Résolution de problème à deux dimensions de mécanique des fluides sur le multiprocesseur DIRMU

de

B. Couapel*, G. Scheuerer**, J. Volkert***

Juillet1987

* Institut de Formation Supérieure en
Informatique et Communication
Université de Rennes I
Avenue du Général Leclerc
F—35042 Rennes

** Lehrstuhl für Strömungsmechanik
Universität Erlanqen-Nürnberg
Egerlandstr. 13
D—8520 Erlangen

*** Institut für Mathematische Maschinen
und Datenverarbeitung III
Universität Erlangen-NUrnberq
Martensstr. 3
D—8520 Erlanqen

Résumé

Ce rapport décrit l'implémentation d'une méthode utilisant des volumes finis pour la résolution des équations de Navier—Stokes sur le système multiprocesseur DIRMU. L'exemple testé est le courant laminaire incompressible sur une marche descendante, afin de le comparer & sa résolution sur un ordinateur séquentiel.
La méthode de calcul utilise l'algorithme SIMPLE dans lequel on travaille sur des systèmes d'équations linéaires itératifs pour les deux composantes de la vitesse et une correction des pressions. L'algorithme s'arrête lorsque les équations de masse et de quantité de mouvement sont vérifiées simultanément.

La configuration du calculateur est choisie de telle sorte que la zone calculée soit divisée en plusieurs bandes. Chaque processeur travaille sur une bande. Théoriquement, avec une telle organisation, on doit obtenir un speedup proportionnel au nombre de processeurs utilisés. Les mesures faites dans le cadre de ce travail montrent que des valeurs proches de l'optimal sont obtenues pour des maillages fins. Les pertes dues aux communications entre processeurs vont de 5 à 20 % selon le maillage et le nombre de processeurs. Les améliorations de la méthode de parallélisation doivent surtout porter sur l'algorithme de résolution des systèmes d'équations linéaires. La méthode 'ligne par ligne' est utilisée dans ce travail. Ultérieurement, la recherche sera plus axée sur la parallélisation d'algorithmes plus efficaces.

PREFACE

Ce travail a été réalisé dans le cadre du domaine de recherche défini par la fondation Volkswagen "Solutions d'équations de base de la mécanique des fluides adaptées aux architectures modernes des calculateurs".

Les auteurs remercient le Prof.Dr.W.Händler, Prof.Dr.A.Bode et Prof.Dr.F.Durst pour leur soutien et l'intérêt constant qu'ils ont porté & ce travail. Mme Dipl.Ing.M.Barcus, Mr Dr.I.1.Peric ainsi que Prof.Dr.J.H.Ferziger sont aussi remerciés pour leurs nombreuses discussions et leur aide à la finition de ce travail.

1. Introduction

1.1 Présentation du problème

Les méthodes modernes de calcul de mécanique des fluides s'appuient sur la résolution des équations de Navier-Stokes. Dans le cas laminaire, i.e. avec des nombres de Reynolds petits, celles-ci sont résolues de façon directe.
Pour les courants & turbulences intéressants sur le plan technique, on observe en général des formes de ces équations en relation avec un modèle de turbulence (Bradshaw 1981).
Pour les deux types de courants, il faut intégrer un système d'équations différentielles partielles du deuxième ordre, non linéaires et couplées entre elles.
Pour les courants à recirculation, cette solution consomme beaucoup de temps de calcul et de place mémoire, même pour des géométries à deux dimensions.
Pour les courants à trois dimensions qui sont intéressants à étudier sur le plan pratique, ces besoins sont encore beaucoup plus importants.

Les calculs de configurations techniques complètes telles que avions ou automobiles nécessitent des ordinateurs d'une capacité mémoire d'un milliard de mots et des performances de l'ordre de Giga-Flops.

Ces besoins importants entraînent que, notamment aux U.S.A., la mécanique des fluides ait un rôle de plus en plus important dans le développement de calculateurs & hautes performances (Kutler 1985).

Jusqu'à maintenant il s'agissait de calculateurs vectoriels à processeurs puissants, mais moins de parallélisme. Ces dernières années, on a découvert que des progrès importants ne pouvaient plus être réalisés avec de telles architectures sur le plan de la vitesse de calcul et de la capacité mémoire.

C'est pourquoi on s'oriente vers des calculateurs à haut degré de parallélisme i.e. multiprocesseurs dans lesquels plusieurs processeurs travaillent simultanément.

Théoriquement, on peut obtenir des gains en temps de calcul et en place mémoire proportionnels au nombre de processeurs utilisés.

Ce concept présente les avantages d'une grande flexibilité et de possibilités d'extension. Les multiprocesseurs offrent le moyen de résoudre d'importants problèmes de la mécanique des fluides qui sont impossibles & travailler sur les machines disponibles actuellement.

Le développement des multiprocesseurs n'ayant débuté que depuis quelques années, il n'y a encore que peu de machines comprenant un grand nombre de processeurs.
Cela entraîne qu'il n'y ait encore que peu d'utilisations dans le domaine de la mécanique des fluides.

Le paragraphe suivant traitera de l'état de la recherche dans ce domaine.

1.2 Etat de la recherche

Depuis quelques années, les multiprocesseurs comprenant un nombre restreint de processeurs sont techniquement réalisables et disponibles commercialement, Les systèmes ont actuellement en général 2 à 4 processeurs.
De tels multiprocesseurs sont utilisés dans le domaine commercial pour obtenir une amélioration du rendement (ex. I.B.M. 3801). La tendance est aussi aux multiprocesseurs dans le domaine des calculateurs scientifiques à hautes performances (ex CRAY-XMP, CRAY 2, ETA 10).

La plupart des machine pourvues d'un plus grand nombre de processeurs qui sont commercialisées ont des liaisons relativement faibles entre les différents éléments.
Un exemple est le calculateur ALLIANT qui comprend 8 processeurs sur un même bus.
Les différentes réalisations de 1'HYPERCUBE sont une exception. Elles sont disponibles depuis peu sur le marché et comprennent des systèmes de liaisons pour 2~ processeurs.
Les multiprocesseurs à système de liaison d'un autre ordre de grandeur (couplage par mémoires multiports, réseaux à plusieurs niveau) sont encore & l'état de recherche.

A l'Institut pour Machines Mathématiques et Informatique (IKMD> de l'Université d'Erlangen-Nürnberg on fait des recherches depuis le milieu des années 1970 sur des systèmes couplés par mémoires multiports.
Actuellement, on utilise le multiprocesseur modulaire DIRMU 25 (Händler 1985>. A partir d'éléments de base comprenant un processeur, un coprocesseur et de la mémoire, on peut facilement configurer différentes architectures jusqu'à 26 processeurs (ex anneaux, champs, hypercubes etc). Ce système sert & développer et mesurer les performances de structures et de programmes d'application parallèles.

Sur ce système quelques problèmes de mécanique des fluides ont déjà été travaillés avec succès.
Par exemple la résolution de l'équation de Poisson au moyen de différences finies et la méthode de Gau~3-Seidel comme algorithme de résolution, ainsi que la résolution des équations de Navier-Stokes avec une méthode comprenant plusieurs maillages (Multigrid) (Geus 1985).
Dans les deux cas, le gain en temps mesuré est presque linéaire par rapport au nombre de processeurs utilisés.

Dans le cadre du projet SUPRENUM du ministère fédéral pour la recherche et la technologie, Thole (1985) a travaillé l'équation de Poisson à trois dimensions sur un calculateur Hypercube. Les processeurs étaient organisés sous la forme d'un champ pouvant comprendre jusqu'à 32 éléments.
La résolution à été réalisée au moyen de multigrid avec des cycles V.
Le travail sur un maillage grossier et les temps de communication importants ont donné comme résultat un gain de temps de l'ordre de 50%.

1.3 But du travail

Dans ce travail, nous décrirons une application sur DIRMU 25 concernant la résolution des équations de Navier—Stokes avec une méthode à volumes finis. Comme exemple test, nous prendrons le cas d'un flot à deux dimensions, laminaire et incompressible sur une marche descendante. Les résultats de la version parallèle du programme doivent être comparés avec ceux de la version séquentielle. Le but de ce travail est de mesurer quels gains en temps de calcul sont obtenus lorsqu'on calcule le flot sur un multiprocesseur. En même temps on montrera les points faibles et problèmes dans l'implémentation parallèle ainsi que les propositions de solution.

1.4 Contenu du rapport

Le chapitre 2 décrira la géométrie du problème traitée et le système d'équations utilisé pour sa solution ainsi que les conditions de départ.

Le chapitre 3 consiste en une discussion sur les méthodes de calcul au moyen de volumes finis.

Le chapitre 4 est consacré au multiprocesseur DIRMU. Les principes de base ainsi que les composants matériels et logiciels de ce système sont expliqués.

Le chapitre 5 présente l'implémentation parallèle de l'application numérique. Les paragraphes 5.1 et 5.2 décrivent la configuration du sytème choisie et la différentiation des différents type de variables. Le déroulement du programme parallèle est décrit dans le paragraphe 5.3. Le paragraphe 5.4 traite des aspects importants de la synchronisation entre les processeurs du calculateur.

Les résultats se trouvent chapitre 6 sous forme de tableaux et diagrammes. Il sont discutés et critiqués.

Les conclusions se trouvent dans le chapitre 7 qui termine ce rapport.

2. Modèle mathématique

2.1 Géométrie et système de coordonnées

La figure 2.1 présente la géométrie et le système de coordonnées de l'exemple-test étudié: le courant à deux dimensions, laminaire sur une marche descendante. La zone d'intégration a une longueur L=0,25 ni; la hauteur du canal à la sortie est H=0,05 m. La hauteur de la marche est h=H/2.

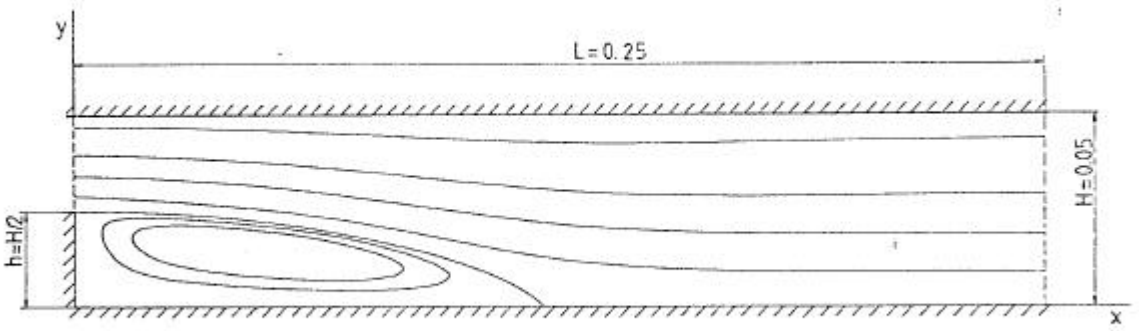

Fig. 2.1 Géométrie de l'exemple test et système de coordonnées.

2.2 Equations de Navier-Stokes

Le courant présenté figure 2.1 peut se décrire par les conservations physiques de la masse et de la quantité de mouvement. Si l'on considère un courant stationnaire, à deux dimensions, laminaire et de densité constante, les équations suivantes pour un fluide Newtonien (Bird **,** 1960) sont valables:

Conservation de la masse:

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \qquad (2.2\text{-}1)$$

Conservation de la quantité de mouvement en x

$$\frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left(\mu \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right) \qquad (2.2\text{-}2)$$

Conservation de la quantité de mouvement en y

$$\frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left(\mu \frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu \frac{\partial v}{\partial y}\right) \qquad (2.2\text{-}3)$$

Dans ces équations, U et V sont les vitesses dans les directions x et y, P est la pression statique. Les grandeurs p et ja sont la densité et la viscosité dynamique du fluide.

Les équations (2.2-1) à (2.2-3) sont appelées équations de Navier-Stokes. Elles constituent un système d'équations différentielles partielles elliptiques couplées. Pour obtenir une solution unique, il est nécessaire de définir des conditions de départ à la périphérie de la zone d'intégration. Celles-ci sont données dans le paragraphe suivant.

2.3 Conditions de départ

Les côtés de la zone d'intégration décrite fig. 2.1 sont formés de parois, d'une zone de flot d'entrée et d'une zone de flot de sortie. Les conditions de départ sont définies comme suit:

Au niveau des parois, les vitesses sont nulles soit:

U = V = 0 (2. 3-1)

A l'entrée du flot, on a un courant qui a un profil de vitesse

parabolique suivant la formule:

$$u = 4\, u_{max} \cdot \frac{\eta}{h}\left(1 - \frac{\eta}{h}\right) \qquad (2.3\text{-}2)$$

n est la valeur du point le plus haut de la marche. La vitesse normale V est nulle à l'entrée de la zone d'intégration. Avec les valeurs choisies Umax = 1,5 m/s, p = 1 kg/m3 et ì = 0,00375 Pas, nous obtenons un nombre de Reynolds égal à 100 par la formule:

$$Re = \frac{u_{max} \cdot h \cdot \rho}{\mu} \qquad (2.3\text{-}3)$$

Du côté du flot de sortie qui doit se trouver suffisamment éloigné de la zone de courant de retour, on supposera négligeables les variations des vitesses U et V dans la direction x.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = 0 \qquad (2.3\text{-}3)$$

A l'aide des relations données, les équations de Navier-Stokes présentée dans le paragraphe 2.2 sont définies clairement.

3. Méthode de résolution numérique

La résolution des équations différentielles de Navier-Stokes est faite au moyen de la méthode de conservation de volumes finis (Finite-Volumen-Methode CAST) décrite par Peric et Scheuerer (1987). Les éléments importants de cette méthode sont les suivants:

- Discrétisation de la zone de calcul (Distribution des volumes de contrôle)
- Discrétisation des équations de conservation
- Algorithmes de résolution des systèmes correspondants d'équations différentielles linéaires.

Ceux-ci seront discutées dans le paragraphe suivant.

3.1 Distribution des volumes de contrôle

Dans la méthode des volumes finis, la zone d'intégration est divisée en un nombre fini de volumes de contrôle. Les points du maillage se trouvent tous au centre de gravité des volumes de contrôle (fig. 3.1). Jusqu'à maintenant, les équations de Navier-Stokes ont été résolues presque exclusivement (Patankar, 1980) au moyen de distribution décalée des volumes de contrôle (staggered grids). Cette méthode a aussi été utilisée pour ce travail.

Les vitesses U et V sont présentées f ig.3.1. Elles sont décalées par rapport à la pression P dans la zone d'intégration.
Les valeurs des vitesses U sont à la limite du volume de contrôle avec x=const, et les vitesses V à la limite correspondante avec y=const.
Cette distribution présente l'avantage d'éviter des problèmes d'oscillations du champ des pressions. Ce point est discuté en détail par Patankar (1980).

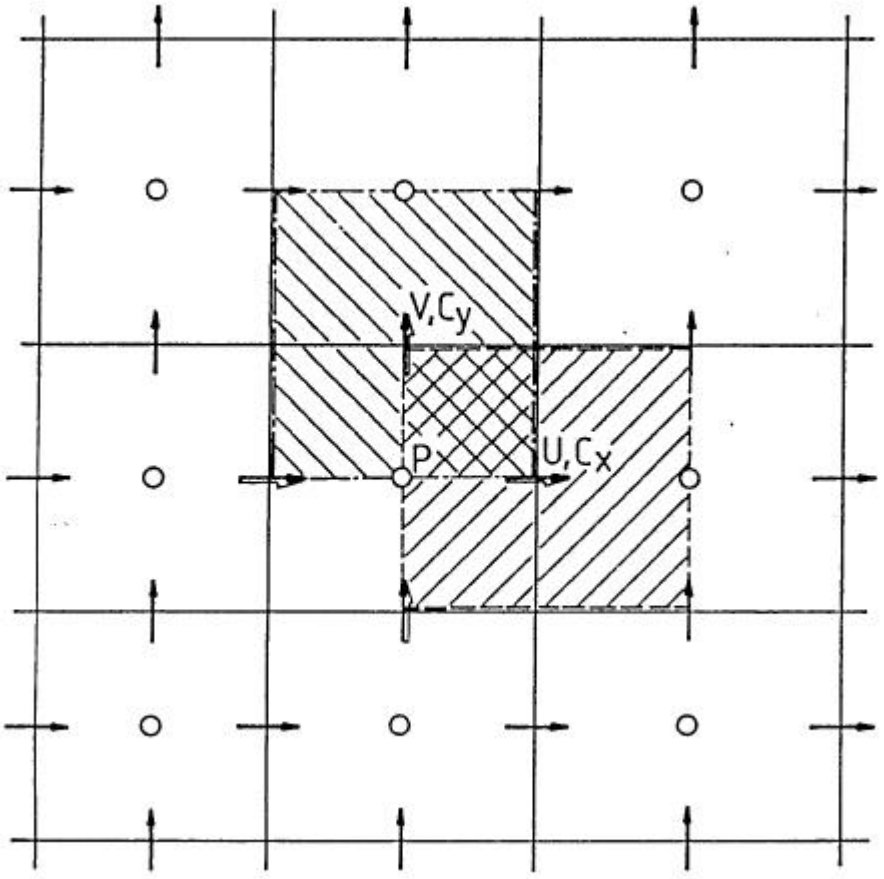

Fig. 3.1: Distribution des volumes de contrôle avec maillages décalés.

3.2 Discrétisation en volumes finis

La discrétisation. est faite selon l'équation de transport généralisée suivante:

$$\frac{\partial(\rho u\phi)}{\partial x} + \frac{\partial(\rho v\phi)}{\partial y} = \frac{\partial}{\partial x}\left(\Gamma_\phi \frac{\partial\phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\Gamma_\phi \frac{\partial\phi}{\partial y}\right) + S_\phi \qquad (3.2\text{-}1)$$

Dans le cas de l'équation de U, ÖU et SÖ=-äP/äx. L'équation de V peut être construite de façon analogue. L'équation (3.2-1) sera formellement intégrée sur un volume de contrôle, dans lequel les intégrales du côté gauche peuvent être réécrites en intégrales de surfaces à l'aide du théorème de Gauß. De la figure 3.2, il suit:

$$\int_s^n \left[\left(\rho u\phi - \Gamma_\phi \frac{\partial\phi}{\partial x}\right)_e - \left(\rho u\phi - \Gamma_\phi \frac{\partial\phi}{\partial x}\right)_w\right] dy +$$

$$\int_w^e \left[\left(\rho v\phi - \Gamma_\phi \frac{\partial\phi}{\partial}\right)_n - \left(\rho v\phi - \Gamma_\phi \frac{\partial\phi}{\partial y}\right)_s\right] dx = \int_{\Delta V} S_\phi \, dV \qquad (3.2\text{-}2)$$

Cette équation intégro-différentielle est équivalente à l'équation de sortie (3.2-1). Elle décrit l'équilibre entre les flux convectif et diffusifs entrant et sortant des surfaces du volume de contrôle, et les termes—sources qui se trouvent dans le volume de contrôle. Dans la méthode des volumes finis, l'équation (3.2-2) sera approximée à la place de, (3.2-1). Ceci donne une discrétisation conservative qui ne doit présenter aucune perte numérique de masse et de quantité de mouvement.

