Parallelisation d'un algorithme de gradient conjugue preconditionne

Jocelyne Erhel

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 189

PARALLÉLISATION D'UN ALGORITHME DE GRADIENT CONJUGUÉ PRÉCONDITIONNÉ

Jocelyne ERHEL

Février 1983

PARALLELISATION D'UN ALGORITHME DE GRADIENT CONJUGUE PRECONDITIONNE

Jocelyne ERHEL

Decembre 1982

Résumé :

La résolution de systèmes linéaires creux intervient fréquemment dans la Méthode des Eléments Finis. On utilise souvent un algorithme du Gradient Conjugué Préconditionné par une Factorisation Incomplète.
On présente une version de cette méthode adaptée à des architectures Multi-Processeurs-Vectoriels.
Des estimations théoriques et simulations numériques prouvent l'efficacité du parallélisme MIMD mis en oeuvre.

Abstract :

The resolution of sparse linear systems appears frequently in Finite Element Methods. The Conjugated Gradient Algorithm Preconditionnated by an Incomplete Cholesky Factorization is often used.
We present a version of this method which is well-suited to Multi-Vector-Processors machines. Estimations and numerical simulations prove the efficiency of the defined MIMD parallelism.

==========

I. Introduction

Les méthodes numériques employées dans les sciences de l'ingénieur nécessitent actuellement beaucoup de calculs, qui doivent en général être exécutés très rapidement.
L'emploi de calculateurs parallèles semble une voie prometteuse pour accroître la vitesse d'exécution des ordinateurs. Il est nécessaire de développer des algorithmes adaptés à ces architectures pour en exploiter au mieux le parallélisme. On s'intéresse plus particulièrement à des machines multi-processeurs où chaque unité de calcul peut effectuer des opérations vectorielles. Des constructeurs, notamment Cray, s'orientent actuellement dans cette voie. [6]

Dans la Méthode des Eléments Finis, une phase fondamentale des calculs résout un ou plusieurs systèmes linéaires, d'ordres très élevés. En outre, une géométrie et un maillage complexe du domaine étudié conduisent à des matrices creuses et irrégulières. Il s'agit alors de trouver un algorithme de résolution qui permette d'exploiter l'aspect MIMD de la machine et le caractère vectoriel des processeurs.

Dans une première étape qui fait l'objet du présent article, on dégage une structure matricielle par blocs du système à résoudre. Les blocs indépendants sont traités en parallèle. Dans une deuxième étape, on étudiera de manière fine la structure interne des blocs de façon à mettre en évidence des calculs vectoriels. [9]
Les outils utilisés sont le graphe de la matrice et les méthodes de renumérotation [7][11][18][20]. On ne s'intéresse toutefois pas au graphe de remplissage, mais uniquement aux dépendances de données.

II. Algorithme de résolution

1) Choix de la méthode

Il s'agit de résoudre le système linéaire

(1) $Ax = b \quad A \in M_N(\mathbb{R}) \quad x \in \mathbb{R}^N \quad b \in \mathbb{R}^N$

où A est une matrice creuse d'ordre N issue d'une Méthode d'Eléments Finis appliquée à un domaine quelconque.

La méthode du Gradient Conjugué préconditionné par une factorisation incomplète s'avère très efficace dans de nombreux cas [15] . Le préconditionnement accélère notablement la vitesse de convergence. Le choix de la renumérotation semble par contre peu l'affecter [13]. Contrairement aux méthodes directes, l'algorithme ne pose aucun problème de remplissage. On peut donc choisir une renumérotation des noeuds adaptée au calcul parallèle. En outre, la vectorisation de la plupart des opérateurs mathématiques de l'algorithme est immédiate. Seules les résolutions (descente/remontée) du système linéaire (préconditionnement) posent des problèmes d'adaptation au calcul parallèle.

2) Préconditionnement

La factorisation incomplète de Choleski est un préconditionnement qui accélère notablement la vitesse de convergence de l'algorithme.

Soit $G_A = (X, E_A)$ le graphe de A et soit $G_L = (X, E_L)$ le graphe de la matrice $(L + {}^tL)$ donné a priori. On cherche une décomposition de la forme :

$$A = L\,L^T + R \quad \text{où : } R_{ij} \neq 0 \rightarrow (v_i, v_j) \notin E_L$$

Les résultats suivants prouvent l'existence d'une telle décomposition sous certaines hypothèses (il ne suffit pas que A soit définie positive, contrairement au cas de la factorisation complète).

Théorème [15] : si A est une M-matrice symétrique ou si A est une matrice définie positive à diagonale dominante, alors la factorisation incomplète est possible pour tout graphe G_L.

Remarque : Dans la suite, on considère le graphe $G_L = G_A$.

Remarque : Une factorisation incomplète de Crout ($A = L\,D\,L^T + R$) conduit à un algorithme similaire.

3) Algorithme séquentiel

Vecteurs u, s, r, d

matrice-creuse A

matrice creuse triangulaire L

$u = u_o$; $s = Au-b$; $r = L^{-t}L^{-1}s$; $d = -r$; $\alpha = \langle s, r \rangle$;

Boucle : $r = Ad$;

$\lambda = \alpha \langle d, r \rangle$; $u = u + \lambda d$; $s = s + \lambda r$;

(2) $r = L^{-t}L^{-1}s$; $\beta = \langle r, s \rangle$;

si $\beta \leq \varepsilon$ aller à Fin ;

$\gamma = \beta/\alpha$; $\alpha = \beta$; $d = -r + \gamma d$;

aller à Boucle

Fin : $x = u$; Stop

A chaque itération, on doit donc effectuer :

1 produit matrice-vecteur

1 descente de système triangulaire

1 remontée de système triangulaire

2 produits scalaires de vecteurs

3 additions vectorielles.

Il faut adapter la résolution des systèmes triangulaires au calcul parallèle, à la fois MIMD et vectoriel.

Les autres opérateurs se parallélisent sans problème.

III. Algorithme parallèle

Il s'agit de résoudre sur plusieurs processeurs vectoriels les systèmes linéaires triangulaires (2).

Remarque : La factorisation de la matrice A se parallélise de manière analogue. On ne détaille donc que l'aspect résolution (descente-remontée).

1) Structure matricielle par blocs

Le graphe de A permet de déterminer les dépendances de données de l'algorithme. Le graphe d'élimination de la factorisation incomplète ne comporte aucun arc de remplissage

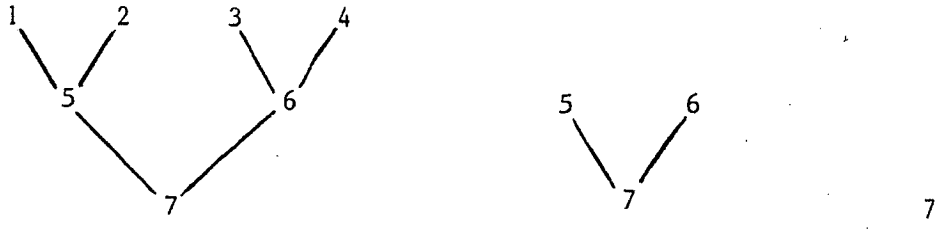
graphe initial

élimination du noeud 1 avec remplissage

élimination du noeud 1 sans remplissage

Figure 1

Des noeuds indépendants dans le graphe de A peuvent être éliminés en parallèle. La même remarque s'applique à des blocs non connectés d'une matrice-bloc. (On étudie alors le graphe-quotient représentant la structure par blocs). [12]

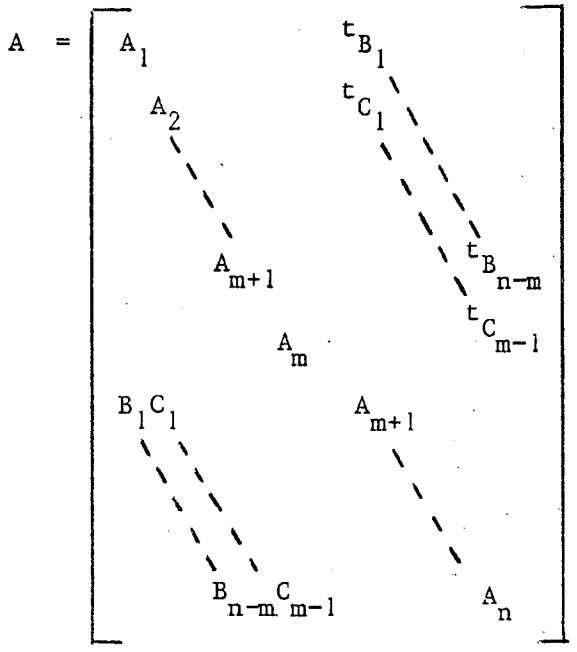


Elimination simultanée : 1) des noeuds 1, 2, 3, 4
2) des noeuds 5, 6.

Figure 2

On choisit une renumérotation des noeuds de façon à optimiser la parallélisme. On ne cherche pas à minimiser le remplissage [11] [18] .

Une renumérotation par fronts de type Cuthill-Mac Kee [7] détermine une matrice bloc-tridiagonale. Après renumérotation des fronts dans l'ordre impair puis pair, on obtient la structure matricielle suivante :



où n est le nombre de fronts
et m = [n/2] est le nombre de fronts impairs.

Figure 3

Les fronts impairs (blocs A_i $1 \leq i \leq m$) se traitent en parallèle sur différents processeurs, de même que les fronts pairs (blocs A_i $m+1 \leq i \leq n$).

Remarque : Dans le cas d'un domaine et d'un maillage réguliers, les fronts sont identiques. On peut alors vectoriser à travers les fronts.[14] [17] Dans le cas présent d'un maillage quelconque, cette méthode nécessite des adaptations et fera l'objet d'une étude ultérieure.

2) Structure interne des blocs

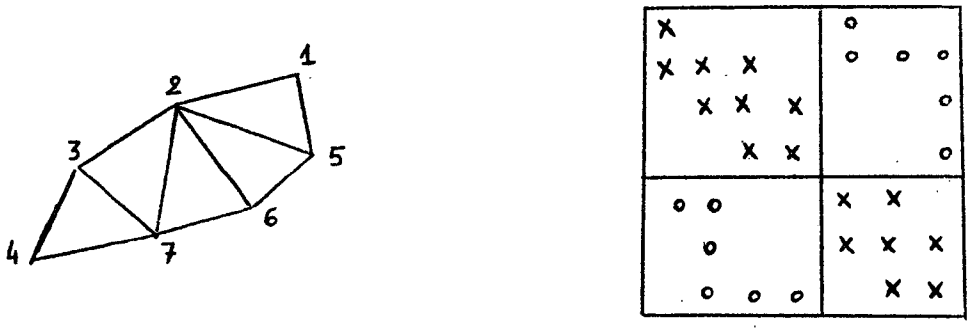
L'existence de blocs indépendants autorise un taux de parallélisme élevé d'un point de vue multi-tâches. Il faut également vectoriser les calculs effectués dans chaque bloc.

On doit :

- inverser les blocs diagonaux A_i $(1 \leq i \leq n)$
- effectuer des produits $B_i\, x_j$ $(1 \leq i \leq n-m)$
  $C_i\, x_j$ $(1 \leq i \leq m)$

où les vecteurs x_j sont connus.

Un choix convenable du front initial et de la renumérotation interne de chaque front définissent en général une structure tridiagonale des blocs A_i $(1 \leq i \leq n)$. La réduction cyclique pair-impair permet alors d'inverser rapidement ces blocs sur des processeurs vectoriels [14].



maillage matrice-bloc

Figure 4

Les blocs extra-diagonaux B_i $(1 \leq i \leq n-m)$ et C_i $(1 \leq i \leq m-1)$ présentent en général une structure irrégulière. Toutefois, les opérations dont ils font l'objet sont purement vectorielles et s'effectuent à l'aide de primitives du type Scatter-Gather. [6]

Les paragraphes suivants détaillent la partie multi-processeurs de l'algorithme. L'aspect vectoriel sera étudié dans un autre article [5].

