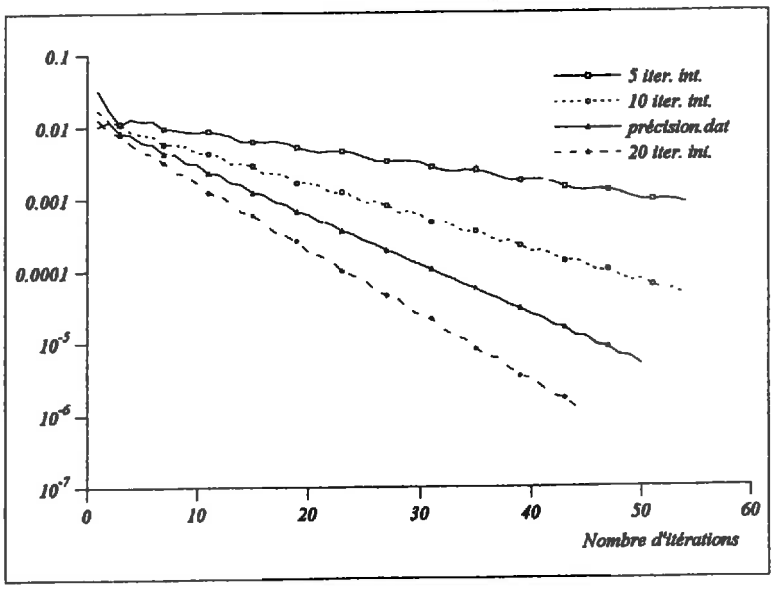


Figure 4.19 : Influence du nombre d'itérations internes sur l'évolution du processus itératif externe

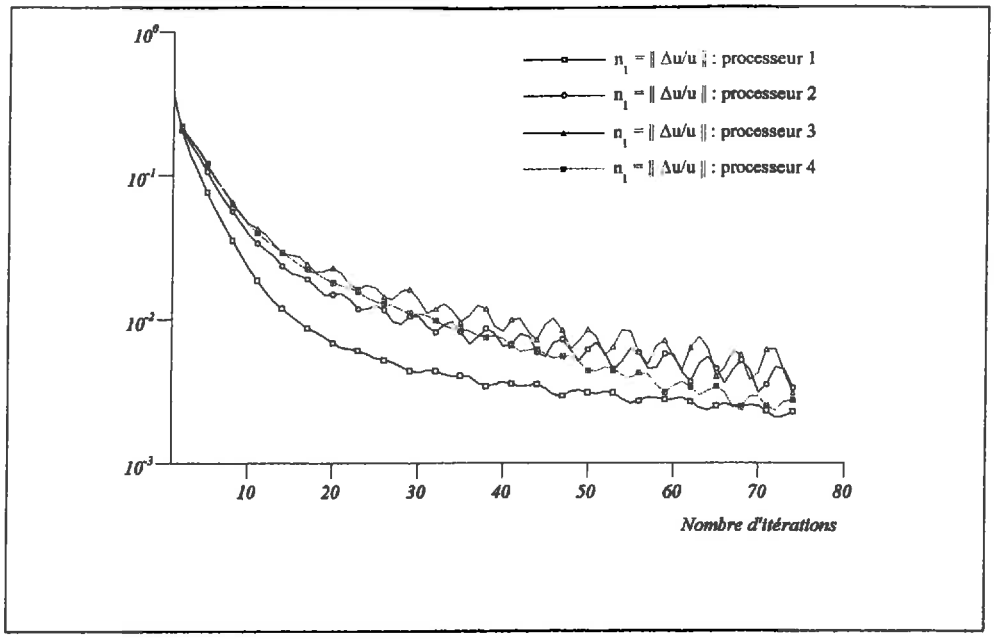


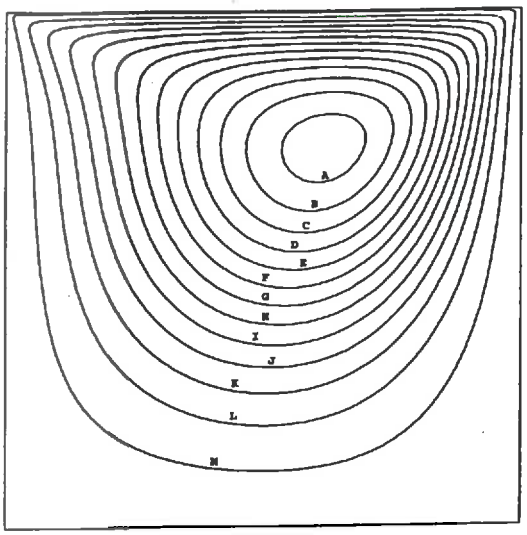
Figure 4.20 : Résolution par JNL – Corrections sur la vitesse par itération - Découpage en colonnes

4.3.3.3. Validation de la solution sur quatre processeurs

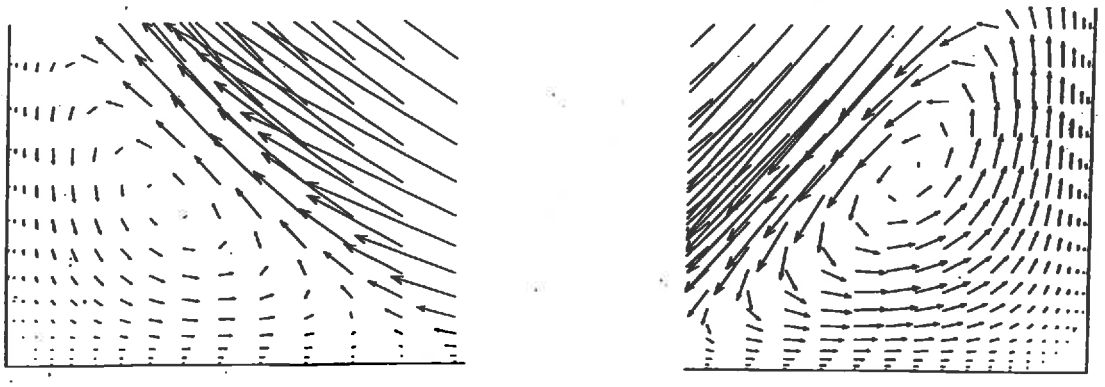
En début de cette section, nous avons validé les solutions obtenues par le solveur parallèle (grille 41x41) par rapport à celles obtenues par le solveur séquentiel, par le biais de critères de convergence globaux (norme du résidu global). Afin qu'il ne subsiste aucun doute, nous avons également vérifié nos solutions par rapport à celles fournies par la littérature en les inspectant localement.

Grâce à la distribution des données sur plusieurs processeurs, nous sommes parvenus à affiner le maillage (grille 81×grille81). Cela nous a permis d'améliorer notre prédiction de l'écoulement et de coller aux résultats de Ghia et al.

Re=100	Ghial et al.	Bruneau & J.	Présent calcul
ψ_{min}	-0,103423	-0,1026	-0,10352
Position des centres	0,6172 0,7344	0,6172 0,7344	0,6172 0,7344



A	-0.09954
B	-0.09157
C	-0.08361
D	-0.07564
E	-0.06768
F	-0.05971
G	-0.05175
H	-0.04378
I	-0.03582
J	-0.02785
K	-0.01989
L	-0.01192
M	-0.003958



igure 4.22 : Ecoulement dans la caviré entraînée pour Re=100 :
lignes de courant du vortex primaire et vecteurs vitesse des structures convectives secondaires.

4.4. Axes de développement actuels

La validation du solveur parallèle ayant été effectuée, nous avons choisi deux axes de développement que nous présentons ci-après.

Les développements en cours suivent les deux orientations suivantes :

* L'application du solveur parallèle à des problèmes moins académiques,
* L'évolution des algorithmes de résolution vers des méthodes itératives plus robustes.

Dans le cadre de la première orientation, nous abordons actuellement un problème relatif à la dynamique des gaz qui nous permet d'appliquer notre procédure de résolution à des problèmes très différents puisqu'il s'agit d'écoulements instationnaires de fluides compressibles (cf. annexe B).

Notre second axe de développement concerne les méthodes de sous espace de Krylov. En effet, les performances de telles méthodes sont étroitement liées au conditionnement de la matrice associée au problème traité. Or, la discrétisation par éléments finis de Galerkin des équations de Navier-Stokes produit des systèmes à matrice non symétrique particulièrement mal conditionnée[6]. C'est pourquoi ces méthodes sont rarement efficaces pour de tels problèmes, excepté si un préconditionnement correct du système est réalisé. Nous effectuons actuellement des tests sur un préconditionnement de type Jacobi par blocs. Voyons les premières tendances qui se dégagent.

Les méthodes CGS et BiCGSTAB ont été appliquées, dans un premier temps, à la résolution de l'équation de Laplace afin de vérifier la fiablilité du solveur.

Afin de ne pas superposer les difficultés, nous avons ensuite abordé un problème de Stokes. Nous avons constaté qu'en l'absence de préconditionneur, les méthodes précitées conduisaient à une stagnation du résidu global. Le préconditionnement du système par une matrice issue d'un découpage vertical régulier en deux blocs, a permis la convergence du processus itératif externe en trois ou quatre itérations.