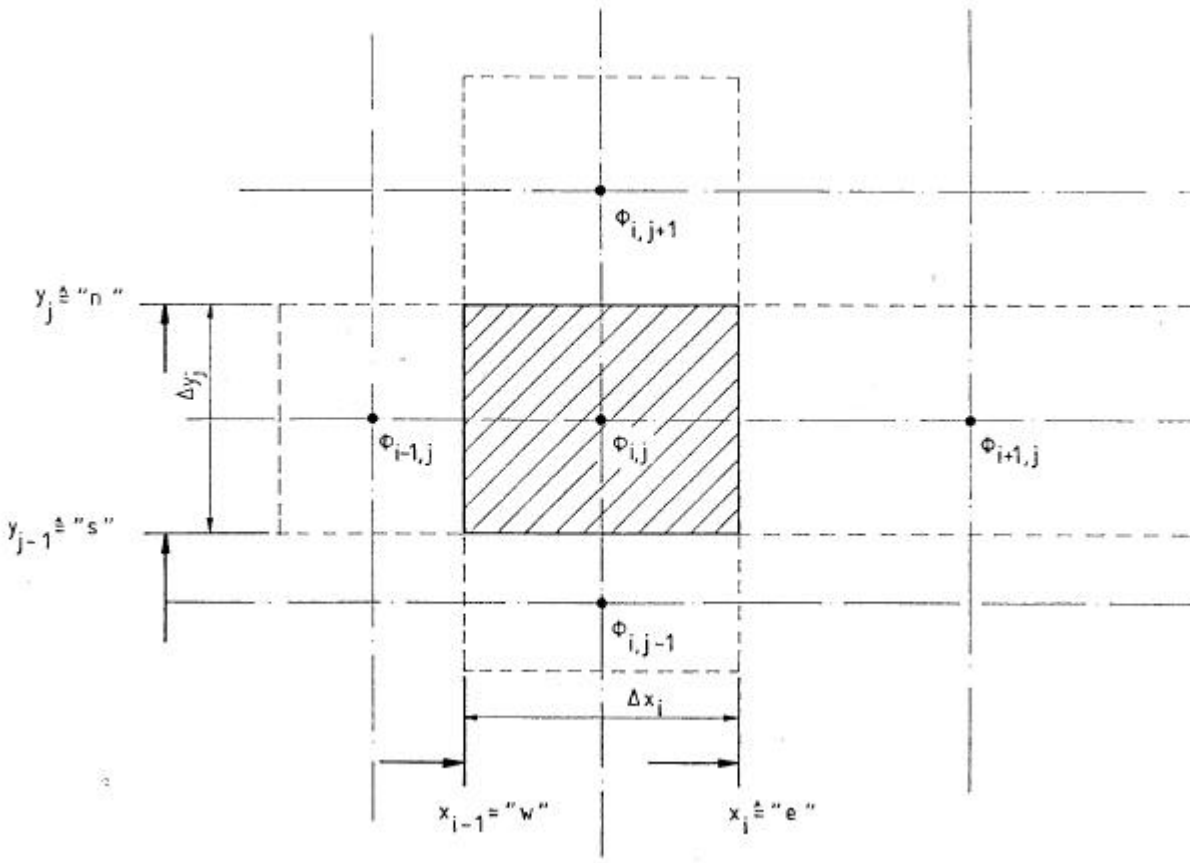

Fig. 3.2 : Géométrie d'un volume de contrôle.

Dans le premier temps de l'approximation, on exploite l'intégrale de l'équation (3.2-2) et l'équation est linéarisée. Pour cela, nous définissons les hypothèses suivantes:

- Les courants le long des volumes de contrôle ainsi que
- les termes-sources dans les volumes de contrôle sont constants.

Pour linéariser la relation, les courants de masse Cx=pU et Cy=pV seront chaque fois calculés avec les valeurs de 1' itération précédente. Cela donne l'expression suivante:

$$\left[C_{x,e}\phi_e - \Gamma_{\phi,e}\left(\frac{\partial\phi}{\partial x}\right)_e\right]\Delta y - \left[C_{x,w}\phi_w - \Gamma_{\phi,w}\left(\frac{\partial\phi}{\partial x}\right)_w\right]\Delta y +$$

$$+\left[C_{x,n}\phi_n - \Gamma_{\phi,n}\left(\frac{\partial\phi}{\partial y}\right)_n\right]\Delta x - \left[C_{x,s}\phi_s - \Gamma_{\phi,s}\left(\frac{\partial\phi}{\partial y}\right)_s\right]\Delta x = \quad (3.2\text{-}3)$$

$$S_\phi \Delta V$$

Le deuxième temps de l'approximation consiste & exprimer les valeurs des fonctions Öe, Öw, Ön, Ös et leurs gradients correspondants sur les surfaces externes des volumes de contrôle par rapport aux points du maillage. Pour les premiers, nous utiliserons la méthode UPWIND (Patankar 1980). La valeur Ö est approchée suivant une fonction en escalier comme fig.3.3, dans laquelle la direction du courant de masse est chaque fois prise en compte. Pour la valeur Öw de la figure 3.3, on obtient par exemple:

$$\phi_w = \phi_{i-1,j} \quad \text{für} \quad C_{x,w} > \sigma \qquad (3.2\text{-}4a)$$

$$\phi_w = \phi_{i,j} \quad \text{für} \quad C_{x,w} \leq \sigma \qquad (3.2\text{-}4b)$$

Dies kann als

$$C_{x,w}\phi_w = \max[C_{x,w}, \sigma]\phi_{i-1,j} - \max[-C_{x,w}, \sigma]\phi_{i,j} \qquad (3.2\text{-}5)$$

où MAX(a,b) signifie la plus grande valeur entre a et b. Les courants de masse Cx,m avec m w, e, n, s sont déterminés par interpolation linéaire des valeurs des points du maillage.

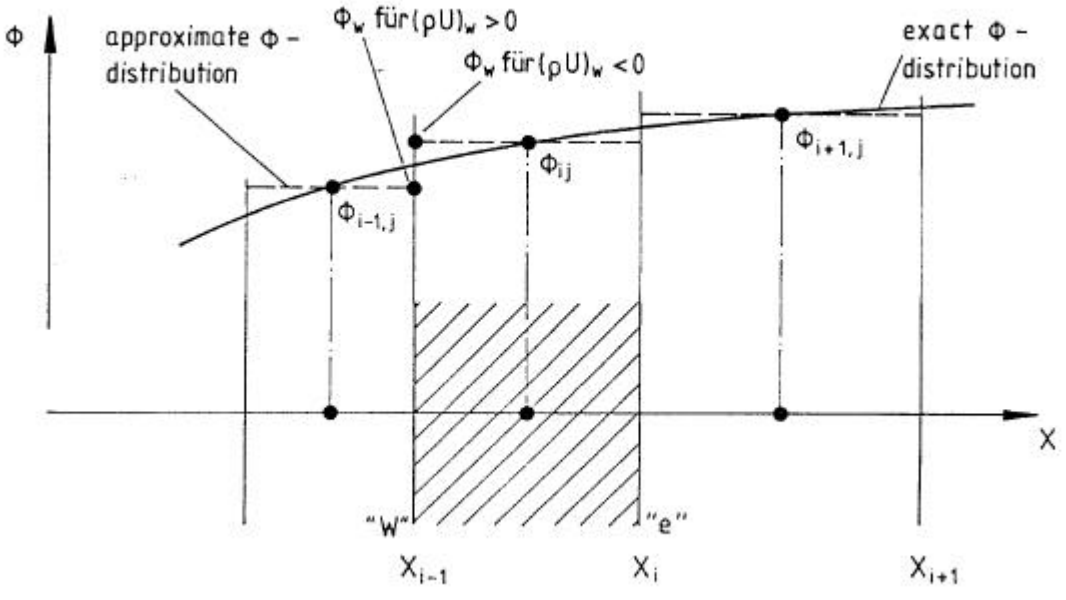

Fig. 3.3 : Approximation des termes de correction avec la méthode Upwind.

Pour les gradients (äÖ/äx)e, (äÖ/äx)w, etc... , nous calculons la différence centrale, i.e. le partage de Ö est approché par des droites. D'après la figure 3.4, nous obtenons:

$$\left(\frac{\partial \phi}{\partial x}\right)_w = \frac{2(\phi_{i,j} - \phi_{i-1,j})}{x_i - x_{i-2}} \qquad (3.2\text{-}6)$$

avec les expressions correspondantes pour les autre gradients dans l'équation (3.2-3).

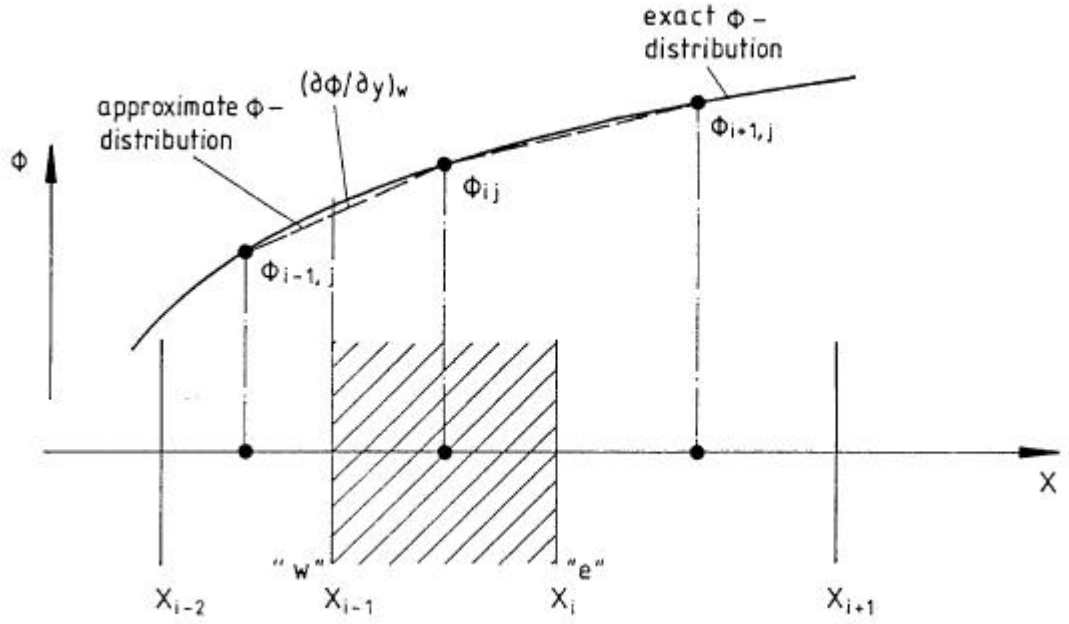

Fig 3.4 Approximation par différence centrale des termes de diffusion.

Si nous employons cette expression dans l'équation nous obtenons pour chaque volume de contrôle une différentielle implicite de la forme:

$$a_P^{ij}\phi_{i,j} = a_E^{ij}\phi_{i+1,j} + a_W^{ij}\phi_{i-1,j} + a_N^{ij}\phi_{i,j+1} + a_S^{ij}\phi_{i,j-1} + b^{ij} \qquad (3.2\text{-}7)$$

mit

$$a_E^{ij} = \max\left[-c_{x,e}, \sigma\right]\Delta y + \frac{\Gamma_{\phi,e}\cdot\Delta y}{x_{i+1} - x_{i-1}} \qquad (3.2\text{-}8a)$$

$$a_W^{ij} = \max\left[c_{x,w}, \sigma\right]\Delta y + \frac{\Gamma_{\phi,w}\,\Delta y}{x_i - x_{i-1}} \qquad (3.2\text{-}8b)$$

$$a_N^{ij} = \max\left[-c_{y,n}, \sigma\right]\Delta x + \frac{\Gamma_{\phi,n}\,\Delta x}{y_{j+1} - y_{j-1}} \qquad (3.2\text{-}8c)$$

$$a_S^{ij} = \max\left[c_{y,s}, \sigma\right]\Delta x + \frac{\Gamma_{\phi,s}\,\Delta x}{y_j - y_{j-1}} \qquad (3.2\text{-}8d)$$

$$b^{ij} = s_\phi\,\Delta V \qquad (3.2\text{-}8e)$$

$$a_P^{ij} = a_E^{ij} + a_W^{ij} + a_N^{ij} + a_S^{ij} \qquad (3.2\text{-}8f)$$

Il faut remarquer ici que les coefficients de la relation (3.2-7) sont différents du fait de l'utilisation de maillage décalé pour les équations de U et V, puisque par exemple, les courants de masse doivent toujours être utilisés à d'autres endroits.

3.3 Couplage pression - vitesse

Les équations discrétisées de quantité de mouvement sont les suivantes:

$$a_P^u u_{ij} = a_E^u u_{i+1,j} + a_W^u u_{i-1,j} + a_N^u u_{i,j+1} + a_S^u u_{i,j-1} - (P_{i+1,j} - P_{ij})\Delta y + b^u \qquad (3.3\text{-}1)$$

$$a_P^v v_{ij} = a_E^v v_{i+1,j} + a_W^v v_{i-1,j} + a_N^v v_{i,j+1} + a_S^v v_{i,j-1} - (P_{i,j+1} - P_{ij})\Delta x + b^v \qquad (3.3\text{-}2)$$

Celles-ci pourraient être résolues avec un algorithme approprié selon U et V, si la pression P était connue. Comme la pression n'a pas d'équation de transport mais qu'il n'y a qu'une condition de compatibilité entre les vitesse (2.2-1), celle-ci doit être recherchée par une méthode spéciale dans le cas de fluide incompressible qui nous intéresse.

Four cela, nous utilisons l'algorithme SIMPLE (Semi-Implicit-Method for Pressure-Linked Equations) développé par Patankar et Spalding (1972). .Les différentes étapes de cet algorithme sont:

1) Evaluation d'un champ de pression EP*], ex. EF*J0

2) Résolution des équations discrétisées de quantité de mouvement (3.3-1) et (3.3-2) avec le champ de pression évalué. Ceci nous donne les champs de vitesse [U*] et [V*].

3) Comme [P*] est inexact, les vitesses calculées [U*] et [V*] ne remplissent pas l'équation de continuité.

4) Les correction de pression et de vitesse IF' J **,** EU' J seront faite de sorte que les valeurs corrigées

[U**] = [U*] + [U'] (3.3-3a)

[V**] = [V*] + [V'] (3.3-3b)

[P**] = [P*] + [P'] (3.3-3b)

correspondent à l'équation de continuité. Après ces corrections, les équations discrétisées de quantité de mouvement ne sont plus exactes.

5) Les étapes 2) - 4) sont répétées jusqu'à ce que les équations de quantité de mouvement et de continuité soient simultanément exactes.
Ensuite nous déduisons les relations correspondantes. La discrétisation de l'équation de continuité en un volume de contrôle de pression donne:

$$s_e U_{ij}\,\Delta y - s_w U_{i-1,j}\,\Delta y + s_n V_{ij}\,\Delta x - s_s V_{i,j-1}\,\Delta x = 0 \qquad (3.3\text{-}4)$$

Dans un maillage décalé, les courants de masse peuvent être déterminés sans interpolation puisqu'ils sont mémorisés au niveau des surfaces externes des volumes de contrôle de pression. D'après les champs de vitesses EU*J et IV*J nous obtenons une 'source de masse' Sm suivant:

$$s_e U^*_{ij}\,\Delta y - s_w U^*_{i-1,j}\,\Delta y + s_n V^*_{ij}\,\Delta x - s_s V^*_{i,j-1}\,\Delta x = S_m \qquad (3.3\text{-}5)$$

Celle-ci peut être annulée si l'on considère les vitesses corrigées par U' et V'. L'équation définissant ces corrections est donc:

$$\begin{aligned} &s_e (U^*_{ij} + U'_{ij})\,\Delta y - s_w (U^*_{i-1,j} + U'_{i-1,j})\,\Delta y + \\ &s_n (V^*_{ij} + V'_{ij})\,\Delta x - s_s (V^*_{i,j-1} + V'_{i,j-1})\,\Delta x = 0 \end{aligned} \qquad (3.3\text{-}6)$$

La soustraction de cette relation à l'équation <3.3-5) donne une expression des corrections des vitesses:

$$s_e U'_{ij}\,\Delta y - s_w U'_{i-1,j}\,\Delta y + s_n V'_{ij}\,\Delta x - s_s V'_{i,j-1}\,\Delta x = -S_m \qquad (3.3\text{-}7)$$

D'un point de vue purement physique, les corrections de vitesses ne peuvent avoir lieu que si les pressions sont modifiées de la valeur correspondant à P' . La corrélation entre les corrections de pressions et de vitesses se déduit des équations discrétisées de la quantité de mouvement. Après la soustraction des équations définies par les champs [U*] et [P*] des équations de quantité de mouvement exactes (3.3-1) et (3.3-2) on obtient une équation de la forme:

$$ {}^{u}a_{P}^{ij}\,u'_{ij} = {}^{u}a_{E}^{ij}\,u'_{i+1,j} + {}^{u}a_{W}^{ij}\,u'_{i-1,j} + {}^{u}a_{N}^{ij}\,u'_{i,j+1} + {}^{u}a_{S}^{ij}\,u'_{i,j-1} - (p'_{i+1,j} - p'_{ij})\,\Delta y \qquad (3.3\text{-}8) $$

Dans la méthode SIMPLE, l'influence implicite des corrections de vitesse sur les points voisins de Ui,j est négligée, ce qui donne l'équation semi-implicite

$$ {}^{u}a_{P}^{ij}\,u'_{ij} = (p'_{ij} - p'_{i+1,j})\,\Delta y \qquad (3.3\text{-}9) $$

Des expressions analogues pour les autres corrections de vitesse peuvent être déduites dans l'équation (3.3—7). La substitution de ces expressions donne l'équation de correction de pression

$$ {}^{p}a_{P}^{ij}\,p'_{ij} = {}^{p}a_{E}^{ij}\,p'_{i+1,j} + {}^{p}a_{W}^{ij}\,p'_{i-1,j} + {}^{p}a_{N}^{ij}\,p'_{i,j+1} + {}^{p}a_{S}^{ij}\,p'_{i,j-1} - S_m \qquad (3.3\text{-}10) $$

mit den Koeffizienten

$$ {}^{p}a_{E}^{ij} = \frac{s_e\,\Delta y^2}{{}^{u}a_{P}^{ij}} \qquad (3.3\text{-}11a) $$

$$ {}^{p}a_{W}^{ij} = \frac{s_w\,\Delta y^2}{{}^{u}a_{P}^{i-1,j}} \qquad (3.3\text{-}11b) $$

$$ {}^{p}a_{N}^{ij} = \frac{s_n\,\Delta x^2}{{}^{v}a_{P}^{ij}} \qquad (3.3\text{-}11c) $$

$$ {}^{p}a_{S}^{ij} = \frac{s_s\,\Delta x^2}{{}^{v}a_{P}^{ij-1}} \qquad (3.3\text{-}11d) $$

$$ {}^{p}a_{P}^{ij} = {}^{p}a_{E}^{ij} + {}^{p}a_{W}^{ij} + {}^{p}a_{N}^{ij} + {}^{p}a_{S}^{ij} \qquad (3.3\text{-}11e) $$

L'équation de correction de pression à la même forme que les équations discrétisées de quantité de mouvement et peut être résolue suivant la même méthode. Avant cela, il faut spécifier les conditions correspondantes à la périphérie. Ceci est décrit dans la suite de ce rapport. Les détails sont donnés par Pakantar (1980).

Dans l'exemple de courant sur une marche descendante, les vitesses normales à la périphérie sont connues. Sur les parois elles sont nulles et sur les plans d'entrée et de sortie, elles sont connues. Ceci peut être utilisé pour la formulation des conditions de correction de pression à la périphérie.

Les corrections de vitesse sont nulles au niveau des parois, ce qui après reformulation conduit à des coefficients correspondants nuls pour l'équation de correction des pressions. Ceci correspond physiquement à une condition de Neumann pour la correction des pressions P' à la périphérie, dans laquelle les gradients normaux aux parois de la zone d'intégration sont nuls.

3.4 Méthode de sous relaxation

D'après les hypothèses qui ont mené à l'équation (3.3-9), les corrections de pression qui en résultent sont un peu trop importantes. Ceci peut mener à une divergence de l'algorithme SIMPLE si l'on ne fait pas une sous relaxation. C'est pourquoi dans le programme CAST, la pression n'est pas corrigée au moyen de l'équation (3.3-3c) mais par:

$$[P^{**}] = [P^{*}] + \acute{a}^{P}\,[P'] \qquad (3.4\text{-}1)$$

$\acute{a}^{P}$ est le facteur de sous relaxation pour la pression.

Les vitesses U et V seront sous relaxées comme suit. A gauche de l'équation algébrique (3.2-7) pour la variable générale Ö=(U,V), la valeur Ö de l'itération précédente sera ajoutée et soustraite de la façon suivante:

$$\phi_{ij} = \phi^{*}_{ij} + \left(\frac{\sum_m a^{ij}_m \phi_m + b^{ij}}{a^{ij}_\phi} - \phi^{*}_{ij} \right) \qquad (3.4\text{-}2)$$

L'index m court sur les point voisins E, W, K, et S. Le terme qui se trouve entre parenthèses correspond à la variation de Ö pour une itération. Celui-ci est sous relaxé par le facteur áÖ dans le programme, soit:

$$\phi_{ij} = \phi^{*}_{ij} + \alpha^{\phi} \left(\frac{\sum_m a^{ij}_m \phi_m + b^{ij}}{a^{ij}_\phi} - \phi^{*}_{ij} \right) \qquad (3.4\text{-}3)$$

Nous obtenons une bonne convergence de la résolution des équations de Navier-Stokes lorsque (Peric 1987):

$$\alpha^{u} \cong 1 - \alpha^{p} \quad ; \quad \alpha^{v} = \alpha^{u} \qquad (3.4\text{-}4)$$

Dans ce travail les facteurs de sous relaxation suivants seront utilisés:

$$\alpha^{u} = \alpha^{v} = 0.8 \quad ; \quad \alpha^{p} = 0.2 \qquad (3.4\text{-}5)$$

3.5 Algorithme de résolution

Les équations (3.3-1), (3.3-2) et (3.3-10) présentent des systèmes linéaires et algébriques pour le calcul des champs [U], [V] et [P']. Elles peuvent s'écrire sous forme de matrices. La matrice de coefficients [A] comporte, dans le cas de la discrétisation UPWIND, cinq éléments différents de zéro, positifs par ligne et à dominante diagonale. Les méthodes de résolution de tels systèmes étant très coûteuses, nous utilisons des méthodes itératives habituelles. Nous allons présenter deux de ces méthodes qui ont été utilisées dans ce travail, la méthode Gauâ-Seidel paragraphe(3.5.2) et la méthode itérative ligne par ligne paragraphe (3.5.1).