IV. Contrôle des tâches

1) Graphe de dépendances

On s'intéresse dorénavant à la mise en oeuvre de l'algorithme MIMD. On abordera ultérieurement la vectorisation des calculs sur chaque front.

La renumérotation choisie a dégagé une structure matricielle adaptée au calcul parallèle. On décrit maintenant l'algorithme à l'aide d'un graphe [5]. Les noeuds du graphe sont des tâches qui représentent les calculs effectués sur un front donné. Un processeur exécute une tâche sans communiquer avec les autres. Les tâches non connectées dans le graphe peuvent être lancées simultanément.

Les arcs du graphe définissent les dépendances de données entre les tâches, qui nécessitent des synchronisations entre les processeurs (figures 6, 7, 8, 9).

2) Synchronisation

La répartition des tâches entre les processeurs s'effectue dynamiquement à l'initialisation. Le contrôle est distribué. Des primitives de synchronisations réalisent la coopération des processeurs (figure 10).

- La primitive ENVOYER valide le résultat d'une tâche.
- La primitive RECEVOIR est bloquante jusqu'à réception de la donnée utilisée.

La notation PRSC désigne le calcul arborescent d'un produit scalaire et la transmission du résultat final (figures 8, 9, 10).

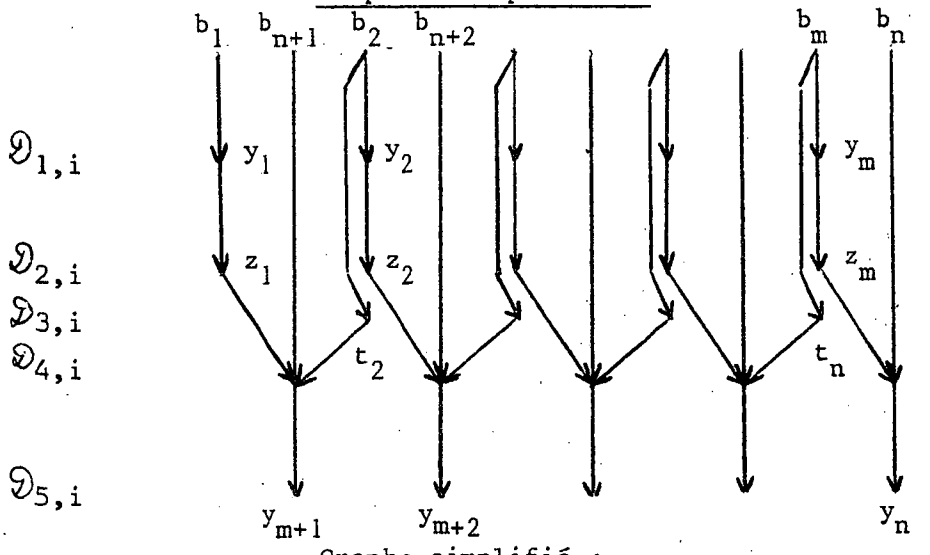
Ddescente : résolution de Ly = b

$$L = \begin{bmatrix} L_1 & & & \\ & L_m & & \\ R_1 S_1 & & L_{m+1} & \\ & & & \\ R_{n-m} S_{m-1} & & & L_n \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ \\ \\ y_n \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ \\ \\ b_n \end{bmatrix}$$

Algorithme :

$\mathcal{D}_{1,i}$	$y_i = L_i^{-1} b_i$	$1 \le i \le m$
$\mathcal{D}_{2,i}$	$z_i = R_i y_i$	$1 \le i \le n-m$
$\mathcal{D}_{3,i}$	$t_i = S_{i-1} y_i$	$2 \le i \le m$
$\mathcal{D}_{4,i}$	$b_{i+m} = b_{i+m} - z_i - t_{i+1}$	$1 \le i \le n-m$
$\mathcal{D}_{5,i}$	$y_{i+m} = L_{i+m}^{-1} b_{i+m}$	$1 \le i \le n-m$

Graphe de dépendances :



Graphe simplifié :

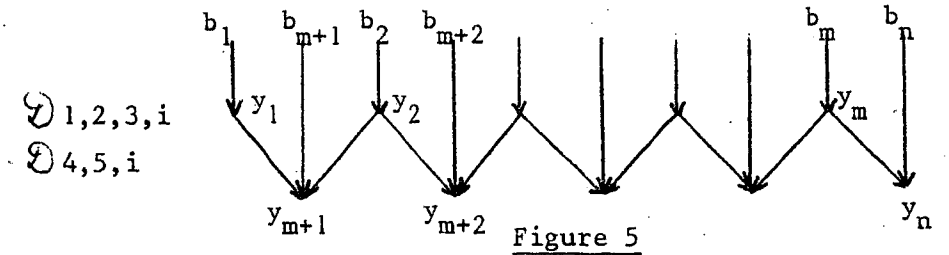


Figure 5

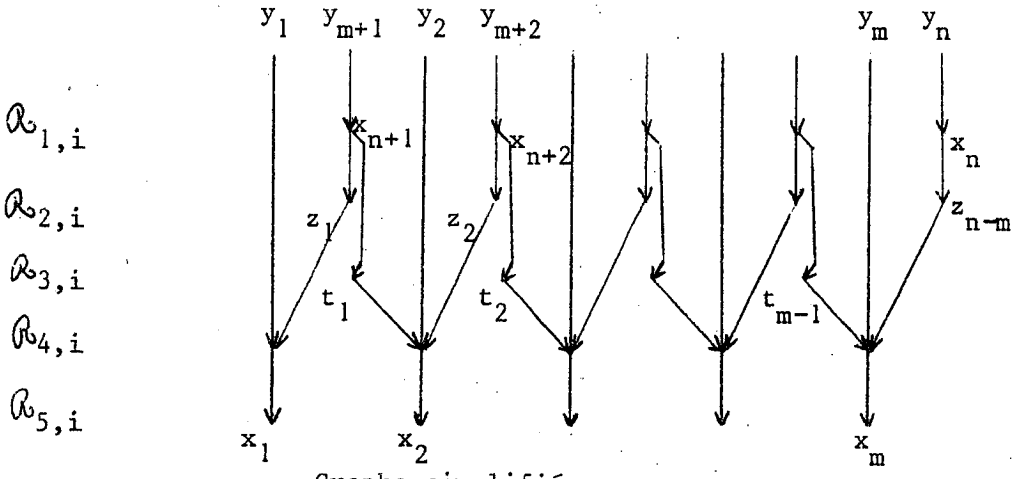
REMONTEE : RESOLUTION DE $^tLx = y$

$$^tL = \begin{bmatrix} ^tL_1 & & ^tR_1 & & \\ & ^tL_2 & ^tS_1 & & \\ & & & & \\ & ^tL_m & & ^tS_{m-1} & ^tR_{n-m} \\ & & ^tL_{m+1} & & \\ & & & & \\ & & & & ^tL_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \\ \\ \\ y_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ \\ \\ \\ x_n \end{bmatrix}$$

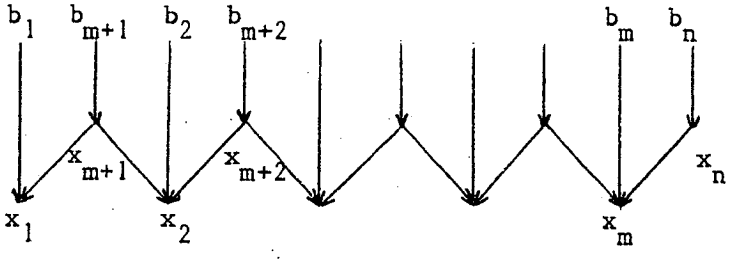
Algorithme :

$\mathcal{R}_{1,i}$	$x_{i+m} = L_{i+m}^{-t} y_{i+m}$		$1 \le i \le n-m$
$\mathcal{R}_{2,i}$	$z_i = {}^tR_i \, x_{i+m}$		$1 \le i \le n-m$
$\mathcal{R}_{3,i}$	$t_i = {}^tS_i \, x_{i+m}$		$1 \le i \le m-1$
$\mathcal{R}_{4,i}$	$y_i = y_i - z_i - t_{i-1}$		$1 \le i \le m$
$\mathcal{R}_{5,i}$	$x_i = L_i^{-t} y_i$		$1 \le i \le m$

Graphe de dépendances :



Graphe simplifié :



dépendances de données

Figure 6

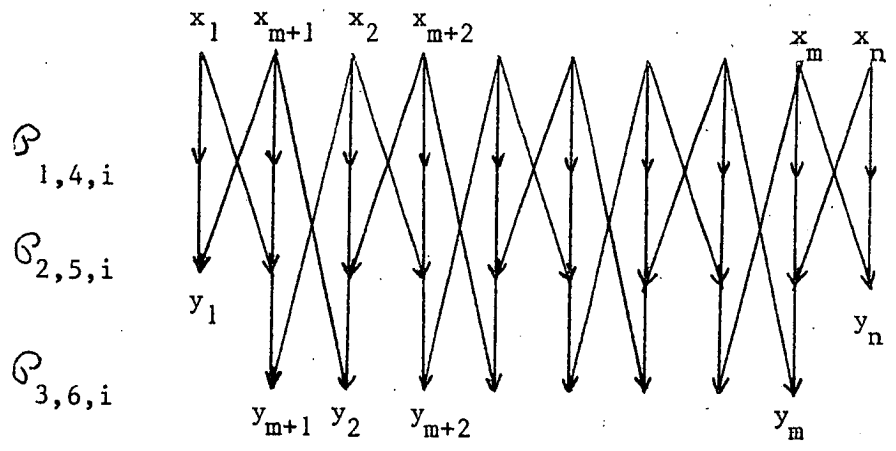
PRODUIT MATRICE-VECTEUR y = Ax

$$A = \begin{bmatrix} A_1 & & & {}^tB_1 & \\ & A_2 & & {}^tC_1 & \\ & & A_m & {}^tC_{m-1} & {}^tB_{n-m} \\ B_1 & C_1 & & A_{m+1} & \\ & & B_{n-m} & C_{m-1} & A_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ \\ \\ \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \\ \\ \\ y_n \end{bmatrix}$$

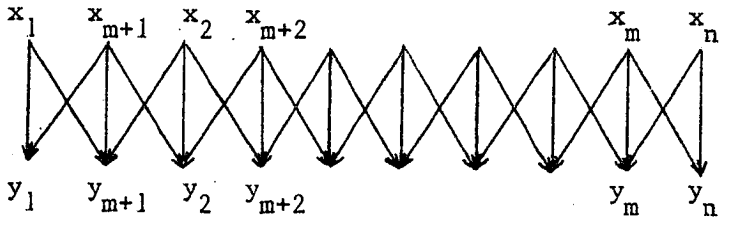
Algorithme :

$\mathcal{P}_{1,i}$	$y_i = A_i x_i$	$1 \le i \le m$
$\mathcal{P}_{2,i}$	$y_i = y_i + {}^tB_i x_{i+m}$	$1 \le i \le n-m$
$\mathcal{P}_{3,i}$	$y_i = y_i + {}^tC_i x_{i+m-1}$	$2 \le i \le m$
$\mathcal{P}_{4,i}$	$y_{i+m} = A_{i+m} x_{i+m}$	$1 \le i \le n-m$
$\mathcal{P}_{5,i}$	$y_{i+m} = y_{i+m} + B_i x_i$	$1 \le i \le n-m$
$\mathcal{P}_{6,i}$	$y_{i+m} = y_{i+m} + C_i x_{i+1}$	$1 \le i \le m-1$

Graphe de dépendances :



Graphe simplifié :



dépendances de données

Figure 7

PRODUIT SCALAIRE S = <x,y>

$$x = \begin{bmatrix} x_1 \\ \\ x_n \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ \\ y_n \end{bmatrix}$$

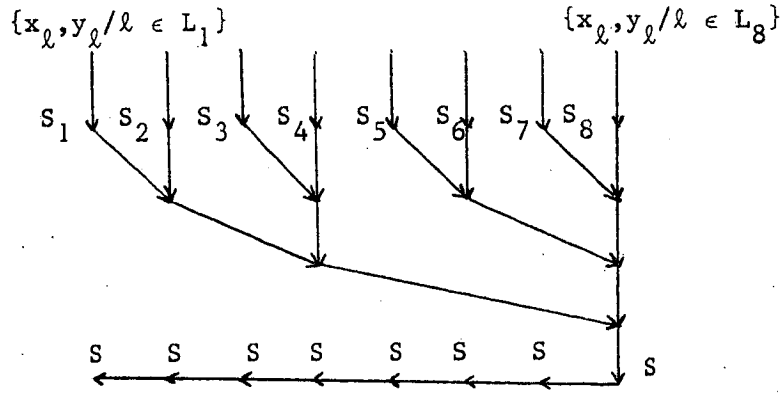
Algorithme :

- Calcul des sommes partielles S_i $(1 \leq i \leq P)$ sur chaque processeur

  $S_i = \sum_{\ell \in L_i} \langle x_\ell, y_\ell \rangle \qquad L_i = \{\ell / num(\ell) = i\}$

- Sommation arborescente des résultats S_i avec synchronisations adéquates du type ENVOYER-RECEVOIR (la structure d'arbre minimise la durée d'exécution [8]).