Les calculs ont été étendus à la résolution des équations de Navier-Stokes. La méthode BiCGSTAB s'est révélée plus robuste que CGS. Nous avons noté, en accord avec la littérature, que la précision imposée au niveau du processus itératif interne (résolution du système linéaire issu d'une linéarisation de Picard ou de Newton-Raphson) conditionnait la

précision finale du processus externe. En effet, pour obtenir une précision finale de l'ordre de α_r, il nous a fallu imposer une précision de l'ordre de $10^{-2}\times\alpha_r$ au processus interne.

D'autre part, il a été remarqué une convergence très irrégulière au niveau de la résolution des systèmes linéaires consécutifs. Nous entendons par irrégulier le fait que les vecteurs résidus successifs varient énormément en ordre de grandeur. Pour autant, cela n'a pas induit la divergence du processus itératif global. Sur ce point, une linéarisation de type Newton-Raphson atténue significativement ces oscillations par rapport aux itérations de Picard. Toutefois, lorsque les effets des non linéarités ont été augmentées (Re=200 et 400), il a été nécessaire d'initialiser les itérations de Newton-Raphson par quelques itérations de Picard, et de démarrer une nouvelle résolution à partir de la solution obtenue pour une valeur plus faible du nombre de Reynolds.

Conclusion

Conclusion

Le sujet du présent mémoire concernait l'application du calcul parallèle à la résolution des équations de Navier-Stokes dans le cadre de la Méthode des Eléments Finis. Il s'inscrit dans un thème de recherche nouveau au sein de l'équipe, dont la finalité est d'élaborer une bibliothèque d'unités de programmes pour la construction de solveurs aux éléments finis. La contribution de la présente étude a été de doter cet environnement de programmation de la possibilité d'une mise en œuvre sur architecture parallèle.

Le travail a essentiellement résidé dans la mise en place d'un solveur parallèle itératif. Celle-ci a consisté dans un premier temps à définir une approche générale et les moyens de la concrétiser. Les objectifs à atteindre en matière de développement couvraient principalement deux domaines :

1. *La programmation* :
    - possibilité d'adapter les programmes à d'autres machines parallèles (portabilité),
    - minimisation et localisation des modifications à apporter au programme lors du passage d'une mise en œuvre séquentielle à une mise en œuvre parallèle,
2. *Les algorithmes :*
    - choix d'algorithmes à fort degré de parallélisme permettant la résolution des systèmes algébriques issus d'une discrétisation par éléments finis,
    - facilité d'adaptation à des problèmes divers (flexibilité), en particulier à des géométries complexes afin de ne pas perdre les avantages offerts par la Méthode des Eléments Finis.

Après avoir examiné les principales approches actuelles en matière de Calcul Parallèle, il nous est apparu que la meilleure voie, au vu de nos objectifs, était la distribution de données. Afin d'inscrire l'étape de résolution dans ce modèle de programmation, nous avons défini une procédure numérique basée sur un découpage algébrique de la matrice associée au problème.

Nous avons dégagé deux critères permettant de choisir entre parmi les découpages algébriques possibles ; l'un est relatif à l'encombrement mémoire, l'autre au coût d'exploitation. Le premier conduit à choisir des découpages qui réduisent la largeur de bande de la matrice associée. Ce choix est lié à la numérotation des nœuds qui doit être effectuée de sorte que, au

sein d'un même élément, les écarts entre indices soient minimes. Il justifie, d'autre part, l'utilisation d'une méthode de résolution directe (LU) eu égard à la taille des sous-matrices (blocs actifs) résultantes.

Le second critère consiste à minimiser la diminution du degré de couplage existant entre les variables nodales, qui résulte de tout découpage par blocs (contribution dans le second membre du bloc inactif) et qui a pour effet d'augmenter le nombre d'itérations externes.

Une seconde phase de l'étude a été consacrée à la validation du solveur parallèle. Celle-ci implique la résolution d'un problème relativement académique afin de disposer de solutions de référence. La seconde partie du mémoire s'est donc façonnée autour d'une série de tests numériques effectués à partir de la simulation de l'écoulement au sein d'une cavité carrée entraînée. La mise au point du modèle numérique a été l'occasion de préciser nos idées concernant le traitement couplé des équations de Navier-Stokes incompressible en variables primitives. Une discussion a été ouverte sur l'adéquation ou non d'une référence de pression. Il est apparu nécessaire d'imposer cette référence, en l'absence de quoi le champ de pression ne se stabilise pas.

C'est donc dans le cadre d'un modèle avec référence de pression que les solutions obtenues à l'aide du solveur parallèle ont été confrontées à celles issues du calcul séquentiel précédemment comparées à celles fournies par la littérature. La finalité du calcul sur quatre processeur a été d'affiner la grille dans le but de décrire plus précisément la structure de l'écoulement (mise en évidence de zones tourbillonnaires secondaires).

Une étude paramétrique a été menée, mettant en œuvre différents découpages (deux puis quatre processeurs). Ces derniers ont été comparés en s'appuyant sur les critères précédemment définis. Il en ressort qu'une réduction significative de la largeur de la bande implique le retour à une numérotation séquentielle des nœuds et à un découpage géométrique perpendiculaire à la direction de balayage des nœuds. La réduction de l'affaiblissement du degré de couplage entre les valeurs nodales équivaut, de par la correspondance existant entre le plan algébrique et le plan géométrique, à la minimisation de l'étendue des frontières (internes) artificielles. La combinaison de ces deux critères conduit à choisir un découpage vertical régulier pour des calculs sur deux processeurs et un découpage en damier pour des calculs sur quatre processeurs.

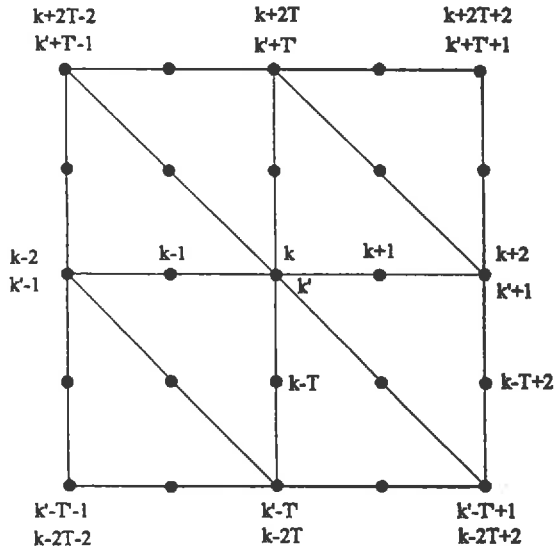
L'exploitation des résultats (cas test de la cavité entraînée) indique, au stade où en sont nos développements, que le calcul parallèle tel que nous l'appliquons ne se justifie pas. Toutefois, l'accroissement de la taille du problème pour un nombre de processeurs fixé s'accompagne d'une augmentation du « speed-up », ce qui est encourageant. Pour vérifier si cette tendance se

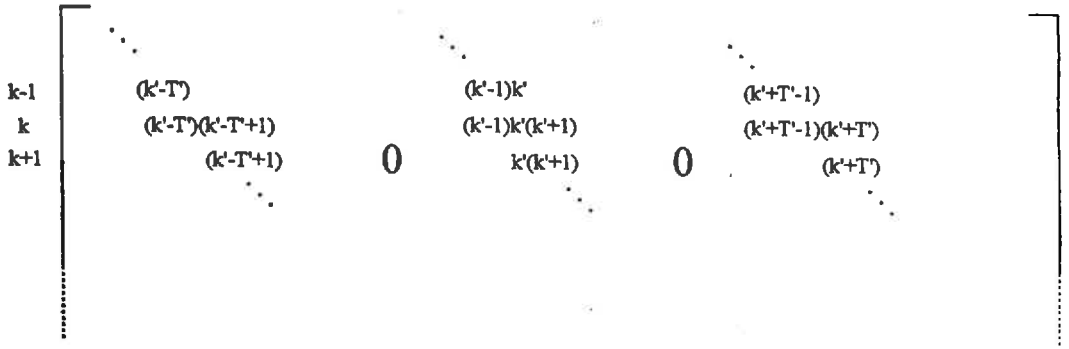
maintient pour des tailles supérieures, il aurait fallu disposer d'une mémoire par processeur plus importante.