3.5.1 Méthode itérative ligne par ligne

La matrice de coefficients [A] peut, comme présenté fig. 3.5a, être écrite sous forme d'un système tridiagonal. Les sous matrices [Ap] sont tridiagonales. [As] et [An] sont des matrices diagonales. Chacun des vecteurs (Öj) comporte une ligne o ù y=const dans le maillage numérique.

La transformation du système d'équations présenté fig. 3.5b constitue la base du processus itératif de la méthode ligne par ligne. A gauche, il n'y a que des matrices tridiagonales qui peuvent être résolues efficacement par l'algorithme de Thomas. Le processus itératif fonctionne comme suit:

La solution est d'abord calculée sur la deuxième ligne de volumes de contrôle, i.e. le vecteur [Ö2]. Pour cela, le produit [An][Ö3] de la partie droite du système (fig. 3.5) est nécessaire. Comme [Ö3] n'est pas encore disponible, on utilise au départ des valeur évaluées. Après que le vecteur [Ö2] ait été calculé avec l'algorithme de Thomas (i.e. avec une méthode directe), on passe au vecteur [Ö4] puis plus généralement au vecteur [Öj].

Ici, les produits [As][Öj-1] et [An][Öj+1] sont nécessaires. Des valeurs de départ évaluées seront aussi utilisées.
L'ensemble de la zone de résolution est ainsi balayé avec à chaque fois des valeurs évaluées ou de la dernière itération (fig. 3.5b) pour la partie droite de l'équation.
Au deuxième passage sur le champ d'intégration, le calcul commence sur la troisième ligne.
Les produits [As][Öj-1] et [An][Öj+1] n écessaires pour le calcul seront ceux calculés au premier passage. puis on passe deux lignes plus loin etc, jusqu'à ce que la zone de travail soit balayée une deuxième fois.

a)

$$\begin{bmatrix} A_P & A_N & & & & \\ A_S & A_P & A_N & & & \\ & A_S & A_P & A_N & & \\ & & A_S & A_P & A_N & \\ & & & A_S & A_P & A_N \\ & & & & A_S & A_P \end{bmatrix} \begin{bmatrix} \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \\ \phi_6 \end{bmatrix} = \begin{bmatrix} S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \end{bmatrix}$$

Fig. 3.5: Structure de la matrice de coefficients.

a) Matrice de résultats

b) Itération

/// Valeurs améliorées
/// lors du 1er sweep.

b)

$$\begin{bmatrix} A_P & & & & & \\ & A_P & & & & \\ & & A_P & & & \\ & & & A_P & & \\ & & & & A_P & \\ & & & & & A_P \end{bmatrix} \begin{bmatrix} \phi_2^{\nu} \\ \phi_3^{\nu} \\ \phi_4^{\nu} \\ \phi_5^{\nu} \\ \phi_6^{\nu} \end{bmatrix} = \begin{bmatrix} S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \end{bmatrix} - \begin{bmatrix} & A_N & & & & \\ A_S & & A_N & & & \\ & A_S & & A_N & & \\ & & A_N & & A_S & \\ & & & A_S & & \end{bmatrix} \cdot \begin{bmatrix} \phi_2^{\nu} \\ \phi_3^{\nu-1} \\ \phi_4^{\nu} \\ \phi_5^{\nu-1} \\ \phi_6^{\nu} \end{bmatrix}$$

Cela nous donne un nouveau vecteur de solution complet [Ö].

La procédure entière est répétée jusqu'à ce que le critère de convergence ou nombre d'itérations (sweeps) donné soit atteint.

Dans l'algorithme SIMPLE, il n'est pas nécessaire à chaque fois de résoudre chaque équation jusqu'à la convergence, puisque après la résolution, les valeurs de la fonction obtenues sont corrigées et les matrices de coefficients sont recalculées.

Il faut cependant être sûr que les équations U, V et notamment l'équation de. .correction de pression, convergent, sinon l'ensemble de l'algorithme divergera après quelques itérations.

Dans ce travail, le nombre d'itérations (Sweeps), i.e. le nombre de passages sur les matrices de coefficients, est prédéfini. A chaque itération, il y a 5 Sweeps pour les équations U et V, et 40 pour l'équation de correction des pressions.

Les équations U et V sont plus simples à résoudre, puisque à chaque fois, on peut utiliser comme valeurs évaluées celles qui se trouvent dans les champs après 1'itération précédente. D'autre part, il y a des conditions Diriclet sur trois côtés de la zone de calcul.
Ceci n'est pas valable pour la correction des pressions; le calcul est à chaque fois démarré avec [P']=0 à défaut de meilleures valeurs. D'autre part, pour l'équation de pression, les côtés de la zone remplissent la condition de Neumann, ce qui défavorise la convergence.

3.5.2 Méthode Gauâ-Seidel

La méthode Gauâ-Seidel itère sur les points. Le calcul se fait selon

$$\phi_{i,j} = \frac{1}{a_P^{ij}} \left(a_E^{ij} \phi_{i+1,j} + a_W^{ij} \phi_{i-1,j} + a_N^{ij} \phi_{i,j+1} + a_S^{ij} \phi_{i,j-1} + b^{ij} \right) \qquad (3.5\text{-}1)$$

Dans ce travail, nous utilisons la méthode Red-Black. La zone de calcul est donc balayée deux fois, comme pour la méthode ligne par ligne.
Au premier passage, les nouvelles valeurs de Ö seront calculées
La zone de travail est représentée comme un échiquier. A chaque fois on saute le point voisin (on ne calcule que les points 'rouges')
Du côté droit de l'équation (3.5-1) les valeurs sont évaluées pour le premier sweep. Pour les passages suivants, les valeurs sont celles calculées au sweep précédent.
Au deuxième passage, ce sont les points 'noirs' qui sont calculés. Four cela on utilise dans la partie droite de l'équation (3.5-1) les valeurs $Ö'_{ij}$ calculées au premier passage.
L'algorithme continue comme décrit dans le paragraphe précédent jusqu'à ce que le critère de convergence ou le nombre d'itérations prédéfini soit atteint.

3.5.3 Evaluation des algorithmes de résolution

Les algorithmes de résolution sont une partie importante du programme car ce sont eux qui demandent le plus de temps de calcul.
Il est donc indispensable d'utiliser des méthodes les plus efficaces possibles. Il faut cependant trouver un compromis pour les calculateurs parallèles.
La méthode Gauâ-Seidel décrite dans le paragraphe 3.5.2 est très efficacement parallélisable. Cependant, elle devient excessivement lente avec l'augmentation du nombre de points de maillage (surtout, pour des problèmes à conditions périphérique de Neumann comme pour l'équation de correction des pressions **).** Le gain de temps obtenu par la parallélisation peut même dans le cas extrême être annulé par la mauvaise efficacité de cet algorithme.
Ceci explique qu'après son utilisation au début de ce travail il ait été abandonné.

L'efficacité de l'algorithme ligne par ligne est améliorée lorsque la résolution directe du système tridiagonal ne se fait pas seulement suivant la direction y=const. mais de façon alternée y=const et x=const. Mais dans ce cas, son degré de parallélisation diminue. Cette possibilité n'a donc pas été utilisée dans ce travail.

Dans la première version du programme CAST (Peric et Scheuerer 1987), la méthode de Stone (1968) a été utilisée pour l'équation de correction des pressions. Celle-ci se base sur une décomposition incomplète. L'efficacité de cette méthode est très supérieure pour la résolution de l'équation de correction des pressions aux deux algorithmes présentés dans ce rapport (Peric 1987 et Barcus 1987). Mais l'algorithme de Stone (1968) n'est pas suffisamment parallélisable. Dans un travail ultérieur, on calculera si une telle méthode n'a pas une efficacité globale meilleure malgré un degré de parallélisation plus faible.

3.6 Critère de convergence

Le critère de convergence est le suivant:
Après que les coefficients de l'équation (3.2-7) soient calculés, les résidus pour les volumes de contrôle U et V sont établis à partir de:

$$R_{ij} = a_E^{ij}\phi_{i+1,j} + a_W^{ij}\phi_{i-1,j} + a_N^{ij}\phi_{i,j+1} + a_S^{ij}\phi_{i,j-1} + b^{ij} - a_P^{ij}\phi_{ij} \qquad (3.6\text{-}1)$$

Dans le cas de l'équation de continuité, le résidu est la source de masse Sm. Pour toutes les variables, la somme des résidus absolus est calculée et mémorisée dans la grandeur FÖ. Dans ce travail, on utilise pour FÖ les masses entrantes et les flots de quantité de mouvement. Il est alors attendu que ces grandeurs soient inférieures à un critère de convergence prédéfini å i.e.:

$$\frac{\sum |R_{ij}|}{F_\phi} \leq \varepsilon \qquad (3.6\text{-}2)$$

Pour ce travail, on a choisi å=0.005.

4. Multiprocesseur DIRMU

(DIstributed Reconfigurable MUltiprocessor kit)

4.1 Matériel

Le multiprocesseur DIRMU 25 (Händler 1~85) a été développé à l'université d'Erlangen—Nürnberg pour tester les configurations de multiprocesseurs et la programmation parallèle.

L'idée de départ de DIRMU est de proposer un élément de base permettant de construire un large éventail de configurations de multiprocesseurs adaptées à diverses tâches.

Un module de base est présenté figure 4. 1, Il comprend un module processeur (P-module) et un module mémoire multiport (M-module). Chacun comprend 8 connexions (ports). Les P-ports 1 - 7 peuvent être connectés aux M-ports d'autres modules DIRMU. Les M-modules sont connectés à plusieurs processeurs qui peuvent ainsi avoir directement accès aux données de la mémoire multiport.

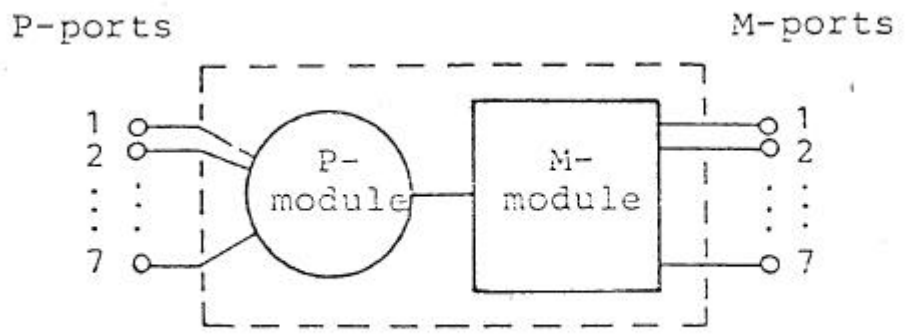

Fig. 4.1 : module de base DIRMU.

A partir de ces éléments, on peut construire différentes configurations de calculateurs. La figure 4.2 montre plusieurs exemples.

En théorie, on peut combiner autant de processeurs que l'on veut. Cependant, il n'y au plus que huit processeurs directement reliés entre eux, il s'agit donc d'un structure à voisinage limité.

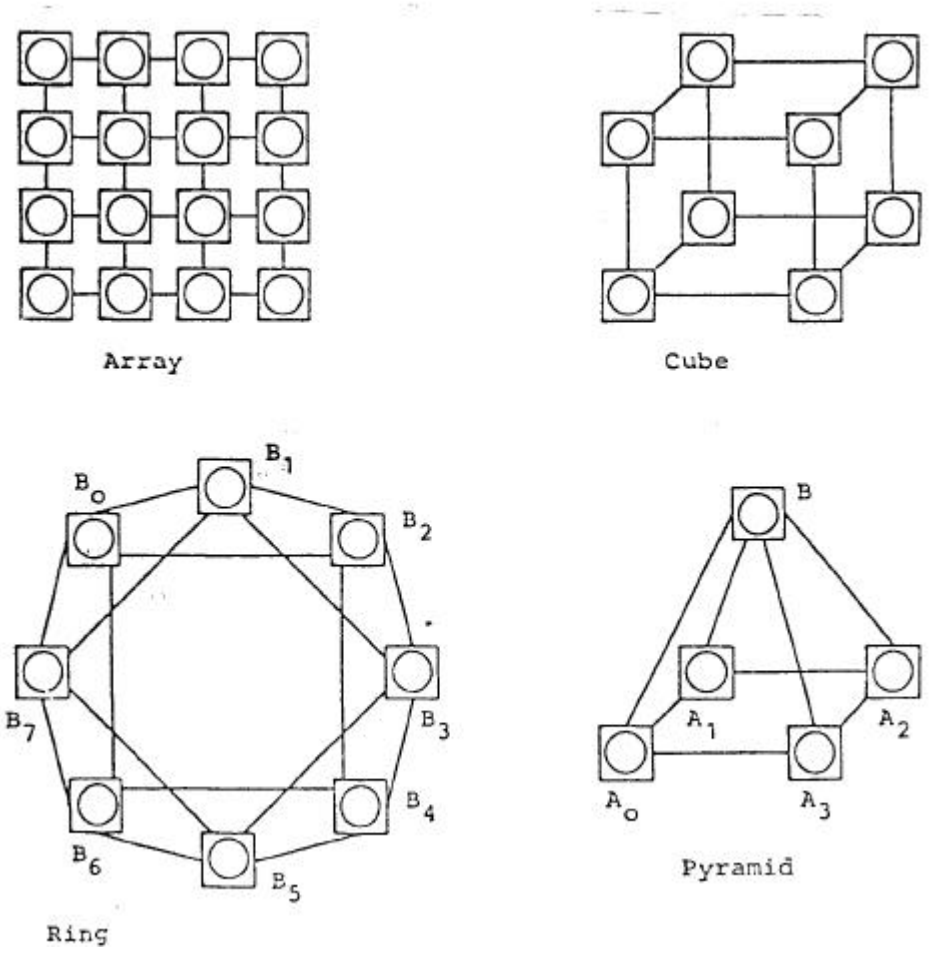

Fig. 4.2 Exemples de configurations DIRMU.

Un module de base est présenté en détail figure 4.3.

Le P-module contient un microprocesseur 8086 et un coprocesseur 8087 de la firme Intel, une mémoire RAM/ROM privée (actuellement de 320K/16K octets), des interfaces d'Entrées/Sorties (terminal, disque, imprimante, etc...) et 8 P-ports.

Le M-module est une mémoire multiport (64 K octets) comprenant 8 M-ports.

Par le P-port 0 et le M-port 0, chaque processeur peut accéder son propre M-module qui fait partie de son espace mémoire. Les P-ports 1,2,...,7 peuvent être connectés au moyen de câbles aux M-ports des autres unités DIRMU. Dans ce cas, les M-modules sont partagés entre plusieurs processeurs ce qui permet de partager les données se trouvant dans la mémoire multiport.

La figure 4 montre un exemple simple.

La zone de données A est vue par le processeur F0 à partir de l'adresse 1:6000H (l=segment, 6000H=offset). La même zone de données peut être accédée par P2 à partir de l'adresse 2:6000H et par P1 à partir de l'adresse 7:6000H.

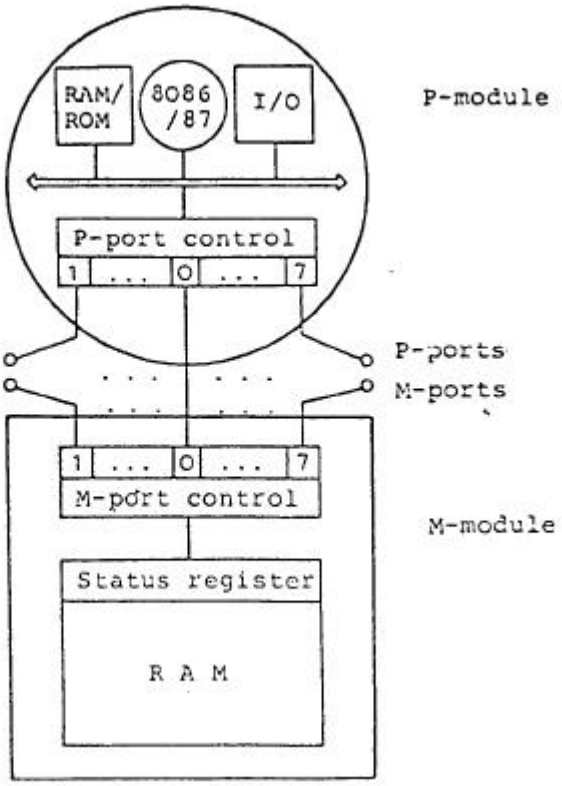

Fig. 4.3 : Structure d'un module de base DIRMU

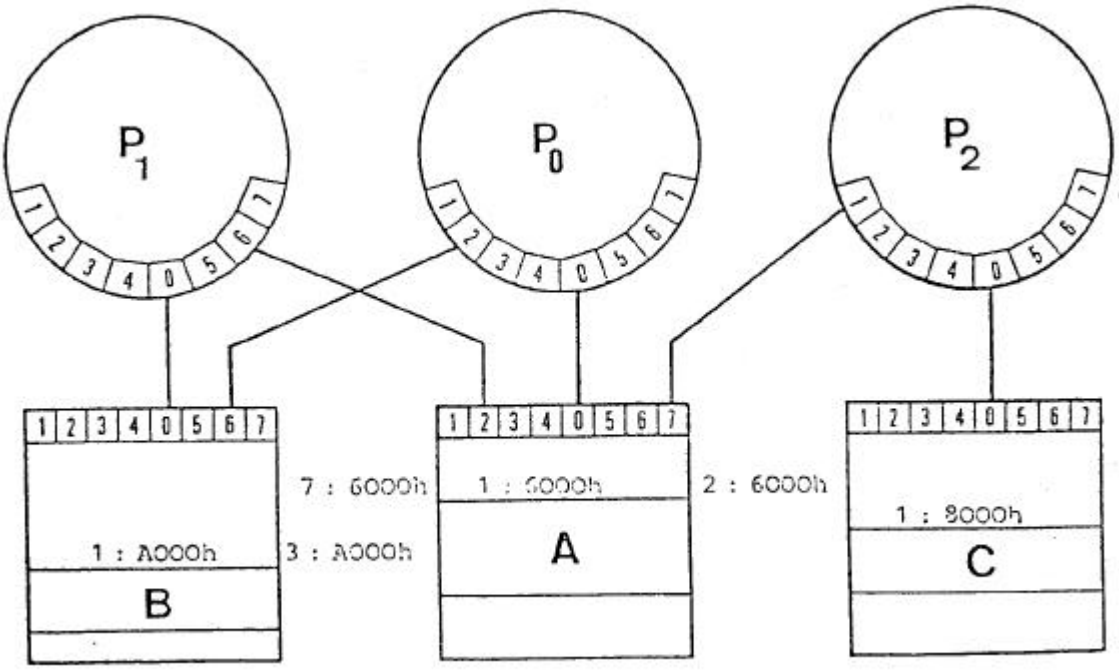

Fig. 4.4 : Adressage dans les M-modules

Les adresses du 8086 sont décodées par le contrôleur du P-port. Lorsque le contrôleur du P-port décode le segment i, i=1,2,...,8, il sélecte le M-port i-1 et accède au module correspondant. L'offset reste inchangé.

Les mots de données sont de 16 bits. Le temps d'accès est de 800ns pour la mémoire locale, et de 1,2ìs pour la mémoire multiports (sans conflits d'accès).
Le temps d'accès plus important pour la mémoire multiports est dû aux mémoires RAM plus lentes et aux retards des contrôles des P- ports et M-ports. Il faut souligner que toutes les mémoires multiports connectées à un processeur ont le même temps d'accès.