Graphe de dépendances :



Notation

Dans ce qui suit, le graphe ci-dessus est noté PRSC et schématisé comme suit :

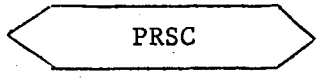
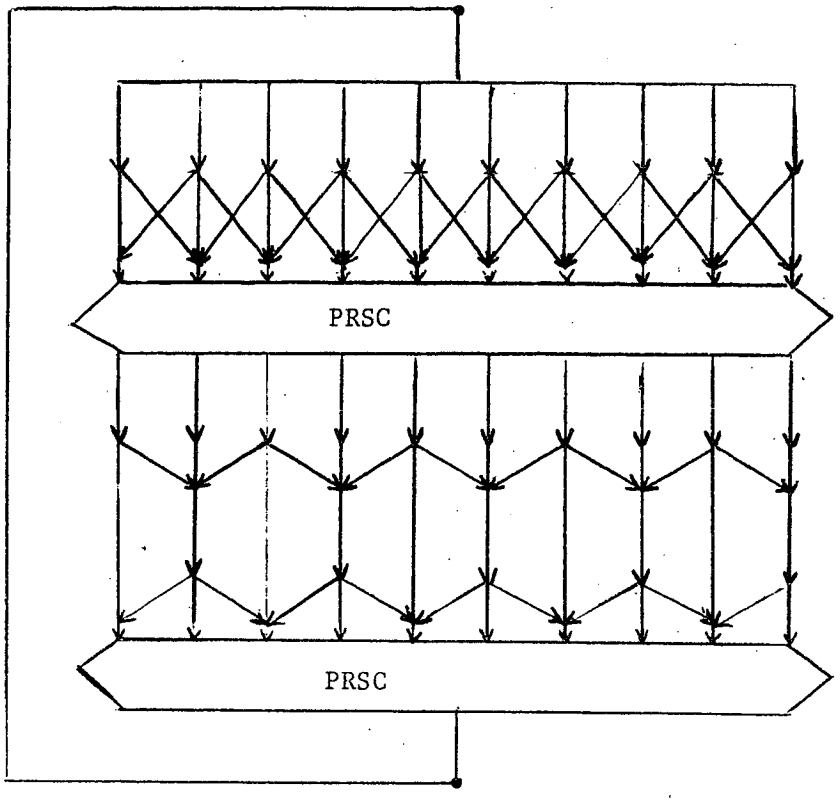


Figure 8

GRAPHE DE DEPENDANCE DE L'ALGORITHME



addition vectorielle

produit matrice-vecteur

produit scalaire
calculs scalaires

addition vectorielle

descente

remontée

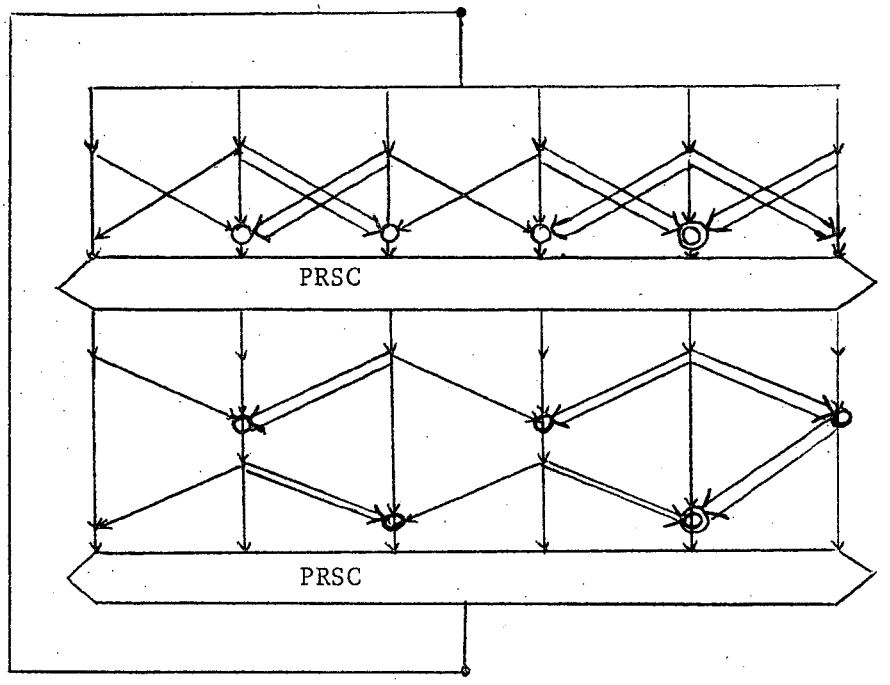
produit scalaire
test de convergence

dépendances de données

PRSC — calcul du produit scalaire (suivi d'un FORK).

Figure 9

UN EXEMPLE DE SYNCHRONISATIONS



addition vectorielle

produit matrice-vecteur

produit scalaire

descente

remontée

produit scalaire

⟹ PRIMITIVE ENVOYER

o PRIMITIVE RECEVOIR

PRSC NOTATION DESIGNANT LE PRODUIT SCALAIRE

Exemple : P = 4 n = 6

num(1) = num(2) = 1

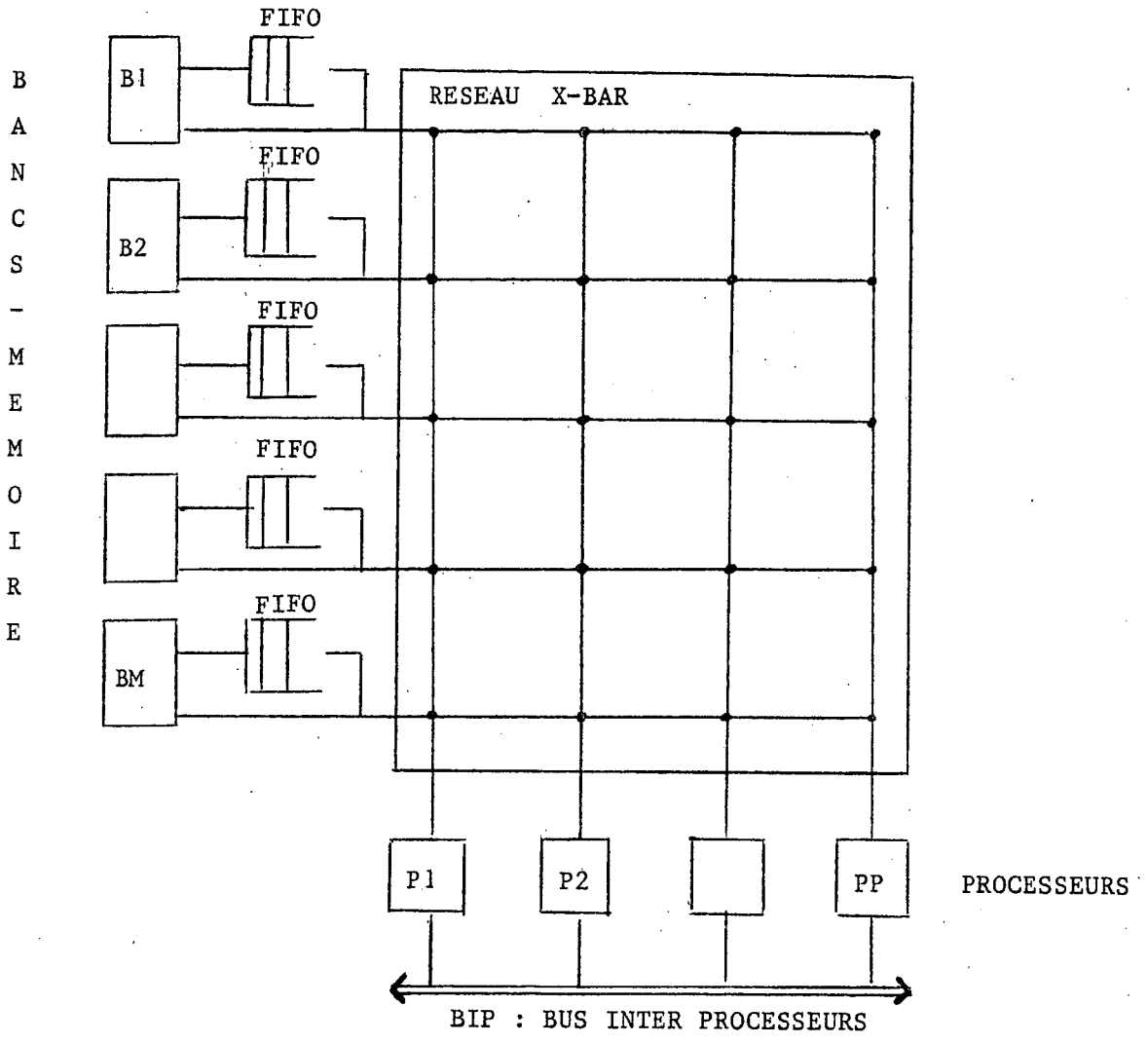
num(3) = num(4) = 2

num(5) = 3

num(6) = 4

Figure 10

ARCHITECTURE DE MUPI



- une mémoire partagée divisée en bancs dotés de queues de type FIFO
- des processeurs indépendants et asynchrones
- un réseau de connexion de type cross-bar
- un Bus Inter Processeurs pour des signaux de relance.

Figure 11

V. Simulation de l'algorithme

1) Description du simulateur

MUPI est un simulateur de machine MIMD qui permet la programmation effective d'algorithmes parallèles et la mesure de performances [8] [9] .

Les processeurs accèdent à une mémoire partagée et divisée en bancs par l'intermédiaire d'un réseau cross-bar.
Les conflits-mémoire sont gérés par des files d'attente de type FIFO (figure 11).

Les processeurs communiquent entre eux par des variables partagées et se synchronisent grâce à des primitives élémentaires opérant sur des sémaphores (instructions Test and Set, Test and Clear, Wait).

2) Répartition des tâches

Dans cette simulation, on a fixé l'allocation des tâches, alors qu'elle serait gérée dynamiquement par le système d'une machine réelle.

Dans le cas séquentiel, le travail effectué sur un processeur est sensiblement proportionnel au nombre de noeuds traités. L'algorithme de répartition des tâches attribue les fronts aux processeurs en équilibrant le partage des noeuds. Il est exécuté à l'initialisation.

Dans le cas vectoriel, il faut tenir compte de la structure interne des blocs matriciels pour optimiser la répartition des fronts [9].

3) Synchronisations

Les primitives ENVOYER et RECEVOIR, de même que le calcul en parallèle d'un produit scalaire (noté PRSC), sont programmés à l'aide de sémaphores et d'instructions de synchronisations.