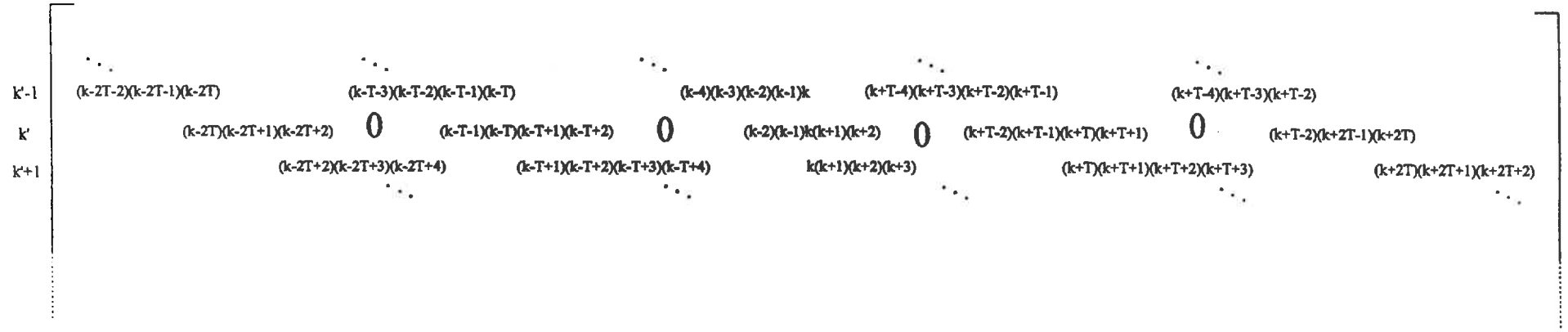
Les phases d 'élaboration et d'implémentation ont été accomplies conformément aux objectifs initiaux, y compris celles concernant les méthodes des sous-espaces de Krylov. La phase de validation a été entamée. Sa poursuite est prévue, notamment celle relative aux méthodes précédemment citées. En effet, avant de passer à des géométries plus complexes, il faut que le solveur soit suffisamment robuste (le rayon de convergence le moins restreint possible) afin de permettre la résolution de systèmes fortement non linéaires.

L'ensemble du présent travail a été mené à bien sur une seule et même plate-forme et n'a pas bénéficié de la nette évolution technologique survenue dans un passé récent. Il va sans dire qu'il pourrait être enrichissant de le reprendre avec les architectures les plus performantes actuellement sur le marché.

Annexes







Structure des matrices creuses issues d'une discrétisation basée sur des éléments de Taylor-Hood

Ci-contre sont présentées un ensemble de figures complémentaires à la publication précédente :

- Figure 1 -

Maillage du domaine de calcul à base d'éléments triangulaires isoparamétriques (18760 inconnues).

- Figure 2 -

Zoom sur le maillage au voisinage de la sortie de la tuyère.

- Figure 3, 4, 5, 7 -

Evolution temporelle de l'écoulement. Visualisation des surfaces à nombre de Mach constant.

- Figure 6 -

Visualisation de la partie supersonique du jet à un temps égal à 9 fois le temps mis par le signal acoustique pour parcourir la tuyère.

- Figure 8 -

Visualisation de la partie supersonique du jet à un temps égal à 12 fois le temps mis par le signal acoustique pour parcourir la tuyère.

Les deux dernières figures représentent l'évolution des vecteurs vitesse pour différentes valeurs du temps jusqu'au régime permanent.

Parallel computation of an unsteady compressible flow

E. Onuphre and A. Chambarel

Institut de Recherche sur les Phénomènes Hors d'Equilibre
UMR138-CNRS, 1 rue Honnorat, 13003 Marseille France

Abstract. In the present paper, a general background is presented for developping parallel applications in the domain of Computational Fluid Dynamics. This frame of work is based upon block Jacobi preconditioned iterative methods for solving partial differential equations. It is shown how the parallelism potential of such a preconditioning can be efficiently exploited by associating it with Finite Element discretization and Object Oriented Programming. The resulting parallel applications are characterized by coarse granularity, ease of maintaining good load balance and the possibility of using the same object in both a serial or a parallel computating context. As an application of our parallel approach, the simulation of an unsteady compressible flow is discussed.

1 Introduction

Computational fluid dynamics (CFD) often require high computational power. This demand has led to a large amount of work in the domain of Parallel Computing. Among the various investigated tracks, Domain Decomposition is still a very active area of research which refers to methods for solving linear or nonlinear systems of equations arising from the discretization of partial differential equations (PDEs). Its popularity mainly lies in the fact that it is a natural device to achieve coarse grain parallelism. In spite of the amount of available algorithms, Domain Decomposition techniques are nevertheless not widespread in the engineering and scientific computing community. It makes sense to say that the lack of available libraries and software tools for parallel computers has something to share with this neglect. Obviously, when partitioning the initial domain into many subdomains, the handling of either the interfaces or the overlapping regions (specially from a programming point of view), as well as the maintain of load balance become ticklish, which seems to be a serious obstacle to automatic Domain Decomposition.
This is the reason why we have chosen another strategy for developing parallel CFD applications which is based upon algebraic matrix splitting. The numerical framework presented in chapter 2, covers a large variety of iterative methods and exploits parallelism inherent to block Jacobi preconditioning. As a matter of fact, this approach is not entirely disconnected from a Domain Decomposition one, namely the additive Schwarz method [1]. On the other hand, we show in chapter 3 that, by associating block Jacobi preconditioning with the Finite

Element numerical background and the Object Oriented Programming (OOP) paradigm, the previously mentioned drawbacks of Domain Decomposition can be avoided. The advantages of such an approach are illustrated in the last chapter by means of the numerical simulation of an unsteady compressible flow.

2 Numerical Framework

When solving PDEs by means of the Finite Element Method (FEM), one has to deal with band matrix systems of algebraic equations. In the case of nonlinear equations, a linearization procedure is generally performed which results in a linear system $Ax = b$. Our parallel approach in solving such systems is numerically based upon a very basic device that leads to many iterative solvers. It consists in splitting the A matrix of the above linear system into two matrices. The matrix splitting we use is $A = B - (B - A)$, where the B matrix is constructed from the block diagonal submatrices $(A_{ii})_{i=1,\ldots,p}$ of A. This matrix is derived from the block Jacobi iteration :

$$B(x^{(k+1)} - x^{(k)}) = b - Ax^{(k)}$$

Obviously, this iterative process can also be thought of to be the Richardson method applied to the initial system preconditioned by B. Thus, since there is no guarantee that this simple iteration converges, the B matrix of the splitting may be used, in a broader context, as a preconditioning step in a more robust iterative method written as :

$$B\Delta x^{(k)} = \sum_j \alpha_j \psi_j^{(k')} = \Psi^{(k')} \quad with \quad k' \leq k+1$$

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$$

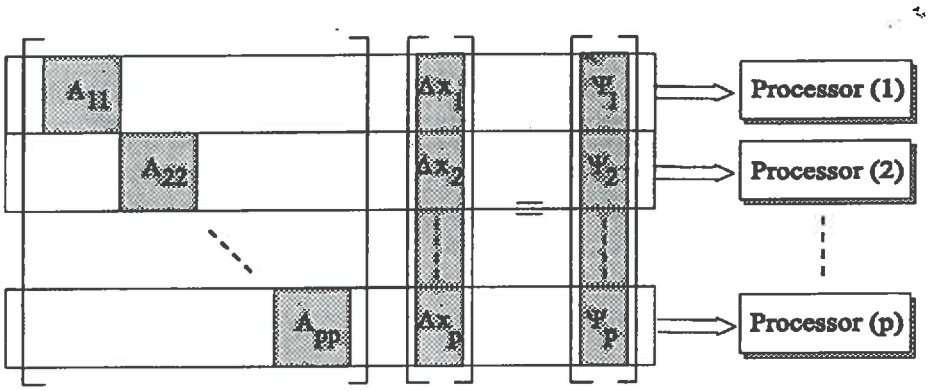


Fig. 1. Parallel implementation of a block Jacobi preconditioned system

Even in this case, the convergence properties strongly depend on the choice of the blocks. Usually, the most important motivation when preconditioning is that the preconditioned system is much more easy to solve than the initial one. In the present case, it mainly lies in the parallelism potential it confers to the global algorithm. In figure 1, wee can see that a straightforward manifestation of this potential is the ability of each processor to handle one subsystem independently of the others. As discussed hereafter, this property results in an ease of programming specially when using Object Oriented Programming paradigm.

3 Parallel Implementation

Our parallel approach practically leads, at every iteration (k), to the four following tasks:

I Send the preceding iteration solution to all the processors,
II Supply every processor with some matrix and vector blocks,
III Compute on every processor one of the sections of the new vector solution,
IV Construct the new vector solution.

In the case of Shared Memory (SM) architectures, tasks I and IV would disappear. As far as Distributed Memory (DM) machines are concerned, they result in message passing procedures which present no difficulty thanks to the communication tools generally available on such computers. Therefore, we focus our attention on tasks II and III.

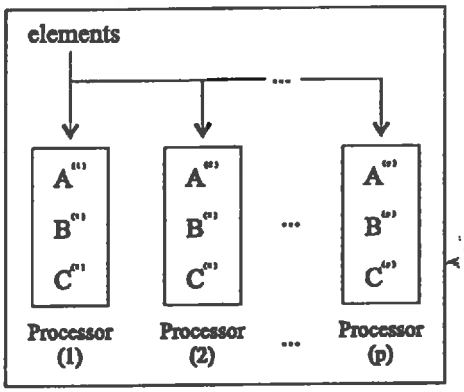


Fig. 2. Distribution of tasks between processors

Roughly speaking, the FEM may be decomposed into three steps : computation of the elementary matrices and vectors (step A), assembling of the global matrix system (step B) and solving (step C).