Chaque processeur a sa propre horloge. Si plus d'un processeur essaie d'accéder en même temps à la mémoire multiport, il y a un conflit d'accès qui est résolu par le contrôleur de M-port.

Un processeur attend jusqu'à l'obtention de la donnée. La règle est telle qu'un processeur ne peut pas accéder 2 fois à la mémoire de suite si un autre processeur attend d'accéder à la même mémoire.

Des mesure ont montré (Maehle 1985) que le temps d'accès plus important pour le M-module, ainsi que les retards dus aux conflits d'accès n'ont qu'une très faible influence sur le temps total d'exécution d'un programme d'application, car seules les données partagées sont stockées dans les M-modules. La majorité des accès mémoire se font sur la zone de code et les données dans la mémoire privée.

Un processeur peut bloquer la mémoire multiports pendant plus d'un cycle mémoire. Cela est nécessaire pour certaines primitives de synchronisation (ex: instruction EXCHANGE) et la consistance des données qui sont plus larges qu'un mot mémoire (ex. variables réelles).

La communication entre processeurs se fait par deux registres dans l'espace d'adressage de la mémoire multiports. Deux processeurs qui partagent le même M-module peuvent s'envoyer des interruptions en positionnant les bits de ces registres.
Ce mécanisme peut être utilisé pour le trafic de messages entre processeurs.

Un autre registre status supporte le diagnostic d'erreur des processeurs voisins. Si une erreur est détectée dans une unité (ex problème d'alimentation, instruction HALT illégale etc.), un bit correspondant est positionné dans le registre status et une interruption est envoyée à tous les voisins.
Le diagnostic du voisinage est un prérequis important pour les configurations DIRMU résistant aux pannes.
Chaque unité DIRMU a sa propre alimentation et se trouve dans un rack 3/4 19" (voir fig. 4.5). On peut contrôler la connexion des P-modules et M-modules ainsi que la charge des ports sur la face avant par des diodes électro-luminescentes.

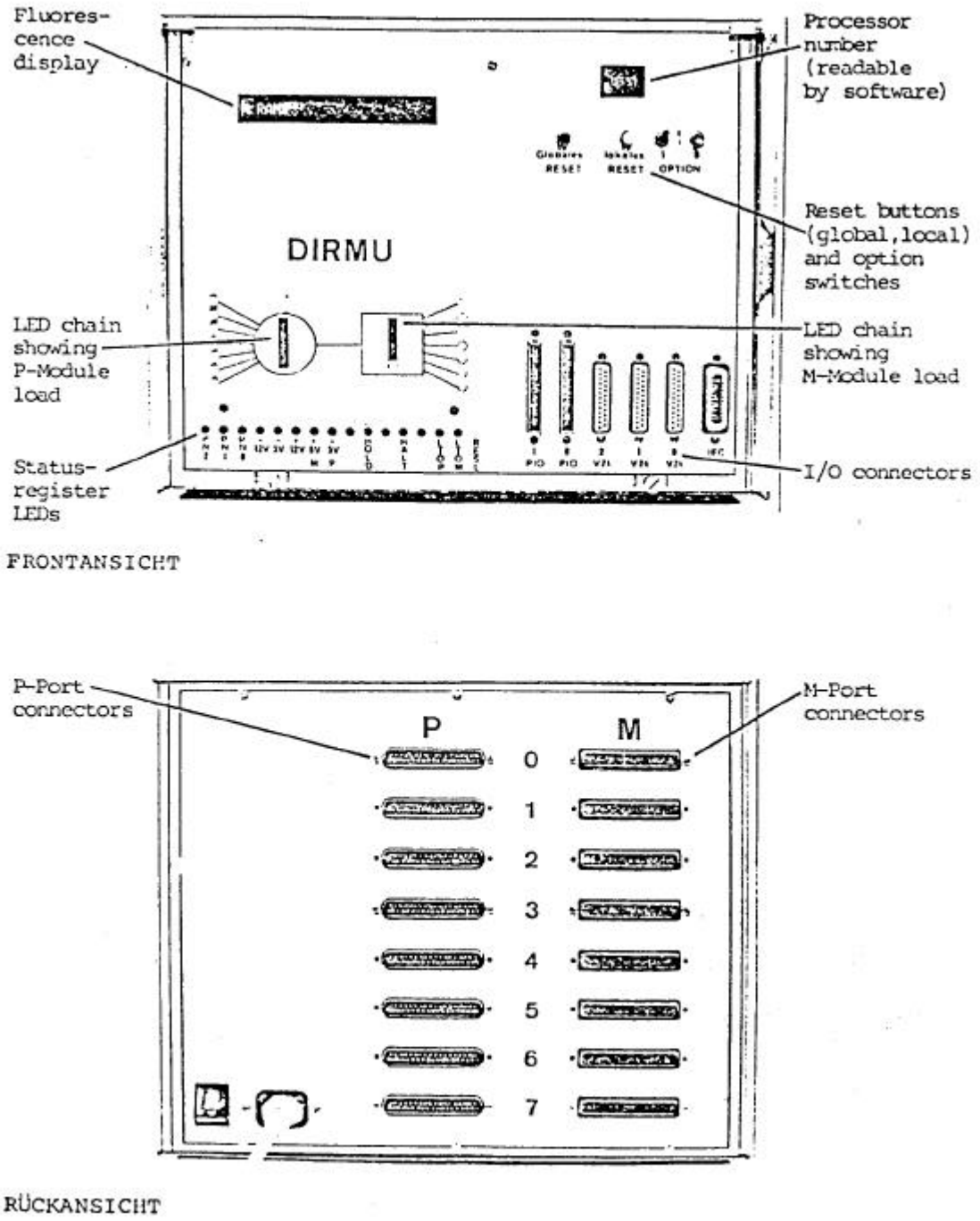

Fig. 4.5 Faces avant et arrière d'un module DIRMU.

4.2 Système d'exploitation et langage de programmation

Nous décrirons dans ce chapitre le sytème fonctionnant sur DIRMU, le DIRMOS (Diagnosis Integrated Reconfiguring Multiprocessor Operating System). Il se greffe sur le système Concurent CP/M.

Le développement d'applications parallèles sur DIRMU est basé sur le concept de rnaître/esclave.

Chaque unité DIRMU équipée d'un disque et d'un terminal (MASTER) peut être utilisée comme unité de développement en mode monoprocesseur. L'exécution d'une application parallèle est supportée par DIRMOS.

Dans les applications parallèles, le MASTER fait toutes les E/S disques et le dialogue avec l'utilisateur. Les autres processeurs coopérant à l'application parallèle sont appelés SLAVES.

Différents utilisateurs peuvent se partager la machine de telle sorte que chaque SLAVE soit assigné à un MASTER à la fois (space sharing).

Une des caractéristiques de DIRMOS est l'intégration de la gestion du diagnostic et de la configuration.
Pendant les moments libres, le diagnostic calcule la configuration de processeurs en bon état. Cela comprend les processeurs en bon état et (par un chemin d'accès) les processeurs atteignables ainsi que leurs connexions.
La configuration utilisée par le programme d'application ('application configuration') est mappée par DIRMOS (Configuration Finder) sur la machine en bon état.
DIRMOS fournit le numéro de port du voisin logique pour chaque processeur ainsi que sa position dans la configuration appelée par l'application.
D'autre part, DIRMOS comporte un chargeur parallèle qui charge l'application parallèle dans la mémoire privée de chaque processeur participant à celle-ci.

DIRMOS consiste en trois parties principales:

- La partie résidente COMINT.
- Le FINDER.
- Le MANAGER.

COMINT est la partie résidente de DIRMOS. Il est chargé dans la mémoire privée de tous les modules à l'initialisation du système.
En tapant le caractère de commande à la console, l'utilisateur change l'état du processeur correspondant de FREE SLAVE en MASTER et entre dans 1'interpréteur de commandes.
L'application est démarrée en tapant le nom du programme parallèle. Avant que celui-ci ne soit chargé dans les SLAVES, l'utilisateur doit spécifier la configuration pour laquelle le programme a été écrit. Le FINDER approprié est chargé dans le MASTER, cherche la configuration la plus grande de ce type et fournit le nombre maximal de processeurs

disponibles. L'utilisateur peut alors choisir la nombre désiré de processeurs pour son application.
Les processeurs réservés pour l'application sont mis en état de SLAVE par le configurateur et le programme est chargé en pipeline **à** partir du MASTER (fig. 4.6).

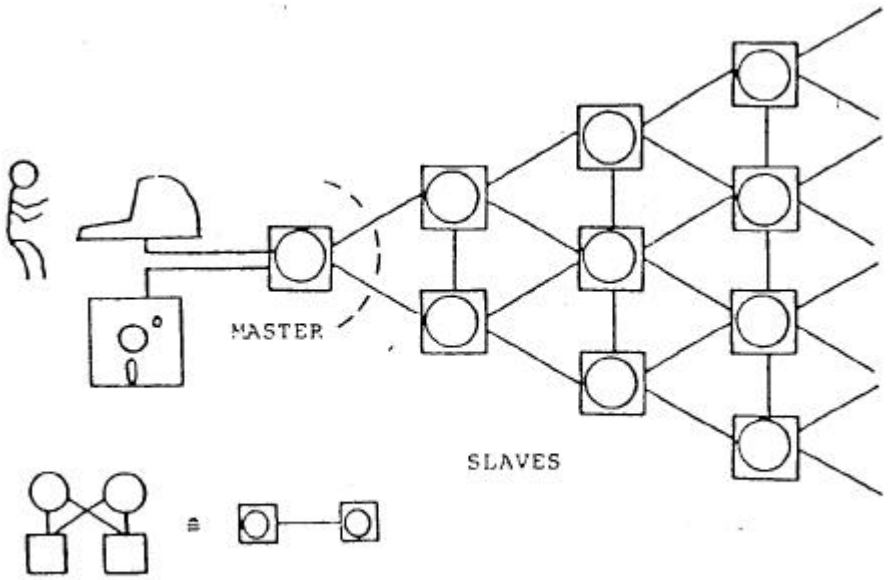

Fig. 4.6 : Chargement du programme parallèle.

L'application parallèle est chargée au dessus du COMINT en overlay.
Le MANAGER est un module en librairie qui est linké au programme d'application. Le MANAGER trouve les informations concernant le mapping fait par le FINDER dans la partie résidente (COMINT). Le MANAGER peut ainsi calculer la position logique du processeur dans la configuration de l'application. Le nombre de ports connectés aux voisins logiques est aussi calculé.

Après l'initialisation du MANAGER, l'APPLICATION a accès au numéro de port de ses voisins logiques et peut créer des structures de données partagées en appelant le LOCATOR.
Le LOCATOR est un module en librairie qui calcule les adresses des données partagées.

Le système DIRMOS supporte trois principaux types de configuration:

- anneau (RING).. y
- tableau <ARRAY)
- cube à n dimensions(N-CUBE>

Si l'application demande une autre configuration (ex. une pyramide), il faut utiliser le configurateur spécial. L'utilisateur décrit alors sa propre configuration.

Le langage de programmation utilisé est le MODULA2 (Wirth 1985) qui s'apparente au PASCAL.

5. Parallélisation de la méthode de résolution

Dans ce chapitre, nous décrirons la parallélisation de la méthode de résolution présentée dans le chapitre 3.
Le premier paragraphe présentera la configuration du calculateur utilisée.
Le partage des variables du programme d'application et le déroulement de la méthode parallèle seront décrits dans les paragraphes 5.2 et 5.3.
La synchronisation nécessaire au déroulement parallèle du programme sera présentée paragraphe 5.4.

5.1 Configuration du calculateur

Le premier temps de la parallélisation d'une méthode de résolution est le choix d'une configuration de calculateur adaptée.
Pour le problème ici traité d'une zone d'intégration à deux dimensions rectangulaire, nous avons le choix entre deux structures (voir fig. 4.2)

- anneau
- champ

Du fait de la facilité des problèmes de synchronisation et de la mise en oeuvre aisée de la méthode de résolution ligne par ligne sans alternance d'orientation des lignes, la structure choisie est l'anneau d'une longueur maximale de 26 processeurs (voir fig. 5.1).
La zone de calcul est divisée en bande comme figure 5.2. Chaque bande est travaillée par un processeur. Sur l'anneau de la figure 5.2, le MASTER est noté M et les SLAVES Sj.

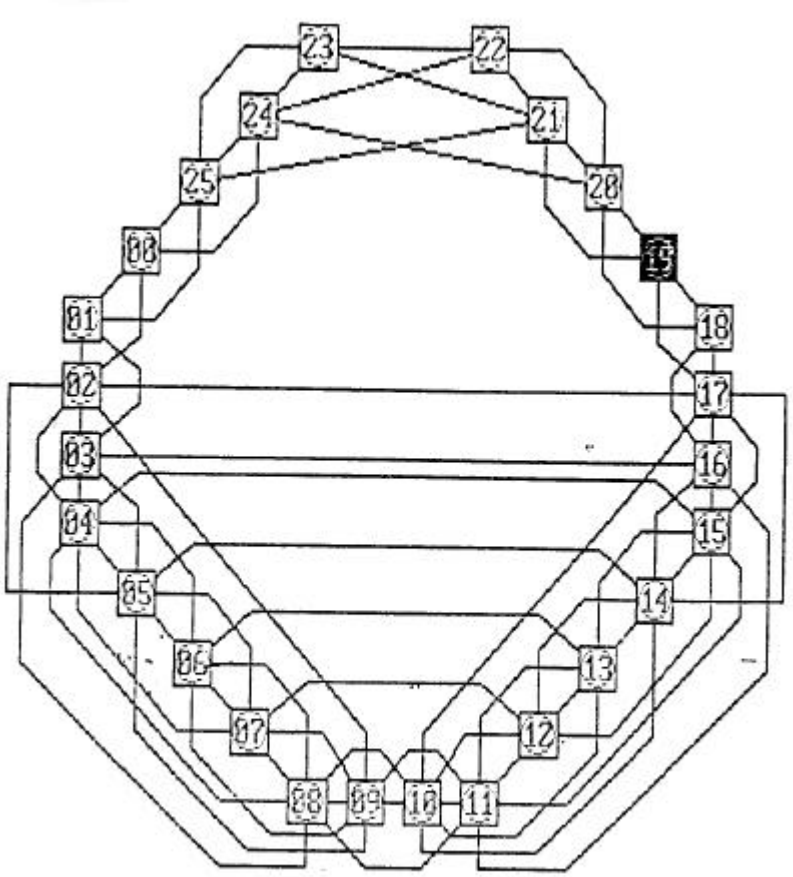

Fig. 5.1 : Configuration du multiprocesseur DIRMU

Processeurs Zone de calcul

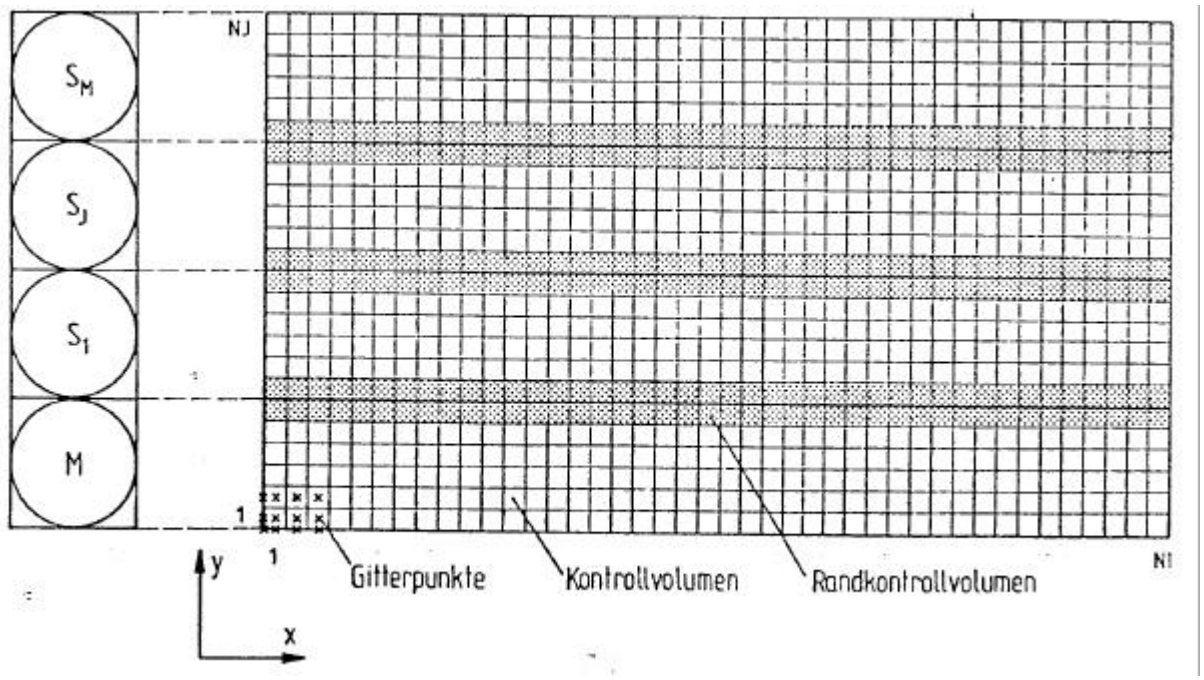

Fig. 5.2 : Partage des variables

5.2 Partage des variables

Le deuxième temps de la parallélisation consiste à partager les variables entre les processeurs.

Les variables dépendantes du problème traité sont les champs de vitesses [U] et [V] ainsi que le champ de pressions [P]. Ce sont des matrices de longueur NX*NY où NX est le nombre de points de maillage dans la direction X et NY le nombre de points dans la direction Y. Leur calcul nécessite l'utilisation de douze autres matrice du même type pour

- les coefficients des volumes finis (équation (3.2-8)) 5
- les courants de masse Cx et Cy aux surfaces des volumes de contrôle 2
- le champ de correction des pressions 1
- le terme source S 2
- des champs accessoires pour le calcul des coefficients de l'équation de correction des pressions. 2

L'utilisation de vecteurs de longueur NY pour la mémorisation des courants aux bords des volumes de contrôle par exemple n'entre presque pas en ligne de compte pour le calcul des besoins en mémoire.

Pour le partage des variables, les concepts de variables locales, régionales et globales seront utilisés. Il sont définis comme suit

- Les variables locales ne sont pas partagées, elles sont en mémoire privée et initialisées dans le programme par le processeur Sj (ou M)
- Les variables régionales sont partagées par des processeurs voisins. Elles se trouvent en mémoire multiport.
- Les variables globales sont partagées par tous les processeurs utilisés dans l'application. Elles se trouvent aussi en mémoire multiport.

Les matrices de longueur NX*NY seront traitées comme des variables régionales, comme le montre la figure 5.2, chaque processeur mémorise une bande comprenant les données de NI et NJ/M volumes de contrôle. M est le nombre de processeurs, le nombre de volumes de contrôle correspond à NI=NX-2 et NJ=NY-2. Les processeurs 1 et M situés aux extrémité de la zone de travail ont une ligne de maillage en plus.
Les sommes de résidus absolus utilisées pour le critère de convergence, le courant de masse vers l'extérieur de la zone d'intégration et la valeur de référence pour la pression sont traitées comme variables globales.

Pour ce travail, l'ensemble des sous-matrices est mémorisé dans les mémoires multiports. Il serait plus efficace de ne mémoriser en mémoire multiport que les rangées de volumes de

contrôles situées aux bords des bandes, le reste des données étant situé en mémoire privée (zone hachurée fig. 5.2). Ainsi, lors du calcul notamment de maillages fins, où contrairement à la figure 5.2 le rapport entre zone locale et zone régionale partagée entre processeurs est élevé, on pourrait remplacer la plupart des accès à la mémoire multiport par des accès à la mémoire locale plus rapide.
Cependant, ceci aurait pour effet d'augmenter sensiblement le travail de programmation et la transparence d'un programme parallèle (la 'sous-matrice' devrait être séparée en deux objets local et régional). C'est pourquoi cette possibilité n'a pas été retenue.
La démarche sera la même pour les prochains systèmes multiprocesseurs d'Erlangen. La mémoire multiport sera augmentée par rapport à la mémoire privée, afin que des objets de ce type puissent être mémorisés. La mémoire privée ne contiendra plus que le système d'exploitation et les données locales.

Ce travail comporte une limitation qui est due au caractère de test du programme. Le maître contient l'ensemble des champs de variables et les rassemble à la fin du calcul.
Ceci entraîne que la grandeur maximale du problème traité est limitée à la place mémoire du MASTER. Cette limitation a pour avantage de faciliter les mesure d'efficacité et du speedup.