Proposition : Les synchronisations respectent les dépendances de données, sans risque d'interblocage ni de famine [8] .

4) Stockage de la matrice A

Le stockage de la matrice tient compte de la structure bloc : [11]

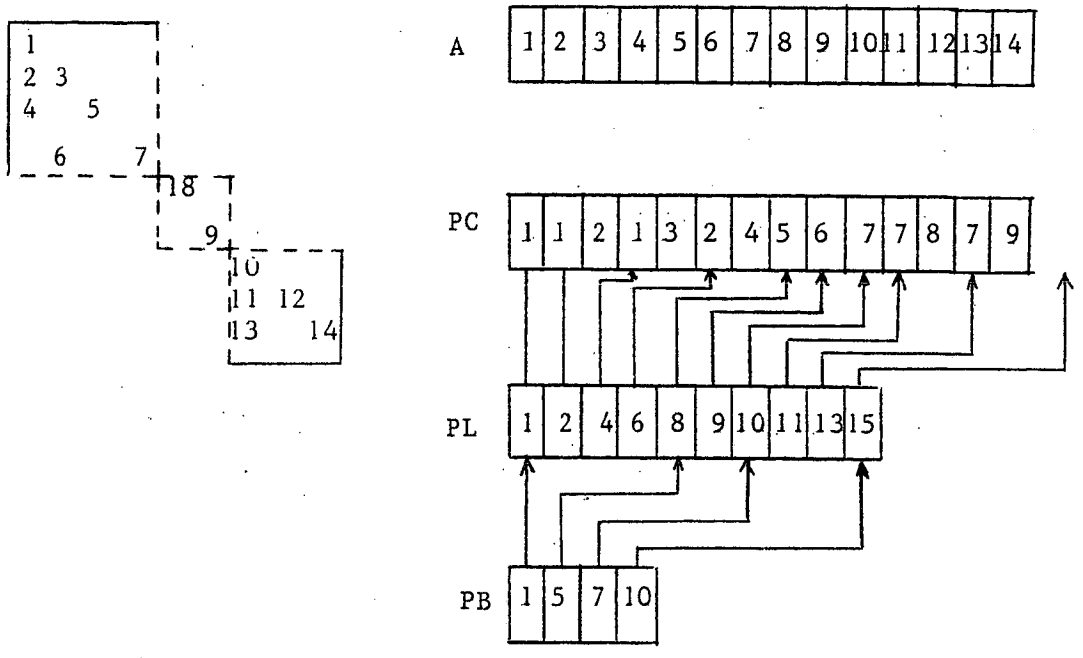


Figure 12

A : vecteur contenant les termes non nuls
PC(k) : numéro de colonne de l'élément A(k)
PL(i) : indice dans A et PC du 1er élément non nul de la ligne i
PB(l) : numéro de la lère ligne du bloc l

Le lème bloc contient les lignes de numéro i $PB(l) \leq i < PB(l+1)$
La ième ligne contient les éléments A(k) $PL(i) \leq k < PL(i+1)$
A(k) est l'élément a_{ij}, avec j = PC(k)

Dans le cas vectoriel, il faut en outre définir un stockage de chaque bloc adapté au calcul vectoriel. [9]

VI. Evaluations des performances

1) Hypothèses et notations

- On suppose que tous les noeuds ont le même nombre de voisins. Dans le cas de processeurs non vectoriels, le nombre d'opérations effectuées est alors sensiblement proportionnel au nombre de noeuds traités.

- On suppose que le traitement en parallèle ne modifie pas la vitesse de convergence. On peut donc estimer l'accélération en calculant la durée d'une itération.

- Les coûts supplémentaires dus aux synchronisations et aux conflits-mémoire sont faibles devant la durée d'exécution.

- La durée des conflits-mémoire est proportionnelle au nombre de noeuds traités.

- P est le nombre de processeurs

- M est le nombre de bancs-mémoire

- N est le nombre de noeuds (ordre du système)

- n est le nombre de fronts (i.e. de blocs)

- NMAX (P,N,n) est le nombre maximal de noeuds attribués à un processeur.

- T(1,N) est la durée séquentielle d'une itération.

- T(P,M,N) est la durée parallèle d'une itération.

- S(P,M,N) = T(1,N)/T(P,M,N) est l'accélération.

- E(P,M,N) = S(P,M,N)/P est l'efficacité.

2) Estimations

Par hypothèse $T(1,N) = tN$.

Pour calculer T(P,M,N) on distingue :

- la durée d'exécution $T_e(P,M,N)$
- la durée des synchronisations $T_s(P,M,N)$
- la durée des conflits-mémoire $T_c(P,M,N)$

On a les estimations suivantes :

$$T_e(P,M,N) = t\ NMAX(P,N,n)$$

$$T_s(P,M,N) = \alpha_s(P,N)\ \ T(P,M,N)$$

$$T_c(P,M,N) = \alpha_c(P,M)\ \ T(P,M,N)$$

où : α_s est le taux moyen de synchronisations par processeur. α_s est indépendant de M car les attentes sont réalisées sans envoyer de requête-mémoire superflue ;
et α_c est le taux moyen de conflits-mémoire par processeur. T_c et T sont proportionnels à NMAX, donc α_c est indépendant de N.

Proposition : L'accélération obtenue par l'algorithme parallèle vaut

$$S(P,M,N) = (1-\alpha_s(P,N)-\alpha_c(P,M))\ N/NMAX(P,N,n)$$

L'accélération maximale vaut donc

$$SMAX(P,M,N) = N/NMAX(P,N,n)$$

3) Modélisation des conflits-mémoire

De nombreux auteurs ont analysé de manière générale les conflits-mémoire dans un système multiprocesseurs, à l'aide par exemple de chaînes de Markov [2] [3] [4] . Le modèle proposé ici repose sur des hypothèses simplificatrices.
Il permet de trouver une bonne approximation du taux des conflits-mémoire dans l'algorithme étudié.

On modélise l'architecture par un système clients/serveurs du type D/D/M/P/P [1], où les M serveurs sont les bancs-mémoire et les P clients les processeurs.

Le modèle est dérivé des hypothèses suivantes :

- Les données sont également réparties sur les bancs-mémoire (les serveurs sont donc tous identiques).
- Les tâches de calculs sont équidistribuées entre les processeurs (les clients sont donc tous identiques).
- Les temps de réponse de la mémoire et de traversée du réseau sont constants. (d'où un temps de service constant).
- Les requêtes-mémoire sont uniformément distribuées dans le programme (d'où un intervalle de temps constant entre deux requêtes).
- Une queue de type FIFO gère les conflits d'accès aux serveurs.

MODELISATION DES CONFLITS-MEMOIRE

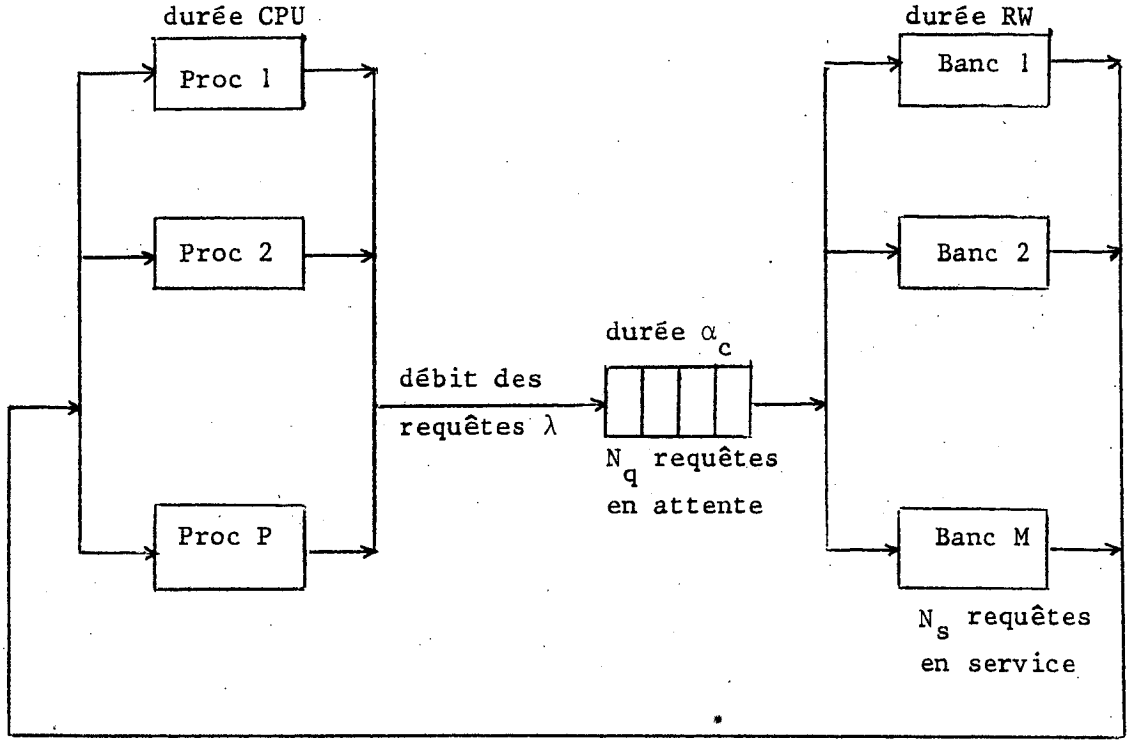


Figure 13

Proposition :

Soit P le nombre de processeurs et M le nombre de bancs-mémoire. Soit CPU la durée moyenne de calcul entre deux requêtes-mémoire et RW le temps moyen de réponse du système mémoire/réseau. Alors le taux moyen α_c de conflits-mémoire (relatif à la durée totale d'exécution) est donné par :

$$\boxed{\alpha_c = (P/(1 + \frac{CPU}{RW})\ M - 1)^+}$$

Preuve :

En régime stationnaire, chaque processeur effectue des cycles de durée $(CPU+RW)(1+\alpha_c)$.

Soit λ le débit des requêtes. On a alors d'une part :

$$\lambda = P/(CPU+RW)(1+\alpha_c)$$

Et d'autre part :

$$\lambda = \min\ (\frac{P}{CPU+RW}, \frac{M}{RW})$$

D'où l'on tire α_c. Dans le 1er cas, le débit est suffisant pour ne créer aucun conflit-mémoire.

VII. Résultats

L'algorithme a été programmé et exécuté sur MUPI. On a analysé différents contextes, en faisant varier le maillage et les paramètres de simulation (nombre de processeurs, de bancs-mémoire, vitesse). On a mesuré les accélérations obtenues dans chaque cas, ainsi que les coûts supplémentaires dus aux conflits-mémoire et aux synchronisations.

1) Maillage

On a étudié trois cas (figures 14 et 15)

- GR : Une Grille Régulière rectangulaire d'Eléments Finis P1 comportant NX × NY noeuds.
- A1 : Un anneau d'Eléments Finis P1 comportant N = 649 noeuds avec le front initial 1 (maillage réalisé à l'aide du logiciel MODULEF [16]).
- A2 : Le même anneau avec le front initial 2.

Les résultats obtenus sont analogues dans les trois cas, et comparables aux évaluations théoriques.

On a mesuré la variation des performances en fonction des dimensions NX et NY de la grille (GR). (figure 17). Les nombres de processeurs et de bancs-mémoire sont fixés à 10.
L'influence de la taille des fronts NX est négligeable. Le nombre de fronts NY conditionne la répartition des tâches et par suite l'accélération.

2) Nombre de processeurs P

On a fait varier le nombre de processeurs P en gardant un nombre de bancs-mémoire égal à P.

- L'accélération croît sensiblement linéairement avec P jusqu'à P = 10. L'efficacité est alors supérieure à 80 %. Puis l'efficacité décroît en restant supérieure à 70 % jusqu'à P = 15. L'accélération maximale obtenue vaut environ 10 (figures 16, 18, 19).
- Le taux de conflits-mémoire est indépendant de P, et avoisine 7 % (figure 20).