In this context, tasks II and III may be carried out in the following manner (fig. 2) :

1. Distribution of the elements between the processors,
2. Computation by every processor (i) of the adequate elementary matrices and vectors (step $A^{(i)}$),
3. Assembling on every processor (i) of the subsystem (step $B^{(i)}$),
4. Solving on every processor (i) of the assembled subsystem (step $C^{(i)}$)

The FEM programs we develop uses the C++ programming language because, thanks to the concept of class and the inheritance mechanism between these classes, this language supports OOP. This programming paradigm has initially been preferred, among other advantages, for it yields programs which are shorter and easier to understand, debug and maintain. For more details on these programs, one will refer to [2].In the present parallel computing framework, this choice happens to have another interest. Indeed, all our sequential Finite Element programs have the same structure made of three objects, the first two ones performing step (A) and the third one steps (B) and (C) :

"element" Object	Receive the number of the given element (i_elem) Compute the entities required to perform numerical integration (interpolation functions, jacobian matrix ...)
"elementary matrices and vectors" Object	Receive the number of the given element (i_elem) Construct the "element" object Compute the corresponding elementary matrix and vector
"assembling and solving" Object	For i_elem = 1 to n do Construct the "elementary matrices and vectors" object Insert the corresponding elementary matrices (resp. vectors) into the global matrix (resp. vector)

Thanks to the inheritance mechanism, the implementation of steps (2), (3) and (4) becomes straightforward. It is achieved by constructing from the corresponding class, one "assembling and solving" type object for every processor. Consequently, steps (A), (B) and (C) are automatically parallelized. Moreover, all the objects initially elaborated in a sequential framework, are also entirely available in the present parallel computing context. It also means that, when elaborating a new object, one has not to care about whether serial or parallel the computation will be. In the end, the parallel applications are characterized by coarse granularity, ease of maintaining good load balance and memory saving. Here lies the overwhelming reason of our determination in investigating block Jacobi

preconditioning (which is akin to additive Schwarz preconditioning), in spite of the convergence problems it may induce.

4 Numerical Results

As an application of this parallelization technique, we now deal with the simulation of a flow jet bursting out of a converging nozzle. The studied phenomenon, transonic and unsteady, is governed by the compressible Navier-Stokes equations. The Finite Element spacial discretization yields the following nonlinear matrix system :

$$M \frac{du}{dt} + K(u)\, u = f(u)$$

u corresponds to the vector of element nodal point unknowns. M designs the mass matrix, K the stiffness matrix. The f vector provides the forcing functions for the system. The time integration makes use of the fourth order Runge-Kutta method. Thus, the computation of one time step solution requires the solving of four linear systems of the form $Mx = b$. The use of block Jacobi preconditioning induces inner iterations per time step. Here, this supplement of computation can be avoided by making the M matrix diagonal. Given the number of unknowns N, the p blocks of the B matrix are chosen so that the first processor handles the first N/p unknowns, and so on, the last processor handling the $(N - N/p)$ ones. Practically, the diagonal of one block is stored in a vector and every processor holds a N/P long section of this vector in its local memory. Computation is

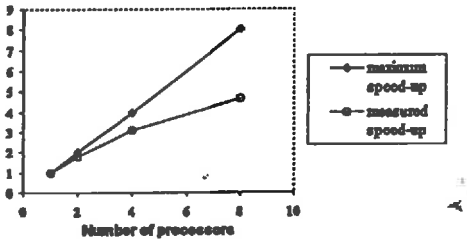


Fig. 3. Speed-up factor

performed with a MIMD-DM architecture, namely an Intel iPSC/860 (8 nodes) parallel computer. The solutions presented hereafter are obtained on a fully unstructured mesh (9387 triangular elements, N =18760 unknowns). Measured speed-up factors are shown in figure 3.
Our numerical results meet with experimental results obtained by A. Maksud [3]. Picture 4 illustrates the time evolution of the phenomenon. It shows the progressive development of the boundary layer within the nozzle. The velocity

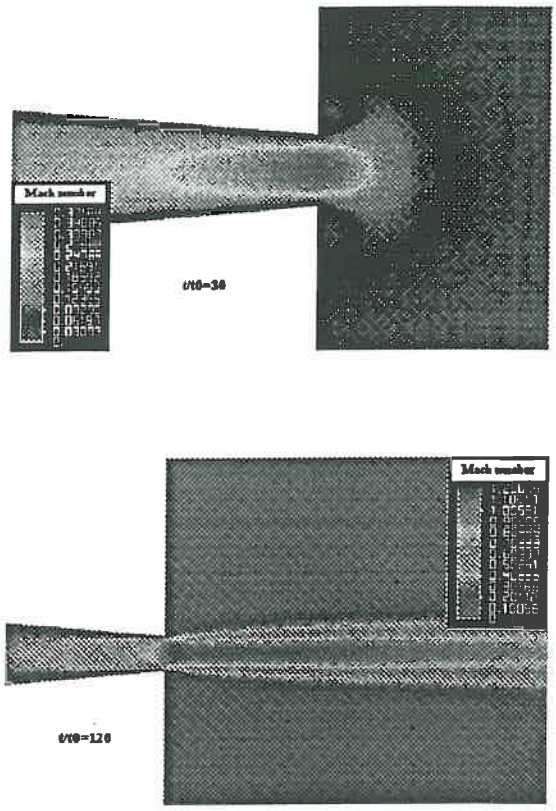


Fig. 3. Iso-Mach levels

References

1. Smith, B., Bjorstad, P., Gropp, W.: Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, 1996.
2. Chambarel, A., Onuphre, E.: Finite Element software based on Object Oriented Progamming, I.A.S.T.E.D. International Conference, Annecy, France, May 18-20 1994.
3. Maksud, A., Brocher, E., Chambarel, A., Cordonnier, P.: Shoch cell structure in a jet at critical pressure ratio, 20th International Symposium on Shock Waves, Caltech, July 23-28 1995.