Lors des mesures, le même problème sera mesuré d'abord sur un seul processeur, puis sur plusieurs, et les temps de calcul seront ensuite comparés.

La limitation peut être évitée par une modification de l'initialisation et de la collecte des résultats afin que pour des applications pratiques on puisse travailler sur des maillages beaucoup plus fins.

5.3 Déroulement du programme

Le premier temps de la méthode est l'initialisation. Les champs de vitesses et de pression sont initialisés avec des valeurs évaluées sur chaque processeur ainsi que les variables régionales et locales.
Ensuite la méthode de résolution se déroule sur chaque processeur.

D'abord, les coefficients des volumes de contrôle et les termes—source de l'équation de quantité de mouvement en X (3.3-1) sont calculés. Pour cela, on utilise les flots convectifs et diffusifs qui passent par les surfaces des volumes de contrôle. Une première boucle calcule les flots à travers les surfaces ouest des volumes de contrôle pour chaque processeur.
Pour les interpolations au niveau des volumes du bord de la zone de calcul, on prend à chaque fois des valeurs des processeurs voisins.
Dans une deuxième boucle, les processeurs calculent les flots à travers les surfaces Sud, Nord et Est des volumes de contrôle. Les flots à travers les surfaces est sont mémorisés dans un vecteur à une dimension de longueur NY et utilisés comme flot Ouest de la colonne suivante.
Aux frontières des sous-matrices, dans la méthode de volumes finis, il faut seulement faire attention à ce que le flot passant par la surface nord du volume de contrôle le plus haut de la zone Sj soit identique à celui passant par la surface sud du volume de contrôle le plus bas de la zone Sj+1.
Le déroulement décrit est le même pour toutes les autres colonnes de volumes de contrôle.

Le calcul des coefficients est facilement parallélisable puisqu'aucune synchronisation n'est nécessaire.

Après le calcul des coefficients de l'équation (3.2-8), les valeurs de [U] en mémoire sont utilisées dans l'équation de volumes finis (3.3-1) et la somme des résidus absolus est calculée dans toutes les sous-matrices selon l'équation (3.6-1). Les valeurs résultantes pour chaque sous-zone seront mémorisées jusqu'au tour de synchronisation qui suit le calcul de la quantité de mouvement en Y et la correction des pressions.

Au cours de cette méthode, l'algorithme de résolution décrit paragraphe 3.5.1 est appelé.
Les étapes de sa parallélisation seront 4écrites à la fin de ce paragraphe.

Après la résolution du champ [U], les valeurs du champ [V] sont calculées de façon analogue.
Ensuite, on fait une correction du flot de masse à la sortie de la zone de résolution, car puisque cette méthode est conservative, la quantité de masse à la sortie doit être égale à la quantité de masse à l'entrée.

Pour la réalisation de cette condition, on calcule le flot de masse sortant en faisant la somme globale des flux de masse sortant de la dernière rangée de volumes de contrôles soit:

$$\dot{m}_E = \sum_{j=1}^{NJ} C_{x,j}\,\delta y_{,j} \qquad (5.3\text{-}1)$$

Chaque processeur calcule d'abord la masse sortant de sa zone de travail, pùis après une synchronisation de tous les processeurs (voir paragraphe 5.4) pour le calcul de la somme globale, on obtient la vitesse à la sortie de la zone de travail par

$$U_{NX,j} = U_{NX-1,j} \cdot \frac{\dot{m}_I}{\dot{m}_E} \qquad (5.3\text{-}2)$$

Les courants de masse Cx et Cy sont recalculés avec les valeurs des champs de vitesses ainsi que le terme source pour l'équation de correction des pressions (3.3-5). En même temps que l'on calcule le courant de masse et le terme source, on détermine les coefficients de l'équation de correction des pressions.
Les coefficients d'équation de correction des pressions sont symétriques, i.e. à chaque fois, le coefficient sud de la ligne j de volumes de contrôle est le même que le coefficient nord de la ligne j-1.
Lors du calcul des coefficients des volumes de contrôle aux frontières des zones de chaque processeur, il faut soit une synchronisation régionale, soit un calcul spécifique pour ces valeurs (comme réalisé dans ce travail).
Après la détermination des coefficients, le résidu de masse est formé de la somme des valeurs absolues des termes source Sm (équ. (3.3-5)) et distribué à tous les processeurs.
Ensuite, l'algorithme de résolution est appelé.

Il faut ici faire attention à ce que l'équation de correction des pressions soit résolue de façon suffisamment précise, sinon on risque d'avoir des corrections de pression irréelles et de faire diverger l'algorithme.

Après la résolution de l'équation de correction des pressions, les pressions et les vitesses sont corrigées avec la relation (3.3-3). Cela est exécuté parallèlement sur toutes les sous-zones.
Ainsi se termine une itération complète de la méthode. Ensuite, une synchronisation globale permet d'ajouter les valeurs résiduelles des équations de correction de U, V et P pour obtenir des valeurs globales. Ces valeurs sont comparées à des conditions de fin données.
Si la valeur maximale de ces trois résidus est plus grande que la condition de fin, 1'itération décrite est relancée avec les nouvelles valeurs en mémoire.
Sinon le programme se termine.
Le maître envoie aux esclaves un ordre de récupération des données qui retournent leurs sous champs [U], [V] et [P]. Les résultats sont rassemblés et affichés.

Pour terminer ce paragraphe, nous décrirons la parallélisation de la méthode ligne par ligne.
Comme nous l'avons montré dans le paragraphe 3;5.1, cet algorithme se déroule en deux passages. Lors du premier, on utilise des valeurs évaluées pour la première itération, sinon les valeurs de l'itération précédente.
C'est pourquoi le premier passage peut se faire sur tous les processeurs simultanément sans synchronisation.
Par contre une synchronisation régionale est nécessaire lors du deuxième passage puisque le processeur Si a besoin de certains résultats des processeurs Sj-1 et Sj+1 et doit les attendre.
Les résolutions directes se faisant uniquement pour des lignes où y=const, une structure en anneau de processeurs permet un degré relativement important de parallélisation de la méthode ligne par ligne.

5.3 Synchronisation

La synchronisation est un élément primordial dans le développement d'algorithmes parallèles.
Nous distinguons deux catégories:

- synchronisation régionale

- synchronisation globale

La synchronisation régionale ne joue qu'entre processeurs voisins sur des variables communes. Chaque processeur interrompt le déroulement de son programme jusqu'à ce que les variables de son voisin aient pris les valeurs désirées.

La synchronisation globale est nécessaire par exemple lors du calcul de la somme des résidus absolus.
Elle se déroule comme suit:

1. Le maître écrit sa somme partielle dans la mémoire de son voisin Si et positionne une variable booléenne a VRAI.
2. Cela signale & Si que la somme partielle est valable.
3. Si ajoute sa somme partielle à cette valeur, repositionne la variable booléenne du maître à FAUX et envoie la nouvelle somme partielle à S2 de la même façon etc...
4. De cette manière, tous les processeurs sont interrogés.
5. Le maître reçoit un signal du dernier processeur SM et lit la somme globale des résidus absolus dans sa mémoire multiports.

De la même façon, la somme globale des résidus est envoyée à tous les processeurs de l'anneau.
Si la somme des résidus est inférieure à une valeur donnée, le programme s'arrête.

Le type de synchronisation globale choisi présente le désavantage d'une perte de temps importante pour des anneaux comprenant de nombreux processeurs. Pour le cas de la machine DIRMU à 26 processeurs, ces pertes de temps sont encore acceptables

6. RESULTATS

6.1 Courants

La figure 6. 1 montre le vecteurs de vitesse calculés et les lignes de courant.
Les vitesses et pressions calculées par ce programme parallèle ont été comparées aux résultats d'un programme séquentiel fonctionnant sur le calculateur CDC Cyber 855 du centre de calcul régional d'Erlangen.
Les valeurs sont les mêmes à la précision de calcul près.

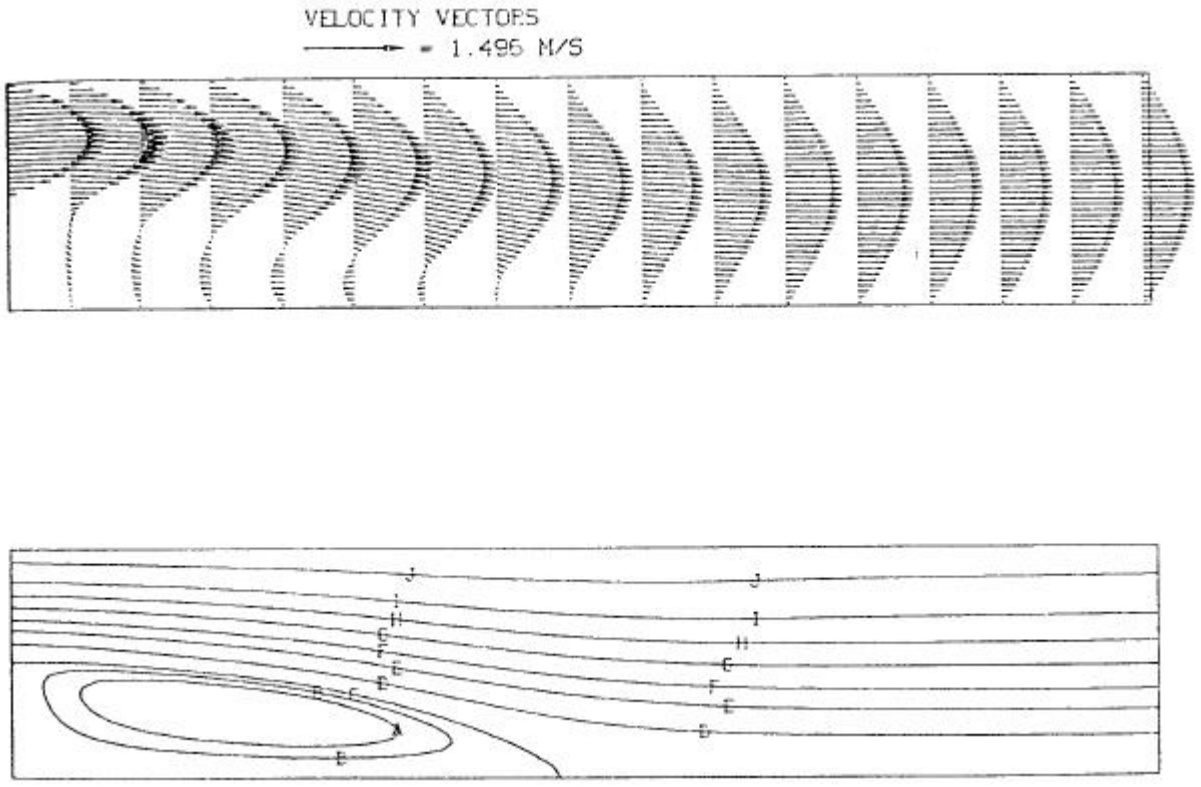

Figure 6. i Vecteurs de vitesse et lignes de courants.

6.2 Accélération grâce à la paralléllisation

Les mesures de temps de calcul ont été effectués pour deux maillages différents.
Le premier comprend 20x10 volumes de contrôle dans les direction respectives x et y.
Le deuxième comprend 20x46 volumes de contrôle.

Les temps de calcul, accélération grâce à la parallélisation (speedup) et l'efficacité sont regroupés dans les tableaux 6-1 et 6-2.
D'autre part, les speedup des deux tests sont présentés sous forme de fonction sur les figures 6.2 et 6.3.

Matrice 22x12 points.
RESOR: U=2,3E-6 V=2,24E-7 P=4,19E-3

Nombre de Processeurs	Nombre d'iterations	Durée (s)	Speedup	Efficacité (%)
1	29	446.700	1.00	100
2	29	265.523	1.68	84
3	29	187.245	2.39	80
4	29	179.876	2.48	62
6	29	98.981	4.51	75

Tableau 6-1 Resultats pour 20x10 volumes de contrôle.

Le speedup est défini comme le temps de calcul avec un processeur divisé par le temps de calcul pour M processeurs. L'efficacité est la relation entre le speedup réel par rapport au speedup théorique. Le speedup idéal est égal au nombre de processeurs.

Les deux tableaux montrent que pour des maillages plus fins, on obtient un speedup et une efficacité nettement meilleurs que pour un maillage grossier.

Matrice 22x48 points.
RESOR: U8,27E-4 V=3,35E-5 P=4,79E-3

Nombre de Processeurs	Nombre d'iterations	Durée (s)	Speedup	Efficacité (%)
2	235	8619.997	2.00	100.0
3	235	5828.656	2.96	98.6
4	235	4399.256	3.92	98.0
6	235	2962.094	5.82	97.0
8	235	2242.709	7.69	96.1
12	235	1523.944	11.31	94.3
16	235	1461.889	11.79	73.7
24	235	807.000	21.36	89.0

Tableau 6-2 Résultats pour 20x46 volumes de contrôle.

Dans le cas de maillage plus fin, on obtient une efficacité supérieure à 90% jusqu'à 12 processeurs. On a donc un plus grand degré de parallélisme.
Lorsqu'on dépasse 12 processeurs, l'efficacité tombe au dessous de 90%. La raison est que les bandes travaillées par chaque processeur sont de plus en plus faibles avec l'augmentation du nombre de processeurs, et donc le temps de Synchronisation devient plus important.
Cependant, même avec 24 processeurs, on gagne un facteur 21 sur le temps de calcul avec un seul processeur.

Pour le maillage grossier, l'efficacité reste au dessous de 90%. Ceci est aussi dû au temps de synchronisation important par rapport au calcul.
Le décrochage de la courbe du speedup (ex. lorsqu'on passe de 12 à 16 processeurs fig. 6.3) est dû aux inégalités de charge entre les différents processeurs. Tous les processeurs n'ont pas le même nombre de volumes de contrôle à calculer, ce qui entraîne que les processeurs les moins chargés doivent attendre les processeurs les plus chargés.

La meilleure efficacité serait obtenue si le maximum de place mémoire de chaque processeur était utilisé. Le temps de synchronisation diminuerait par rapport au temps de calcul ce qui ferait augmenter l'efficacité.

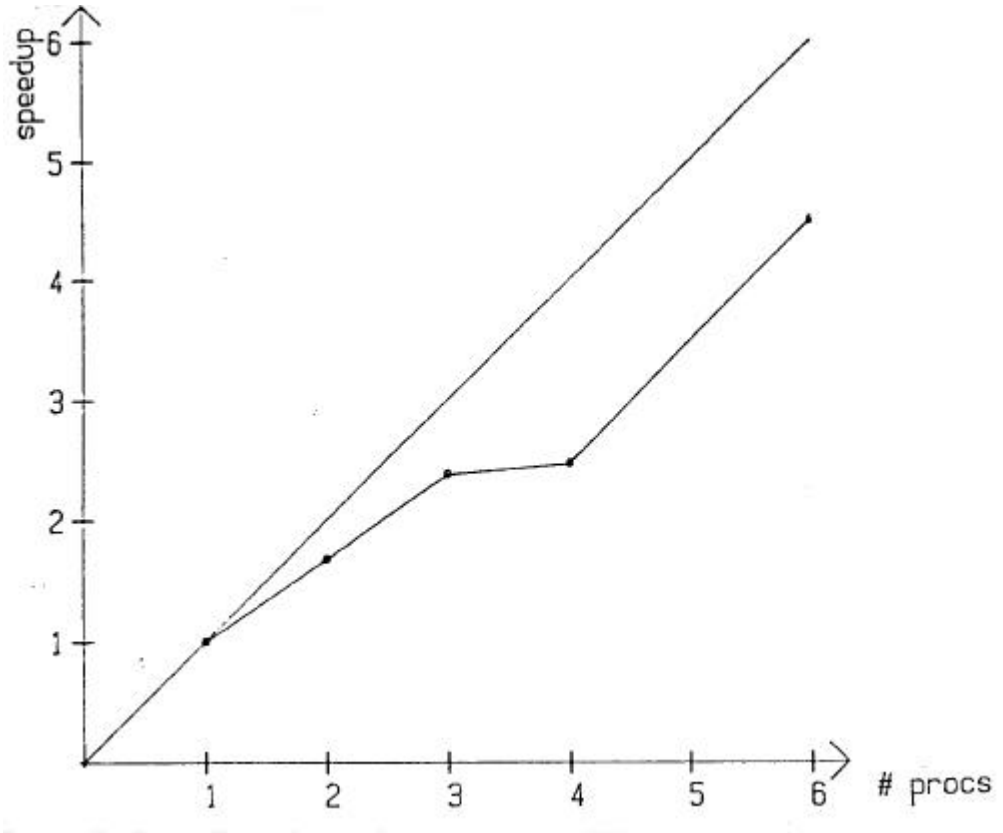

Fig. 6.2 : Speedup obetnu pour 20x10 volumes de contrôle

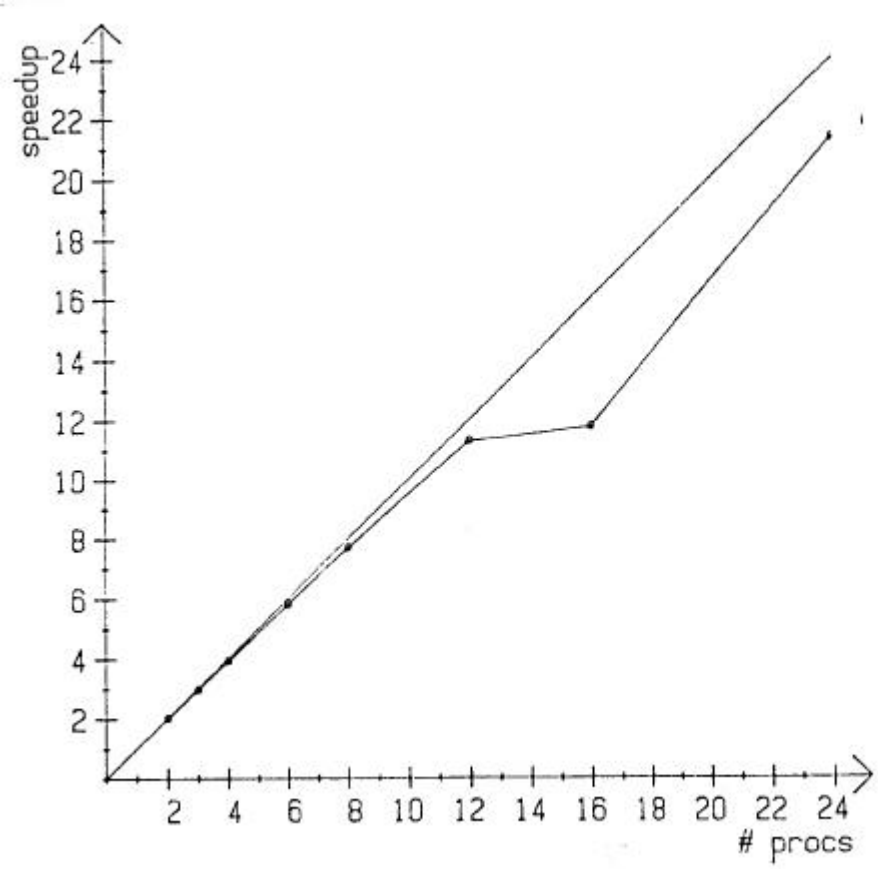

Fig. 6.3 : Speedup obetnu pour 20x10 volumes de contrôle

6.3 Discussion et propositions d'améliorations

Dans ce paragraphe nous ferons quelques remarques sur l'optimisation de la méthode.

6.3.1 Algorithmique

L'algorithme de résolution est la partie qui demande le plus de temps de calcul. Dans la résolution séquentielle, on utilise 1'algorithme de Stone (1968) qui est efficace et basé sur un découpage. incomplet. Cette méthode présente le désavantage de ne pas être parallélisable.
Dans le 'programme parallèle, nous avons utilisé l'algorithme point par point Gauâ-Seidel et la méthode ligne par ligne qui sont très parallélisables mais peu efficaces pour la résolution de grandes matrices. Ceci est un problème, surtout pour la résolution de l'équation de correction des pressions qui remplit les conditions de Neumann pour les bords et dont la convergence avec l'algorithme de Gauâ-Seidel est très mauvaise. L'utilisation de la méthode ligne par ligne n'apporte pas d'amélioration déterminante.