Par contre, le coût des synchronisations est lié à P. (figure 21).
En effet, la répartition des tâches dépend du nombre des processeurs, et induit des attentes plus ou moins importantes.

3) Nombre de bancs-mémoire M

Pour un nombre de processeurs P donné, on a fait varier le nombre de bancs-mémoire M.

- L'accélération croît fortement jusqu'à une valeur maximale atteinte pour M = P environ (figure 22). Cette variation est due aux conflits-mémoire qui sont inversement proportionnels à M (figures 24, 25).

- Les synchronisations sont comme prévu indépendantes de M, grâce à l'usage du Bus Inter Processeur (figure 23).

4) Vitesse de la machine

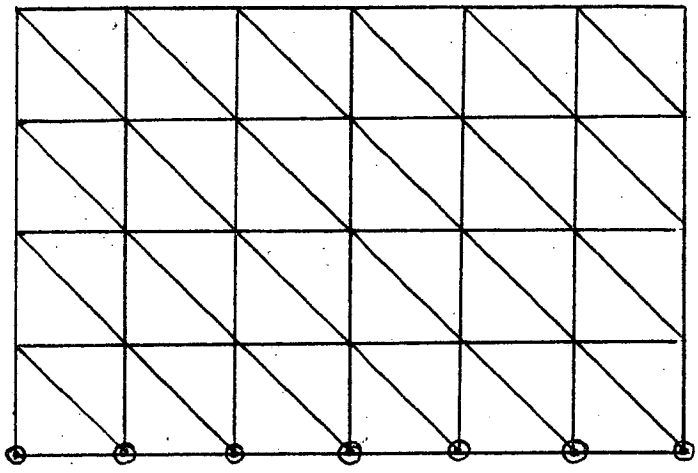
P et M étant fixes et égaux, on a étudié essentiellement deux configurations de machines liées à la vitesse des processeurs, du réseau, et de la mémoire.

- Processeur rapide, mémoire lente (T1)
- Processeur lent, mémoire rapide (T2)

- L'accélération est légèrement supérieure dans le cas d'une mémoire rapide (figure 26). En effet, les conflits-mémoire sont très faibles (de l'ordre de 2 à 3 % au lieu de 7 %) (figures 28, 29).

- Les attentes sur le BIP restent inchangées (figure 27).

MAILLAGE REGULIER D'UNE GRILLE RECTANGULAIRE



NX : nombre de noeuds dans la direction x

NY : nombre de noeuds dans la direction y

FRONT INITIAL : noeuds ⊙

Figure 14

MAILLAGE P1 D'UN ANNEAU

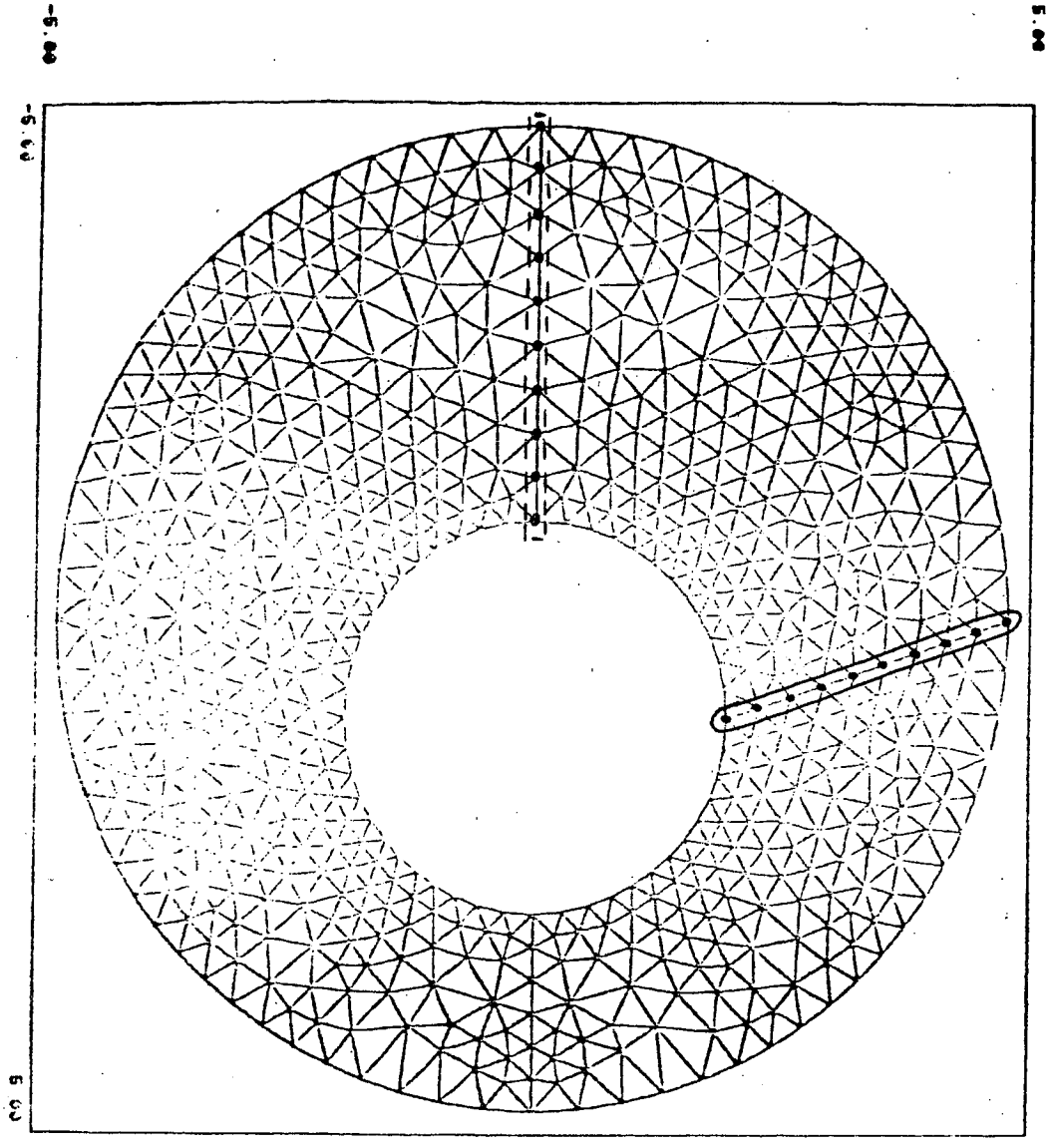
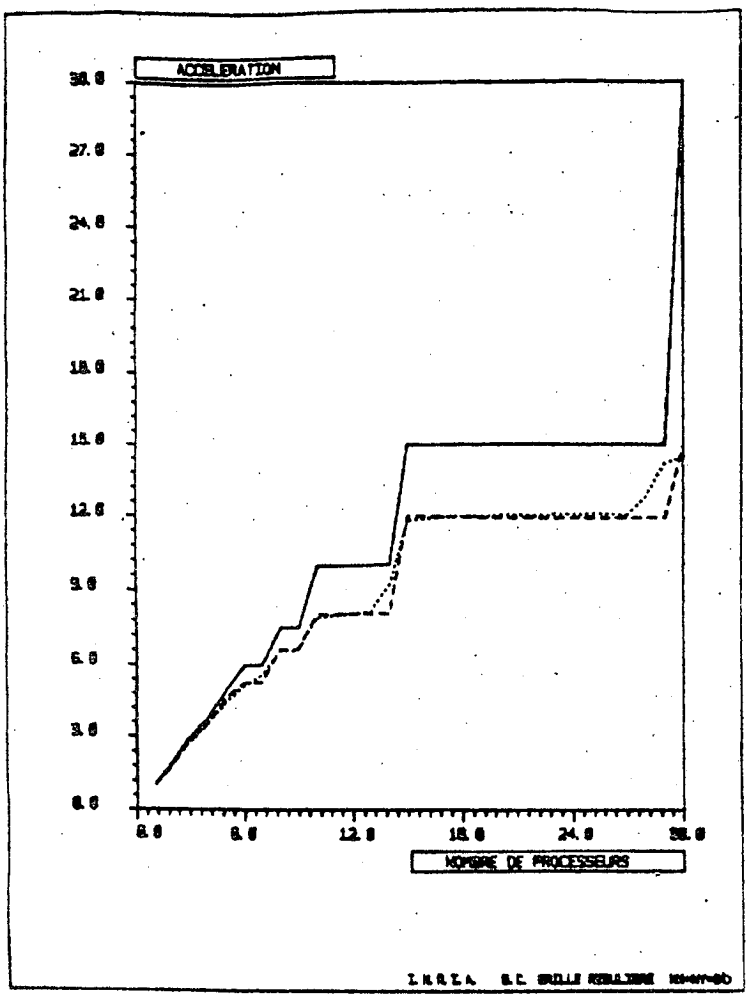


Figure 15



GRILLE REGULIERE

(NX = NY = 30 - N = 900)

VARIATION SUIVANT P

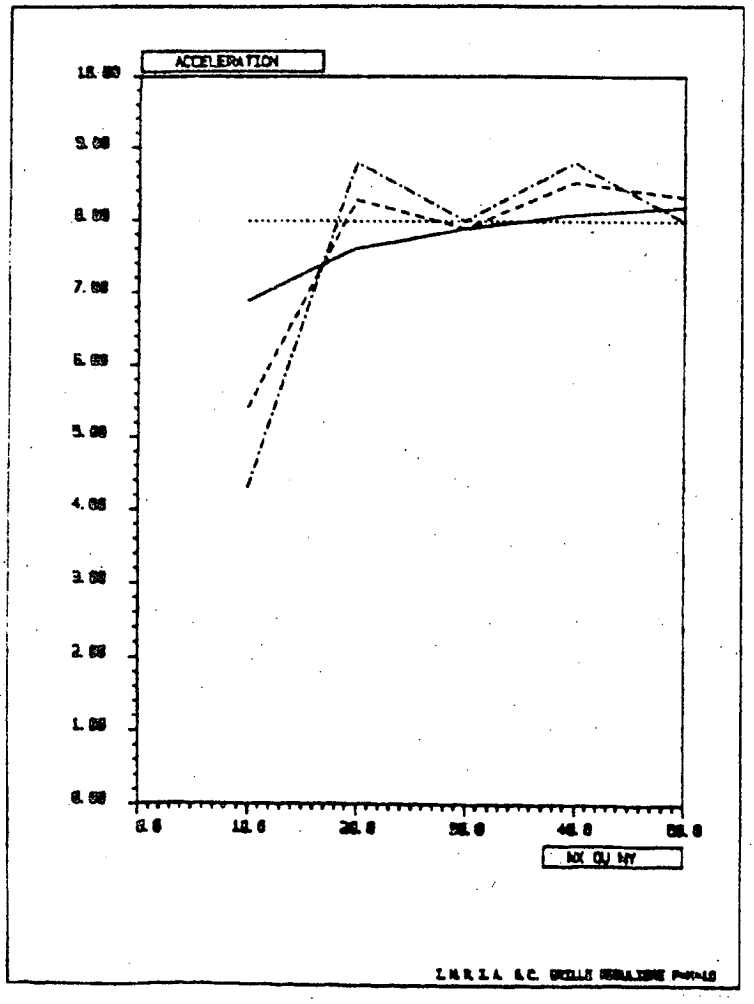
(M = P)

——— SP MAXIMALE

– – SP EVALUEE

• • • • SP MESUREE

Figure 16



VARIATION SUIVANT N

(P = M = 10)