This article was processed using the LaTeX macro package with LLNCS style

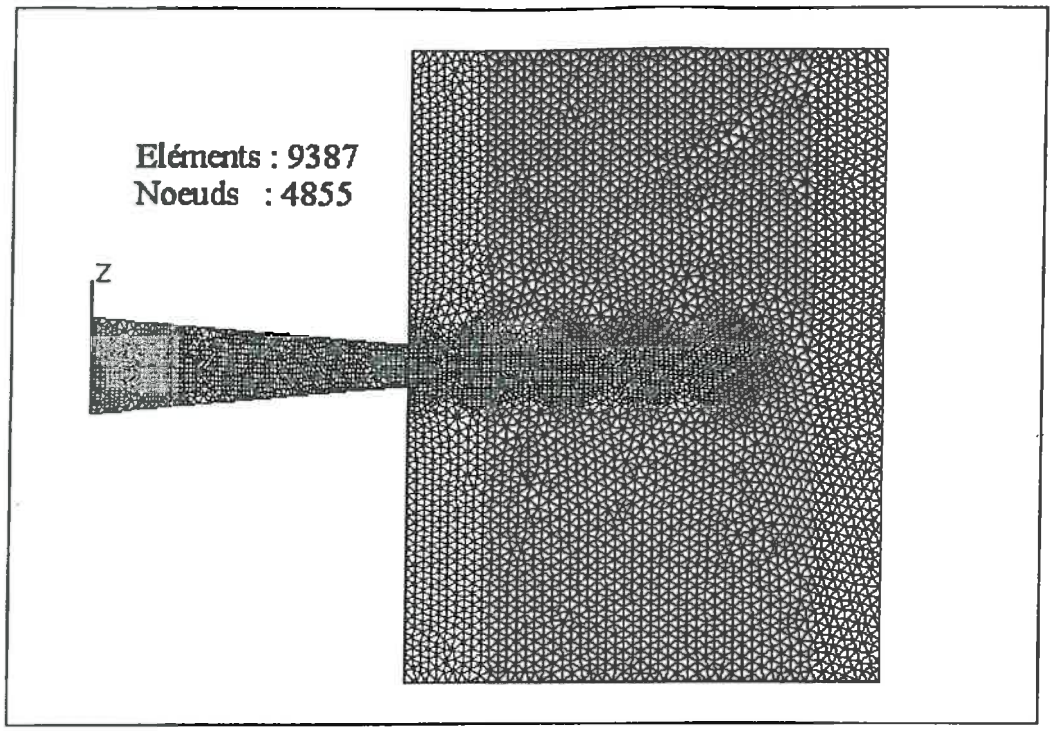


Figure 1

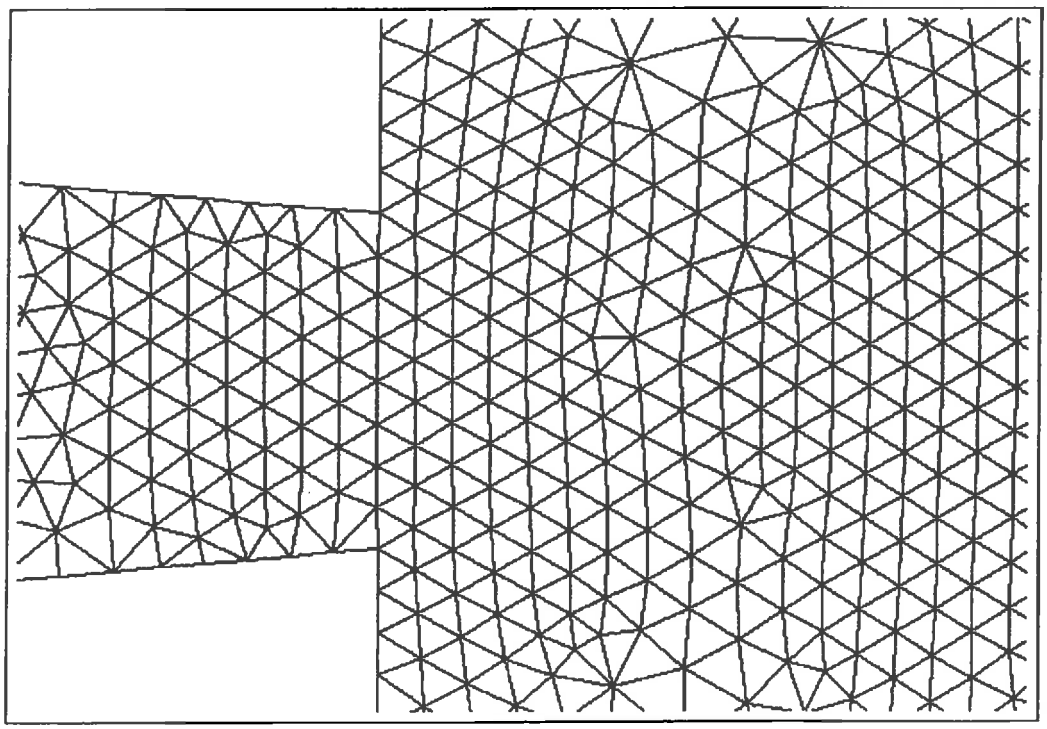


Figure 2