Il sera donc important de développer et tester ultérieurement des algorithmes parallélisables performants. Il faudra aussi vérifier dans quelle mesure l'utilisation de méthodes travaillant

sur des points couplés est valable. Ce sont des méthodes itératives sur des points qui résolvent pour chaque point un système pour les composantes de la vitesse et la pression (Vanka 1986). Ces méthodes de relaxation peuvent être accélérées en faisant des cycles avec plusieurs maillages (multigrid).

Le calcul des coefficients et la méthode ligne par ligne sont très parallélisables sur une configuration en anneau. Une synchronisation n'est nécessaire qu'aux limites Nord-Sud entre les sous-zones.
Dans le cas d'une configuration en champ, il faut ajouter à la synchronisation Nord-Sud une synchronisation Est-Ouest aux limites des sous-zones. Ceci entraîne une augmentation du temps de synchronisation. D'autre part, le nombre de volumes de contrôle calculés par processeur diminue quand on passe de l'anneau au champ.
Le choix entre l'anneau et le champ dépend de la grosseur du problème et du nombre de processeurs disponibles.
Pour un grand nombre de processeurs, on peut choisir une structure de champ plus intéressante que l'anneau sur le plan de la synchronisation (bandes étroites, chaque synchronisation demande n étapes au lieu de sqrt(n) pour le champ).
Les pertes de temps sont dues à la synchronisation. Les synchronisations globales demandent deux tours d'anneau le premier pour le calcul de la variable et le deuxième pour la diffusion de sa valeur globale à tous les processeurs. Les synchronisations doivent donc être réduites au minimum. Ceci a été fait pour ce travail.
Une synchronisation peut parfois être évitée grâce à un calcul d'une variable par deux processeurs voisins. Ceci est plus rapide même au prix d'un calcul un peu plus long, qu'une synchronisation.

6.3.2 Matériel

La machine DIRMU a été conçue dans les années 70 et travaille avec un processeur 8086/8087 à la vitesse de 5 MHz.
Il est clair que l'utilisation de processeurs 32 bits de la nouvelle génération augmenterait sensiblement la vitesse de calcul.
D'autre part, le développement de coprocesseurs microprogrammables spécialisés dans le calcul d'une partie souvent utilisée de l'algorithme (ex. algorithme de résolution Gauâ-Seidel) permettrait un gain important en temps de calcul.

7. Conclusion

Ce travail a consisté en la parallélisation d'une méthode à volumes finis des équations de Navier-Stokes à deux dimensions et incompressibles sur le multiprocesseur DIRMU.
La configuration choisie est l'anneau de processeurs. La zone de calcul est divisée en bandes travaillées chacune par un processeur.
L'exemple test est le courant laminaire sur une marche descendante. Les résultats ont été comparés à ceux obtenus sur une machine séquentielle.

On peut tirer les conclusions suivantes du travail réalisé:

- La méthode à volumes finis est très efficacement paralléllisable. Le travail sur une zone de 20x46 volumes de contrôle avec 24 processeurs permet de gagner un facteur 21,4 ce qui représente une efficacité de 89%.Une diminution du nombre de processeurs fait augmenter l'efficacité (97% pour 6 processeurs) puisque le temps de synchronisation diminue.Le caractère de test du programme n'a pas permis le travail sur des maillages fins.

- L'efficacité importante obtenue ne doit pas cacher le fait que l'utilisation des méthodes Gauâ-Seidel et ligne par ligne demandent beaucoup plus d'itérations (sweeps) que la méthode séquentielle de Stone (1968) qui présente le désavantage d'être difficilement parallélisable. L'utilisation d'algorithmes parallèles diminue l'efficacité de la méthode. Il faut calculer si le gain obtenu grâce à la parallélisation compense cette perte d'efficacité. Sinon, la parallélisation de la méthode ne présente aucun intérêt.

A partir de ces résultats, les travaux suivants devront être réalisés:

1. Test de différentes méthodes de résolution sur le plan de la parallélisation mais aussi de leurs performances.

2. Ajout aux algorithmes de solution d'une méthode par plusieurs niveaux de maillage (multigrid) pour la résolution des équations de Navier-Stokes. Cette méthode qui accé1ère la convergence doit être absolument utilisée pour avoir dans l'avenir des temps de calcul acceptables sur des machines parallèles pour des maillages fins.

3. Comparaison entre des méthodes couplées ou non pour la résolution des équations de Navier—Stokes sur des calculateurs parallèles. L'algorithme SIMPLE utilisé est une méthode non couplée: trois matrices sont successivement résolues pour les deux vitesses et la correction des pressions. Dans la méthode couplée, on résout une matrice par volume d~ contrôle, i.e. pour chaque volume de contrôle, on résout un système d'équations. Il y a dans cette méthode une part plus importante de temps de calcul par rapport à la synchronisation, ce qui pourrait améliorer l'efficacité de la parallélisation. Ces arguments doivent cependant être confirmés par des essais.

8 Bibliographie

Barcus, M., 1987, “Berechnung zweidimensionaler Strömungsprobleme mit Mehrgitterverfahren“, Diplomarbeit, Lehrstuhl für Strömungsmechanik, Universität Erlangen-Nürnberg.

Bird, R.B., Stewart, W.E., Lightfoot, E.N., 1960, ‘Transport Phenomena“, J. Wiley & Sons Inc., New York.

Bradshaw, P., Cebeci, T., Whitelaw, J.H., 1981, “Engineering Calculation Methods for Turbulent Flow“, Academic Press, London.

Geus, L., 1985, “Parallelisierung eines Mehrgitterverfahrens für die Navier-Stokes-Gleichungen auf EGPA-Systemen“, Arbeitsberichte des Inst. f. Mathem. Maschinen und Datenverarbeitung, Universität Erlangen-Nürnberg, Band 18, Nummer 3.

Händler, W., Maehle, E., Wirl, K., 1985, “The DIRI4U Testbed for High-Performance Multiprocessor Configurations“, Proc. of the First Intern. Conf. on Supercomputing Systems, St. Petersburg, FL, pp. 468—475.

Kutler, P., 1985, “A Perspective of Theoretical and Applied Computational Fluid Dynamics“, AIAA-J., Vol. 23, pp. 328-341.

Maehle, E., Wirl, K., Japel, D., 1985, “Experiments with Parallel Programs on the DIRMU Multiprocessor“, Parallel Computing, Berlin, pp. 515-520.

Patankar, S.V., 1980, “Numerical Heat Transfer and Fluid Flow“, Hemisphere Publ. Corp., Washington.

Patankar, S.V., Spalding, D.B., 1972, “A Calculation Procedure for Heat, Mass and Momentum Transfer in Three—Dimensional Parabolic Flows‘, Int. J. Heat Mass Transfer, Vol. 15, pp. 1787—1806.

Peric, M., 1987, “Efficient Semi-Implicit Solving Algorithm for Nine-Diagonal Coefficient Matrix“, Num. Heat Transfer, Vol. 11, pp. 251—279.

Peric, M., Scheuerer, G., 1987, “CAST - A Finite Volume Method for the Computer Simulation of Two-Dimensional Flows“, Lehrstuhl f. Strömungsmechanik, Universität Erlangen-Nürbnerg, LSTM-Bericht 165/T/87.

Stone, H.L., 1968, “Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations‘, SIAM J. Num. Anal., Vol. 5, pp. 530-558.

Thole, C.-A., 1985, “Experiments with Multigrid Methods on the CalTech-Hypercube“ Gesellschaft f. Mathem. und Datenverarbeitung, St. Augustin, GMD-Studien Nr. 103.

Vanka, S.P., 1986, “Block-Implicit Multigrid Solution of Navier-Stokes Equations in Primitive Variables“, J. Comp. Physics, Vol. 65, pp. 138—158.

Wirth, N., 1985, “Programming in MODULA-2“, 3. ed., Springer Verlag, Berlin.

Annexe: Programme en MODULA2

```
(*$R-*)
(*$S-*)
(*$T-*)
MODULE BCDIG;
FROM Terminal IMPORT KeyPressed,Read,Write;
FROM Termbase IMPORT AssignWrite, UnAssignWrite;
FROM Usarts   IMPORT U2Write;
FROM RealInOut IMPORT ReadReal,WriteReal;
FROM InOut IMPORT
WriteLn,ReadCard,WriteCard,ReadInt,WriteInt,ReadString,WriteString;
FROM SYSTEM IMPORT TSIZE;
FROM Storage IMPORT ALLOCATE,Available, DEALLOCATE;
FROM Locator IMPORT Allocate;
FROM Spinlock IMPORT Signal,SignalArrived;
FROM FacitFunctions IMPORT ClearScreen,CursorPos;
FROM NumberConversion IMPORT CardToString;
FROM Timer IMPORT RestartTime,GetTime;
FROM RingManager IMPORT right,left,len,pos;
CONST MAXX=22;
      MAXY=24;
      MAXXX=MAXX;
      MAXYY=48;
TYPE TMATRIX=ARRAY [1..MAXXX],[1..MAXYY] OF REAL;
VAR ptrU,ptrV,ptrP:POINTER TO TMATRIX;
    ptrY,ptrFY:POINTER TO ARRAY[1..MAXYY] OF REAL;
    i,j,k,jj,posi,posj,ofsi,ofsj,stepx,stepy,mn,sec,ms:CARDINAL;
    r,coef:REAL;
    test:BOOLEAN;
    c,matrix:CHAR;
    ch:ARRAY [1..255] OF CHAR;



(* ----------------------- affichage -------------------------------------
*)
PROCEDURE affich(VAR x:TMATRIX);
VAR i,j,xx,yy:CARDINAL;
    K:INTEGER;
BEGIN
CursorPos(0,0);Write(matrix);xx:=1+ofsi;yy:=1+ofsj;
i:=0;WHILE xx<=maxxx DO i:=i+1;
CursorPos(0,i*4);WriteCard(xx,4);xx:=xx+stepx END;

j:=0;WHILE yy<=maxyy DO
j:=j+1;CursorPos(j,0);WriteCard(yy,3);Write(':');xx:=1+ofsi;
 i:=0;WHILE xx<=maxxx DO
   i:=i+1;CursorPos(j,i*4);r:=x[xx,yy]*coef;
IF r>=0. THEN IF r>999. THEN WriteString('++++')
                       ELSE WriteCard(TRUNC(r),4) END
         ELSE IF r<-999. THEN WriteString('----')
                        ELSE WriteInt(-INTEGER(TRUNC(-r)),4) END
END;
xx:=xx+stepx  END; yy:=yy+stepy END;
CursorPos(23,50);WriteString('Coef.=');WriteReal(coef,6);
END affich;
```

```
PROCEDURE affctrl;
VAR i,j,xx,yy:CARDINAL;
    k:INTEGER;
BEGIN
WITH ptrm^ DO
CursorPos(0,0);Write(matrix);xx:=1+ofsi;yy:=1+ofsj;
FOR i:=1 TO 19 DO CursorPos(0,i*4);WriteCard(xx,4);xx:=xx+stepx END;
FOR j:=1 TO 22 DO CursorPos(j,0);WriteCard(yy,3);Write(':');xx:=1+ofsi;
 FOR i:=1 TO 19 DO
   CursorPos(j,i*4);
   IF ((xx<=maxx) AND (yy<=maxy))
       THEN WriteCard(ctrl[xx,yy],4);xx:=xx+stepx
       ELSE WriteString('    ') END;
 END;
yy:=yy+stepy END;Write(CHR(7));
END (* with *)
END affctrl;

PROCEDURE affiche(c:CHAR);
BEGIN
matrix:=c;
CASE c OF
'U':affich(ptrU^); |
'V':affich(ptrV^); |
'P':affich(ptrP^); |
'K':affctrl;
END
END affiche;

PROCEDURE init2(VAR x:TMATRIX);
VAR i,j:CARDINAL;
BEGIN
FOR i:=1 TO MAXXX DO
 FOR j:=1 TO MAXYY DO x[i,j]:=0. END END;
END init2;

(* ------------------------- CALCUL --------------------------------- *)
TYPE tmatrix=ARRAY [1..MAXX],[1..MAXY] OF REAL;
     tmemult=RECORD u,v,p,PP,CX,CY,DV:tmatrix;
                    ctrl:ARRAY[1..MAXX],[1..MAXY] OF CARDINAL;
                    X,FX:ARRAY[1..MAXX] OF REAL;
                    Y,FY:ARRAY[1..MAXY] OF REAL;
                    err:ARRAY[1..3] OF REAL;
                    GSN:ARRAY[1..3] OF CARDINAL;
                    OMEGA,flowg,PPREF,SUM,URFU,URFV,URFP:REAL;
                    DENSIT,FLOWIN,VISCOS:REAL;

maxx,maxy,n0,tempo,syncg,syncd,tredblack,redblack:CARDINAL;
                    sem:BOOLEAN;
                    stat:CHAR;
                    END;
VAR ptrm,gauche,droite,ptrx:POINTER TO tmemult;
    ptr1:POINTER TO RECORD AP,AE,AS,AW,AN:tmatrix END;
    ptr2:POINTER TO RECORD
         SU,SP,VIS,DU,DEN:tmatrix;
         CW,DW:ARRAY[1..MAXY] OF REAL;
         END;
    SORMAX,XMAX,YMAX:REAL;
    ITER:CARDINAL;
    SNORM,RESOR:ARRAY[1..3] OF REAL;
    maxxx,maxyy:CARDINAL;
```

```
PROCEDURE init(VAR x:tmatrix);
VAR i,j:CARDINAL;
BEGIN
FOR i:=1 TO MAXX DO
 FOR j:=1 TO MAXY DO x[i,j]:=0. END END;
END init;

PROCEDURE INITCOEF;
BEGIN
WITH ptrm^ DO
LOOP
ClearScreen;
WriteLn;
WriteString('A:URFU (.8) =');WriteReal(URFU,6);WriteLn;
WriteString('B:URFV (.8) =');WriteReal(URFV,6);WriteLn;
WriteString('C:URFP (.2) =');WriteReal(URFP,6);WriteLn;
WriteString('D:GSU  (5)  =');WriteCard(GSN[1],3);WriteLn;
WriteString('E:GSV  (5)  =');WriteCard(GSN[2],3);WriteLn;
WriteString('F:GSP  (40) =');WriteCard(GSN[3],3);WriteLn;
WriteString('G:TRB 1/2/3 =');WriteCard(tredblack,3);WriteLn;
WriteString('H:OMEGA 1..2=');WriteReal(OMEGA,6);WriteLn;
WriteString('I:SORMAX    =');WriteReal(SORMAX,6);WriteLn;
WriteString('J:maxxx     =');WriteCard(maxxx,3);WriteLn;
WriteString('K:maxyy     =');WriteCard(maxyy,3);WriteLn;
WriteString('L:XMAX      =');WriteReal(XMAX,6);WriteLn;
WriteString('M:YMAX      =');WriteReal(YMAX,6);WriteLn;
WriteString('N:DENSIT    =');WriteReal(DENSIT,6);WriteLn;
WriteString('O:VISCOS    =');WriteReal(VISCOS,6);WriteLn;
WriteLn;
WriteString('Modif :');Read(c);i:=ORD(CAP(c))-64;
IF ((i<1) OR (i>16)) THEN EXIT END;
CursorPos(i,14);
CASE i OF
 1:ReadReal(URFU); |
 2:ReadReal(URFV); |
 3:ReadReal(URFP); |
 4,5,6:ReadCard(GSN[i-3]); |
 7:ReadCard(tredblack); |
 8:ReadReal(OMEGA); |
 9:ReadReal(SORMAX); |
10:ReadCard(maxxx);maxx:=maxxx; |
11:ReadCard(maxyy);maxy:=maxyy DIV len; |
12:ReadReal(XMAX); |
13:ReadReal(YMAX); |
14:ReadReal(DENSIT); |
15:ReadReal(VISCOS);
END; (* case *)
END; (* loop *)
ClearScreen;
maxx:=maxxx;maxy:=maxyy DIV len;
stepx:=maxxx DIV 19;IF maxxx MOD 19<>0 THEN stepx:=stepx+1 END;
stepy:=maxyy DIV 22;IF maxyy MOD 22<>0 THEN stepy:=stepy+1 END;
ofsi:=0;ofsj:=0;posi:=1;posj:=1;
END (* with *)
END INITCOEF;

PROCEDURE INITVAL;
VAR COEF,XMOMIN:REAL;
    JST:CARDINAL;
BEGIN
```

```
WITH ptrm^ DO
FOR i:=1 TO MAXXX DO FOR j:=1 TO MAXYY DO
ptrU^[i,j]:=0.;ptrV^[i,j]:=0.;ptrP^[i,j]:=0. END END;
X[1]:=0.;r:=XMAX/FLOAT(maxxx-2);FX[1]:=0.;FX[maxxx]:=0.;
FOR i:=2 TO maxxx DO X[i]:=X[i-1]+r END;X[maxxx]:=X[maxxx-1];
FOR i:=2 TO maxxx-1 DO FX[i]:=(X[i]-X[i-1])/(X[i+1]-X[i-1]) END;
ptrY^[1]:=0.;r:=YMAX/FLOAT(maxyy-2);ptrFY^[1]:=0.;ptrFY^[maxyy]:=0.;
FOR j:=2 TO maxyy DO ptrY^[j]:=ptrY^[j-1]+r END;ptrY^[maxyy]:=ptrY^[maxyy-
1];
FOR j:=2 TO maxyy-1 DO ptrFY^[j]:=(ptrY^[j]-ptrY^[j-1])/(ptrY^[j+1]-
ptrY^[j-1]) END;
JST:=maxyy DIV 2+1;
COEF:=1.;FLOWIN:=0.;XMOMIN:=0.;
FOR j:=JST TO maxyy-1 DO
 ptrU^[1,j]:=COEF;
 r:=DENSIT*(ptrY^[j]-ptrY^[j-1])*ptrU^[1,j];
 FLOWIN:=FLOWIN+r;
 XMOMIN:=XMOMIN+r*ptrU^[1,j];
 ptrV^[1,j]:=r;
END; (* for *)
flowg:=FLOWIN;
PPREF:=0.;
FOR i:=2 TO maxxx DO FOR j:=2 TO maxyy-1 DO ptrU^[i,j]:=0.5*COEF END END;
SNORM[1]:=1./XMOMIN;SNORM[2]:=SNORM[1];SNORM[3]:=1./FLOWIN;
ITER:=0;
IF test THEN CursorPos(23,50);WriteString('FLOWIN');WriteReal(FLOWIN,6)
END;
END; (* with *)
END INITVAL;

PROCEDURE INIT;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO

FOR j:=1 TO MAXY DO
 CW[j]:=0.;DW[j]:=0.;
 FOR i:=1 TO MAXX DO
  v[i,j]:=0.;p[i,j]:=0.;VIS[i,j]:=VISCOS;DEN[i,j]:=DENSIT;
(* CX[i,j]:=0.; *)
  CY[i,j]:=0.;SU[i,j]:=0.;SP[i,j]:=0.;DU[i,j]:=0.;DV[i,j]:=0.;
  PP[i,j]:=0.;AP[i,j]:=0.;AE[i,j]:=0.;AW[i,j]:=0.;AN[i,j]:=0.;AS[i,j]:=0.;
END END; (* for for *)
END END END (* with *)
END INIT;

PROCEDURE handshake;
BEGIN
WITH ptrm^ DO
REPEAT UNTIL gauche^.syncg=0;gauche^.syncg:=1;
REPEAT UNTIL droite^.syncd=0;droite^.syncd:=1;
REPEAT UNTIL syncg=1;syncg:=0;
REPEAT UNTIL syncd=1;syncd:=0;
END END handshake;

(* ------------------------ solution ------------------------------- *)
PROCEDURE GS(VAR PHI,PHIG,PHID:tmatrix;IFI,IEND,JEND:CARDINAL);
VAR PHIOLD,RESL,RESAB:REAL;
    deb,k,trb:CARDINAL;

PROCEDURE point;
BEGIN
```

```
WITH ptrm^ DO WITH ptr1^ DO WITH ptr2^ DO
deb:=redblack+1;
IF (ODD(pos) AND ODD(maxy)) THEN deb:=5-deb END;

IF ((pos<>0) AND (JEND<>0)) THEN                          (* d// *)
    i:=deb;REPEAT
       PHIOLD:=PHI[i,1];
       PHI[i,1]:=OMEGA*((AW[i,1]*PHI[i-
1,1]+AE[i,1]*PHI[i+1,1]+AN[i,1]*PHI[i,2]+AS[i,1]*PHIG[i,maxy]+SU[i,1])/AP[i
,1])+(1.-OMEGA)*PHI[i,1];
RESL:=PHIOLD-PHI[i,1];
RESAB:=RESAB+ABS(RESL);
i:=i+2;UNTIL i>IEND
END;(* if *)                                          (* f// *)

 FOR j:=2 TO JEND DO deb:=5-deb;
  i:=deb;REPEAT
   PHIOLD:=PHI[i,j];
   PHI[i,j]:=OMEGA*((AW[i,j]*PHI[i-
1,j]+AE[i,j]*PHI[i+1,j]+AN[i,j]*PHI[i,j+1]+AS[i,j]*PHI[i,j-
1]+SU[i,j])/AP[i,j])+(1.-OMEGA)*PHI[i,j];
   RESL:=PHIOLD-PHI[i,j];
   RESAB:=RESAB+ABS(RESL);
i:=i+2;UNTIL i>IEND;
END;(* for *)

IF pos<>len-1 THEN                                  (* d// *)
   deb:=5-deb;i:=deb;REPEAT
       PHIOLD:=PHI[i,maxy];
       PHI[i,maxy]:=OMEGA*((AW[i,maxy]*PHI[i-
1,maxy]+AE[i,maxy]*PHI[i+1,maxy]+AN[i,maxy]*PHID[i,1]+AS[i,maxy]*PHI[i,maxy
-1]+SU[i,maxy])/AP[i,maxy])+(1.-OMEGA)*PHI[i,j];
RESL:=PHIOLD-PHI[i,maxy];
RESAB:=RESAB+ABS(RESL);
i:=i+2;UNTIL i>IEND;
END; (* if *)

END END END; (* with *)
END point;

PROCEDURE ligne;
VAR A,C:ARRAY[1..MAXX] OF REAL;
    IM:CARDINAL;
    TERM:REAL;
BEGIN
WITH ptrm^ DO WITH ptr1^ DO WITH ptr2^ DO
deb:=redblack;
IF (ODD(pos) AND ODD(maxy)) THEN deb:=3-deb END;
j:=deb;
IF j=1 THEN j:=3;
 IF ((pos<>0) AND (JEND<>0)) THEN                   (* d// *)
   A[1]:=0.;C[1]:=PHI[1,1];
  FOR i:=2 TO IEND DO
   IM:=i-1;
   TERM:=1./(AP[i,1]-AW[i,1]*A[IM]);
   A[i]:=AE[i,1]*TERM;

C[i]:=(AN[i,1]*PHI[i,2]+AS[i,1]*PHIG[i,maxy]+SU[i,1]+AW[i,1]*C[IM])*TERM;
  END;(* for *)
  FOR i:=IEND TO 2 BY -1 DO
   PHI[i,1]:=OMEGA*(A[i]*PHI[i+1,1]+C[i])+(1.-OMEGA)*PHI[i,1];
```

```
    END; (* for *);
END END;(* if if *)                                                (* f// *)

WHILE j<=JEND DO
  A[1]:=0.;C[1]:=PHI[1,j];
  FOR i:=2 TO IEND DO
   IM:=i-1;
   TERM:=1./(AP[i,j]-AW[i,j]*A[IM]);
   A[i]:=AE[i,j]*TERM;
   C[i]:=(AN[i,j]*PHI[i,j+1]+AS[i,j]*PHI[i,j-
1]+SU[i,j]+AW[i,j]*C[IM])*TERM;
  END;(* for *)
  FOR i:=IEND TO 2 BY -1 DO
   PHI[i,j]:=OMEGA*(A[i]*PHI[i+1,j]+C[i])+(1.-OMEGA)*PHI[i,j];
  END; (* for *)
j:=j+2;END; (* while *)

IF j=maxy THEN
  IF pos<>len-1 THEN                              (* d// *)
  A[1]:=0.;C[1]:=PHI[1,maxy];
  FOR i:=2 TO IEND DO
   IM:=i-1;
   TERM:=1./(AP[i,maxy]-AW[i,maxy]*A[IM]);
   A[i]:=AE[i,maxy]*TERM;
   C[i]:=(AN[i,maxy]*PHID[i,1]+AS[i,maxy]*PHI[i,maxy-
1]+SU[i,maxy]+AW[i,maxy]*C[IM])*TERM;
  END;(* for *)
  FOR i:=IEND TO 2 BY -1 DO
   PHI[i,maxy]:=OMEGA*(A[i]*PHI[i+1,maxy]+C[i])+(1.-OMEGA)*PHI[i,maxy];
  END; (* for *)
END END; (* if if *)                                  (* f// *)

IF k=GSN[IFI]*2 THEN handshake;RESAB:=0.;
 IF ((pos<>0) AND (JEND<>0)) THEN
    FOR i:=2 TO IEND DO
    RESL:=-
AP[i,1]*PHI[i,1]+AS[i,1]*PHIG[i,maxy]+AN[i,1]*PHI[i,2]+AW[i,1]*PHI[i-
1,1]+AE[i,1]*PHI[i+1,1]+SU[i,1];
    RESAB:=RESAB+ABS(RESL);
    END END; (* for if *)
 FOR j:=2 TO JEND DO
  FOR i:=2 TO IEND DO
  RESL:=-AP[i,j]*PHI[i,j]+AS[i,j]*PHI[i,j-
1]+AN[i,j]*PHI[i,j+1]+AW[i,j]*PHI[i-1,j]+AE[i,j]*PHI[i+1,j]+SU[i,j];
  RESAB:=RESAB+ABS(RESL);
 END END; (* for for *)
 IF pos<>len-1 THEN
   FOR i:=2 TO IEND DO
   RESL:=-AP[i,maxy]*PHI[i,maxy]+AS[i,maxy]*PHI[i,maxy-
1]+AN[i,maxy]*PHID[i,1]+AW[i,maxy]*PHI[i-
1,maxy]+AE[i,maxy]*PHI[i+1,maxy]+SU[i,maxy];
   RESAB:=RESAB+ABS(RESL);
 END END; (* for if *)
END; (* if *)
END END END (* with *)
END ligne;

BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO

k:=0;redblack:=2;
```

```
IF tredblack=3 THEN trb:=1 ELSE trb:=tredblack END;
REPEAT
k:=k+1;redblack:=3-redblack;
handshake;
IF ODD(k) THEN RESAB:=0.;IF tredblack=3 THEN trb:=3-trb END END;

IF trb=1 THEN point  ELSE ligne END;

UNTIL k=GSN[IFI]*2;
RESOR[IFI]:=RESAB;
END END END; (* with *)
handshake;
END GS;


(* -------------------------- calcu ----------------------------------
*)
PROCEDURE CALCU;
VAR deb,fin:CARDINAL;
CN,DN,CS,DS,CE,DE,DX,DY,VISN,DYNR,URFUR,HL,RESP:REAL;
PROCEDURE init;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
URFUR:=1./URFU;
(* initialize west side *)
IF pos<>0 THEN                                          (* d// *)
   DW[1]:=VIS[2,1]*(Y[1]-gauche^.Y[maxy])/(X[2]-X[1]);
   CW[1]:=0.5*(CX[2,1]+CX[1,1]);
END;                                                    (* f// *)

IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;
FOR j:=2 TO fin DO
 DW[j]:=VIS[2,j]*(Y[j]-Y[j-1])/(X[2]-X[1]);
 CW[j]:=0.5*(CX[2,j]+CX[1,j]);
END; (* for *)

END END END; (* with *)
END init;

PROCEDURE assemblage;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
IF pos<>0 THEN deb:=1 ELSE deb:=2 END;
IF pos<>len-1 THEN fin:=maxy ELSE fin:=maxy-1 END;
FOR j:=deb TO fin DO
 FOR i:=2 TO maxx-2 DO
  AP[i,j]:=(AE[i,j]+AW[i,j]+AN[i,j]+AS[i,j]+SP[i,j])*URFUR;
  SU[i,j]:=SU[i,j]+(1.-URFU)*AP[i,j]*u[i,j];
  DU[i,j]:=DU[i,j]/AP[i,j];
END END; (* for for *)
END END END; (* with *)
END assemblage;

PROCEDURE mlgauche;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
IF pos<>0 THEN                                        (* d// *)
   DY:=Y[1]-gauche^.Y[maxy];
   VISN:=(VIS[i,2]*FY[1]+VIS[i,1]*(1.-FY[1]))*(1.-
FX[i])+(VIS[i+1,2]*FY[1]+VIS[i+1,1]*(1.-FY[1]))*FX[i];
   CS:=0.5*(gauche^.CY[i,maxy]+gauche^.CY[i+1,maxy]);  (* ??? *)
```

```
   CN:=0.5*(CY[i,1]+CY[i+1,1]);
   CE:=0.5*(CX[i,1]+CX[i+1,1]);
   DYNR:=2.0/(Y[2]-gauche^.Y[maxy]);
   DS:=((VIS[i,1]*gauche^.FY[maxy]+VIS[i,1]*(1.-gauche^.FY[maxy]))*(1.-
FX[i])+(VIS[i+1,1]*gauche^.FY[maxy]+VIS[i+1,1]*(1.-
gauche^.FY[maxy]))*FX[i])*DX*(2.0/(Y[1]-gauche^.Y[maxy-1]));
(* ~ ??? *)
   DN:=VISN*DX*DYNR;
   DE:=VIS[i+1,1]*DY/(X[i+1]-X[i]);
   DU[i,1]:=DY;
   (* coef *)
   AE[i,1]:=DE;IF CE<0. THEN AE[i,1]:=AE[i,1]-CE END;
   AW[i,1]:=DW[1];IF CW[1]>0. THEN AW[i,1]:=AW[i,1]+CW[1] END;
   AN[i,1]:=DN;IF CN<0. THEN AN[i,1]:=AN[i,1]-CN END;
   AS[i,1]:=DS;IF CS>0. THEN AS[i,1]:=AS[i,1]+CS END;
   (* sour. terms *)
   SU[i,1]:=DY*(p[i,1]-p[i+1,1]);
   SP[i,1]:=0.;
   DW[1]:=DE;
   CW[1]:=CE;
END; (* if *)                                          (* f// *)
END END END; (* with *)
END mlgauche;

PROCEDURE mldroite;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
IF pos<>len-1 THEN                                        (* d// *)
   DY:=Y[maxy]-Y[maxy-1];
   VISN:=(VIS[i,maxy]*FY[maxy]+VIS[i,maxy]*(1.-FY[maxy]))*(1.-
FX[i])+(VIS[i+1,maxy]*FY[maxy]+VIS[i+1,maxy]*(1.-FY[maxy]))*FX[i];
(* ~ *)
   CS:=CN;
   CN:=0.5*(CY[i,maxy]+CY[i+1,maxy]);
   CE:=0.5*(CX[i,maxy]+CX[i+1,maxy]);
   DYNR:=2.0/(droite^.Y[1]-Y[maxy-1]);
   DS:=DN;
   DN:=VISN*DX*DYNR;
   DE:=VIS[i+1,maxy]*DY/(X[i+1]-X[i]);
   DU[i,maxy]:=DY;
   (* coef *)
   AE[i,maxy]:=DE;IF CE<0. THEN AE[i,maxy]:=AE[i,maxy]-CE END;
   AW[i,maxy]:=DW[maxy];IF CW[maxy]>0. THEN AW[i,maxy]:=AW[i,maxy]+CW[maxy]
END;
   AN[i,maxy]:=DN;IF CN<0. THEN AN[i,maxy]:=AN[i,maxy]-CN END;
   AS[i,maxy]:=DS;IF CS>0. THEN AS[i,maxy]:=AS[i,maxy]+CS END;
   (* sour. terms *)
   SU[i,maxy]:=DY*(p[i,maxy]-p[i+1,maxy]);
   SP[i,maxy]:=0.;
   DW[maxy]:=DE;
   CW[maxy]:=CE;
END; (* if *)                                          (* f// *)
END END END; (* with *)
END mldroite;

BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
init;
(* initialize south side *)
FOR i:=2 TO maxx-2 DO
 DYNR:=2.0/(Y[2]-Y[1]);
```

```
 DX:=0.5*(X[i+1]-X[i-1]);
 CN:=0.5*(CY[i,1]+CY[i+1,1]);
 DN:=(VIS[i+1,1]*FX[i]+VIS[i,1]*(1.-FX[i]))*DX*DYNR;

(* main loop *)
mlgauche;

FOR j:=2 TO maxy-1 DO
 DY:=Y[j]-Y[j-1];
 VISN:=(VIS[i,j+1]*FY[j]+VIS[i,j]*(1.-FY[j]))*(1.-
FX[i])+(VIS[i+1,j+1]*FY[j]+VIS[i+1,j]*(1.-FY[j]))*FX[i];
 CS:=CN;
 CN:=0.5*(CY[i,j]+CY[i+1,j]);
 CE:=0.5*(CX[i,j]+CX[i+1,j]);
 DYNR:=2.0/(Y[j+1]-Y[j-1]);
 DS:=DN;
 DN:=VISN*DX*DYNR;
 DE:=VIS[i+1,j]*DY/(X[i+1]-X[i]);
 DU[i,j]:=DY;
(* coefficients *)
 AE[i,j]:=DE;IF CE<0. THEN AE[i,j]:=AE[i,j]-CE END;
 AW[i,j]:=DW[j];IF CW[j]>0. THEN AW[i,j]:=AW[i,j]+CW[j] END;
 AN[i,j]:=DN;IF CN<0. THEN AN[i,j]:=AN[i,j]-CN END;
 AS[i,j]:=DS;IF CS>0. THEN AS[i,j]:=AS[i,j]+CS END;
(* source terms *)
 SU[i,j]:=DY*(p[i,j]-p[i+1,j]);
 SP[i,j]:=0.;
 DW[j]:=DE;
 CW[j]:=CE;
END; (* for j *)

mldroite;

END; (* for i *)

(* modifications *)
IF pos=0 THEN deb:=2 ELSE deb:=1 END;
IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;
FOR j:=deb TO fin DO AE[maxx-2,j]:=0. END;

(* final assembly *)
assemblage;

(* appel a la resolution du systeme *)
GS(u,gauche^.u,droite^.u,1,maxx-2,maxy-1);

END END END (* with *)
END CALCU;


(* --------------------------- calcv -----------------------------------
*)
PROCEDURE CALCV;
VAR Vmaxy,deb,fin:CARDINAL;
    VISE,DXER,CN,DN,CS,DS,CE,DE,DX,DY,VISN,DYNR,URFUR:REAL;

PROCEDURE init;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
URFUR:=1./URFV;
(* initialise west side *)
```

```
DXER:=2.0/(X[2]-X[1]);
IF pos<>0 THEN                                          (* d// *)
   DY:=0.5*(Y[2]-gauche^.Y[maxy]);
   VISE:=VIS[1,2]*FY[1]+VIS[1,1]*(1.-FY[1]);
   DW[1]:=VISE*DY*DXER;
   CW[1]:=0.5*(CX[1,1]+CX[1,2]);
END; (* if *)                                          (* f// *)

FOR j:=2 TO Vmaxy DO
 DY:=0.5*(Y[j+1]-Y[j-1]);
 VISE:=VIS[1,j+1]*FY[j]+VIS[1,j]*(1.0-FY[j]);
 DW[j]:=VISE*DY*DXER;
 CW[j]:=0.5*(CX[1,j]+CX[1,j+1]);
END; (* for *)

IF pos<>len-1 THEN                                     (* d// *)
   DY:=0.5*(droite^.Y[1]-Y[maxy-1]);
   VISE:=VIS[1,maxy]*FY[maxy]+VIS[1,maxy]*(1.-FY[maxy]);    (* ~ *)
   DW[maxy]:=VISE*DY*DXER;
   CW[maxy]:=0.5*(CX[1,maxy]+droite^.CX[1,1]);
END; (* if *)                                       (* f// *)

END END END (* with *)
END init;

PROCEDURE assembly;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
(* final assembly *)
IF pos<>0 THEN deb:=1 ELSE deb:=2 END;
IF pos<>len-1 THEN fin:=maxy ELSE fin:=maxy-2 END;
FOR j:=deb TO fin DO
 FOR i:=2 TO maxx-1 DO
  AP[i,j]:=(AE[i,j]+AW[i,j]+AN[i,j]+AS[i,j]+SP[i,j])*URFUR;
  SU[i,j]:=SU[i,j]+(1.-URFV)*AP[i,j]*v[i,j];
  DV[i,j]:=DV[i,j]/AP[i,j];
END END; (* for for *)

END END END; (* with *)
END assembly;

PROCEDURE mlgauche;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
                                                        (* d// *)
 DY:=0.5*(Y[2]-gauche^.Y[maxy]);
 DYNR:=1./(Y[2]-Y[1]);
 VISE:=(VIS[i,2]*FY[1]+VIS[i,1]*(1.-FY[1]))*(1.-
FX[i])+(VIS[i+1,2]*FY[1]+VIS[i+1,1]*(1.-FY[1]))*FX[i];
 CS:=0.5*(CY[i,1]+gauche^.CY[i,maxy]);                  (* ??? *)
 CN:=0.5*(CY[i,2]+CY[i,1]);
 CE:=0.5*(CX[i,2]+CX[i,1]);
 DS:=VIS[i,1]*DX*(1./(Y[1]-gauche^.Y[maxy]));           (* ??? *)
 DN:=VIS[i,2]*DX*DYNR;
 DE:=VISE*DY*DXER;
 DV[i,1]:=DX;
(* coef *)
 AE[i,1]:=DE;IF CE<0. THEN AE[i,1]:=AE[i,1]-CE END;
 AW[i,1]:=DW[1];IF CW[1]>0. THEN AW[i,1]:=AW[i,1]+CW[1] END;
 AN[i,1]:=DN;IF CN<0. THEN AN[i,1]:=AN[i,1]-CN END;
 AS[i,1]:=DS;IF CS>0. THEN AS[i,1]:=AS[i,1]+CS END;
```

```
(* sour terms *)
 SP[i,1]:=0.;
 SU[i,1]:=DX*(p[i,1]-p[i,2]);
 DW[1]:=DE;
 CW[1]:=CE;
                                                        (* f// *)
END END END; (* with *)
END mlgauche;

PROCEDURE mldroite;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
                                                        (* d// *)
 DY:=0.5*(droite^.Y[1]-Y[maxy-1]);
 DYNR:=1./(droite^.Y[1]-Y[maxy]);
 VISE:=(VIS[i,maxy]*FY[maxy]+VIS[i,maxy]*(1.-FY[maxy]))*(1.-
FX[i])+(VIS[i+1,maxy]*FY[maxy]+VIS[i+1,maxy]*(1.-FY[maxy]))*FX[i];      (*
~ *)
 CS:=CN;
 CN:=0.5*(droite^.CY[i,1]+CY[i,maxy]);
 CE:=0.5*(droite^.CX[i,1]+CX[i,maxy]);
 DS:=DN;
 DN:=VIS[i,maxy]*DX*DYNR;                               (* ~ *)
 DE:=VISE*DY*DXER;
 DV[i,maxy]:=DX;
(* coef *)
 AE[i,maxy]:=DE;IF CE<0. THEN AE[i,maxy]:=AE[i,maxy]-CE END;
 AW[i,maxy]:=DW[maxy];IF CW[maxy]>0. THEN AW[i,maxy]:=AW[i,maxy]+CW[maxy]
END;
 AN[i,maxy]:=DN;IF CN<0. THEN AN[i,maxy]:=AN[i,maxy]-CN END;
 AS[i,maxy]:=DS;IF CS>0. THEN AS[i,maxy]:=AS[i,maxy]+CS END;
(* sour terms *)
 SP[i,maxy]:=0.;
 SU[i,maxy]:=DX*(p[i,maxy]-droite^.p[i,1]);
 DW[maxy]:=DE;
 CW[maxy]:=CE;
                                                       (* f// *)
END END END; (* with *)
END mldroite;

BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
IF pos<>len-1 THEN Vmaxy:=maxy-1 ELSE Vmaxy:=maxy-2 END;

init;
(* initialize south side *)
FOR i:=2 TO maxx-1 DO
 DX:=X[i]-X[i-1];
 DXER:=2./(X[i+1]-X[i-1]);
 DYNR:=1./(Y[2]-Y[1]);
 DN:=VIS[i,2]*DX*DYNR;
 CN:=0.5*(CY[i,2]+CY[i,1]);

(* main loop east and north sides *)
IF ((pos<>0) AND (Vmaxy<>0)) THEN mlgauche END;

FOR j:=2 TO Vmaxy DO
 DY:=0.5*(Y[j+1]-Y[j-1]);
 DYNR:=1./(Y[j+1]-Y[j]);
 VISE:=(VIS[i,j+1]*FY[j]+VIS[i,j]*(1.-FY[j]))*(1.-
FX[i])+(VIS[i+1,j+1]*FY[j]+VIS[i+1,j]*(1.-FY[j]))*FX[i];
```

```
 CS:=CN;
 CN:=0.5*(CY[i,j+1]+CY[i,j]);
 CE:=0.5*(CX[i,j+1]+CX[i,j]);
 DS:=DN;
 DN:=VIS[i,j+1]*DX*DYNR;
 DE:=VISE*DY*DXER;
 DV[i,j]:=DX;

(* coefficients *)
 AE[i,j]:=DE;IF CE<0. THEN AE[i,j]:=AE[i,j]-CE END;
 AW[i,j]:=DW[j];IF CW[j]>0. THEN AW[i,j]:=AW[i,j]+CW[j] END;
 AN[i,j]:=DN;IF CN<0. THEN AN[i,j]:=AN[i,j]-CN END;
 AS[i,j]:=DS;IF CS>0. THEN AS[i,j]:=AS[i,j]+CS END;

(* source terms *)
 SP[i,j]:=0.;
 SU[i,j]:=DX*(p[i,j]-p[i,j+1]);
 DW[j]:=DE;
 CW[j]:=CE;
END; (* for j *)

IF pos<>len-1 THEN mldroite END;
END; (* for i *)

(* modifications *)
IF pos=0 THEN deb:=2 ELSE deb:=1 END;
IF pos=len-1 THEN fin:=maxy-2 ELSE fin:=maxy END;
FOR j:=deb TO fin DO AE[maxx-1,j]:=0. END;

(* assembly *)
assembly;

(* appel a la resolution du systeme *)
GS(v,gauche^.v,droite^.v,2,maxx-1,Vmaxy);

END END END (* with *)
END CALCV;

(* ---------------------------------- calcp ---------------------------- *)
PROCEDURE CALCP;
VAR deb,fin:CARDINAL;
VAR DENE,DENN,DPE,DPN,DX,DY:REAL;

PROCEDURE mlgauche;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
 DY:=Y[1]-gauche^.Y[maxy];                              (* d// *)
 DENE:=DEN[i+1,1]*FX[i]+DEN[i,1]*(1.-FX[i]);
 DENN:=DEN[i,2]*FY[1]+DEN[i,1]*(1.-FY[1]);
 CX[i,1]:=DENE*DY*u[i,1];
 CY[i,1]:=DENN*DX*v[i,1];
(* coef *)
 AW[i,1]:=AE[i-1,1];
 AE[i,1]:=DENE*DY*DU[i,1];
 AS[i,1]:=(DEN[i,1]*gauche^.FY[maxy]+DEN[i,1]*(1.-
gauche^.FY[maxy]))*DX*gauche^.DV[i,maxy];
(* ~ *)
 AN[i,1]:=DENN*DX*DV[i,1];
(* sour terms *)
 SP[i,1]:=0.;
```

```
 SU[i,1]:=CX[i-1,1]-CX[i,1]-
CY[i,1]+(DEN[i,1]*gauche^.FY[maxy]+DEN[i,1]*(1.-
gauche^.FY[maxy]))*DX*gauche^.v[i,maxy];          (* ~ *)
 SUM:=SUM+SU[i,1];                                (* f// *)
END END END; (* with *)
END mlgauche;

PROCEDURE mldroite;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
 DY:=Y[maxy]-Y[maxy-1];                                    (* d// *)
 DENE:=DEN[i+1,maxy]*FX[i]+DEN[i,maxy]*(1.-FX[i]);
 DENN:=DEN[i,maxy]*FY[maxy]+DEN[i,maxy]*(1.-FY[maxy]);     (* ~ *)
 CX[i,maxy]:=DENE*DY*u[i,maxy];
 CY[i,maxy]:=DENN*DX*v[i,maxy];
(* coef *)
 AW[i,maxy]:=AE[i-1,maxy];
 AE[i,maxy]:=DENE*DY*DU[i,maxy];
 AS[i,maxy]:=AN[i,maxy-1];
 AN[i,maxy]:=DENN*DX*DV[i,maxy];
(* sour terms *)
 SP[i,maxy]:=0.;
 SU[i,maxy]:=CX[i-1,maxy]-CX[i,maxy]+CY[i,maxy-1]-CY[i,maxy];
 SUM:=SUM+SU[i,maxy];                                      (* f// *)
END END END; (* with *)
END mldroite;

PROCEDURE correction;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO

IF pos=0 THEN deb:=2 ELSE deb:=1 END;

FOR j:=deb TO maxy-1 DO
 FOR i:=2 TO maxx-1 DO
  DPE:=PP[i,j]-PP[i+1,j];
  DPN:=PP[i,j]-PP[i,j+1];
  u[i,j]:=u[i,j]+DU[i,j]*DPE;
  v[i,j]:=v[i,j]+DV[i,j]*DPN;
  CX[i,j]:=CX[i,j]+AE[i,j]*DPE;
  CY[i,j]:=CY[i,j]+AN[i,j]*DPN;
  p[i,j]:=p[i,j]+URFP*(PP[i,j]-PPREF);
END END; (* for for *)

IF pos<>len-1 THEN                                        (* d// *)
 FOR i:=2 TO maxx-1 DO
  DPE:=PP[i,maxy]-PP[i+1,maxy];
  DPN:=PP[i,maxy]-droite^.PP[i,1];
  u[i,maxy]:=u[i,maxy]+DU[i,maxy]*DPE;
  v[i,maxy]:=v[i,maxy]+DV[i,maxy]*DPN;
  CX[i,maxy]:=CX[i,maxy]+AE[i,maxy]*DPE;
  CY[i,maxy]:=CY[i,maxy]+AN[i,maxy]*DPN;
  p[i,maxy]:=p[i,maxy]+URFP*(PP[i,maxy]-PPREF);
END END; (* for if *)                                    (* f// *)

END END END; (* with *)
END correction;

BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
SUM:=0.;
```

```
(* initialize west side *)
IF pos<>0 THEN CX[1,1]:=DEN[1,1]*(Y[1]-gauche^.Y[maxy])*u[1,1] END; (* //
*)

IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;

FOR j:=2 TO fin DO
 DY:=Y[j]-Y[j-1];
 CX[1,j]:=DEN[1,j]*DY*u[1,j];
END; (* for *)

(* initialize south side *)
FOR i:=2 TO maxx-1 DO
 DX:=X[i]-X[i-1];
 CY[i,1]:=DEN[i,1]*DX*v[i,1];

IF pos<>0 THEN mlgauche END;

(* main loop east and north sides *)
FOR j:=2 TO maxy-1 DO
 DY:=Y[j]-Y[j-1];
 DENE:=DEN[i+1,j]*FX[i]+DEN[i,j]*(1.-FX[i]);
 DENN:=DEN[i,j+1]*FY[j]+DEN[i,j]*(1.-FY[j]);
 CX[i,j]:=DENE*DY*u[i,j];
 CY[i,j]:=DENN*DX*v[i,j];

(* coefficients *)
 AW[i,j]:=AE[i-1,j];
 AE[i,j]:=DENE*DY*DU[i,j];
 AS[i,j]:=AN[i,j-1];
 AN[i,j]:=DENN*DX*DV[i,j];

(* source terms *)
 SP[i,j]:=0.;
 SU[i,j]:=CX[i-1,j]-CX[i,j]+CY[i,j-1]-CY[i,j];
 SUM:=SUM+SU[i,j];
END; (* for j *)

IF pos<>len-1 THEN  mldroite END;
END; (* for i *)

(* final assembly *)
IF pos=0 THEN deb:=2 ELSE deb:=1 END;
IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;

FOR j:=deb TO fin DO
 FOR i:=2 TO maxx-1 DO
  PP[i,j]:=0.;
  AP[i,j]:=AE[i,j]+AW[i,j]+AN[i,j]+AS[i,j]+SP[i,j];
END END; (* for for *)

(* appel a la resolution du systeme *)
GS(PP,gauche^.PP,droite^.PP,3,maxx-1,maxy-1);

(* corrections *)

IF pos=0 THEN REPEAT UNTIL SignalArrived(sem);
              PPREF:=PP[2,2];Signal(droite^.sem);
              REPEAT UNTIL SignalArrived(sem);Signal(sem)
        ELSE REPEAT UNTIL SignalArrived(sem);
```

```
              PPREF:=gauche^.PPREF;SUM:=SUM+gauche^.SUM;
              Signal(droite^.sem);
         END;

correction;

IF test THEN CursorPos(23,50);WriteString('Sum =
');WriteReal(gauche^.SUM,10) END;

END END END; (* with *)
END CALCP;

(* ------------------------- outbc ---------------------------------- *)
PROCEDURE OUTBC;
VAR FAC,FLOW,FLOWOLD,CE:REAL;
    deb,fin:CARDINAL;
BEGIN
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
FLOW:=0.;

IF pos<>0 THEN                                          (* d// *)
   CE:=DENSIT*(Y[1]-gauche^.Y[maxy])*u[maxx-2,1];
   FLOW:=FLOW+CE;
   CX[maxx-1,1]:=CE;
END; (* if *)                                          (* f// *)

IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;

FOR j:= 2 TO fin DO
 CE:=DENSIT*(Y[j]-Y[j-1])*u[maxx-2,j];
 FLOW:=FLOW+CE;
 CX[maxx-1,j]:=CE;
END; (* for *)

IF pos=0 THEN REPEAT UNTIL SignalArrived(sem);
              flowg:=FLOW;Signal(droite^.sem);
              REPEAT UNTIL SignalArrived(sem);
              flowg:=gauche^.flowg;Signal(droite^.sem);
         ELSE REPEAT UNTIL SignalArrived(sem);
              flowg:=gauche^.flowg+FLOW;Signal(droite^.sem);
              REPEAT UNTIL SignalArrived(sem);
              flowg:=gauche^.flowg;Signal(droite^.sem);
         END;


FAC:=FLOWIN/flowg;

IF pos=0 THEN deb:=2 ELSE deb:=1 END;
IF pos=len-1 THEN fin:=maxy-1 ELSE fin:=maxy END;

FOR j:=deb TO fin DO
 u[maxx-1,j]:=u[maxx-2,j]*FAC;
 u[maxx,j]:=u[maxx-1,j];
 CX[maxx-1,j]:=CX[maxx-1,j]*FAC;
 CX[maxx,j]:=CX[maxx-1,j];
END; (* for *)

handshake;
IF test THEN CursorPos(23,50);WriteString(' Fac= ');WriteReal(FAC,6) END;

END END END; (* with *)
```

```
END OUTBC;

(* ------------------------ maincalc / slave ----------------------- *)
PROCEDURE MAINCALC;
BEGIN
WITH ptrm^ DO
IF test THEN stat:='T';droite^.stat:='T'
        ELSE stat:='X';droite^.stat:='X' END;
REPEAT
Signal(droite^.sem);
 ITER:=ITER+1;CursorPos(23,1);WriteString('ITERATION
');WriteCard(ITER,3);
 CALCU;CALCV;OUTBC;CALCP;
REPEAT UNTIL SignalArrived(sem);
FOR i:=1 TO 3 DO err[i]:=RESOR[i] END;
Signal(droite^.sem);REPEAT UNTIL SignalArrived(sem);
FOR i:=1 TO 3 DO err[i]:=gauche^.err[i]*SNORM[i] END;
CursorPos(22,1);
WriteString('RESORU=');WriteReal(err[1],10);
WriteString('  RESORV=');WriteReal(err[2],10);
WriteString('  RESORP=');WriteReal(err[3],10);
UNTIL (((err[1]<=SORMAX) AND (err[2]<=SORMAX) AND (err[3]<=SORMAX)) OR
KeyPressed());
droite^.stat:='e';Signal(droite^.sem);
(* REPEAT UNTIL SignalArrived(sem); *)
END; (* with *)
END MAINCALC;

PROCEDURE slave;
BEGIN
WITH ptrm^ DO
droite^.stat:=stat;
REPEAT
Signal(droite^.sem);
 CALCU;CALCV;OUTBC;CALCP;
REPEAT UNTIL SignalArrived(sem);
FOR i:=1 TO 3 DO err[i]:=gauche^.err[i]+RESOR[i] END;
Signal(droite^.sem);REPEAT UNTIL SignalArrived(sem);
UNTIL stat='e';droite^.stat:='e';Signal(droite^.sem);
END; (* with *)
END slave;
(* ---------------------- write on printer ------------------------- *)

PROCEDURE MyWrite(ch : CHAR);
BEGIN
  U2Write(ch);
END MyWrite;

(* ---------------------- edit ------------------------------------- *)

PROCEDURE edit(m:TMATRIX);
VAR done : BOOLEAN;
    deb:CARDINAL;
BEGIN
  AssignWrite(MyWrite,done);

  Write(CHR(13));Write(CHR(10));
  deb:=0; WHILE deb<maxxx DO
  i:=deb;WriteString('  i=>     ');REPEAT i:=i+1;WriteCard(i,10);
         UNTIL (((i MOD 12)=0) OR (i=maxxx));Write(CHR(13));Write(CHR(10));
  i:=deb;WriteString('----------');REPEAT i:=i+1;WriteString('----------');
```

```
        UNTIL (((i MOD 12)=0) OR (i=maxxx));Write(CHR(13));Write(CHR(10));
  FOR j:=maxyy TO 1 BY -1 DO
  i:=deb;WriteString('  j=');WriteCard(j,4);WriteString('  ');REPEAT
i:=i+1;WriteReal(m[i,j],10);
        UNTIL (((i MOD 12)=0) OR (i=maxxx));Write(CHR(13));Write(CHR(10));
  END; (* for *)
  Write(CHR(13));Write(CHR(10));deb:=i;
  END; (* while *)

  UnAssignWrite(done);
END edit;
(* ---------------------- duree ------------------------------------ *)

PROCEDURE duree();
BEGIN
GetTime(mn,sec,ms);
CursorPos(23,20);WriteString('Duree : ');
WriteCard(mn,2);WriteString(' mn ');WriteCard(sec,2);WriteString(' sec ');
WriteCard(ms,3);WriteString(' ms');
END duree;

(* --------------------- programme principal ------------------------ *)
BEGIN
ClearScreen;Write('x');test:=FALSE;
NEW(ptr1);NEW(ptr2);NEW(ptrx);
NEW(ptrU);NEW(ptrV);NEW(ptrP);NEW(ptrY);NEW(ptrFY);
Allocate(ptrm,TSIZE(tmemult),0,"X");ptrm^.sem:=FALSE;
REPEAT Allocate(gauche,TSIZE(tmemult),left,"X") UNTIL gauche<>NIL;
REPEAT Allocate(droite,TSIZE(tmemult),right,"X") UNTIL droite<>NIL;
WITH ptr1^ DO WITH ptr2^ DO WITH ptrm^ DO
XMAX:=0.25;YMAX:=0.05;DENSIT:=1000.;VISCOS:=0.25;SORMAX:=0.005;
URFU:=0.8;URFV:=0.8;URFP:=0.2;GSN[1]:=5;GSN[2]:=5;GSN[3]:=40;
tredblack:=2;syncg:=0;syncd:=0;tempo:=1;OMEGA:=1.;
maxxx:=MAXX;maxyy:=12;
maxx:=maxxx;maxy:=maxyy DIV len;
stepx:=maxxx DIV 19;IF maxxx MOD 19<>0 THEN stepx:=stepx+1 END;
stepy:=maxyy DIV 22;IF maxyy MOD 22<>0 THEN stepy:=stepy+1 END;
init2(ptrU^);init2(ptrV^);init2(ptrP^);INIT;
FOR i:=1 TO MAXX DO FOR j:=1 TO MAXY DO ctrl[i,j]:=0 END END;
coef:=1.;ofsi:=0;ofsj:=0;posi:=1;posj:=1;
IF pos=0 THEN affiche('U') END;
LOOP
REPEAT UNTIL KeyPressed() OR SignalArrived(ptrm^.sem);
IF KeyPressed() THEN Read(c);IF c=CHR(27) THEN Read(c) END;
   RestartTime;
   CASE CAP(c) OF
   'T':test:=NOT(test); |
   'U':affiche('U'); |
   'V':affiche('V'); |
   'P':affiche('P'); |
   'K':matrix:='K';affctrl; |
   20c:edit(ptrP^); |
   25c:edit(ptrU^); |
   26c:edit(ptrV^); |
   'A':IF posj>1 THEN posj:=posj-1 END; |
   'B':IF posj*stepy+1+ofsj<=maxyy THEN posj:=posj+1 END; |
   'C':IF posi*stepx+1+ofsi<=maxxx THEN posi:=posi+1 END; |
   'D':IF posi>1 THEN posi:=posi-1 END; |
   '4':IF ofsi>0 THEN ofsi:=(ofsi-1) MOD stepx;affiche(matrix) END; |
   '6':IF stepx>1 THEN ofsi:=(ofsi+1) MOD stepx;affiche(matrix) END; |
   '8':IF ofsj>0 THEN ofsj:=(ofsj-1) MOD stepy;affiche(matrix) END; |
```

```
   '2':IF stepy>1 THEN ofsj:=(ofsj+1) MOD stepy;affiche(matrix) END; |
   '*':coef:=coef*10.; |
   ':':coef:=coef/10.; |
    'I':INIT;INITVAL;
        n0:=len;WHILE n0>0 DO jj:=0;n0:=n0-1;
        CursorPos(23,1);WriteString('INIT PR. N0:');WriteCard(n0,3);
        FOR j:=(n0*maxy)+1 TO (n0+1)*maxy DO jj:=jj+1;
        Y[jj]:=ptrY^[j];FY[jj]:=ptrFY^[j];
          FOR i:=1 TO maxx DO
            u[i,jj]:=ptrU^[i,j];CX[i,jj]:=ptrV^[i,j];
        END END; (* for for *)
        IF n0>0 THEN droite^.stat:='i';Signal(droite^.sem);
                     REPEAT UNTIL SignalArrived(sem) END;
        END; (* while *)
        duree();affiche(matrix); |
    'J':init2(ptrU^);init2(ptrV^);init2(ptrP^);affiche(matrix);ITER:=0; |
    '?':INITCOEF;affiche(matrix); |
    'S':MAINCALC;duree(); |
    'R':k:=0;ptrx^:=ptrm^;REPEAT jj:=0;
       CursorPos(23,1);WriteString('REC. PR. N0:');WriteCard(k,3);
         FOR j:=(k*maxy)+1 TO (k+1)*maxy DO jj:=jj+1;
          FOR i:=1 TO maxx DO
           ptrU^[i,j]:=u[i,jj];ptrV^[i,j]:=v[i,jj];ptrP^[i,j]:=p[i,jj];
         END END; (* for for *)
       IF k<>len-1 THEN
ptrm^:=droite^;droite^.stat:='R';Signal(droite^.sem);
          REPEAT UNTIL SignalArrived(sem) END;
       k:=k+1 UNTIL k>len-1;ptrm^:=ptrx^;affiche(matrix); |
    '/':CASE matrix OF
       'U':ReadReal(r);ptrU^[(posi-1)*stepx+1+ofsi,(posj-
1)*stepy+1+ofsj]:=r; |
       'V':ReadReal(r);ptrV^[(posi-1)*stepx+1+ofsi,(posj-
1)*stepy+1+ofsj]:=r; |
       'P':ReadReal(r);ptrP^[(posi-1)*stepx+1+ofsi,(posj-
1)*stepy+1+ofsj]:=r; |
       'K':ReadCard(k);ctrl[(posi-1)*stepx+1+ofsi,(posj-1)*stepy+1+ofsj]:=k
       END; |
  33c:droite^.stat:='f';Signal(droite^.sem);CursorPos(23,0);EXIT
   ELSE (* WriteCard(ORD(c),4) *) Write(CHR(7))
   END;CursorPos(posj,posi*4)
ELSE (* signalarrived *)
   IF pos=0 THEN Write(CHR(7))
            ELSE
   CASE ptrm^.stat OF
   ' ':droite^.stat:=' ';Signal(droite^.sem); |
   'i':IF gauche^.n0=pos THEN ptrm^:=gauche^;droite^.stat:=' ';INIT
                         ELSE ptrm^:=gauche^;droite^.stat:='i' END;
       Signal(droite^.sem); |
   'R':ptrm^:=droite^;droite^.stat:='R';Signal(droite^.sem); |
   'f':droite^.stat:='f';Signal(droite^.sem);EXIT; |
   'X':slave; |
   'T':slave;
   END END;
END; (* if *)
END;(* loop *)
END END END (* with *)
END BCDIG.
```

Sommaire