- NX VARIABLE NY = 30

——— SP MESUREE

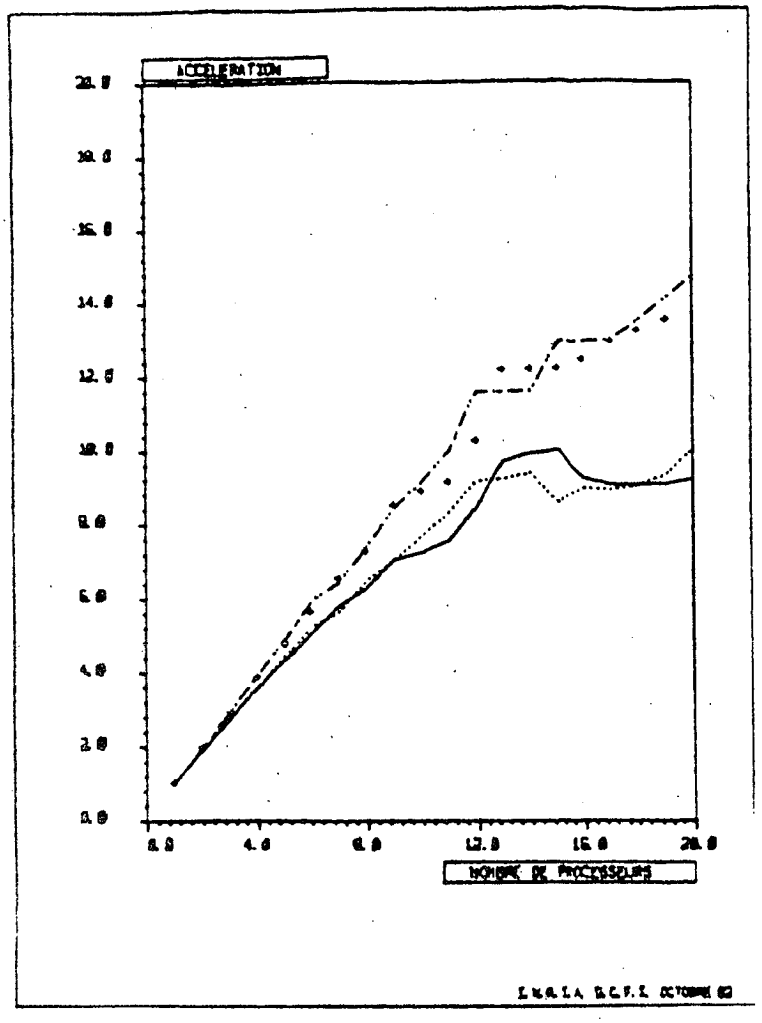
• • • • SP EVALUEE

- NY VARIABLE NX = 30

– – – SP MESUREE

—• – • SP EVALUEE

Figure 17



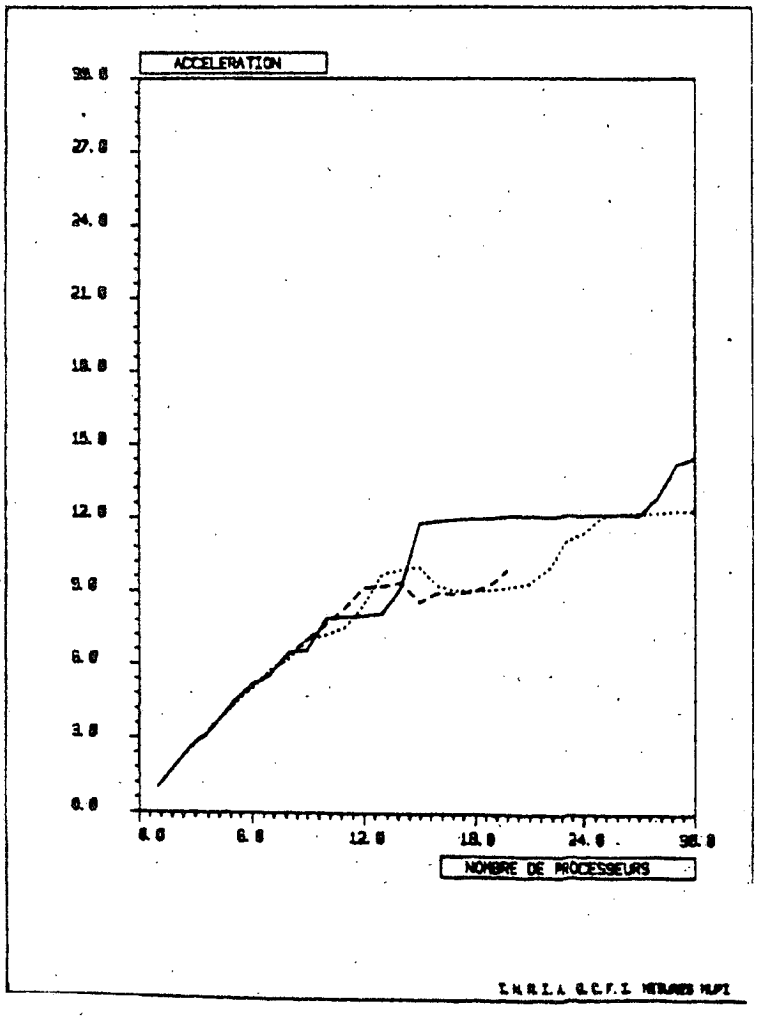
ANNEAU (N = 649)

VARIATION SUIVANT P

(M = P)

- FRONT INITIAL 1 A1
  - —— SP MESUREE
  - —·— SP MAXIMALE
- FRONT INITIAL 2 A2
  - ··· SP MESUREE
  - \+ SP MAXIMALE

Figure 18



VARIATION SUIVANT LE MAILLAGE

(SP MESUREE)

(M = P)

—— GR

GRILLE REGULIERE (N = 900)

···· A1

ANNEAU (N = 649)

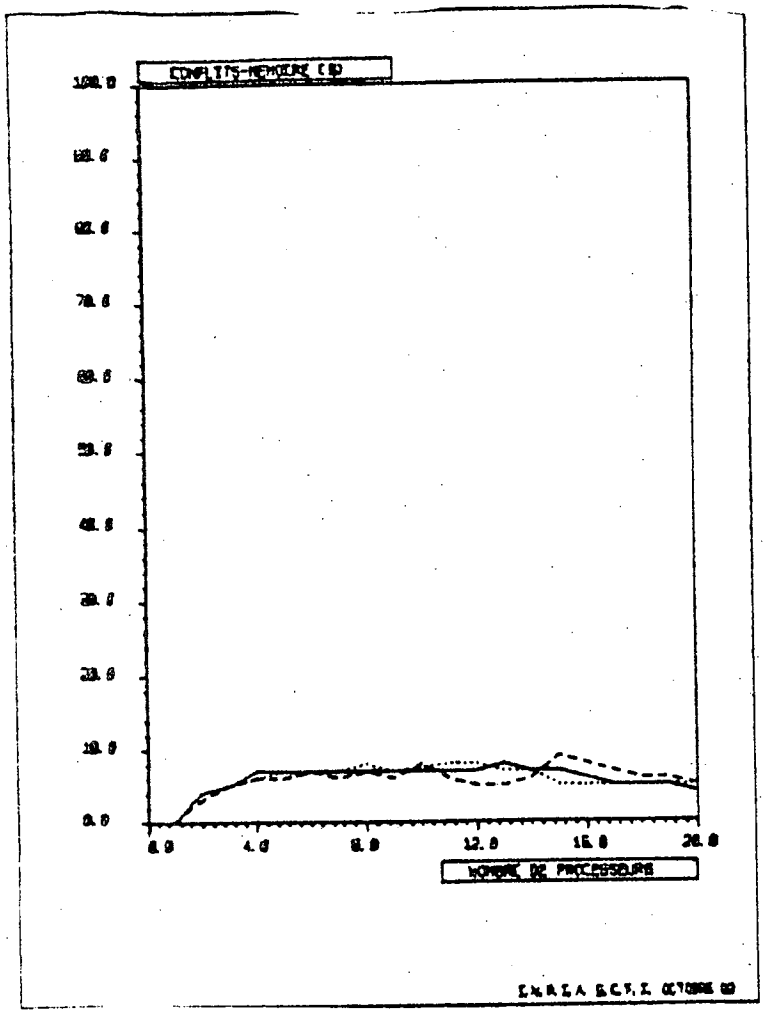
FRONT INITIAL 1

- - - A2

ANNEAU (N = 649)

FRONT INITIAL 2

Figure 19



VARIATION SUIVANT P ET LE MAILLAGE (M = P)

____ GR

GRILLE REGULIERE (N = 900)

. . . . A1

ANNEAU (N = 649) FRONT INITIAL 1

_ _ A2

ANNEAU (N = 649) FRONT INITIAL 2

Figure 20

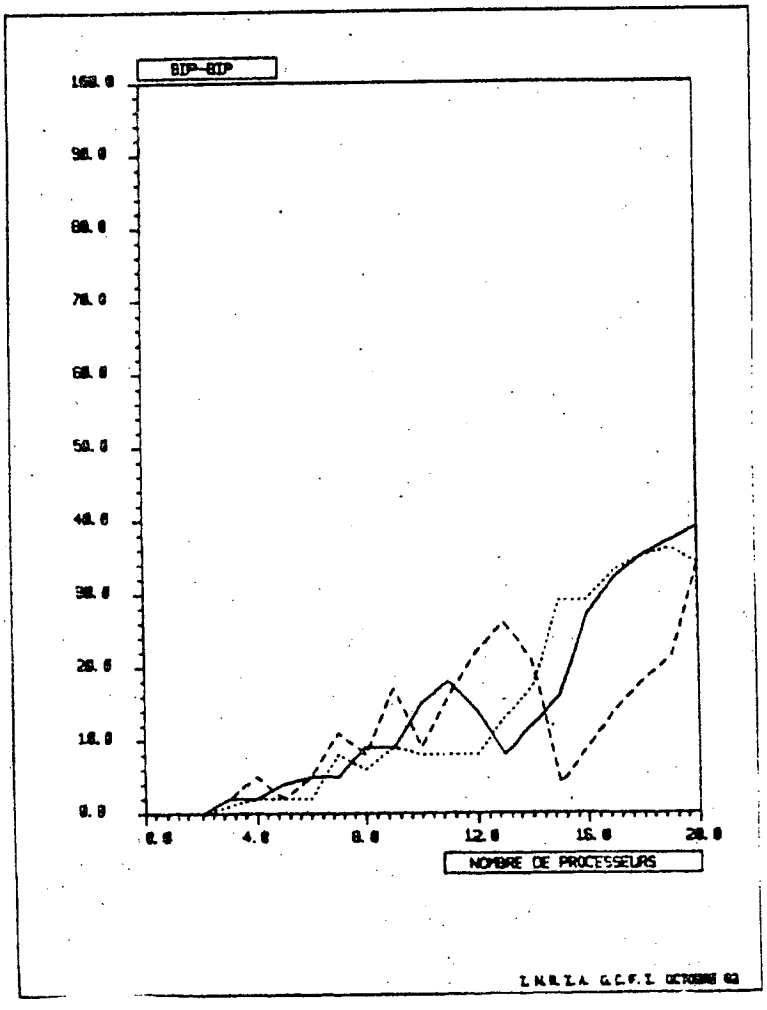
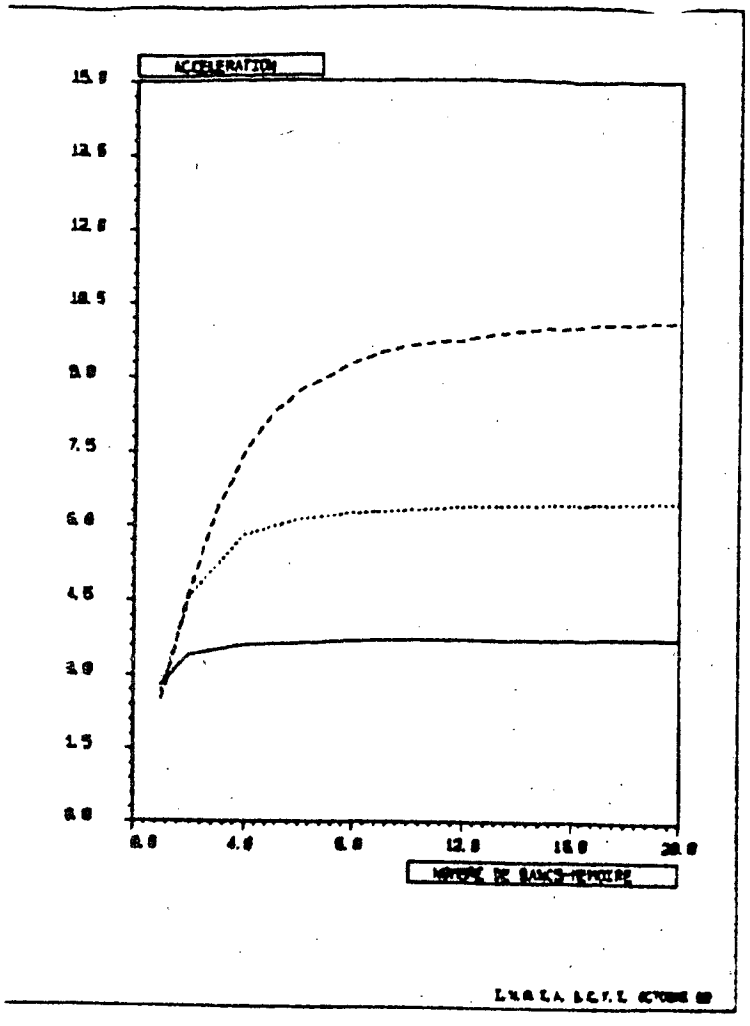