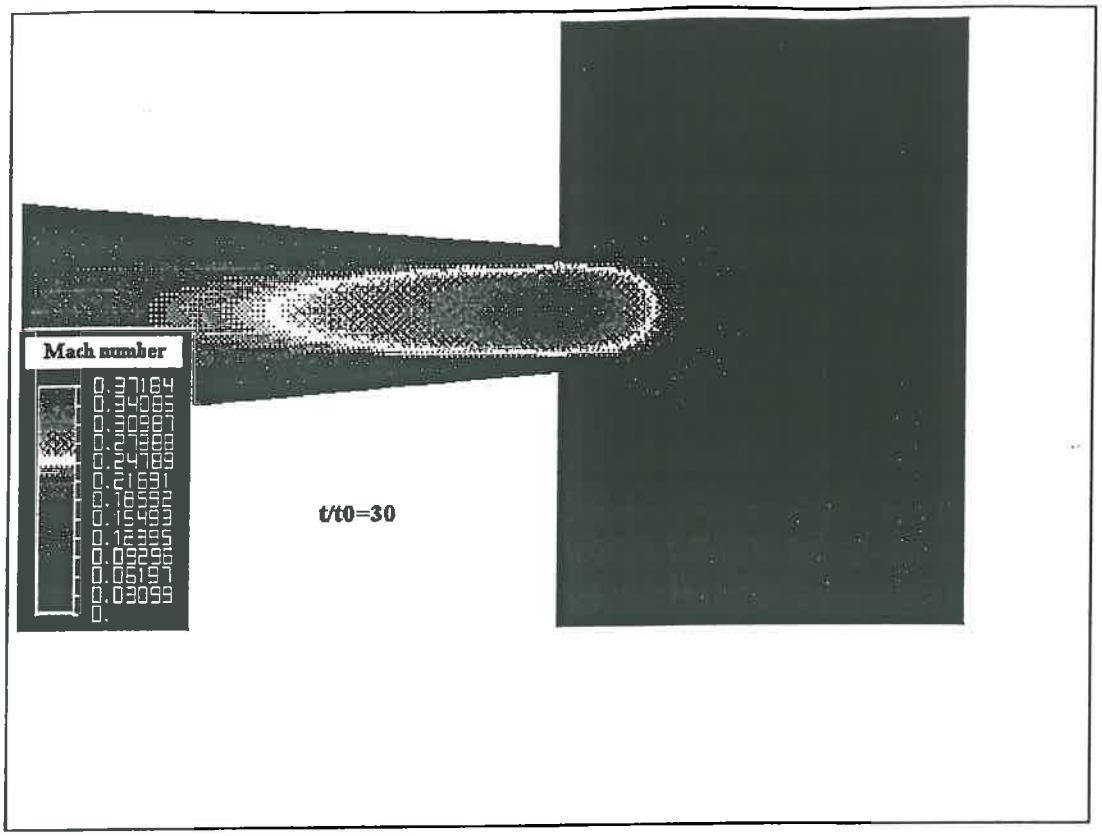


Figure 3

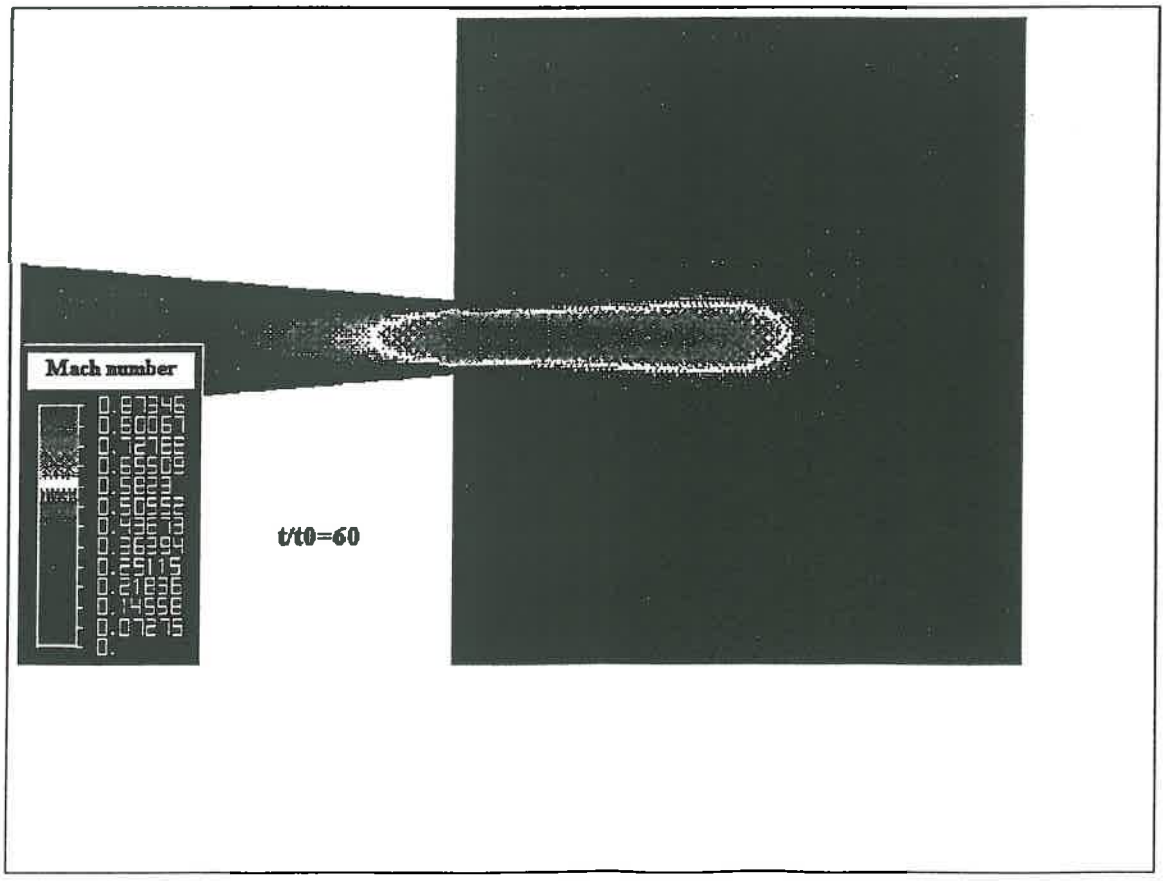


Figure 4

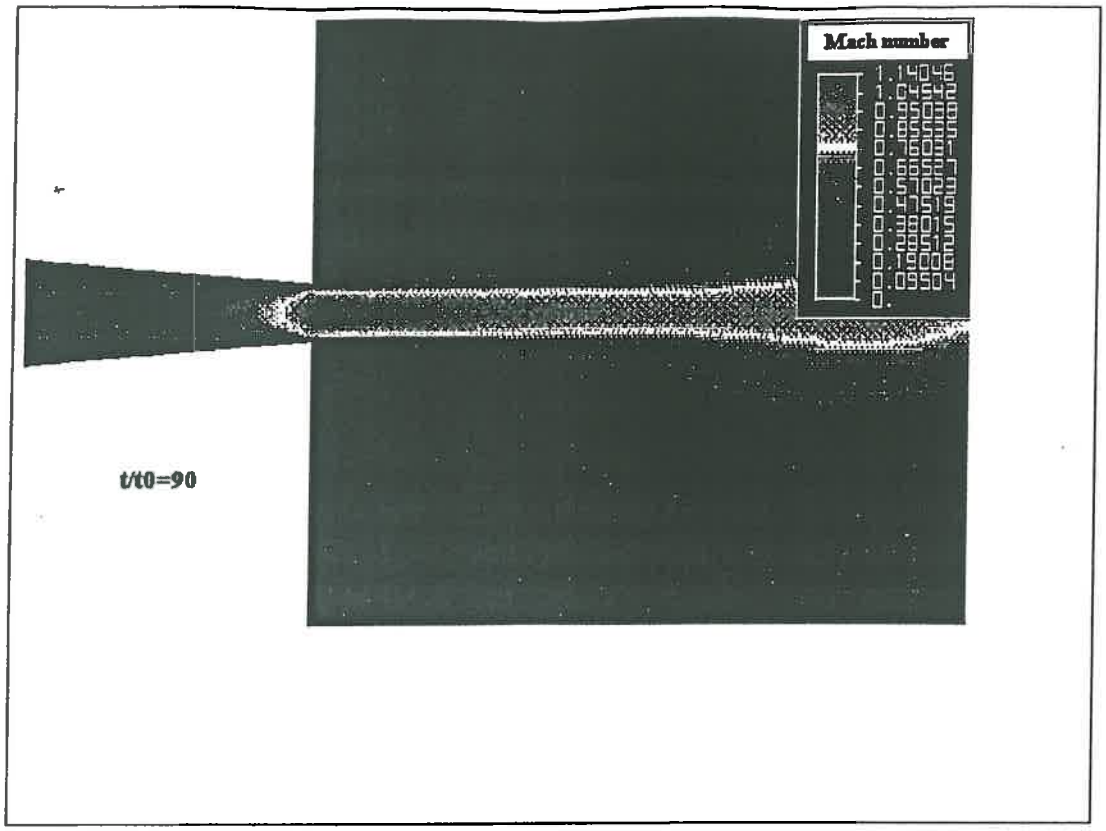


Figure 5

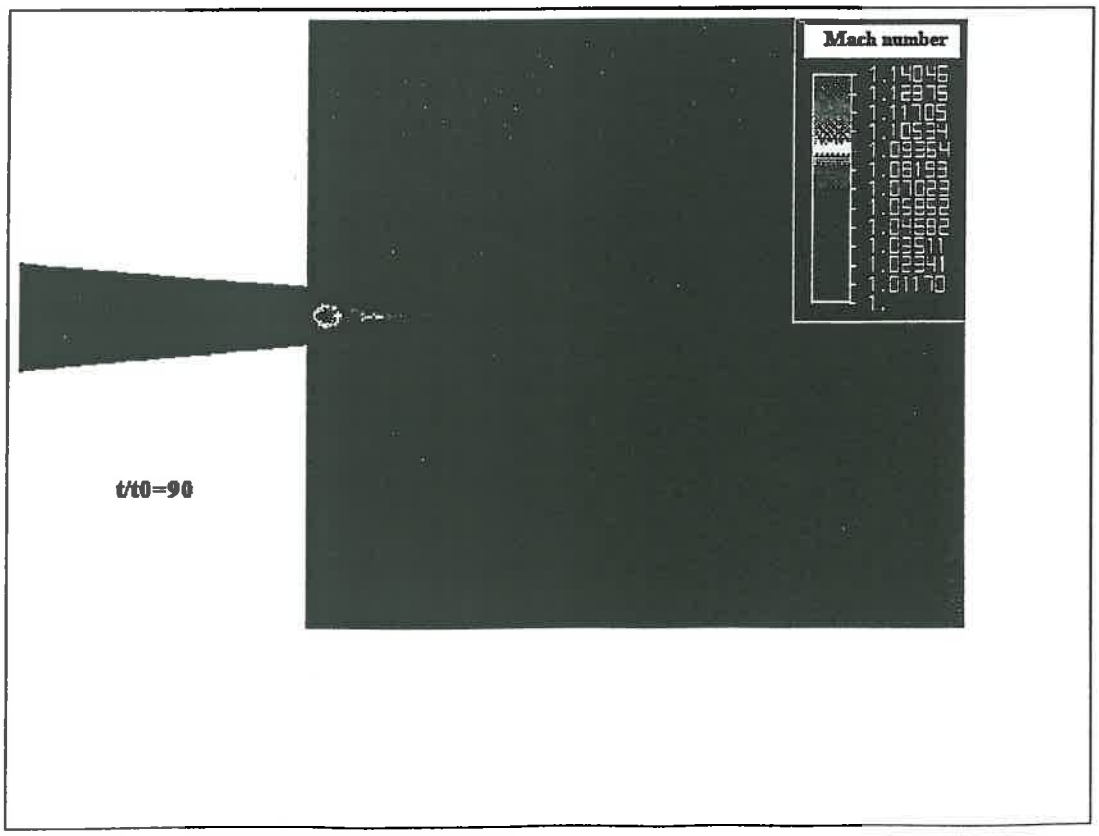


Figure 6

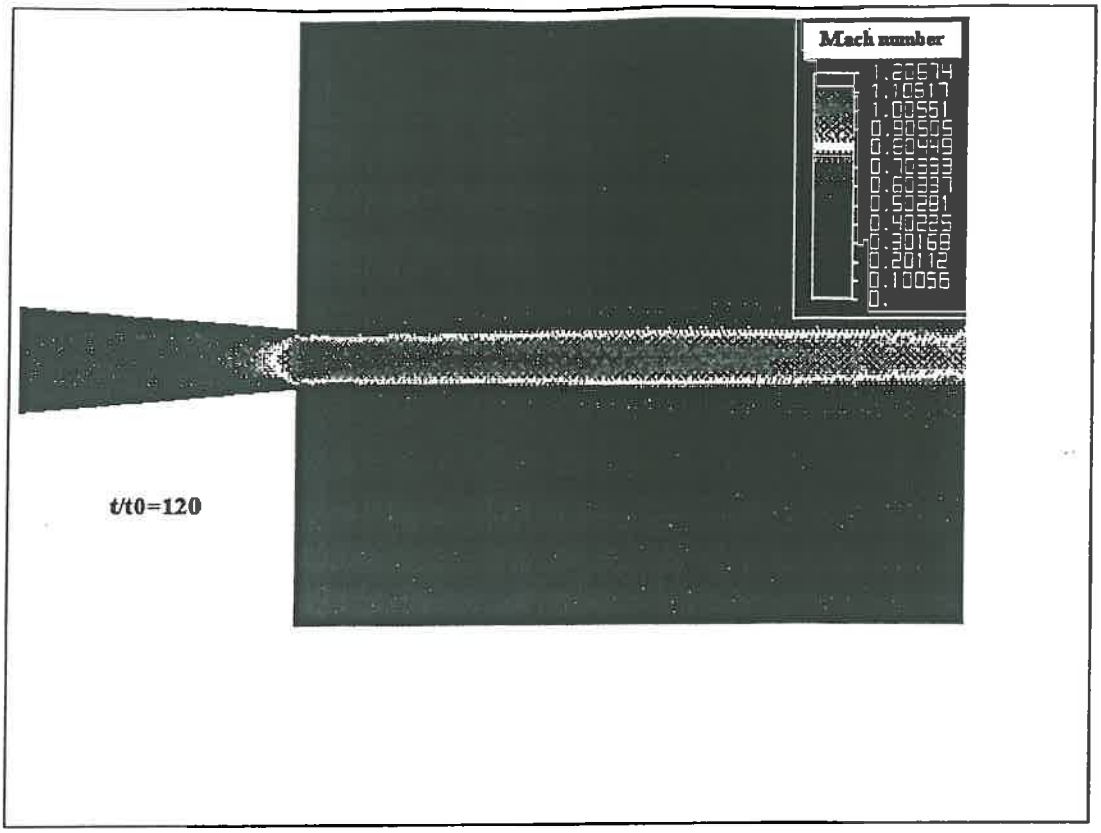


Figure 7

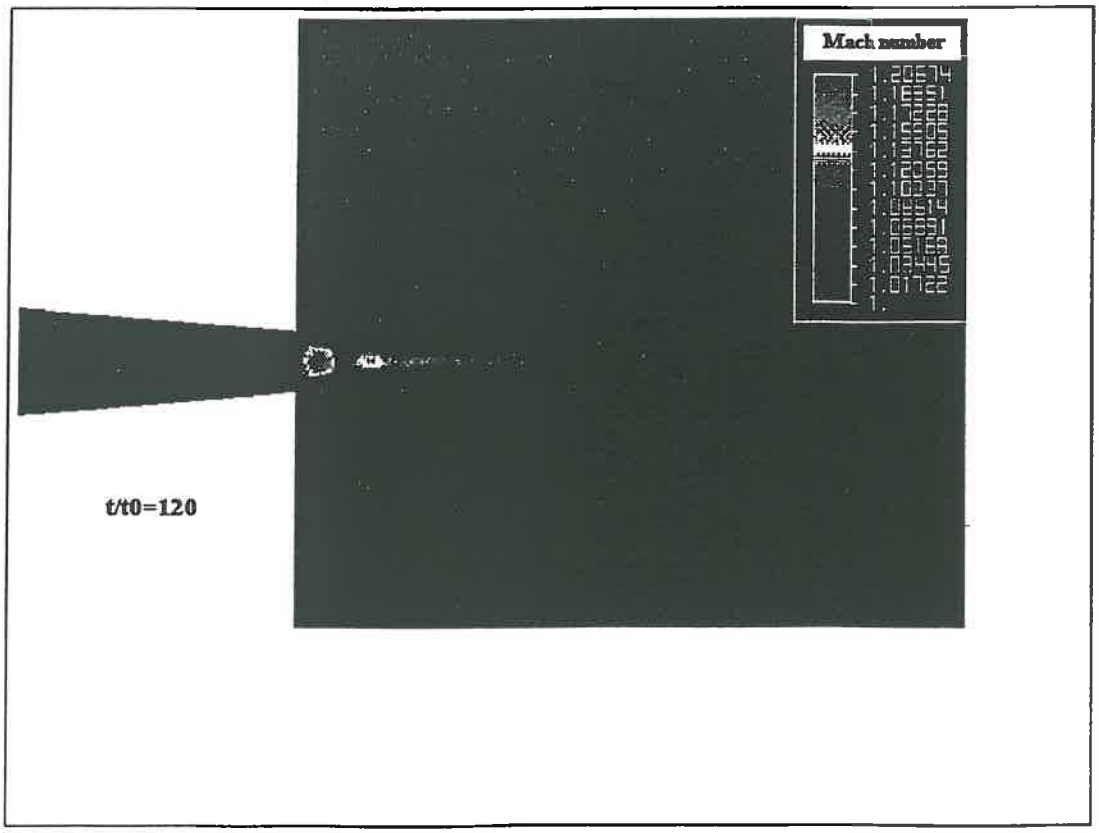
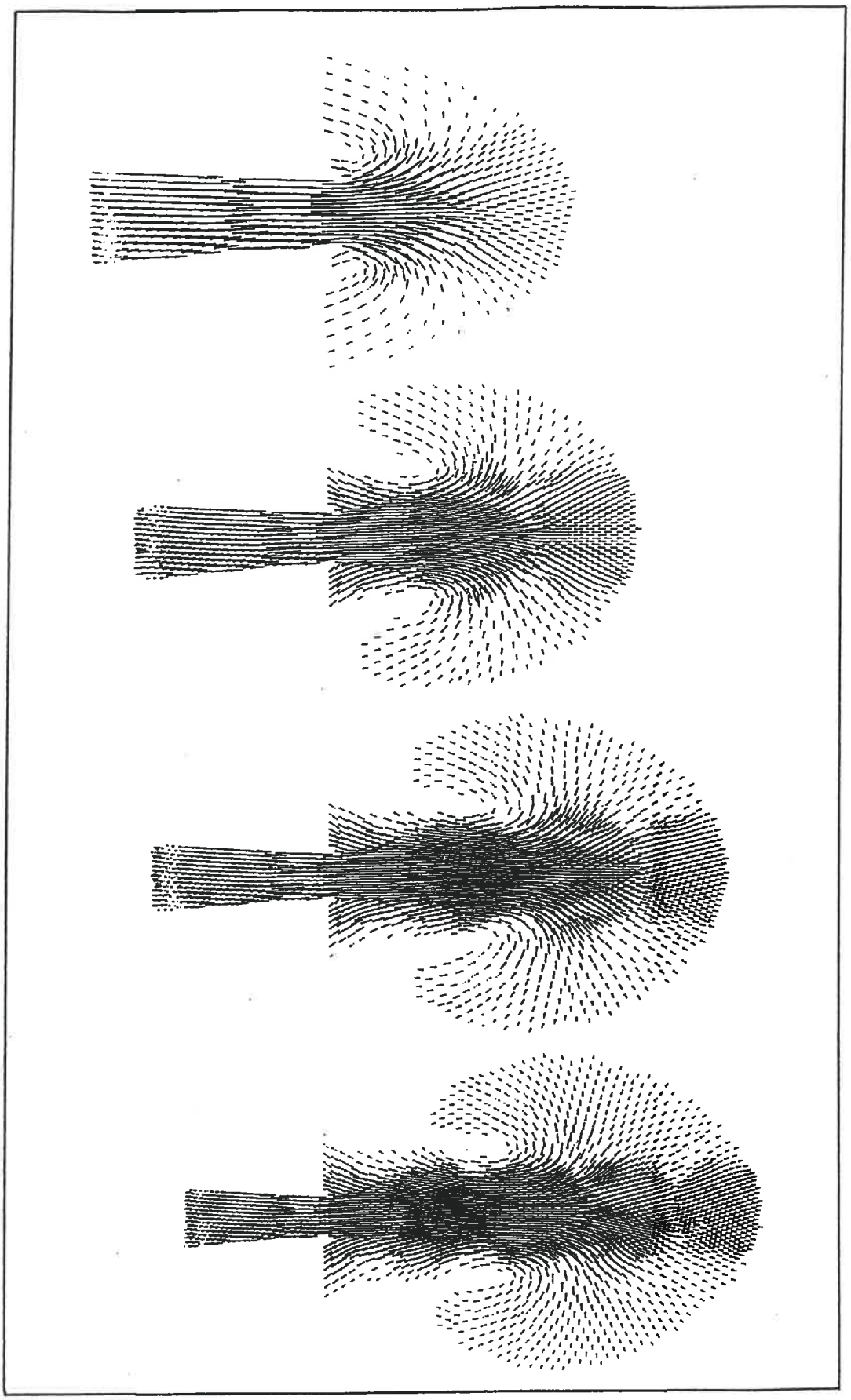
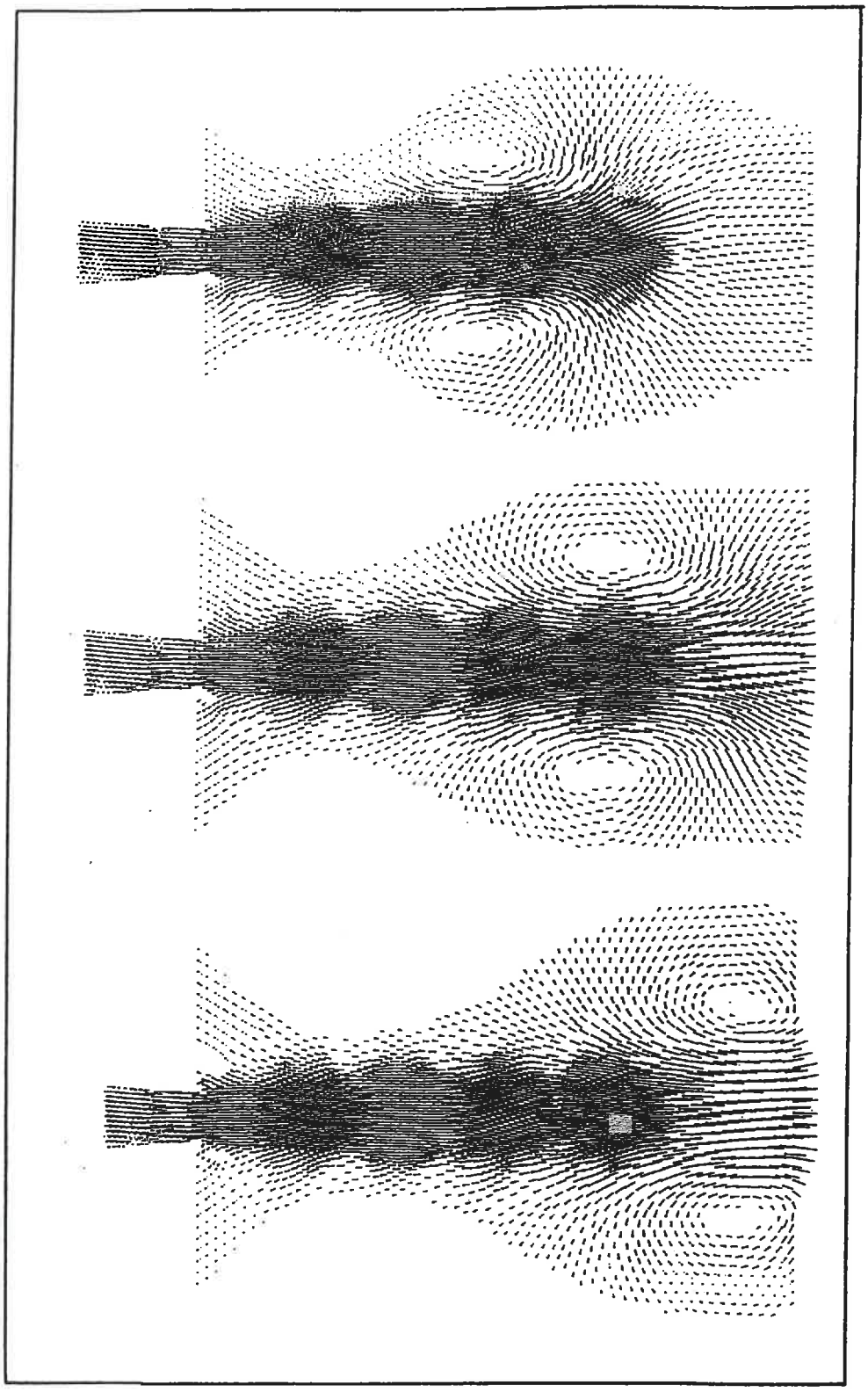


Figure 8





Bibliographie

[1] O. Axelsson, L. Yu, editors, *Preconditioned Conjugate Gradient Methods*, Berlin, 1990, Nijmegen 1989, Springer Verlag. Lecture Notes in Mathematics 1457

[2] E. Barragy, G. F. Carey, and R. Van De Geijn, *Performance and scalability of Finite Element for Distributed Parallel computation.* Journal of parallel and distributed computing 21, 202-212, 1994

[3] M.E. Braaten, W. Shyy, *Comparison of Iterative And Direct Solution Methods For Viscous Flow Calculations In Body-fitted Coordinates*, Int. J. Numer. Meth. Eng., vol. 6, pp. 325–349, 1996

[4] M.E. Braaten, S.V. Patankar, *A Block-Corrected Subdomain Solution Procedure For Recirculating Flow Calculations*, Numerical Heat Transfer, Part B, vol.15, pp. 1-20, 1989

[5] C. H. Bruneau, C. Jouron, An Efficient Scheme For Solving Steady Incompressible Navier-Stokes Equations, J. Comp. Physics, vol. 89, 389-413, 1990

[6] A. Chambarel, E. Onuphre, Finite Element Software Based On Object Programming, Proceedings of the Twelfth IASTED International Conference, Annecy, France, 1994

[7] G. Dhatt, G. Touzot, *Une Présentation De La Méthode Des Éléments Finis*, Collection Université De Compiègne, 2° édition, 1987

[8] M. S. Engelman, I. Hasbani, *Matrix-Free Solution Algorithms In A Finite Element Context*, Technical Report 88-1, Fluid Dynamics International, Evanston, Ill, 1988

[9] P. Fischer, *Spectral Element Solution Of The Navier-Stokes Equations On High Performance Distributed-Memory Parallel Processors*, PhD Thesis, Massachussets Institute of Technology, 1989

[10] P. Fischer, A. Patera, *Parallel simulation of viscous incompressible flows*, Annu. Rev. Fluid Mech., 26 : 483-527, 1994

[11] P. Fischer, A. T. Patera, *Parallel Spectral Element Solution Of The Stokes Problem*, J. Comp. Physics, 92, pp. 380-421, 1991

[12] R. Fletcher, *Conjugate Gradient Methods For Indefinite Systems*, Lectures Notes in Mathematics, 506, pp. 73-89, Springer-Verlag, Berlin – Heidelberg – New York, 1976

[13] M. J. Flynn, *Some Computer Organizations and their effectiveness*, I.E.E. Trans. On Computers, Vol. C-21, pp. 948-960, 1972

[14] R. W. Freund, N. M. Nachtigal, QMR : *A Quasi-Minimal Residual Method For Non-Hermitian Linear Systems*, Num. Math., 60, pp. 315-339, 1991

[15] U. Ghia, K. N. Ghia, C.T. Shin, *High-Re Solutions For Incompressible Flow Using The Navier-Stokes Equations And A Multigrid Method*, J. Comp. Physics, vol. 48, pp. 387-411, 1982

[16] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Second edition, The Johns Hopkins University Press, Baltimore and London, 1989

[17] M. R. Hestenes, E. Stiefel, *Methods Of Conjugate Gradients For Solving Linear Systems*, J. Res. Natl. Bur. Stand, 49, pp. 409-436, 1954

[18] P. Hood, *A Finite Element Solution Of The Navier-Stokes Equations For Incompressible Contained Flow*, M. Sc. Thesis, University Of Wales, Swansea, 1970

[19] T.J.R. Hughes, W. K. Liu, A. Brooks, *Finite Element Analysis Of Incompressible Viscous Flows By The Penalty Function Formulation*, Journal Of Computational Physics, 30, pp. 1-60, 1979

[20] C.P. Jackson and P.C. Robinson. *A numerical study of various algorithms related to the preconditioned Conjugate Gradient method. Int. J. for num.* Meth. In Eng., 21:1315-1338, 1985

[21] K.C. Jea and D. M. Young. *Generaliezd conjugate-gradient acceleration of nonsymetrizable iterative methods.* Lin. Algebra Appl., 34:159-194, 1980

[22] A. A. Johnson, T. E. Tezduyar, *Parallel Computation Of Incompressible Flows With Complex Geometries*, Int. J. Num. Methods Fluids, vol. 24, pp. 1321-1340, 1997

[23] M. Kaiho, M. Ikegawa, C. Kato, *Parallel Overlapping Scheme For Viscous Incompressible Flows*, Int. J. Num. Methods Fluids, vol. 24, pp. 1341-1352, 1997

[24] P. M. Kogge, *The Architecture of Pipelined Computers*, Mc. Graw-Hill Ed., 1981

[25] C. Lanczos, *Solution Of Systems Of Linear Equations By Minimized Iterations*, J. Res. Natl. Bur. Stand, 49, pp. 33-53, 1952

[26] J.T. Oden, *The Finite Element Method In Fluid Mechanics*, Lecture For NATO Advanced Study Institute On Finite Element Methods In Continuum Mechanics, lisbon, 1971

[27] E. Onuphre, C. Chambarel, Parallel Computation Of An Unsteady Compressible Flow, Lectures Notes in Computer Science, Springer-Verlag, à paraître.

[28] C. C. Paige, M. A. Saunders, *Solution Of Sparse Indefinite Systems Of Linear Equations*, SIAM J. Numer. Anal., 12, pp. 617-629, 1975

[29] C. C. Paige, M. A. Saunders, LSQR *: An Algorithm For Sparse Linear Equations And Sparse Least Squares*, ACM Trans. Math. Soft., 8, pp. 43-71, 1982

[30] B. Ramaswamy, *Theory And Implementation Of A Semi-implicit Finite Element Method For Viscous Incompressible Flow*, Computers Fluids, vol. 22, No 6, pp. 725-747, 1993

[31] Y. Saad, M. H. Schultz, *GMRES: A Generalized Minimal Residual Algorithm For Solving Non-Symmetric Linear Systems*, SIAM J. Sci. Statist. Comput., 7, pp 856-869, 1986

[32] G. L. G. Sleijpen, H. A. van der Vorst, D. R. Fokkema, *Bi-CGSTAB(l) And Other Hybrid Bi-CG Methods*, Numerical Algorithms, 7, pp. 75-109, 1994

[33] B. Smith, P. Bjorstad et W. Gropp, *Domain Decomposition : Parallel Multilevel Methods For Elliptic Partial Differential Equations*, Cambridge University Press, 1996

[34] P. Sonneveld, *CGS : A Fast Lanczos-Type Solver For Nonsymmetric Linear Systems*, SIAM J. Sci. Statist. Comput., 10, pp. 36-52, 1989

[35] C. Taylor, P. Hood, *A Numerical Solution Of The Navier-Stokes Equations Using The Finite Element Technique*, Computers & Fluids, 1, pp. 73-100, Pergamon Press, 1973

[36] J. Thomas, R. Hugues, Wing Kam Liu and Alec Brooks, *Finite Element Analysis of Incompressible Viscous Flows by the Penalty Function Formulation*, Journal of Computational Physics 30 : 1-60, 1979

[37] E. E. Tyrtyshnikov, *New approaches to deriving parallel algorithms*, Parallel Computing 15 261-265, 1990

[38] H. A. van der Vorst, Bi-CGSTAB : *A Fast And Smoothly Converging Variant Of Bi-CG For The Solution Of Non-Symmetric Linear Systems*, SIAM J. Sci. Statist. Comput., 13, pp 631-644

[39] S.P. Vanka, *Fully Coupled Calculation Of Fluid Flows With Limitated Use Of Computation Storage*, Argonne National Laboratory Report ANL-83-87, Argonne, Ill., 1983

[40] S.P. Vanka, G.K. Leaf, *Fully Coupled Solution Of Pressure-Linked Fluid Flow Equations*, Argonne National Laboratory Rept. ANL-83-73, Argonne, Ill., 1983

[41] R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs N. J., 1962

[42] O. Widlung, *A Lanczos Method For A Class Of Non-Symmetric Systems Of Linear Equations*, SIAM J. Numer. Anal., 15, pp. 801-812, 1978

[43] S.O. Wille, *A Structured Tri-Tree Search Method For Generation Of Optimal Unstructured Finite Element Grids In Two And Three Dimensions*, Int. J. Num. Methods Fluids, 14, 861-881, 1992

[44] K. G. Wilson, *Grand Challenges to Computational Science*, Future Generation Computer Systems, 5, Nos 2-3, September 1989

[45] A. Yeckel, J. W. Smith, J. J. Derby, *Parallel Finite Element Calculation Of Flow In A Three Dimensional Lid-Driven Cavity Using The CM-5 and T3D*, Int. J. Num. Methods Fluids, 24, pp. 1449-1461, 1997

[46] O.C. Zienkiewicz, *The Finite Element Method In Engineering Science*, Mc Graw Hill, New-York, 3rd edition, 1977