Figure 21



ANNEAU A1
(FRONT 1 - N = 649)

VARIATION SUIVANT M

—— P = 4

···· P = 8

--- P = 15

Figure 22

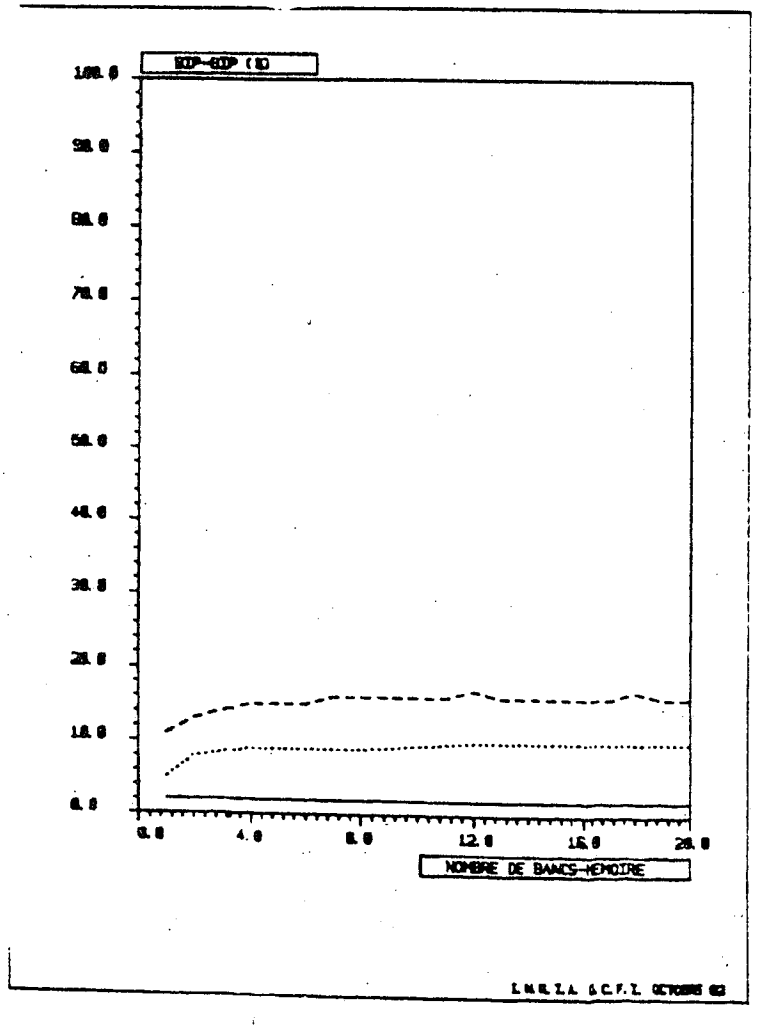
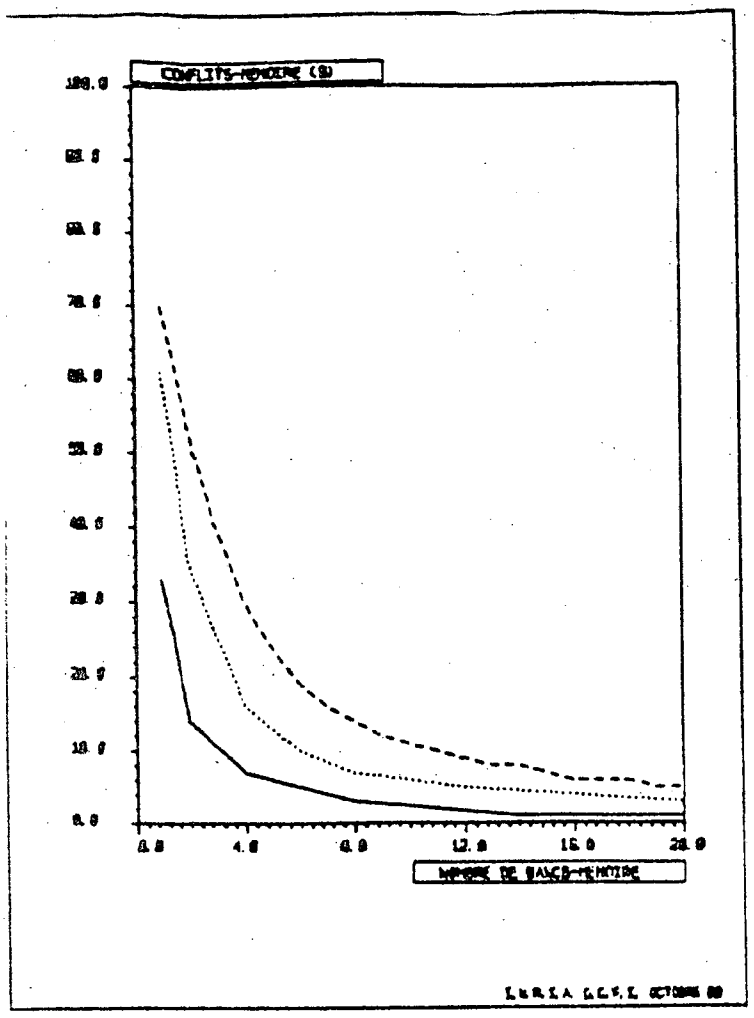


Figure 23



ANNEAU A1

(FRONT 1 - N = 649)

VARIATION SUIVANT M

—— P = 4

···· P = 8

--- P = 15

Figure 24

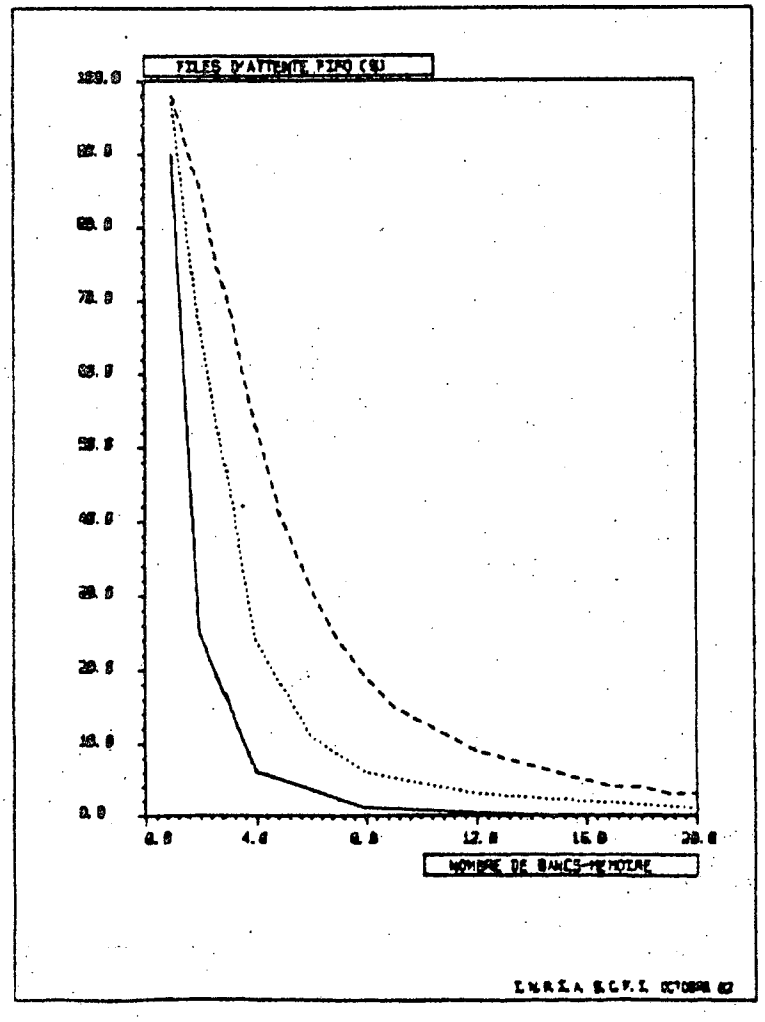
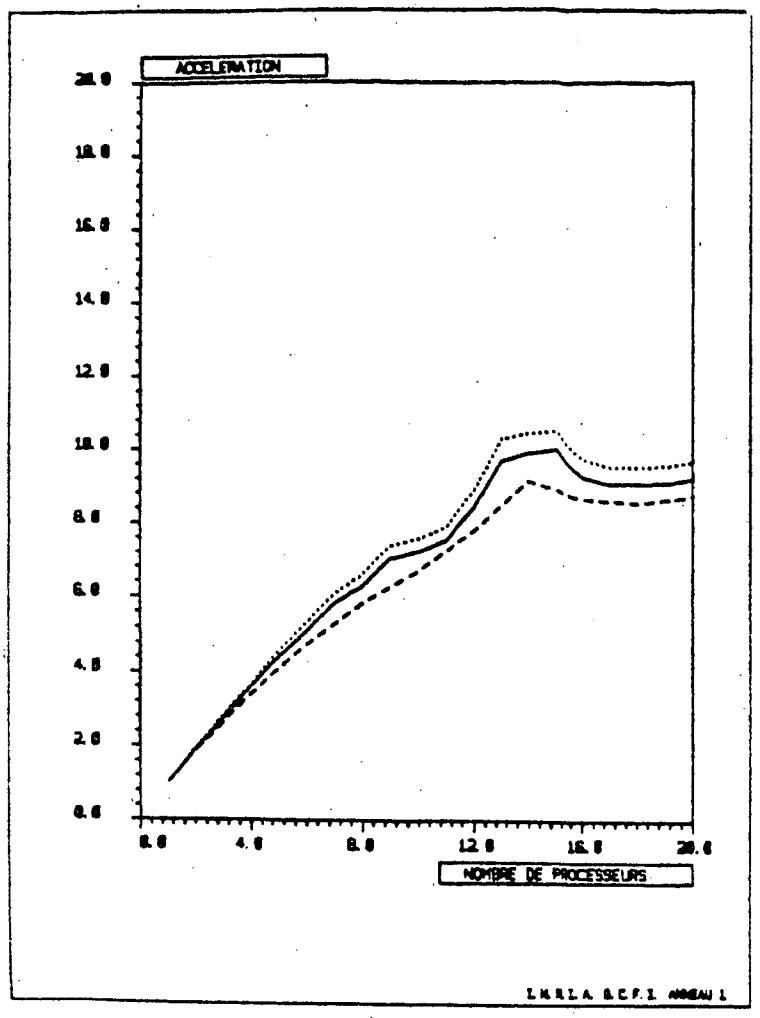


Figure 25



ANNEAU A1

(FRONT 1 - N = 649)

VARIATION SUIVANT

LA VITESSE DE LA MACHINE

—— M = P T1

PROCESSEUR RAPIDE

MEMOIRE ET RESEAU LENTS

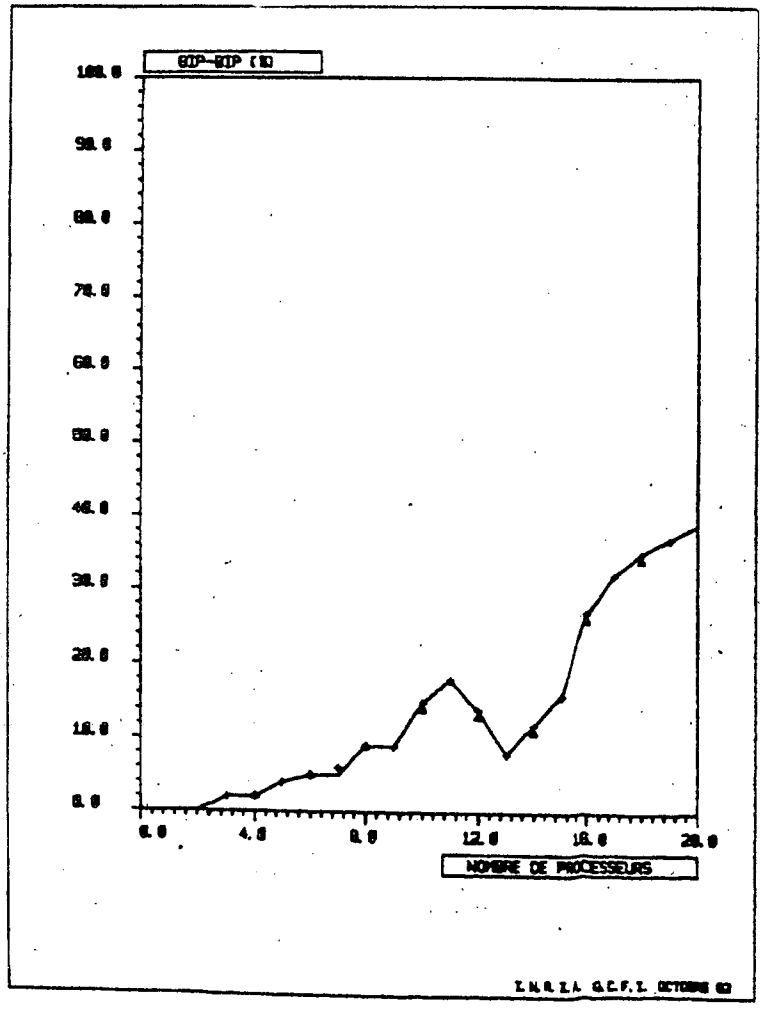
– – M = P/2 T1

••• M = P T2

PROCESSEUR LENT

MEMOIRE ET RESEAU RAPIDES

Figure 26

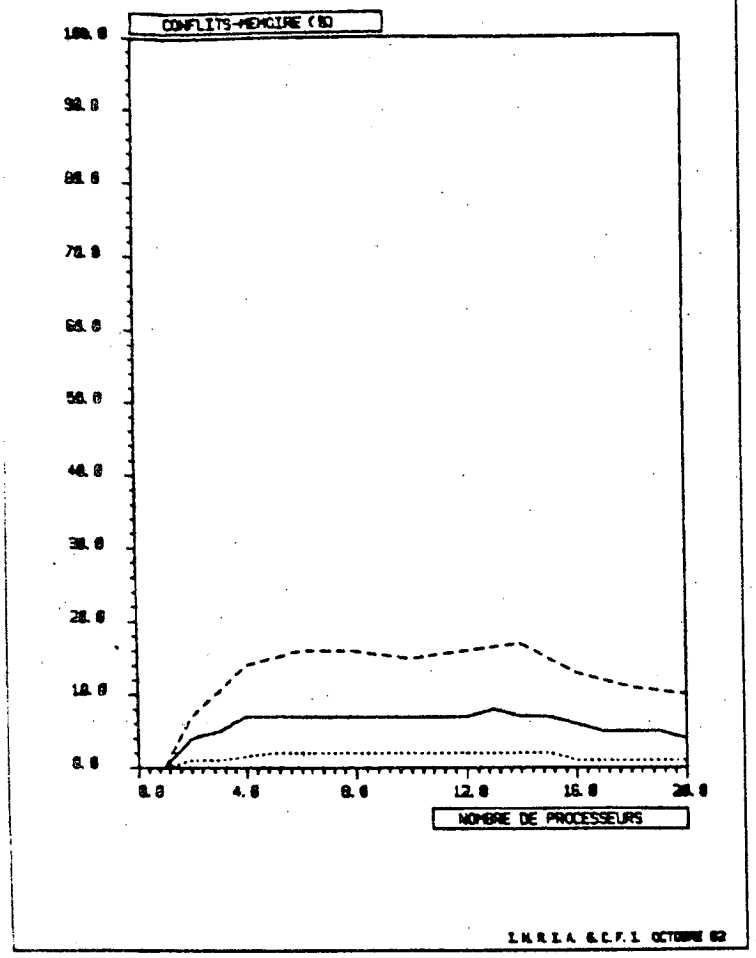


—— M = P T1

+ M = P/2 T1

▲ M = P T2

Figure 27



ANNEAU A1

(FRONT 1 - N = 649)

VARIATION SUIVANT

LA VITESE DE LA MACHINE

—— M = P T1

•••• M = P/2 T1

– – M = P T2

Figure 28

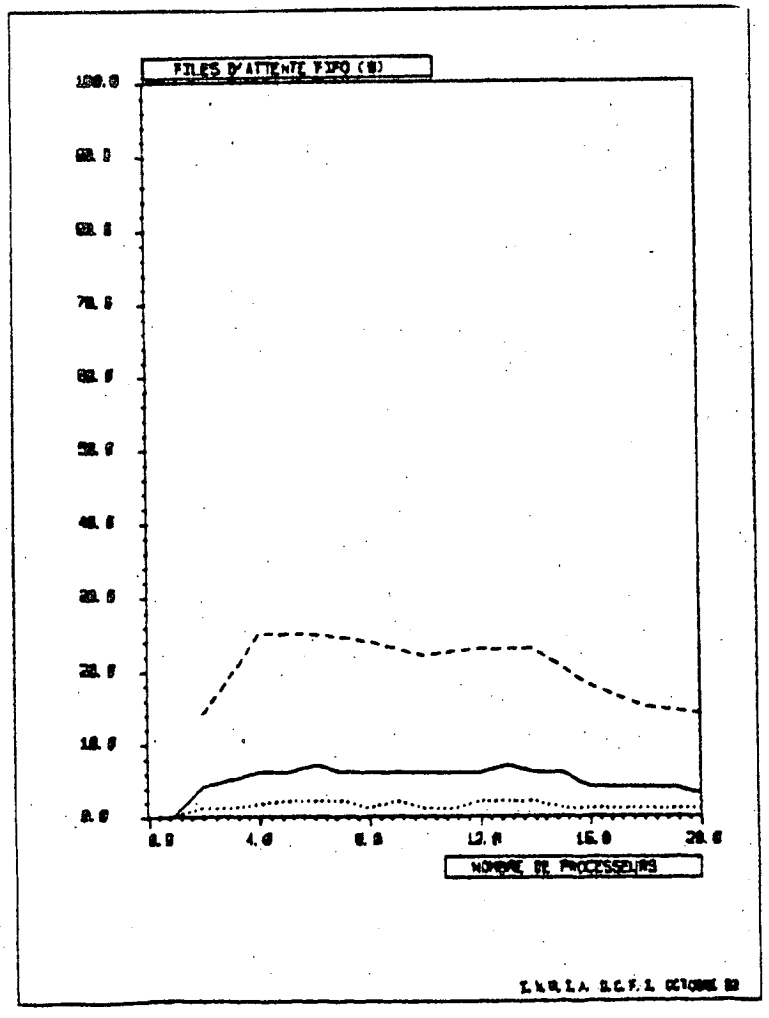


Figure 29

Conclusion

L'algorithme étudié comporte deux niveaux de parallélisme :

- un découplage en tâches qui exploite une architecture Multi-Processeurs.
- une vectorisation des calculs internes à une tâche.

Une renumérotation adéquate des noeuds de la matrice, à l'aide du graphe associé, permet de dégager cette structure Multi-Vectorielle.

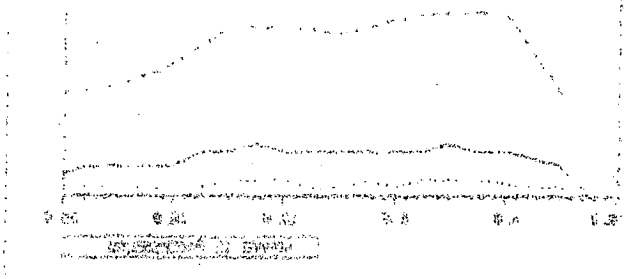
Les estimations et les résultats de simulation montrent de bonnes performances au niveau MIMD.

- On obtient une efficacité supérieure à 75 % pour une matrice quelconque d'Eléments Finis (d'ordre 649) jusqu'à environ 16 processeurs.

- Le coût des synchronisations dépend directement de la répartition des tâches entre les processeurs.

- Le taux des conflits-mémoire, que l'on peut modéliser simplement, est sensiblement proportionnel à P/M.

L'aspect vectorisation sera abordé ultérieurement [9] .



Bibliographie

[1] A. O. Allen "Probability, Statistics and Queueing Theory" Computer Science and Applied Mathematics, W. Rheinboldt Ed.

[2] F. Bashett, A.J. Smith "Interference in Multiprocessor Computer Systems with Interleaved Memory" Comm. de l'ACM (Juin 1976).

[3] D. Bhandarkar "Analysis of Memory Interference in Multiprocessors" IEEE Trans. on Comp. (Sept. 1975).

[4] D. Chang "On the effective bandwith of parallel memories" IEEE Trans. on Comp. (Mai 1977).

[5] N.F. Chen "An analysis of scheduling algorithms in multiprocessor computing systems" University of Illinois at Urbana-Champeign (Mai 1975).

[6] Cray Research "Cray-XMP" (1982).

[7] E. Cuthill "Several strategies for reducing the bandwith of matrices" in Sparse Matrices and their Applications D. Rose and R. Willoughby Ed. Plenum Press New York (1972).

[8] J. Erhel, Thèse de 3ème cycle, Université Paris VI (Mars 1982).

[9] J. Erhel "Résolution parallèle de systèmes linéaires creux" (à paraître).

[10] J. Erhel, A. Lichnewsky, F. Thomasset "Multi-processeur-INRIA : Structure et fonctionnement" Rapport technique INRIA N° 14 (Juillet 1982).

[11] J.A. George "Solution of linear systems of equations : direct methods for finite element problems" in Sparse Matrix Techniques Barker Ed., Springer (1977).

[12] L. Johnsonn "Gaussian Elimination on Sparse Matrices and Concurrency : A complexity Analysis", California Institute of Technology (Décembre 1980).

[13] A. Lichnewsky "Sur la résolution de systèmes linéaires issus de la méthode des éléments finis par une machine multi-processeur" Rapport INRIA N° 119 (Février 1982).

[14] A. Lichnewsky "Solving some linear systems arising in finite element methods on parallel processors", SIAM (Juillet 1982).

[15] J.A. Meijerink et H.A. Van der Vorst "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix", Math. Comp., 31 (1977).

[16] A. Perronnet "Modulef-le module APMEFI", INRIA (Mai 1981).

[17] G. Rodrigue "Incomplete Bloc Cyclic Reduction", Atelier Calcul Vectoriel EDF-CISI (Octobre 1982).

[18] D.J. Rose "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations" in Graph theory and Computing, Academic Press (1971).

[19] H.S. Stone "Parallel tridiagonal equations solvers", ACM Trans. on Math. Software (Décembre 1975).

[20] R.E. Tarjan "Graph theory and Gaussian Elimination" in Sparse Matrix Computations" Bunch Rose Ed., Academic Press (1976).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique