88/936

UNIVERSITE DE NANCY I

Sc N 88/133 A

THESE

# CONTRIBUTION A L'EXTENSION DU CALCUL PARALLELE APPLICATION A L'IDENTIFICATION ET A LA COMMANDE DES SYSTEMES INTERCONNECTES

Présentée devant l'Université de Nancy I
Pour obtenir le grade de

**DOCTEUR ès-SCIENCES**
**Par**

**Javad FOULADGAR**

Ingénieur de la Faculté technique de l'Université de Téhéran
Docteur-Ingenieur en Automatique de l'Université Paul SABATIER de Toulouse

UNIVERSITE DE NANCY

Soutenue le 19 Juillet 1988 devant la Commission d'Examen

JURY

| | | |
|---|---|---|
| MM. | C. HUMBERT | |
| | G. VILLERMAIN LECOLIER | Rapporteurs |
| | G. KRZAKALA | |
| | M. AUBRUN | |
| | C. SAINT REMY | Examinateurs |
| | A. SCHNEIDER | |

1988

BIBLIOTHEQUE SCIENCES NANCY 1
D 095 145844 2

UNIVERSITE DE NANCY I

---

THESE

---

# CONTRIBUTION A L'EXTENSION DU CALCUL PARALLELE APPLICATION A L'IDENTIFICATION ET A LA COMMANDE DES SYSTEMES INTERCONNECTES

Présentée devant l'Université de Nancy I
Pour obtenir le grade de

**DOCTEUR ès-SCIENCES**
**Par**
**Javad FOULADGAR**

Ingénieur de la Faculté technique de l'Université de Téhéran
Docteur-Ingenieur en Automatique de l'Université Paul SABATIER de Toulouse

Soutenue le 19 Juillet 1988 devant la Commission d'Examen

---

JURY

MM. C. HUMBERT
G. VILLERMAIN LECOLIER — Rapporteurs
G. KRZAKALA

M. AUBRUN
C. SAINT REMY — Examinateurs
A. SCHNEIDER

1988

à Farah

à Sepideh

Cette thèse a été préparée au Centre de Recherche en Automatique de Nancy au sein du Laboratoire d'Automatique et de Recherche Appliquée de Longwy dirigé par Monsieur le Professeur Gérard KRZAKALA.

Je suis particulièrement heureux de compter parmi les membres de ce Jury Monsieur le Professeur Claude HUMBERT, Directeur du Laboratoire d'Automatique et de Recherche Appliquée de l'ESSTIN.

J'exprime ma plus profonde gratitude à Monsieur le Professeur Gérard VILLERMAIN LECOLIER, Directeur du Laboratoire d'Automatique et de Micro-Electronique de l'Université de Reims pour sa participation à ce Jury.

Je suis très sensible à l'intérêt qu'a bien voulu porter à ce travail Monsieur le Professeur Michel AUBRUN, Directeur de l'Institut Universitaire de Technologie de Nancy-Longwy et du Laboratoire d'Automatique et de Recherche Appliquée de Nancy.

Que Monsieur Christian SAINT REMY, Directeur des Services d'Informatique de la Société Valexy accepte mes remerciements pour sa participation à ce Jury.

Je ne saurais exprimer tout ce que je dois à Monsieur Antoine SCHNEIDER pour tout l'encouragement, l'aide et les conseils précieux qu'il m'a apportés à tous les stades de ma vie de chercheur et d'enseignant.

Ma reconnaissance s'adresse également à tous mes camarades du Laboratoire dont l'aide m'a été précieuse.

Enfin je remercie sincèrement Mesdames F. ORY, N. BALESTRERI, V. SANTORO, P. PINOT et N. BENITO pour le soin et la compétence qu'elles ont apportés à la réalisation de ce mémoire.

# TABLE DES MATIERES


**INTRODUCTION**

**CHAPITRE 1 - ARCHITECTURES PARALLELES**

1-1 Introduction

1-2 Classification des architectures parallèles

1-2-1 Les processeurs M.I.S.D.
1-2-2 Processeurs S.I.M.D.
1-2-3 Multiprocesseurs M.I.M.D.

1-3 Réseaux d'interconnexion

1-3-1 Réseau de Clos
1-3-2 Réseau de Bênes
1-3-3 Réseau Oméga

1-4 Conclusion

**CHAPITRE 2 - CREATION ET ORDONNANCEMENT DES ALGORITHMES PARALLELES**

2-1 Introduction

2-2 Algorithme Parallèle

2-2-1 Parallèlisme des algorithmes numériques
2-2-2 Méthode de création des algorithmes parallèles

2-3 Ordonnancement des algorithmes parallèles

2-3-1 Représentation des algorithmes par des graphes
2-3-2 Pondération du graphe
2-3-3 Fusion du graphe
2-3-4 Détermination de la durée d'exécution des tâches
2-3-5 Ordonnancement de base

2-3-6 Ordonnancement des algorithmes parallèles en présence de contraintes
2-3-7 Ordonnancement des tâches avec durée indéterminée

2-4 Conclusion

**CHAPITRE 3 - CALCUL PARALLELE DES SYSTEMES LINEAIRES**

3-1 Introduction

3-2 Méthodes directes de résolution des systèmes d'équations linéaires

3-2-1 Système triangulaire
3-2-2 Méthode de Gauss
3-2-3 Système symétrique
3-2-4 Méthode de Cholesky

3-3 Méthodes itératives de résolution des systèmes d'équations linéaires

3-3-1 Méthode de Jacobi
3-3-2 Méthode de Gauss-Seidel
3-3-3 Comparaison des méthodes directes et itératives

3-4 Inversion d'une matrice

3-5 Conclusion

**CHAPITRE 4 - ALGORITHMES PARALLELES DE L'ESTIMATION**

4-1 Introduction

4-2 Multiplication de deux matrices

4-3 Validation des données

4-3-1 Méthode globale
4-3-2 Méthode itérative

4-4 Estimation des paramètres

4-4-1 Méthode directe
4-4-2 Méthode itérative
4-4-3 Algorithme récursif

4-5 Filtre de Kalman

4-5-1 Filtre centralisé
4-5-2 Filtre décentralisé

4-6 Conclusion

**CHAPITRE 5 - CALCUL PARALLELE DE LA COMMANDE OPTIMALE**

5-1 Introduction

5-2 Résolution des sytèmes d'équations différentielles

5-2-1 Equations différentielles non linéaires
5-2-2 Equations différentielles linéaires

5-3 Commande optimale des sytèmes linéaires continus

5-3-1 Décomposition en sous-problèmes avec conditions à un seul bout
5-3-2 Décomposition en sous-problèmes avec conditions aux deux bouts
5-3-3 Intégration en parallèle de l'équation de Ricatti

5-4 Commande optimale des systèmes linéaires discrets

5-4-1 Décomposition spatiale
5-4-2 Décomposition temporelle
5-4-3 Décomposition de l'équation de Riccati

5-5 Conclusion

CHAPITRE 6 - ORDONNANCEMENT DU TRANSFERT

6-1 Introduction

6-2 Architecture multiprocesseurs avec un bus commun

6-3 Architecture avec un réseau d'interconnexion

6-4 Conclusion

CONCLUSION GENERALE ET PERSPECTIVES

## TABLE DES SYMBOLES

| | |
|---|---|
| $X, Y, Z$ | Scalaires |
| $\underline{X}, \underline{Y}, \underline{Z}$ | Vecteurs colonnes. |
| $\hat{\underline{X}}, \hat{\underline{Y}}, \hat{\underline{Z}}$ | Vecteurs d'estimation. |
| $\underline{X}^T, \underline{Y}^T, \underline{Z}^T$ | Vecteurs transposés. |
| $\tilde{\underline{X}}, \tilde{\underline{Y}}, \tilde{\underline{Z}}$ | Vecteur d'erreur d'estimation. |
| $\dot{\underline{X}} = \frac{d\underline{X}}{dt}$ | |
| $L$ | Lagrangien. |
| $J$ | Critère. |
| $A, B, C$ | Matrices. |
| $\|\underline{X}\|^2_S = \underline{X}^T S \underline{X}$ | |
| $\underline{\lambda}$ | Paramètre de Lagrange. |
| $\varepsilon$ | Infiniment petit. |
| $t_0, t_f$ | Horizon d'optimisation. |
| $t_{add}$ | Temps d'addition de deux nombres réels. |
| $t_{div}$ | Temps de division de deux nombres réels. |
| $t_{mult}$ | Temps de multiplication de deux nombres réels. |
| $t_{racine}$ | Temps de calcul d'une racine carrée. |

| | |
|---|---|
| $t_{test}$ | Temps de calcul du test pour les algorithmes itératifs. |
| $\tau_{accel}$ | Taux d'accélération des algorithmes parallèles. |
| $\tau_{occup}$ | Taux d'occupation des processeurs. |
| S.I.S.D. | ( Single Instruction, Single Data Stream ). Les processeurs séquentiels. |
| M.I.S.D. | ( Multiple Instruction, Single Data Stream ). Les processeurs pipelines. |
| S.I.M.D. | ( Single Instruction, Multiple Data Stream ). Les processeurs synchrones, vectoriels, matriciels. |
| M.I.M.D. | ( Mutiple Instruction, Multiple Data Stream ). Les multiprocesseurs asynchrones. |
| C.P.M. | ( Critical Path Methode ). Méthode d'ordonnancement des tâches avec la durée déterminée. |
| P.E.R.T. | ( Program Evaluation and Review Technique ). Méthode d'ordonnancement des tâches avec la durée probabiliste. |
| O (M) | de l'ordre M. |

# - CHAPITRE 1 -

# ARCHITECTURES PARALLELES

## 1-1 INTRODUCTION :

Un des objectifs les plus importants dans le développement de l'architecture, des ordinateurs a été et continue à être l'augmentation de la vitesse de calcul et du débit des données. Cet objectif est atteint par trois moyens :

- l'amélioration des modèles et des algorithmes de résolution,
- l'amélioration de la technologie,
- l'amélioration de l'architecture des ordinateurs.

Ces trois techniques sont l'objet d'études approfondies partout dans le monde et leur description dépasse largement le cadre de ce mémoire. Nous abordons brièvement la troisième technique, où le maître mot en matière d'amélioration de l'architecture des ordinateurs est le parallélisme [1-1], à tel point que l'on constate l'utilisation de ce parallélisme dans beaucoup de processeurs pourtant classés comme des processeurs séquentiels, [1-2], [1-3], [1-4], [1-5]. Dans ce chapitre, nous présentons la classification des architectures parallèles, puis nous étudions ensuite les différents aspects de chaque architecture. Nous abordons enfin les réseaux d'interconnexion des modules parallèles.

## 1-2 CLASSIFICATION DES ARCHITECTURES PARALLELLES :

Les différents types d'architectures ont fait l'objet de multiples classifications dont la plus connue est celle de Flynn, basée sur le nombre d'instructions et de données traitées simultanément. [1-6]

7

1-1 INTRODUCTION :

Un des objectifs les plus importants dans le développement de l'architecture, des ordinateurs a été et continue à être l'augmentation de la vitesse de calcul et du débit des données. Cet objectif est atteint par trois moyens :

- l'amélioration des modèles et des algorithmes de résolution,
- l'amélioration de la technologie,
- l'amélioration de l'architecture des ordinateurs.

Ces trois techniques sont l'objet d'études approfondies partout dans le monde et leur description dépasse largement le cadre de ce mémoire. Nous abordons brièvement la troisième technique, où le maître mot en matière d'amélioration de l'architecture des ordinateurs est le parallélisme [1-1], à tel point que l'on constate l'utilisation de ce parallélisme dans beaucoup de processeurs pourtant classés comme des processeurs séquentiels, [1-2], [1-3], [1-4], [1-5]. Dans ce chapitre, nous présentons la classification des architectures parallèles, puis nous étudions ensuite les différents aspects de chaque architecture. Nous abordons enfin les réseaux d'interconnexion des modules parallèles.

1-2 CLASSIFICATION DES ARCHITECTURES PARALLELLES :

Les différents types d'architectures ont fait l'objet de multiples classifications dont la plus connue est celle de Flynn, basée sur le nombre d'instructions et de données traitées simultanément. [1-6]

On distingue :

a) le processeur sequentiel SISD (Single Instruction, Single Data Stream) : l'architecture d'un calculateur monoprocesseur exécutant, à chaque instant une seule instruction sur une seule donnée.

b) le processeur MISD (Multiple Instruction, Single Data Stream) : plusieurs instructions sont exécutées à partir d'une même information.

c) le processeur SIMD (Single Instruction, Multiple Data Stream) : une seule instruction commande le traitement simultané de plusieurs données.

d) le MIMD (Multiple Instruction, Multiple Data Stream) : cette catégorie est celle des systèmes multiprocesseurs. Chaque processeur effectue, indépendamment des autres, les calculs qui lui ont été assignés sur les données correspondantes. La figure (1-1) donne le schéma de ces 4 types d'architectures.

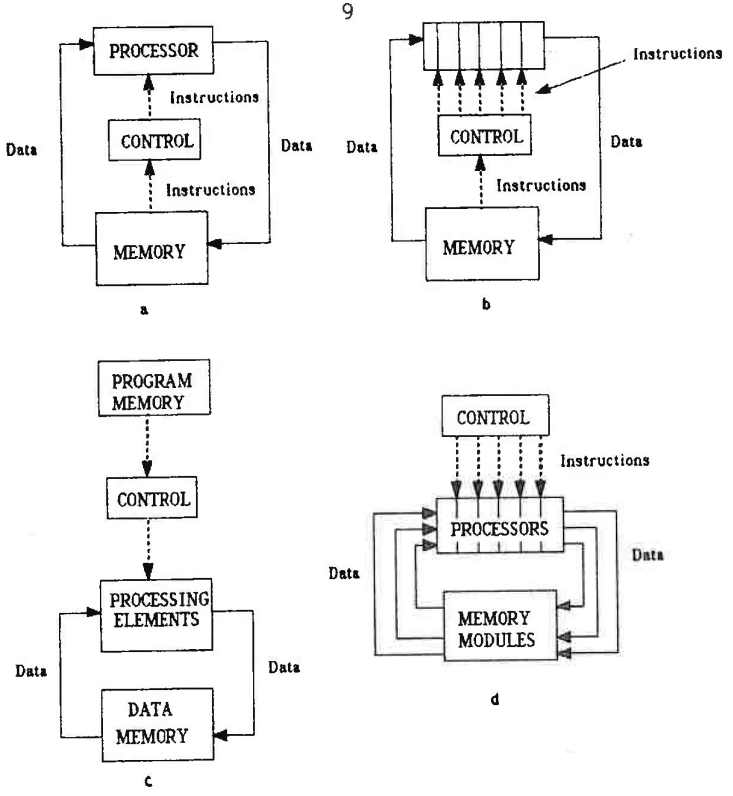



Figure (1-1) : architecture des ordinateurs

a - SISD b- MISD c- SIMD d - MIMD

### 1-2-1 Les processeurs MISD :

Parmi ces processeurs, les processeurs pipeline sont les plus importants. Le principe du pipeline est de segmenter ou de partitionner un processus global en sous-processus qui peuvent être exécutés par des unités ou des modules distincts. Les modules sont liés en série de telle



façon que la sortie d'un module sert d'entrée pour le module suivant. Les temps d'exécution des sous-processus pour les modules différents sont les mêmes. Ainsi, si l'algorithme utilisé fournit un flot continu de données, chaque module libéré peut exécuter son sous-processus sur la donnée suivante.

La méthode du pipeline pour les processeurs, est la même dans son principe, que des chaînes de production en série dans les usines ou les classes successives dans des universités.

Appelons T le temps d'initialisation (start-up time) nécessaire pour l'apparition du premier résultat. Si le nombre des segments ou des modules est K, chaque module exécute son processus en un temps $T/_K$. Ainsi après le premier résultat, il y aura un résultat toutes les $T/_K$ unités de temps. Supposons que le processus global se répète L fois. Le temps d'exécution des L processus est :

$$T_L = T + (L-1).\ T/_K \qquad (1\text{-}2\text{-}1)$$

Dans ce cas, pour amortir le temps de démarrage T, il faut que le nombre L soit assez grand. D'autre part, les données nécessaires pour l'exécution des sous-processus doivent entrer dans le système d'une façon régulière c'est-à-dire comme des vecteurs. C'est pour cette raison que la technique du pipeline est en principe une conception pour les processeurs vectoriels. En effet, la classe des processeurs vectoriels repose sur la combinaison de deux architectures de base : le processeur pipeline et le processeur SIMD [1-1].

Parmi ces processeurs on peut citer :

le CDC STAR 100 [1-7], le ASC de Texas Instruments [1-8], le CRAY-1, et le CRAY x MP [1-1], [1-9].

Niveau de pipeline :

Le principe du pipeline en tant que chevauchement des opérations peut-être utilisé à plusieurs niveaux, c'est-à-dire au niveau des instructions, des groupes d'instructions ou même des sous-programmes. Nous illustrons ce problème par deux exemples :

Exemple 1 : pipeline au niveau des instructions.

On peut décomposer l'exécution d'une instruction c = a*b, (avec * représentant une opération donnée) en étapes suivantes :

1 - recherche de l'instruction
2 - décodage de l'instruction
3 - génération de l'adresse a
4 - recherche de l'opérande a
5 - génération de l'adresse b
6 - recherche de l'opérande b
7 - exécution de l'opération *
8 - génération de l'adresse c
9 - écriture du résultat c

Si un processeur comprend neuf modules, chacun devant exécuter une de ces étapes, alors on peut exécuter une suite d'instructions c = a*b en



pipeline. Dans ce cas, certaines conditions doivent être vérifiées pour assurer l'efficacité de la technique du pipeline :

1 - décomposition de l'instruction en étapes approximativement de même durée.
2 - prévision correcte de la prochaine instruction [1-1], [1-10]
3 - évitement des aléas [1-1], [1-11]

La première condition est évidente et l'exemple suivant illustre la seconde condition.

```
Pour n = 1 à K faire
instruction 1
instruction 2
If (condition) faire instruction 10
instruction 4
--------------------
instruction 10
```

Avant de terminer l'instruction 3, la structure pipeline commence à décoder l'instruction 4, 5 ...., mais si la condition est vrai, ces instructions doivent être vidées, afin de permettre l'exécution de l'instruction 10. Les méthodes de réduction de ces vidages sont décrites dans [1-1].

Les aléas sont des contraintes de succession entre instructions dont les exécutions se recouvrent dans le pipeline. Supposons la séquence d'instruction suivante :

1 - $C_1 = a_1 * b_1$

2 - $C_2 = a_2 * b_2$

3 - $C_3 = a_3 * b_3$

4 - IF ($C_1 < C_3$) Then $C_1 = C_3$

Le schéma de l'exécution de cette suite d'instructions est le suivant figure (1-2).

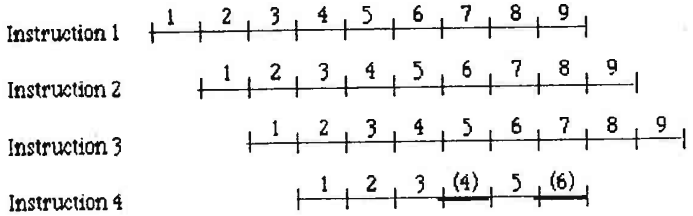



Figure (1-2)

Désignons par t le temps d'exécution de chaque étape. L'instruction 4 a besoin de $C_1$ et $C_3$ respectivement aux instants 6t et 8t, mais ces données ne sont générées qu'aux instants 9t et 11t. Ainsi, pout éviter l'utilisation des anciennes valeurs de $C_1$ et $C_3$ par l'instruction 4, le mécanisme de contrôle doit être à même de bloquer l'exécution de l'instruction 4 jusqu'à la mise à jour des valeurs $C_1$ et $C_3$.

Exemple 2 : <u>pipeline au niveau des sous-programmes.</u>

Soit un système monovariable d'équation dynamique :

$$x(k+1) = ax(k) + f\,u(k) \qquad (1\text{-}2\text{-}2)$$



où x est la variable d'état, u est la commande et k est l'instant d'échantillonnage. Le calcul de la commande optimale pour le système (1-2-2) minimisant un critère quadratique conduit aux équations suivantes [1-12], (voir également chapitre 5 de ce mémoire) :

$$\begin{aligned} x(k+1) &= ax(k) + b\,\lambda(k+1) \\ \lambda(k) &= cx(k) + d\,\lambda(k+1) \\ x(o) &= x_o \text{ et } \lambda(k_f) = cx(k_f) \end{aligned} \qquad (1\text{-}2\text{-}3)$$

où $\lambda$ est le paramètre de Langrange associé, $x_o$ est l'état initial et $K_f$ est l'instant final.

Pour la résolution du système (1-2-3), nous utilisons la méthode itérative alternée présentée en [1-13]. Pour $k_f = 6$, cette méthode est déterminée par :

Début de l'algorithme

- poser l'indice de l'itération égal à 0
- estimer les valeurs $x^{\circ}(k)$, k = 1 à 6 et $\lambda^{\circ}(k)$, k = 2 à 6
- tant qu'un test convenable n'est pas vrai

Début

- calculer $\lambda^{l+1}(k) = c\,x^{l}(k) + d\,\lambda^{l}(k+1)$
- calculer $x^{l+1}(k) = a\,x^{l}(k) + b\,\lambda^{l+1}(k)$
- $l = l + 1$

Fin

Fin de l'algorithme.

Pour deux processeurs en pipeline, le premier calcule les valeurs de $\lambda^{I+1}(K)$ qu'il fournit au deuxième processeur. Ainsi, le schéma du calcul pour une itération est le suivant : figure (1-3).

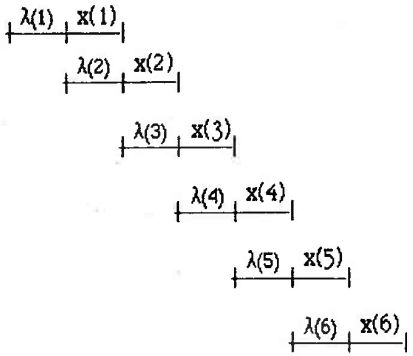


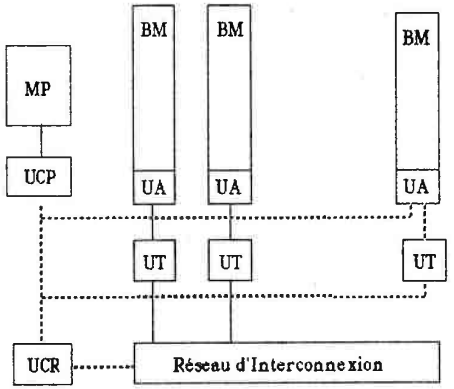

Figure (1-3)

### 1-2-2 Processeurs S.I.M.D :

Dans les processeurs S.I.M.D, le parallèlisme du système est obtenu par le travail simultané d'un certains nombres d'unités de traitement U.T. dont chacune est capable d'exécuter une opération spécialisée dans un flot d'instructions. Dans cette architecture, le système présente une seule unité de contrôle qui décode le flot d'instructions et envoie les signaux de contrôle vers les U.T. Parmi les architectures S.I.M.D, les processeurs vectoriels et matriciels sont les plus utilisés : ces processeurs sont



caractérisés par le fait que la même instruction est exécutée par un nombre élevé N d'U.T, sur un grand ensemble de données. La différence essentielle entre l'architecture S.I.MD et M.I.M.D est que les U.T. de l'architecture S.I.M.D ne sont pas des unités centrales complètes, donc elles ne sont pas capables de réaliser des activités indépendantes. La figure 1-4 présente le schéma d'un processeur S.I.M.D [1-1].




BM Banc de Mémoire — UCP Unité de Commandedu Processeur

UA Unité de Calcul d'Adresses — UCR Unité de Commande du Réseau

UT Unité de Traitement — MP Mémoire du Programme

Figure (1-4) : structure d'un processeur S.I.M.D.

Sur cette figure nous distinguons les différentes composantes d'un processeur S.I.M.D mentionnées auparavant :

- BM est la mémoire centrale du processeur décomposée en N bancs de mémoire,
- N est le nombre d'unités de traitement U.T. qui exécutent les instructions spécialisées : opérations arithmétiques, logiques, etc

- Un réseau d'interconnexion relie les U.T. entre elles,
- Une mémoire de programme MP contient les instructions à décoder,
- Une U.C.P, unité de commande de processeur, décode le flot d'instructions fournies par la Mp, et les diffuse sur les U.T. et les unités de calcul d'adresse UA.
- Les unités de calcul d'adresse UA modifient et traduisent l'adresse générée par l'unité de commande en adresse du BM concerné.

Une difficulté marquante de ce type d'architecture est liée à la nécessité d'accéder simultanément à N opérandes (parallèlisation de la mémoire) et à celle d'amener dans la même unité de traitement deux opérandes qui doivent être composées. Le problème de parallèlisation de la mémoire et d'organisation des données est un sujet très vaste qui dépasse le cadre de ce chapitre. Nous renvoyons le lecteur intéressé à [1-14], [1-15], [1-16], [1-17].

Les processeurs S.I.M.D, surtout les processeurs vectoriels et matriciels conviennent aux problèmes d'algèbres linéaires [1-18], à la résolution numériques des équations différentielles[1-19], et au traitement du signal. Parmi les processeurs S.I.M.D on peut citer notamment l'ILLIAC IV [1-20], et le STARAN [1-21].

### 1-2-3 Les multiprocesseurs M.I.M.D :

Un multiprocesseurs est défini comme suit :



- il contient au moins deux processeurs indépendants. Si ces processeurs ont la même performance, on parle d'un système multiprocesseurs homogène, sinon d'un système multiprocesseurs nonhomogène.

- la mémoire principale du système est accessible par tous les processeurs. Les mémoires locales des processeurs sont de taille réduite. Si les processeurs ont de grandes mémoires locales, on parle d'un système multi-ordinateurs (multimicroprocesseurs).

- les entrées/sorties sont accessibles par tous les processeurs.

- pour tout le système multiprocesseurs, il existe un seul système d'exploitation qui contrôle tous les processeurs. Ce système d'exploitation doit contrôler et assurer les interactions entre les processeurs et leurs programmes au niveau du travail de tâches, d'ensemble de données et de données.

Les multiprocesseurs peuvent avoir différentes topologies et architectures. Les topologies les plus étudiées sont les suivantes :

a) multiprocesseurs à bus commun :

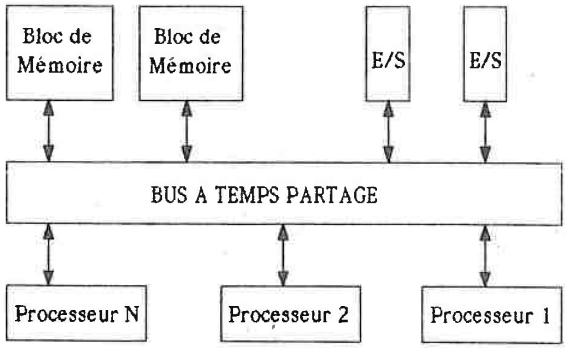



Figure (1-5) : Bus Commun

Cette architecture est la plus simple à mettre en oeuvre et permet une extension aisée du multiprocesseur. Mais comme le bus ne peut être utilisé à un instant donné, que par un seul processeur, le débit d'informations est limité. Cette architecture est donc réservée aux systèmes comportant un nombre limité de processeurs ou demandant peu d'échanges. Au chapitre 6, nous étudions en détail le problème du choix des nombres de processeurs et les limites de cette architecture.

b) matrice de points de croisement : figure (1-6)

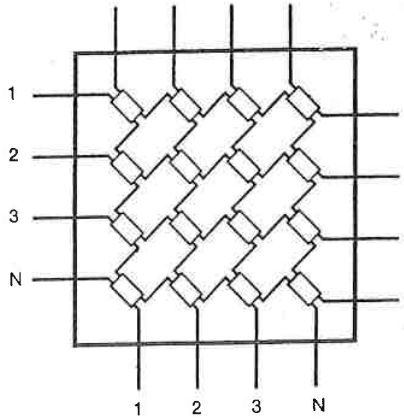



Figure (1-6) : matrice de croisement.

Dans cette architecture, N processeurs communiquent avec au moins autant de bancs de mémoires à travers une matrice de points de croisement. Cette matrice peut être également utilisée pour interconnecter les différents processeurs comme l'illustre la Figure (1-7).



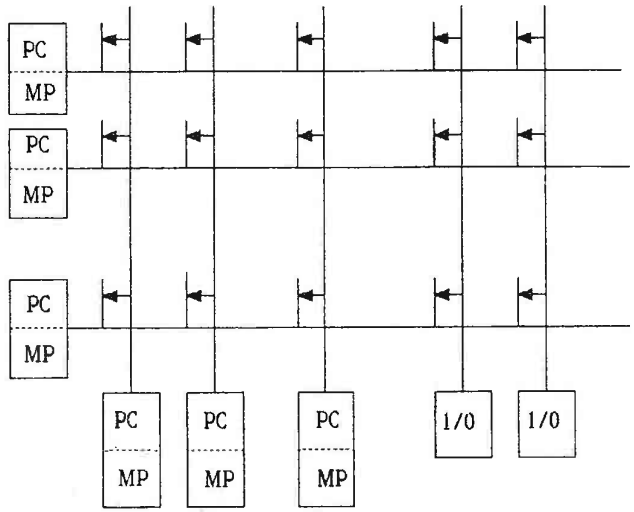



Figure (1-7) interconnexion des processeurs par une matrice de point de croisement.

La matrice de points de croisement réalise toutes les connections entre les processeurs et les mémoires. Malheureusement, son coût de réalisation est très élevé et en particulier quand le nombre de processeurs est approximativement égal au nombre de bancs de mémoires, ce coût est proportionnel à $N^2$. De plus, la liaison entre la première entrée et la dernière sortie passe par 2N commutateurs, ce qui pour N assez grand peut affaiblir le signal d'entrée [1-22]. Ces inconvénients limitent le nombre de processeurs et de bancs de mémoires qu'on peut relier par une matrice de points de croisement, dont une photo est présentée à la figure (1-8) [1-23].

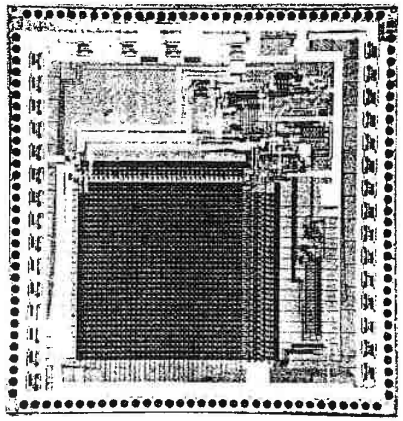


Figure 1-8 : matrice de point de croisement 32x32

En fait, la topologie N processeurs, M bancs de mémoire peut être réalisée avec d'autres réseaux moins coûteux que la matrice de croisement qui sont présentés en 1-3.

c) topologie hiérarchisée :

Dans cette organisation, un ou plusieurs processeurs sont connectés par un bus local aux bancs de mémoires et aux entrées/sorties. Chaque groupe de processeurs forme ainsi un noeud de calcul. Ces noeuds de calcul sont ensuite reliés entre eux à travers des processeurs de niveau supérieur. Figure (1-9).



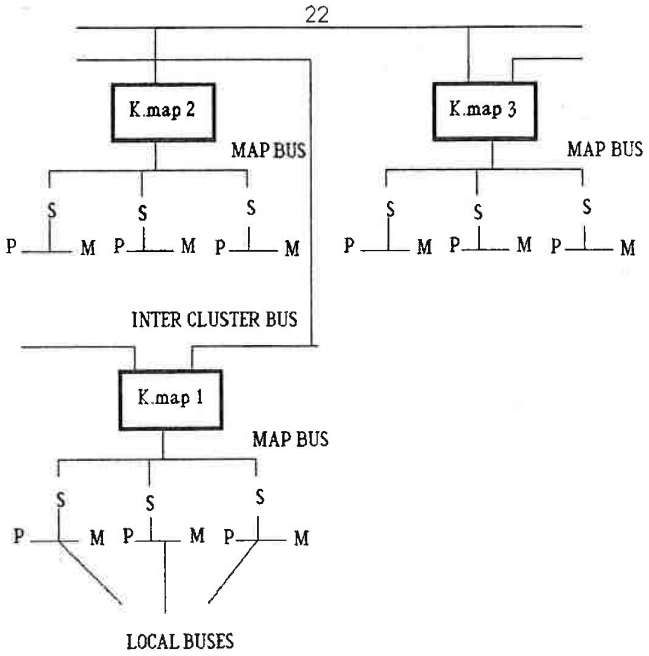



Figure (1-9) : Topologie hiérarchisée.

La tâche des processeurs du niveau supérieur est de réaliser toutes les fonctions nécessaires d'une part pour l'accès des processeurs locaux à la mémoire globale et d'autre part pour l'accès du système externe aux mémoires locales.

Un troisième niveau de hiérarchie consiste à relier les bus de deuxième niveau au bus global.

Ce réseau a l'avantage de réduire le nombre de processeurs sur les bus locaux. Mais pour qu'il soit efficace, il faut que les échanges entre les processeurs de deuxième niveau ne soient pas nombreux. D'autre part,

l'existence des processeurs du deuxième et du troisième niveau rend le système assez coûteux.

## 1-3 RESEAUX D'INTERCONNEXION :

De manière générale, un réseau d'interconnexion est un ensemble de commutateurs et de fils qui établissent la liaison entre les composants liés à ces N lignes d'entrées et des composants liés à ces N lignes de sorties, ces derniers n'étant pas forcément différents des premiers. La fonction la plus fréquente pour un réseau d'interconnexion est de permettre l'échange de données, soit entre les processeurs et les bancs de mémoire, soit entre les processeurs eux-mêmes. Le choix de la structure du réseau et son mode de fonctionnement est lié à l'architecture du système utilisé. Dans un système M.I.M.D, des liaisons entre point d'entrée et point de sortie s'établissent ou se défont, sans que l'enchaînement de ces changements d'état soit prévisible. Dans un système S.I.M.D, au contraire, la permutation d'état de tous les commutateurs du réseau s'effectue au même instant. On appelle *réseau sans blocage* un réseau $R_{N,N}$ de N entrées et N sorties, pour lequel on peut établir une liaison entre une entrée active i et une sortie inactive j sans modifier les liaisons déjà existantes. Les entrées et sorties inactives sont celles qui ne sont pas respectivement liées à une sortie ou une entrée.

Un réseau est *réarrangeable* s'il est toujours possible d'établir une liaison entre une entrée et une sortie inactive, au prix, le cas échéant, de modifications de liaisons déjà établies entre entrées et sorties actives.

Enfin un réseau est *non réarrangeable* s'il existe une probabilité de blocage arbitrairement faible pour l'établissement de liaisons entre les entrées et les sorties inactives [1-24].



Il existe différents réseaux d'interconnexion, sans blocage, réarrangeables et non réarrangeables, conçus pour les deux architectures S.I.M.D et M.I.M.D. Au 1-2-3 nous avons présenté la matrice de points de croisement qui est un réseau sans blocage. Nous présentons ici d'autres réseaux utilisés dans les architectures parallèles.

### 1-3-1 : Réseau de Clos :

Le réseau de Clos [1-25] représenté à la figure (1-10) est un réseau de N entrées et N sorties qui comporte trois étages formés respectivement de r matrices de points de croisement n x m , n matrices r x r et r matrices n x m.

Le réseau de Clos est sans blocage si $m \geq 2n-1$ [1-24]

Le nombre de commutateurs $N_c$ du réseau de Clos est de :

$$N_c = 2\,rmn + m\,r^2 \qquad (1\text{-}3\text{-}1)$$

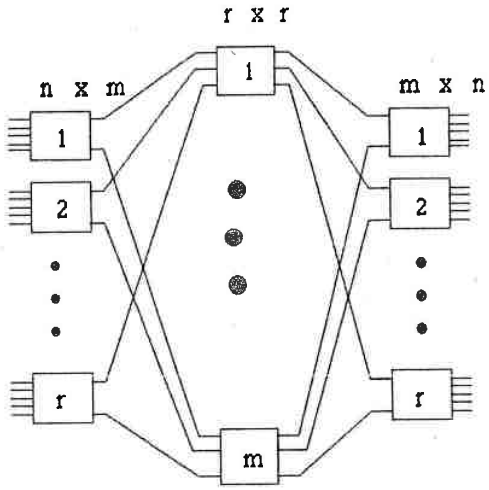



Figure (1-10) réseau de Clos à trois étages.

et si m = 2n-1 alors

$$N_c = 2r\,(2n-1)\,n + (2n-1).\,r^2 \qquad (1\text{-}3\text{-}2)$$

Comme N = r.n, le minimum de $N_c$ est obtenu par $\sqrt{N-\sqrt{N}}$ : c'est-à-dire $n \approx \sqrt{N}$ lorsque N est d'ordre élevé d'où

$$N_c min = 2N\,(2\sqrt{N}-1) + (2+\sqrt{N}-1)\,N \approx 6\,N\sqrt{N} \qquad (1\text{-}3\text{-}3)$$

Le réseau de Clos nécessite ainsi moins de commutateurs que la matrice de points de croisement à partir de N>30.

Le coût de réalisation du réseau de Clos est encore assez élevé. Il est même plus élevé que la matrice de point de croissement pour N<30.

### 1-3-2 Réseaux de Bênes :

On démontre que pour m≥n, le réseau de Clos est réarrangeable [1-25].

Le réseau de Bênes est obtenu à partir d'un réseau de Clos avec m=n=2. Dans ce cas l'étage médian du réseau de Clos se compose de deux commutateurs à N/2 entrées et N/2 sorties. On décompose alors ces deux commutateurs en réseaux de Clos à trois étages et on répète le processus jusqu'à ce que les commutateurs médians soient eux-mêmes des commutateurs 2 entrées x 2 sorties. La figure (1-11) présente un réseau de Bênes pour N=8.



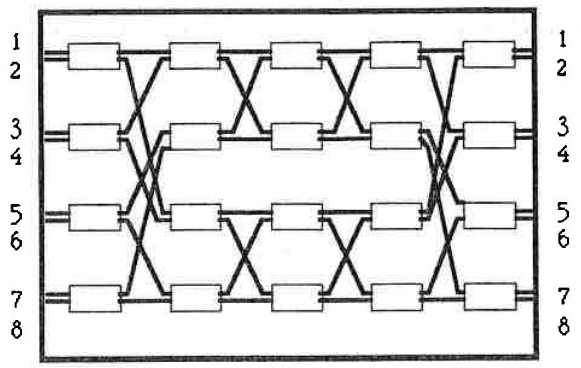



Figure (1-11) : réseau de Bênes.

Le nombre de commutateurs dans un réseau de Bênes est

$$(N/2)\,(2\log_2 N-1) \qquad (1\text{-}3\text{-}4)$$

ce qui constitue une réduction considérable par rapport à la matrice de points de croisement et au réseau de Clos. Pourtant, le fait d'être réarrangeable le rend difficile à utiliser dans les architectures M.I.M.D. Le réseau de Bênes est donc surtout utilisé dans les architectures S.I.M.D. [1-14].

### 1-3-3 Réseau Oméga :

Le réseau Oméga à $N = 2^n$ entrées et autant de sorties, est composéde n étages de $2^{n-1}$ commutateurs élémentaires. Chacune de ces mailles réalise la permutation dite "perfect shuffle", telle que chaque interconnexion prend pour entrée la position dont la représentation binaire est $S_1\ S_2 \ldots\ S_n$ et la permute à la position $S_2\ S_3\ \ldots\ S_n\ S_1$. Après, le commutateur peut permuter la sortie à la position $S_2\ S_3\ \ldots\ S_n 0$ ou $S_2\ S_3\ \ldots$

$S_n1$. La figure (1-12) présente le réseau oméga pour N = 8.

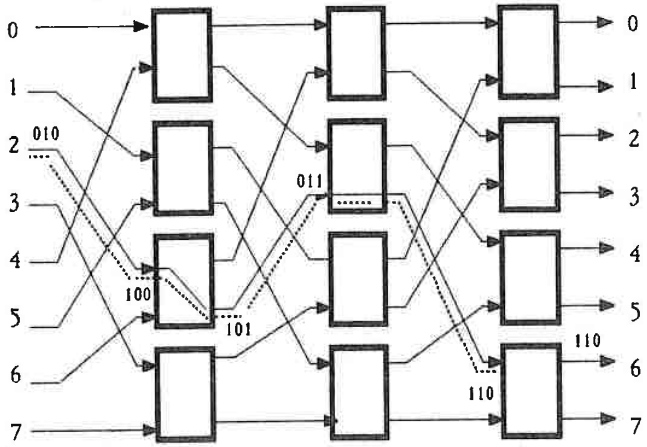



Figure (1-12) : réseau Oméga.

Le nombre de commutateurs du réseau oméga est N/2 $\log_2$N, c'est-à-dire deux fois moins que le réseau de Bênes. En revanche, comme il y a une seule liaison possible entre chaque entrée et chaque sortie, des conflits sont inévitables. La figure (1-13) montre une liaison simultanée de 000 ---->000 et 100 ---->010. où les deux liaisons utilisent une même connexion.

Cette situation conflictuelle est inadmissible, parce que deux signaux différents utilisent le même fil en même temps.



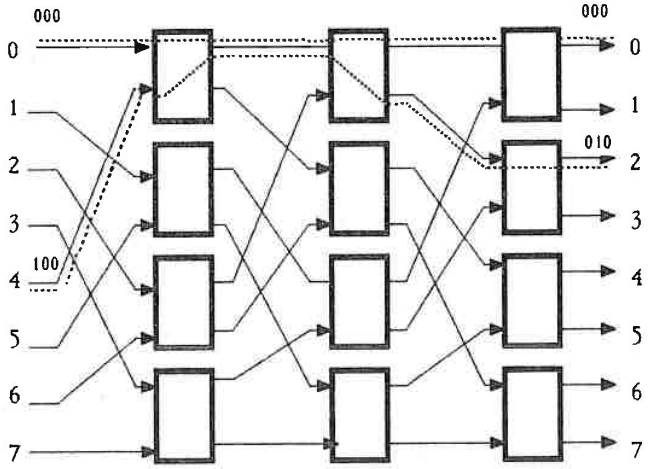



Figure (1-13) : conflit dans un réseau Oméga.

Différentes études, tant par simulation qu'expérimentales ont démontré que ces conflits, sont largement occultés par les conflits d'accès aux bancs : la réduction du débit résultant de l'emploi d'un réseau Oméga à la place d'une matrice de points de croisement reste souvent de l'ordre de 10 à 15% [1-24].

Le réseau Oméga a été étudié pour les deux architectures S.I.M.D et M.I.M.D et il existe des algorithmes d'établissement d'un réseau Oméga, dans les deux cas [1-24], ce réseau Oméga offre également la possibilité de diffusion (figure 1-14).

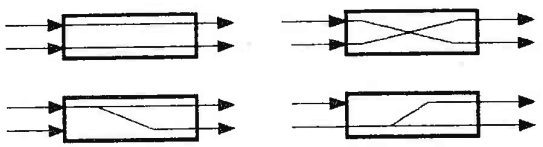


Figure (1-14) : les états des commutateurs d'un réseau Oméga.

## 1-4 CONCLUSION :

Le développement de la technologie L.S.I et V.L.S.I a donné à deux différentes architectures parallèles présentées une grande importance pratique. Nous avons cité très brièvement les problèmes existants tels que l'accès aux mémoires communes et le blocage des réseaux d'interconnexion . Pour les architectures S.I.M.D et M.I.M.D, les réseaux d'interconnexion et de permutation sont désormais une partie intégrante de leurs architectures.

L'application de ces architectures pour l'identification et la commande des systèmes interconnectés dépend largement du problème traité. D'une façon générale ces systèmes sont composés des sous-systèmes de différentes dimensions faiblement couplés. Nous estimons donc qu'une architecture M.I.M.D convient mieux à ce type de problèmes. Le calcul au niveau des sous-systèmes étant essentiellement constitué d'opérations matricielles, on peut choisir à ce niveau les processeurs vectoriels.

# - CHAPITRE 2 -

# CREATION ET ORDONNANCEMENT DES ALGORITHMES PARALLELES

## 2-1 INTRODUCTION:

Un nombre important d'algorithmes numériques conçus pour le calcul en série ont une formulation fortement parallèle.

Par contre d'autres doivent être modifiés en vue de l'implantation sur les systèmes parallèles.

Nous commençons donc ce chapitre par la présentation du parallélisme pour certains algorithmes et par les techniques de sa création pour d'autres.

Nous étudions ensuite les méthodes d'ordonnancement des différentes tâches d'un algorithme. Pour cela nous représentons ces tâches et leurs relations par un graphe. A l'aide de la théorie des graphes nous abordons l'ordonnancement temporel de l'algorithme dans le cas où la durée et d'autres caractéristiques des tâches peuvent être déterminées a priori.

L'ordonnancement par le rang, utile dans le cas où la prédiction de ces caractéristiques est impossible, est abordé en fin de chapitre.

## 2-2 ALGORITHME PARALLELE:

### 2-2-1 Parallélisme des algorithmes numériques

Nous appelons parallélisme la possibilité d'exécuter plusieurs tâches

d'un algorithme simultanément. Dans cette définition nous supposons naturellement que les ressources pour exécuter ces tâches sont illimitées.

Dans ce cas le parallélisme de l'algorithme est déterminé uniquement par l'indépendance de ces tâches.

C'est au travers de la logique d'exécution que l'on décèle la dépendance ou l'indépendance de ces tâches. Ainsi dans un algorithme strictement de type série, les tâches doivent être exécutées l'une après l'autre.

Par contre dans un algorithme parallèle, la fin d'une tâche peut déclencher l'exécution de plusieurs autres tâches.

Nous allons illustrer ce phénomène par quelques exemples.

Exemple 1 :

Dans certains calculateurs on utilise l'algorithme suivant pour la division de deux nombres décimaux N et D : [2-1] , [2-2].

Début de l'algorithme,

par le décalage à droite, rendre le nombre D inférieur à 1

poser i = 0 ; D(i) = D ; N(i) = N ;

tant que (| D(i) - 1| < $\epsilon$) faire

Début

D (i+1) = (2 - D(i)) x D(i) ;



N (i+1) = (2 - D(i)) x N(i) ;

i = i + 1;

fin

Fin de l'algorithme.

A la fin du calcul, D(i) s'approche de 1 et N(i) donne le rapport N/D.

Dans cette algorithme, le calcul de N(i+1) et D(i+1) peut être effectué en parallèle sur deux processeurs. Mais l'exécution du test (| D(i) - 1| < $\epsilon$) ne peut commencer qu'après le calcul de D(i).

Le schéma du calcul est dont le suivant : (figure 1).

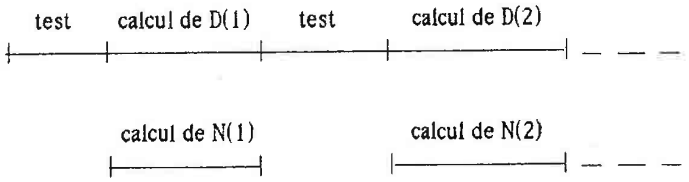



Figure (2-1) Schéma du calcul de N/D

Pour le calcul de D(1) et N(1), deux étapes, une addition et une multiplication, sont nécessaires. Ainsi si l'algorithme converge après k itérations, le calcul de N/D se termine en 3k étapes. Si on dispose d'une structure pipeline on peut modifier cet algorithme pour le rendre plus parallèle :

Début de l'algorithme

Par le décalage à droite, rendre le nombre D inférieur à 1

Poser i=0 ; D(0) = D ; N(0) = N ;

Faire

Début

D(i+1) = (2-D(i))x D(i) ;

N(i+1) = (2-D(i ))x N(i) ;

Fin

Tant que (|D(i-1)-1| <= e)

Fin de l'algorithme

Dans ce cas on peut effectuer le test (|D(i-1)-1|< = e en parallèle avec le calcul de N(i+1) et D(i+1). Le schéma du calcul est alors le suivant :

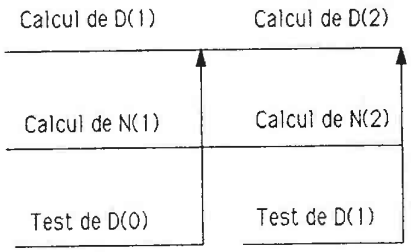



Figure (2-2) Schéma du calcul de N/D utilisant le principe de pipeline

Exemple 2 :

Méthode d'élimination de Gauss pour la résolution d'un système



d'équation linéaire. Supposons le système :

$$A\underline{x} = \underline{b}$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} ; \underline{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} ; \underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

(2-2-1)

L'algorithme de Gauss de triangularisation de A est réalisé par :

Début de l'algorithme

Poser $(a_{ij})^1 = a_{ij}$ ; (i,j = 1 à n)

Pour l=1 à (n-1)

Début

Calculer $[(a_{ll})^l]^{-1}$ ;

Calculer $D_{il} = [-(d_{il}) \times [(d_{ll})^l]^{-1}$ ; i = l+1 à n

Calculer $(a_{ij})^{l+1} = a_{ij} + D_{il}\, a_{lj}$ ; i,j = l+1 à n

Fin

Fin de l'algorithme.

A chaque étape de triangularisation, la suite des calculs dépend du calcul de $[(a_{ll})^l]^{-1}$. Mais le calcul de $[(a_{ll})^l]^{-1}$ déclenche le calcul de tous les $D_{il}$ lequel rend possible le calcul de tous les éléments $(a_{ij})^{l+1}$ de la ligne i.

Ainsi, sur un système multiprocesseurs, on peut effectuer le calcul de la façon suivante :

1- Calcul de $[(a_{ll})^{l}]^{-1}$ sur un processeur en une étape,

2- calcul des $D_{il}$ en parallèle sur au maximum (n-1) processeurs en une étape,

3- calcul des $(a_{ij})^{l+1}$ en parallèle sur au maximum $(n-1)^2$ processeurs en deux étapes.

Chaque étape de triangularisation nécessite quatre opérations. La méthode de Gauss se termine donc en 4(n-1) étapes en utilisant un nombre maximum de $(n-1)^2$ processeurs.

Si on utilise la stratégie du pivot maximum le nombre des étapes du calcul augmente d'une façon considérable.

Pour la recherche du pivot maximum, parmi n pivots, il faut $\log_2 n$ comparaisons (Cf 2-2-2-a). Ainsi, le nombre d'opérations à chaque étape est log n + 4 et le temps total de calcul est de l'ordre de n $\log_2(n^!)$ + 4 (n-1)

### 2-2-2 Méthodes de création des algorithmes parallèles:

Les deux exemples précédents montrent le parallélisme naturel existant dans les algorithmes pourtant conçus pour les calculateurs séries. Mais cette formulation parallèle n'est pas toujours aussi évidante. Dans ces



cas, une simple conversion des algorithmes séries en algorithmes parallèles ne révèle pas tout le parallélisme contenu dans l'algorithme. L'exploitation de ce parallélisme caché nécessite souvent la modification de l'algorithme original. Dans les ouvrages spécialisés il existe des méthodes de création d'algorithmes parallèles que nous allons illustrer par des exemples :

a- Méthode de dichotomie:

Cette méthode consiste à diviser une tâche de dimension N en deux tâches de même complexité, de dimensions N/2 et à continuer cette division récursivement.

Exemple:

Dans la méthode du gradient conjugué, pour le calcul de la matrice de conditionnement, on utilise souvent le schéma suivant : [2-3], [2-4].

$$M^{-1} = \sum_{n=0}^{N} N_i A^n \qquad (2\text{-}2\text{-}2)$$

Dans le calcul série de $M^{-1}$, on utilise la règle de Horner :

$W_n = 1$ ;

$W_{n-1} = (\mu_{n-1} + \mu_n) W_n$ ;

ce qui donne $W_0 = M^{-1}$

Cet algorithme ne convient pas au calcul parallèle, car il nécessite N

multiplications et N additions strictement en série.

La méthode de dichotomie peut être utilisée pour créer un certain parallélisme pour le calcul de $M^{-1}$. Supposons N=8, on peut écrire :

$$M^{-1} = (\mu_0 I + \mu_2 A^2 + \mu_4 A^4 + \mu_6 A^6 + \mu_8 A^8) + A(\mu_1 I + \mu_3 A^2 + \mu_5 A^4 + \mu_7 A^6)$$
$$= F_1(A) + AF_2(A)$$

on calcule donc

1- $A^2$ en une étape

2- $F_1(A)$ et $AF_2(A)$ sur deux calculateurs parallèles en dix étapes

3- $F_1(A) + AF_2(A)$ en une étape

Le calcul de $M^{-1}$ se fait donc en 12 étapes, en utilisant deux processeurs. On peut continuer encore la méthode de dichotomie :

$$F1(A) = (\mu_0 I + \mu_4 A^4 + \mu_8 A^8) + A^2(\mu_2 + \mu_6 A^4)$$
$$F2(A) = (\mu_1 I + \mu_5 A^4) + A^2(\mu_3 + \mu_7 A^4)$$

on calcule donc :

- $A^4$ en deux étapes (au passage on calcule également $A^2$)
- $F_1(A)$ et $F_2(A)$ en 6 étapes sur 4 processeurs
- $F_1(A) + AF_2(A)$ en deux étapes



ainsi le calcul de $M^{-1}$ se fait en 10 étapes utilisant au maximum 4 processeurs.

La méthode de dichotomie est également utilisée pour le calcul de F.F.T. [2-6], du tri rapide [2-7] et beaucoup d'autres exemples.

b- Méthode de vectorisation de l'algorithme

On convertit un algorithme série de calcul direct d'un système en un algorithme itératif qui converge rapidement vers la solution du problème sous certaines conditions.

Exemple:

Résolution du système linéaire

$$A\underline{x} = \underline{b} \qquad (2\text{-}2\text{-}3)$$

avec A matrice tridiagonale.

La méthode directe de résolution de cette équation est décrite par :

Début de l'agorithme

Poser l=1 ; $(A_{ij})^1 = A_{ij}$ i=1 à n ; j=i-1, i, i+1 $(1 \leq j \leq n)$

Pour (l=1 à n-1)

Début

Calculer $[(a_{ll})^l]^{-1}$

Calculer $D_{il} = -(a_{kl})^l . [(a_{ll})^l]^{-1}$ $i=l+1$ $i \leq n$

Calculer $(a_{ij})^{l+1} = (a_{ij})^l + D_{il}(a_{lj})$ $i = l+1 ; j=l+1$ $(i \leq n, j \leq n)$

Fin

Fin de l'algorithme.

A chaque étape de cet algorithme, on ne calcule qu'une seule valeur, c'est donc un algorithme strictement série. Par contre si on utilise les méthodes itératives comme la méthode de Jacobi ou celle du Gradient conjugué, la résolution du système se fait en parallèle sur o(n) processeurs [2-3], [2-4], [2-5].

Nous utilisons cette méthode au chapitre 3 pour la résolution itérative des systèmes linéaires et au chapitre 5 pour le calcul d'une commande optimale.

c - Méthode de partitionnement

Dans cette méthode on décompose le problème en plusieurs sous-problèmes que l'on résoud en parallèle. La solution du problème global est alors une combinaison des solutions des sous-problèmes.

Exemple :

Soit le problème suivant de calcul de commande optimale : minimiser le critère non linéaire :



$$j = h(\underline{x}_f(t)) + \int_0^{t_f} g(\underline{x}, \underline{u}, t)\, dt \qquad (2\text{-}2\text{-}4)$$

relatif au système d'équation dynamique :

$$\frac{d\underline{x}}{dt} = \underline{f}(\underline{x}, \underline{u}, t) \qquad (2\text{-}2\text{-}5)$$

$\underline{x}$ est le vecteur d'état de dimension n, $\underline{u}$ est le vecteur de commande de dimension m et $t_0$ et $t_f$ sont respectivement le début et la fin de la période d'optimisation.

Les conditions d'optimalité pour ce système s'expriment par : [2-8]

$$\frac{d\underline{x}}{dt} = \underline{f}(\underline{x}, \underline{u}, t) \qquad (2\text{-}2\text{-}6)$$

$$\frac{d\underline{\lambda}(t)}{dt} = -\left[\frac{\partial \underline{f}(\underline{x}, \underline{u}, t)}{\partial \underline{x}}\right]^T \underline{\lambda}(t) - \frac{\partial g}{\partial \underline{x}}(\underline{x}, \underline{u}, t) \qquad (2\text{-}2\text{-}7)$$

$$\left[\frac{\partial \underline{f}(\underline{x}, \underline{u}, t)}{\partial \underline{u}}\right]^T \underline{\lambda}(t) + \frac{\partial g}{\partial \underline{u}}(\underline{x}, \underline{u}, t) = 0 \qquad (2\text{-}2\text{-}8)$$

$$\underline{x}(t_0) = \underline{x}_0 \qquad (2\text{-}2\text{-}9)$$

$$\underline{\lambda}(t_f) = \frac{\partial h}{\partial \underline{x}}(\underline{x}(t_f), t_f) \qquad (2\text{-}2\text{-}10)$$

ou $\underline{\lambda}$ est le vecteur des paramètres de Lagrange associé.

Si on calcule $\underline{u}$ dans l'équation (2-2-8) et on obtient à partir de (2-2-6) et (2-2-7) deux équations en $\underline{x}(t)$ et $\underline{\lambda}(t)$

$$\frac{d\underline{H}}{dt} = \underline{\phi}_1(\underline{H}, \underline{\lambda}, t) \qquad (2\text{-}2\text{-}11)$$

$$\frac{d\underline{\lambda}}{dt} = \underline{\phi}_2(\underline{H}, \underline{\lambda}, t) \qquad (2\text{-}2\text{-}12)$$

Supposons $\underline{z} = \begin{bmatrix} \underline{H} \\ \underline{\lambda} \end{bmatrix}$ ; $\underline{\phi} = \begin{bmatrix} \underline{\phi}_1 \\ \underline{\phi}_2 \end{bmatrix}$ et $M = \begin{bmatrix} \underline{H}(t_0) - \underline{H}_0 \\ \underline{\lambda}(t_f) - \frac{\partial h}{\partial \underline{H}}(\underline{H}(t_f), t_f) \end{bmatrix}$

on a alors :

$$\frac{d\underline{z}}{dt} = \underline{\phi}[\underline{z}, t] \qquad (2\text{-}2\text{-}13)$$

$$M(\underline{z}(t_0), \underline{z}(t_f)) = 0 \qquad (2\text{-}2\text{-}14)$$

Pour la partition de cet algorithme, on peut utiliser la méthode suivante : [2-1]

- on divise l'intervalle $[t_0, t_f]$ en (k-1) sous-intervalles

$t_0 = t_1 < t_2 < t_3 \text{ -------} < t_n = t_f$

- on estime une solution $\underline{z}_i$ pour le système 2-2-13 dans chaque intervalle et on résout pour chaque intervalle les systèmes d'équations



$$\frac{d\underline{z}}{dt} = \phi[\underline{z}, t], \quad \underline{z}_i = \underline{z}_i$$

Chaque solution $\underline{z} = \underline{z}(t, \underline{z}_i)$ est définie sur $[t_i, t_{i+1}]$.

On partitionne ainsi le problème original en m-1 sous-problèmes que l'on peut résoudre sur m-1 processeurs en parallèle. La solution globale est une combinaison de ces solutions [1]. Cette méthode est également utilisée pour la recherche du zéro d'une fonction [2-9], [2-10].

d) Implantation asynchrone des algorithmes :

Dans un système multiprocesseurs, on peut avoir des algorithmes répartis sur différents processeurs qui exécutent leurs tâches indépendamment les uns des autres. A partir d'un certain moment, ces tâches doivent être synchronisées pour la suite des calculs. C'est le cas lorsque le commencement d'une nouvelle tâche sur chaque processeur dépend de la fin des plusieurs tâches précédentes. Les processeurs qui ont terminé leurs calculs doivent alors atteindre la fin des tâches les plus longues. Si la différence de temps d'exécution est assez grande ces attentes peuvent réduire considérablement l'efficacité globale du système. Une solution pour surmonter ce problème est d'utiliser un algorithme dit "asynchrone" : chaque processeur reçoit une ou plusieurs variables globales, en fonction de ces variables et des résultats précédents, il effectue les modifications nécessaires sur ces mêmes variables globales ou d'autres variables, avant de commencer son prochain cycle. Ainsi, le processeur n'a pas à attendre les données des autres processeurs. Cette configuration peut

accélérer considérablement l'exécution de la tâche globale sous certaines conditions [2-11].

Exemple :

Considérons le problème de calcul du point fixe :

$x_{i+1} = \phi(x_i)$ (i = 0, 1, .....)

où $x_0$ est donné.

Si $\phi(x) = f(g_1(x)\, g_2(x), ....., g_n(x))$ alors on peut écrire

$x_{i+1} = f(g_1(x_i), g_2(x_i), ....., g_n(x_i))$

Avec un algorithme synchronisé, $g_1(x_i)$, $g_2(x_i)$ .....$gn(x_i)$ sont calculés sur n processeurs; mais pour le calcul de $x_{i+1}$ et donc le commencement de la prochaine itération, le calcul de tous les $g_k(x_i)$ doit être terminé Pour un algorithme asynchrone, dès qu'il y a une nouvelle valeur de $g_k(x_i)$, la valeur de $x_{i+1}$ est modifiée, le premier processeur qui a terminé le calcul de $g_k(n_i)$ le plus vite commence l'itération suivante avec cette nouvelle valeur. On peut trouver les conditions de convergence de ces algorithmes dans [2-12]. Ils ont en particulier été utilisés pour le calcul de commandes optimales des systèmes linéaires avec des contraintes non linéaires sur la commande. [2-13], [2-14].

## 2-3 ORDONNANCEMENT DES ALGORITHMES PARALLELES:



L'ordonnancement d'un algorithme parallèle est la façon d'organiser le calcul de ses différentes tâches afin de minimiser ou de maximiser un critère donné. Dans le calcul parallèle, un critère très important à minimiser est le temps de calcul en présence de contraintes telles que la limitation du nombre de processeurs, la capacité mémoire, etc...

Deux des méthodes les plus connues pour ce type de problème sont la méthode C.P.M dans le cas où le temps d'exécution des tâches est déterministe, et la méthode P.E.R.T dans le cas où le temps présente une distribution probabiliste. Pour mettre en oeure ces méthodes, il est nécessaire de représenter les algorithmes par des graphes.

### 2-3-1 Représentation des algorithmes par des graphes:

Chaque algorithme est composé d'un système de tâches ST = (T,<) où $T = (T_1, T_2, ----, T_n)$ est une ensemble de tâches et < est l'ordre partiel sur T. La relation < est représentée comme suit: si $T_i < T_j$ la tâche $T_i$ doit être terminée avant le commencement de la tâche $T_j$. Dans le cas de ressources illimitées, cet ordre partiel est déterminé par la logique d'algorithme. Dans le cas des ressources limitées, il est déterminé également par les contraintes imposés par ces ressources.

Exemple :

Etudions la résolution du système d'équations linéaires suivants :

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \qquad (2\text{-}3\text{-}1)$$

La liste des tâches et leurs ordres partiels sont les suivants :

| | |
|---|---|
| 1- $x_1 = a_{11}^{-1}\ b_1$ | $T_1 = 1$ |
| 2- $a_{21}x_1$ | $T_2 = 2$ |
| 3- $a_{31}x_1$ | $T_3 = 2$ |
| 4- $b'_2 = b_2 - a_{21}x_1$ | $T_4 = 3$ |
| 5- $b'_3 = b_3 - a_{31}x_1$ | $T_5 = 3$ |
| 6- $x_2 = a_{22}^{-1}\ b_2$ | $T_6 = 4$ |
| 7- $a_{32}x_2$ | $T_7 = 5$ |
| 8- $b''_3 = b'_3 - a_{32}x_2$ | $T_8 = 6$ |
| 9- $x_3 = a_{33}^{-1}\ b''_3$ | $T_9 = 7$ |

Tableau (2-1) l'ordre partiel des tâches du système (2-3-1).

On peut représenter le système des tâches ST = (T,<) par un graphe orienté G=(N,M) où l'ensemble des noeuds N est identique à l'ensemble des tâches T et M est l'ensemble des arcs reliant les noeuds N tels que :



$$T_i < T_j \iff (T_i, T_j) \in M \qquad i \neq j \quad i,j = 1, 2 \cdots\cdots > N$$

Ainsi le graphe représentant l'algorithme de résolution du système (2-3-1) est le suivant :

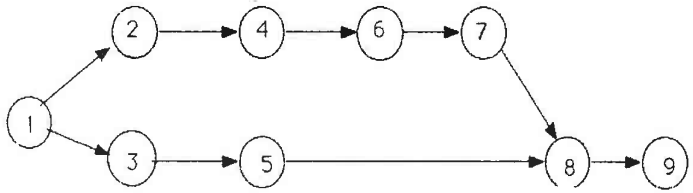



Figure (2-3): Graphe représentant le résolution du système (2-3-1)

Dans cette représentation nous avons éliminé toutes les connections redondantes (connections 1-4, 1-6 ...) pour raison d'inclusion.

### 2-3-2 Pondération du graphe:

Dans le cas d'implantation du système des tâches ST sur un système multiprocesseurs, on peut affecter à chaque tâche un tableau des valeurs nécessaires pour son exécution. Ces valeurs peuvent être par exemple la durée d'exécution de la tâche, ses besoins en place mémoire, en réseau de transfert, etc... Ainsi on peut compléter par exemple le tableau 2-1 par le

tableau 2-2 suivant :

| Tâche | t exécution | besoin en place mémoire |
|---|---|---|
| 1 | t div | 3 |
| 2 | t mult | 3 |
| 3 | t mult | 3 |
| 4 | t add | 3 |
| 5 | t add | 3 |
| 6 | t div | 3 |
| 7 | t mult | 3 |
| 8 | t add | 3 |
| 9 | t mult | 3 |

Tableau (2-2) Caractéristiques des tâches du système( 2-3-1)

Nous appelons pondération du graphe G(N,M) l'affectation de ces valeurs à chaque noeud de G avant l'exécution de l'algorithme. La pondération du graphe n'est pas toujours possible.

Dans le cas où les tâches sont des tâches composées à l'intérieur desquelles il existe des boucles d'une longueur indéterminée (while) ou des branchements logiques, la détermination de la durée d'exécution est impossible. Par exemple ajoutons au système (2-3-1) la condition suivante :

(si $|a_{ii}|<\varepsilon_i$ ) alors 10



$\varepsilon_i$ est une valeur positive prédéterminée et 10 est une tâche qui modifie les valeurs des $a_{ii}$ et les renvoie aux tâches respectives. Dans ce cas le graphe représentatif de l'algorithme est le suivant :

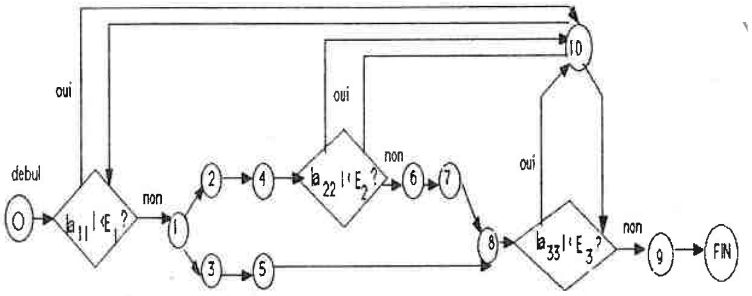



Figure (2-4) Le graphe du système (2-3-1 )avec branchement sur les tâches 1-6 et 9

Si l'ensemble des tâches 1 à 10 forme une tâche composée d'un algorithme plus grand alors la détermination a priori de la durée de cette tâche est impossible. La durée de cette tâche est déterminée par le nombre de boucles du le programme à l'intérieur de cette tâche. Comme les valeurs $a_{ii}$ sont générées dynamiquement par le programme principal, le nombre de fois que le programme boucle à l'interieur de cette tâche est imprévisible. Nous reprendrons ce problème en 2-3-7.

**2-3-3 Fusion du graphe:**

Dans un système parallèle, l'organisation du calcul est l'une des fonctions les plus complexes d'un logiciel. Le temps nécessaire pour que le système réalise cette organisation est en effet beaucoup plus important dans le calcul en parallèle que dans le calcul en série.

A titre d'exemple, le simple transfert d'une donnée d'une mémoire vers le registre ou une mémoire interne d'un processeur pour une architecture utilisant un réseau oméga nécessite déjà la réalisation d'une liaison entre la memoire et le processeur concerné, ce qui peut provoquer le changement d'état de tous les commutateurs du réseau. Si les tâches d'un algorithme sont composées d'opérations élémentaires sur les mots, alors le temps d'organisation du calcul peut même dépasser le temps du calcul proprement dit [2-3], [2-15]. Pour cette raison et pour augmenter la "granulosité" de l'algorithme, il faut fusionner le graphe afin de former des tâches plus importantes. La procédure de fusion dépend de l'algorithme et du système utilisés.

Le choix de tâches de grande dimension diminue normalement le degré de parallélisme de l'algorithme. Si le nombre de processeurs utilisés est assez grand, la fusion de l'algorithme risque de diminuer le temps d'occupation des processeurs et donc de réduire l'efficacité globale du système. La dimension des tâches est donc un compromis entre les possibilités du système et le parallélisme de l'algorithme utilisé.

Une des méthodes de fusion que l'on peut utiliser, en particulier dans l'identification et la commande des systèmes interconnectés, consiste



à travailler sur les blocs de données. Cette idée, largement utilisée pour le calcul de la commande des systèmes à grande dimension, peut avoir beaucoup d'intérêt pour diminuer la part de l'organisation dans le temps total de calcul d'un algorithme parallèle. Nous illustrons ce problème par un exemple

Supposons qu'on veuille effectuer la multiplication de deux matrices A et B sur une architecture en bus partagé.

$$C = A.B$$

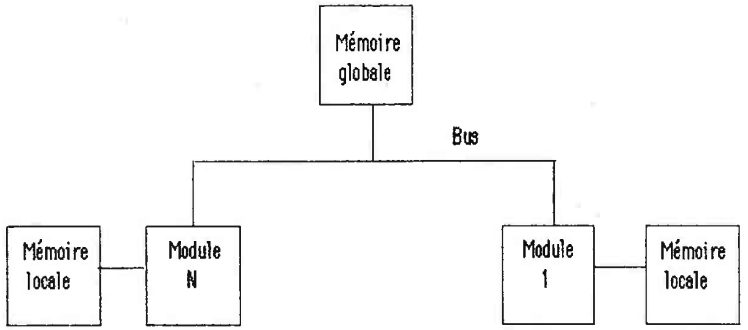



Figure (2-5)

Les matrices A et B sont stockées dans la mémoire globale et la matrice de résultat C est également transférée dans cette mémoire. Pour pouvoir réaliser la multiplication des deux matrices en parallèle on décompose les deux matrices A en N blocs de lignes et la matrice B en N

blocs de colonnes. On effectue la multiplication des deux matrices par les blocs de ligne et de colonne comme cela est montré à la figure (2-6).

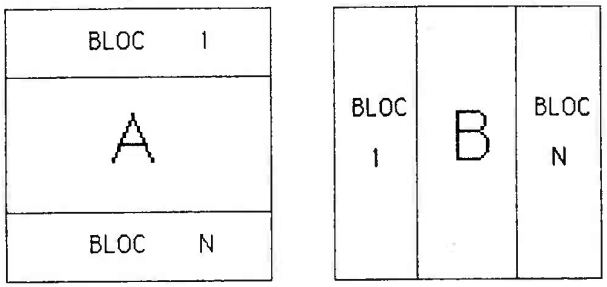



Figure (2-6)

Pour ne pas occuper le bus pendant le calcul, chaque processeur transfère les blocs $A_i$ et $B_i$ correspondants dans sa mémoire locale et libère le bus pour le transfert des autres blocs sur d'autres processeurs. Si n est la dimension de A et B, celle des blocs est de $n^2/N$.

Supposons $t_t$ le temps de transfert d'un nombre réel de la mémoire globale à une mémoire locale et $t_{mult}$ le temps de multiplication de deux nombres réels par un des processeurs. Le transfert des données de mémoire globale aux mémoires locales est propre à l'architecture parallèle; dans une architecture série ce transfert est une notion inexistante.



Le temps de transfert d'un bloc de A ou B est $T_t = (n^2/N)t_t$, et d'un bloc de C est $T'_t = (n^2/N^2)t_t$.

Le temps de multiplication de deux blocs de A et B est

$$T_m = (n^3/N^2)t_{mult}$$

Le schéma de calcul est présenté à la figure (2-7) :

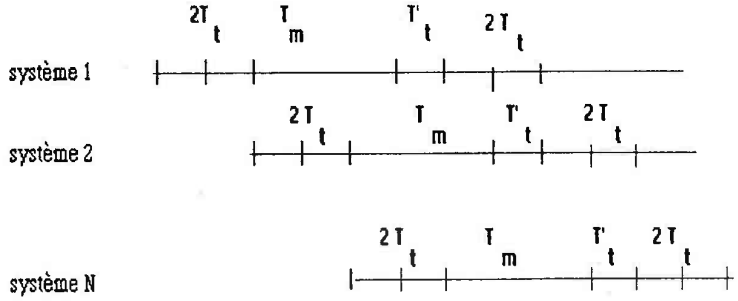



Figure (2-7) : Schéma de calcul pour A.B

Le temps de multiplication de deux matrices $T_{mult}$ est donc :

$$T_{mult} = 2(N-1)T_t + N(T_m + 2T_t + T'_t) \qquad (2\text{-}3\text{-}2)$$

Le temps utile du calcul est de $NT_m$ et le temps de résolution du

système (overhead) est $(4N - 2)\,T_t + NT'_t$ .

On peut donc poser :

$$\frac{t\,\text{système}}{t\,\text{utile}} = \frac{(4N-2) + NT'}{NT_m} = \frac{[(4N-2)\frac{n^2}{N} + \frac{n^2}{N}]t_t}{(n^3/N)t_{mult}} = \frac{4N-1}{n} \times \frac{t_t}{t_{mult}} \qquad (2\text{-}3\text{-}3)$$

pour $(t_t/t_{mult})=1$ , si le nombre de blocs N est supérieur à n/4, le temps de calcul calculé dans (2-3-2) dépasse le temps utile. Pour diminuer le temps de calcul, il faut diminuer le nombre de blocs N et choisir des blocs plus grands. Mais, dans ce cas, si on calcule le taux d'accélération, c'est-à-dire le rapport du temps d'exécution d'algorithme sur un seul processeur avec le temps d'exécution sur le système de multiprocesseurs, on a :

$$\tau_{accel} = \frac{n^3 t_{mult}}{2\,(N-1)\,T_t + N\,(T_m + 2\,T_t + T'_t)} = \frac{n^3\,t_{mult}}{(4\,N-1)\,\frac{n^2}{N}\,t_t + n^3/(Nt_{mult})} = \frac{n.N}{(4N-1)\frac{t_t}{t_{mult}} + n} \qquad (2\text{-}3\text{-}4)$$

La figure (2-8) montre la variation de $\tau_{accel}$ pour n=100 et $(t_t/t_{mult})=1$ en fonction de N



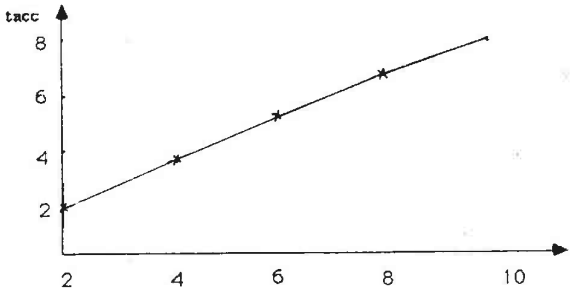



Figure (2-8) Taux d'accéleration de multiplication de deux matrices

Le choix de blocs plus grands (N plus petit) diminue le taux d'accélération. La dimension des blocs est donc choisie par compromis entre le nombre des processeurs utilisés et le taux d'accélération acquis.

### 2-3-4 Détermination de la durée d'exécution des tâches:

D'une façon générale, on peut regrouper les tâches en trois catégories:

1- Les tâches dont la durée d'exécution ne peut pas être déterminée a priori,

2- Les tâches dont la durée peut être déterminée avec exactitude ,

3- Les tâches dont la durée ne peut être déterminée qu'avec une certaine incertitude.

Les tâches de la première catégorie sont celles qui sont composées de boucles de longueurs indéterminées ou de branchements conditionnels (exemple 2-3-1).

Pour les algorithmes composés de ces types de tâches nous présentons une méthode d'ordonnancement dans le paragraphe 2-3-7. Nous étudions donc pour l'instant les deux dernières catégories.

Chaque tâche, quelque soit sa complexité, peut être décomposée en une suite d'instructions du processeur utilisé. Le temps d'exécution de ces instructions est normalement déterminé par les fabricants des processeurs en fonction du cycle d'horloge. Il suffit donc d'écrire le programme d'exécution de la tâche en langage assembleur du processeur concerné et d'en déduire son temps de calcul. Dans [2-16] par exemple, on trouve entre autres, la durée de la multiplication de deux matrices, de l'inversion d'une matrice, et d'une itération de la méthode de Newton, en fonction du cycle d'horloge pour un processeur 77230 de la Société Nec. On trouve également l'évaluation des performances du processeur 56000 de Motorola [2-17] et du processeur 68930 de Thomson [2-18], pour le calcul d'une racine carrée, d'une transformée rapide de Fourrier...etc.

Une méthode moins précise, mais plus rapide, surtout dans le cas du calcul matriciel, est de déterminer le nombre d'opérations mathématiques nécessaires pour l'exécution d'une tâche. La durée de la tâche est déterminée ensuite par la somme des temps nécessaires pour effectuer ces opérations.



Exemples :

- La multiplication de deux matrices de dimensions m.n et n.k nécessite m.n.k multiplications et mk(n-1) additions.

- La résolution d'un système linéaire symétrique par la méthode de Cholesky nécessite [2-19] :
  - n extractions de racines carrées
  - n(n+3)/2 divisions
  - n(n-1)(n+7)/6 multiplications
  - n(n-1)(n+7)/6 additions

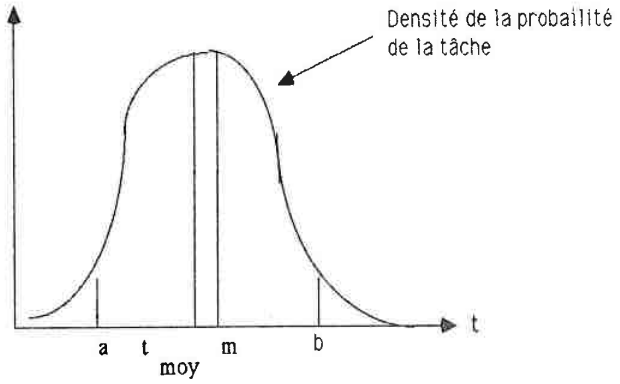
Dans l'annexe A-1, nous donnons une liste des nombres d'opérations mathématiques élémentaires pour les calculs matriciels fréquemment rencontrés dans l'identification et le calcul de la commande des systèmes complexes.

La détermination du temps d'exécution exact d'une tâche n'est pas toujours possible. La durée des opérations mathématiques élémentaires est souvent conditionnée par le mode d'adressage, la précision des données, la présence ou non d'états d'attente (instruction WAIT), le chevauchement des instructions... etc...[2-20].

Dans ce cas, la durée de la tâche est une variable aléatoire pour laquelle il faut déterminer la moyenne et la variance. La durée d'une tâche composée est la somme des durées de ces opérations élémentaires. Si le

nombre de ces opérations est assez grand, on peut appliquer le théorème central limite.La durée de la tâche composée est donc dans ce cas une variable aléatoire normale, dont la moyenne et la variance sont la somme des moyennes et des variances des opérations élémentaires.

Dans le cas où le nombre d'opérations élémentaires d'une tâche n'est pas très élevé, on peut utiliser la méthode des trois temps de la technique de la méthode P.E.R.T. [2-21] présentée à la figure (2-9).




Figure(2-9) Méthode des trois temps

On estime trois temps : le temps optimiste a, le temps pessimiste b et le temps le plus probable m :

- a est la limite inférieur de l'intervalle de confiance à 90 %



- b est la limite supérieur de l'intervalle de confiance à 90 %
- m est la durée la plus probable.

Dans ce cas, la moyenne et la variance de la durée de la tâche sont déterminées par

$$t_{moyenne} = (a+b+4m)/6 \qquad (2\text{-}3\text{-}5)$$

et

$$variance = (b-a)^2/10,2 \qquad (2\text{-}3\text{-}6)$$

### 2-3-5 Ordonnancement de base:

Nous appelons ordonnancement de base l'accomplissement des étapes suivantes :

1- Détermination des relations entre les différentes tâches,

2- Estimation des durées des tâches et de la durée d'exécution de l'algorithme,

3- Estimation du début et de la fin de chaque tâche,

4- Evaluation des ressources nécessaires,

Pour cet ordonnancement nous utilisons la méthode "C.P.M" dans le cas où les durées des tâches sont déterministes et la méthode "P.E.R.T."

dans le cas de durées aléatoires.

L'étape 1 consiste à former le graphe représentatif de l'algorithme qui a été étudié en 2-2-1, 2-2-2 et 2-2-3. Nous avons expliqué une partie de l'étape 2 à 2-2-4 et nous expliquons la suite à partir de l'exemple suivant :

soit à résoudre le système d'équations linéaires, triangulaire par blocs suivant :

$$A\underline{X} = \underline{b} \qquad (2\text{-}3\text{-}7)$$

où A est de la forme suivante :

$$\begin{bmatrix} A_{11} & 0 & 0 & 0 & 0 \\ A_{21} & A_{22} & 0 & 0 & 0 \\ 0 & A_{32} & A_{33} & 0 & 0 \\ A_{41} & 0 & A_{43} & A_{44} & 0 \\ A_{51} & 0 & 0 & A_{54} & A_{55} \end{bmatrix}$$

Nous supposons également que les sous-matrices Aij sont toutes d'ordre 3 et qu'on utilise un système multiprocesseurs pour lequel la



division, la multiplication et l'addition de deux nombres réels se font dans le même temps t.

Les différentes tâches de l'algorithme pour la solution du système (2-3-7) sont résumées dans le tableau (2-3).

| N° | Nature de la tâche | Durée |
|---|---|---|
| 1 | $A_{11}^{-1}$ | 41 t |
| 2 | $A_{22}^{-1}$ | 41 t |
| 3 | $A_{33}^{-1}$ | 41 t |
| 4 | $A_{44}^{-1}$ | 41 t |
| 5 | $A_{55}^{-1}$ | 41 t |
| 6 | $\underline{X}_1 = A_{11}^{-1}\ \underline{b}_1$ | 15 t |
| 7 | $A_{21}\ \underline{X}_1$ | 15 t |
| 8 | $A_{41}\ \underline{X}_1$ | 15 t |
| 9 | $A_{51}\ \underline{X}_1$ | 15 t |
| 10 | $\underline{b}_2^1 = \underline{b}_2 - A_{21}\ \underline{X}_1$ | 3 t |
| 11 | $\underline{b}_4^1 = \underline{b}_4 - A_{41}\ \underline{X}_1$ | 3 t |
| 12 | $\underline{b}_5^1 = \underline{b}_5 - A_{51}\ \underline{X}_1$ | 3 t |
| 13 | $\underline{X}_2 = A_{22}^{-1}\ \underline{b}_2^1$ | 15 t |
| 14 | $A_{32}\ \underline{X}_2$ | 15 t |

Tableau (2-3) Temps d'exécution des tâches du système (2-3-7)



| N° | Nature de la tâche | Durée |
|---|---|---|
| 15 | $\underline{b}_3^1 = \underline{b}_3 - A_{32}\ \underline{X}_2$ | 3 t |
| 16 | $\underline{X}_3 = A_{33}^{-1}\ \underline{X}_3$ | 15 t |
| 17 | $A_{43}\ \underline{X}_3$ | 15 t |
| 18 | $\underline{b}_4^2 = \underline{b}_4^1 - A_{43}\underline{X}_3$ | 3 t |
| 19 | $\underline{X}_4 = A_{44}^{-1}\ \underline{b}_4^2$ | 15 t |
| 20 | $A_{54}\ \underline{X}_4$ | 15 t |
| 21 | $\underline{b}_5^2 = \underline{b}_5^1 - A_{54}\ \underline{X}_4$ | 3 t |
| 22 | $\underline{X}_5 = A_{55}^{-1}\ \underline{b}_5^2$ | 15 t |

Tableau (2-3) suite

La figure (2-10) présente le graphe représentatif de l'algorithme

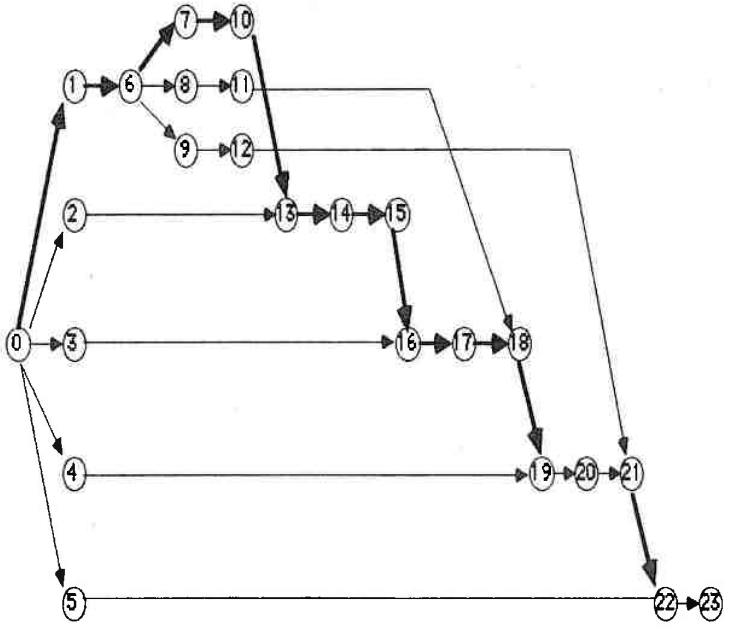



Figure (2-10)

Les tâches 0 et 23 sont respectivement le début et la fin de l'algorithme. Nous désignons par dpt(i) l'instant correspondant au commancement au plus tôt de la tâche i. Si prd(i) est l'ensemble des tâches qui sont prédécesseurs de i, alors on peut écrire

$$dpt(i) = \max (dpt(j) + durée(j)) \; \forall j \in prd(i) \qquad (2\text{-}3\text{-}8)$$



En supposant dpt(0) =0, on peut calculer ainsi l'instant au plus tôt pour le début de chaque tâche. Soit fpt(i) l'instant au plus tôt pour la fin de la tâche i, il est donné par:

$$fpt(i) = dpt(i) + durée(i) \qquad (2\text{-}3\text{-}9)$$

Dans le calcul des fpt(i), on n'a supposé aucune limite sur les ressources (nombre de processeurs, mémoires..) ; fpt(23) est donc la durée limite qu'on peut obtenir avec cet algorithme sur ces type de processeurs. L'augmentation du nombre de processeurs ou de la capacité des mémoires n'affecte pas la valeur de fpt(23).

Soit dptd(i) l'instant au plus tard de début de la tâche i, c'est à dire le temps maximum au delà duquel le commencement de la tâche i entraîne un retard sur la tâche finale. On a donc :

$$dptd(23) = dpt(23)$$

alors si Suc(i) est l'ensemble des successeurs de i on peut écrire

$$dptd(i) = \min\{dptd(j) - durée(j)\} \; \forall j \in Suc(i) \qquad (2\text{-}3\text{-}10)$$

L'instant au plus tard de la fin de la tâche i, fptd(i), est défini alors par :

$$fptd(i) = dptd(i) + durée(i) \qquad (2\text{-}3\text{-}11)$$

et la marge(i) par :

$$marge(i) = dptd(i) - dpt(i) \qquad (2\text{-}3\text{-}12)$$

La marge(i) est le retard qu'on peut tolérer sur la tâche i sans retarder la tâche finale. Le tableau (2-4) résume ces différentes valeurs calculées pour le graphe (2-10).



| N° | DUREE | dpt | fpt | dptd | fptd | MARGE |
|---|---|---|---|---|---|---|
| 1 | 41 | 0 | 41 | 00 | 41 | 0 |
| 2 | 41 | 0 | 41 | 33 | 74 | 33 |
| 3 | 41 | 0 | 41 | 66 | 107 | 66 |
| 4 | 41 | 0 | 41 | 99 | 140 | 99 |
| 5 | 41 | 0 | 41 | 132 | 173 | 132 |
| 6 | 15 | 41 | 56 | 41 | 56 | 0 |
| 7 | 15 | 56 | 71 | 56 | 71 | 0 |
| 8 | 15 | 56 | 71 | 119 | 134 | 63 |
| 9 | 15 | 56 | 71 | 155 | 170 | 99 |
| 10 | 3 | 71 | 74 | 71 | 74 | 0 |
| 11 | 3 | 71 | 74 | 134 | 137 | 63 |
| 12 | 3 | 71 | 74 | 170 | 173 | 99 |
| 13 | 15 | 74 | 89 | 74 | 89 | 0 |
| 14 | 15 | 89 | 104 | 89 | 104 | 0 |
| 15 | 3 | 104 | 107 | 104 | 107 | 0 |
| 16 | 15 | 107 | 122 | 107 | 122 | 0 |
| 17 | 15 | 122 | 137 | 122 | 137 | 0 |

| N° | DUREE | dpt | fpt | dptd | fptd | MARGE |
|---|---|---|---|---|---|---|
| 18 | 3 | 137 | 140 | 137 | 140 | 0 |
| 19 | 15 | 140 | 155 | 170 | 155 | 0 |
| 20 | 15 | 155 | 170 | 155 | 170 | 0 |
| 21 | 3 | 170 | 173 | 170 | 173 | 0 |
| 22 | 15 | 173 | 188 | 173 | 188 | 0 |
| 23 | 0 | 188 | 188 | 188 | 188 | 0 |

Tableau (2-4 ) Caractéristiques temporelles des tâches du graphe (2-10)

On appelle tâches critiques, les tâches dont la marge est égale à 0. Ces tâches ne peuvent pas être retardées sans retarder la tâche finale. Par conséquence, le chemin reliant ces tâches sur le graphe est appelé chemin critique. Nous avons mis en évidence le chemin critique sur le graphe (2-10) par une ligne double.

Le nombre maximum de processeurs pour exécuter cet algorithme avec une durée correspondant au chemin critique est le nombre maximum de tâches de même rang sur le graphe. Pour le graphe de la figure (2-10), ce nombre est 5. Donc avec 5 processeurs on peut résoudre le système (2-3-7) en fpt (23) c'est-à-dire en 188 t. L'exécution du même algorithme sur un seul processeur nécessite un temps :



$$t_{mono} = 388\ t$$

Le taux d'accélération de l'exécution de l'algorithme en parallèle sur 5 processeurs est alors de :

$$taccel = \frac{388\ t}{188\ t} = 2,1$$

Nous appelons taux d'occupation le rapport entre le temps moyen d'occupation des processeurs et le temps d'exécution de l'algorithme. On peut écrire :

$$\tau_{occup} = \frac{\tau_{accel}}{n\ processeurs} \qquad (2\text{-}3\text{-}13)$$

Pour cet exemple :

$$\tau_{occup} = \frac{2,1}{5} = 0,42$$

On peut constater que le taux d'occupation des processeurs est très bas, seulement 42% du temps total.

La question qui se pose maintenant est de savoir si l'on peut diminuer le nombre de processeurs sans retarder la tâche finale.

Nous répondons à cette question dans le paragraphe suivant.

### 2-3-6 Ordonnancement des algorithmes parallèles en présence de contraintes:

Dans 2-3-5, nous avons supposé que les ressources sont considérées comme illimitées. Dans la pratique cette hypothèse n'est pas toujours vérifiée et même si c'est le cas, cela peut conduire à un rendement très insuffisant du système global. Nous avons constaté cette insuffisance dans l'exemple précédent.

En général, le problème d'ordonnancement d'un algorithme parallèle se pose de la façon suivante [2-1] :

On dispose d'un système multiprocesseurs composé de paires :

- MS = (P, R) ou $p = (p_1, P_2, \ldots p_m)$ est un ensemble des processeurs,
- R = (TR, M) est une paire qui détermine les ressources secondaires comme les mémoires, les entrées/sorties, etc....
- $M = (m_1, m_2, \ldots m_p)$, donne le nombre de ressources secondaires de même type.

Par exemple si $TR_2$ désigne des pages mémoires et $m_2 = 10$, on dispose de 10 pages mémoires.



Les processeurs peuvent être identiques ou avoir différentes vitesses ou même différentes fonctions. On dispose également d'un algorithme ou d'un système de tâches ST représenté par un graphe. L'algorithme d'ordonnancement est alors la manière d'organiser le calcul de ST sur MS, afin de satisfaire un critère. Le critère peut être par exemple :

a) minimiser le temps d'exécution de ST sur MS,
b) minimiser le nombre de processeurs et de ressources secondaire pour un temps donné,
c) minimiser le temps moyen d'exécution de ST sur MS.

En automatique, on peut rencontrer ces trois critères de façon directe ou approchée. On peut même avoir plusieurs critères en même temps. Supposons que l'on dispose d'un système interconnecté d'une structure donnée, invariante au cours du fonctionnement. L'implantation d'un dispositif de commande parallèle pour ce système peut se poser de différentes manières :

1- concevoir un système multiprocesseurs qui calcule et exécute la commande en un temps donné (critère b),

2- minimiser le temps de calcul et d'exécution de la commande sur un système multiprocesseur existant (critère a),

3- choisir parmi les multiprocesseurs existants, un système qui convient mieux au système commandé et minimiser ensuite le temps de calcul de la commande sur ce système (critère b et a).

Ainsi par exemple, pour le système d'équation (2-3-7), on peut se poser les questions suivantes :

1- quel est le nombre minimum de processeurs inférieur ou égal à 5 pour ne pas dépasser le temps du chemin critique(critèreb)

2- quel est le temps minimum d'exécution de l'algorithme sur un nombre de processeurs inférieur à 5 (critère a)

On montre dans [2-1], [2-22] que le problème d'ordonnancement optimal est NP complet, c'est-à-dire qu'il a une variation expotentielle en fonction du nombre de tâches. Dans la majorité des cas, on utilise donc, des algorithmes heuristiques, surtout quand le système de tâches est complexe. Un algorithme heuristique qui donne des résultats satisfaisants dans de nombreuses applications est l'algorithme basé sur la méthode C.P.M. [2-1], [2-23], [2-24]. Cet algorithme se déroule de la manière suivante :

Début de l'algorithme

1-Calculer dpt, dptd et la marge de chaque tâche en commençant à t=0

2- Eliminer les tâches dont l'exécution est déjà terminée (à t=0, on a la tâche 0).

Lister ensuite tous les successeurs de ces tâches et réactualiser le nombre de ressources disponibles.

3- Déterminer parmi les successeurs, les tâches dont tous les



prédécesseurs ont déjà été exécutés. Ce sont les tâches prêtes à exécution : T.P.E.

4- Faire une liste des T.P.E
- par ordre croissant de leurs marges
- pour les mêmes marges, par l'ordre croissant de la durée d'exécution
- pour les mêmes marges et durées par ordre décroissant de la place mémoire nécessaire

5- S'il existe des ressources suffisantes, attribuer les tâches dans l'ordre de la liste 4 et réactualiser le nombre des ressources

6- Si toutes les tâches ont été attribuées, terminer le calcul, sinon poser $t_{nouveau} = t_{ancien} + 1$ et reprendre l'étape 2

Fin de l'algorithme.

La complexité de cet algorithme pour le calcul du chemin critique est $0(M)$ et celle de l'ordonnancement de la liste des T.P.E est de $O(N \log_2 N)$ [2-7].

L'application de cet algorithme au système (2-3-7) donne les résultats suivants :

| m processeurs | t exécution | t accel | t occup |
|---|---|---|---|
| 5 | 188 t | 2,06 | 0,41 |
| 3 | 188 t | 2,06 | 0,62 |
| 2 | 207 t | 1,87 | 0,94 |

Tableau (2-5):

Les figures (2-11, a, b, c) montrent respectivement l'ordonnancement du calcul sur cinq, trois et deux processeurs ;

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 7 | 10 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 2 | | 8 | 11 | | | | | | | | | | |
| 3 | | 9 | 12 | | | | | | | | | | |
| 4 | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | |

Figure (2-11 a) : 5 processeurs.



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 7 | 10 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 2 | 4 | | | 8 | 11 | | | | | | | | |
| 3 | 5 | | | 9 | 12 | | | | | | | | |

Figure (2-11 b) : 3 processeurs.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 7 | 10 | 13 | 14 | 15 | 16 | 8 | 5 | | | 12 | |
| 2 | 3 | | | 4 | | | | 17 | 11 | 18 | 19 | 20 | 21 | 22 |

Figure (2-11 c) : 2 processeurs.

On peut remarquer que le nombre minimal de processeurs pour exécuter le programme sur une durée égale à celle du chemin critique est de 3. Le taux d'occuppation dans ce cas est de 69 %. Il est meilleur que pour 5 processeurs, mais il est encore assez bas. Pour deux processeurs, le temps de calcul dépasse de 10 % la durée du chemin critique, mais en revanche le taux d'occuppation passe de 69 % à 94 % qui est un taux excellent. Donc dans cet exemple, si l'exécution de l'algorithme en temps minimal du chemin critique n'est pas une nécessité absolue, le choix de deux processeur est très intéressant. La figure (2-12) fournit le graphe correspondant pour l'ordonnancement de cet algorithme sur 2 processeurs

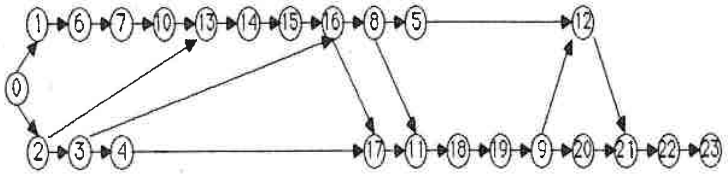



Figure (2-12) Ordonnancement du calcul sur deux processeurs

La comparaison des graphes (2-10) et (2-12) nous apprend que

- de nouveaux arcs ont été crées qui sont purement dus aux contraintes du système (arc 2-3, 3-4, 4-17 etc...)
- d'autres arcs redondants ont disparu ; les arcs 6-9, 6-8, 4-19, etc...
- le nombre d'arcs dans le graphe (2-10) et (2-11) est 29; donc la complexité de programmation en parallèle, liée aux problèmes de syncronisations, (voir[2-23])est pratiquement la même pour les deux graphes.

### 2-3-7 Ordonnancement des tâches avec la durée indéterminée:

Considérons un programme parallèle composé de tâches de type (2-3-1) avec des branchements et des boucles interieures. Dans ce cas la durée des tâches ne peut pas être déterminée a priori. Nous supposons que le programme est donné sous la forme d'un graphe G=(N,T,M), où le noeud $n_i \in N$ correspond à la tâche i, $t_i \in T$ est le temps d'exécution de cette tâche i, (une variable aléatoire dans l'intervalle o et $t_{max}$) et l'ensemble M traduit les relations entre les différentes tâches. Comme dans le cas



précédent, nous formons à chaque instant l'ensemble des tâches prêt à l'exécution T.P.E. Dès qu'un processeur est libre, il faut lui affecter une tâche parmi les T.P.E. Comme la durée des T.P.E. est indéterminée, on choisit la tâche i pour laquelle

$$P(T_i \leq T_i) \geq P(T_j \geq T_i) \forall i,\ j \in \text{T.P.E.} \qquad (2\text{-}3\text{-}14)$$

Supposons $r_i$ le rang de la tâche i par rapport à la tâche finale ; alors on montre dans [2-1] que la stratégie de choix de la tâche avec le rang maximum correspond à la condition (2-3-14). Dans ce cas, chaque fois que la liste des T.P.E. est renouvelée par l'arrivée des nouvelles tâches, il faut ordonner cette liste par ordre décroissant des rangs. Le rang de chaque tâche est constant et peut être calculé avant l'exécution du programme. Mais l'ordonnancement des T.P.E. doit être fait dynamiquement. Cela nécessite que le temps moyen d'exécution des tâches soit assez grand pour amortir le temps d'ordonnancement.

## 2-4 CONCLUSION

Nous avons explicité le parallélisme et les méthodes de création des algorithmes parallèles. Pour faciliter l'étude de ces algorithmes, nous les avons représentés par des graphes. En utilisant les techniques C.P.M et P.E.R.T., nous avons calculé le taux d'accélération maximum pour chaque algorithme. Dans le cas de ressouces limitées, nous avons démontré que le choix de tâches, de marge et de durée les plus petites conduit à un temps d'exécution et à un taux d'occupation raisonnables. Dans le cas de tâches de

durée indéterminée, le choix des tâches avec rang maximum peut améliorer le temps d'exécution du programme.

# - CHAPITRE 3 -

# CALCUL PARALLELE DES SYSTEMES LINEAIRES

## 3-1 INTRODUCTION

Les techniques de l'algèbre linéaire jouent un rôle essentiel dans les problèmes de l'identification et la commande des systèmes interconnectés. En effet de nombreuses méthodes en Automatique utilisent les calculs matriciels, ou les systèmes d'équations linéaires. On peut citer par exemple le filtre de Kalman, la méthode des moindres carrés, la commande optimale des systèmes linéaires, etc... Nous avons donc consacré ce chapître à l'étude des méthodes de résolution parallèle des problèmes d'algèbre linéaire.

Nous étudions d'abord la résolution parallèle des systèmes d'équations linéaires triangulaires. Nous abordons ensuite la méthode de Gauss pour la résolution directe des systèmes d'équations linéaires quelconques. Nous étudions ensuite les systèmes symétriques définis positifs qui ont une importance particulière en Automatique. Nous abordons également les techniques de résolutions parallèles des systèmes linéaires par les méthodes itératives et nous présentons enfin les méthodes parallèles d'inversion d'une matrice. Dans chaque cas nous étudions le problème d'ordonnancement des calculs afin de minimiser, soit le temps de calcul, soit le nombre de processeurs.

## 3-2 Méthode directe de résolution des systèmes d'équations linéaires

### 3-2-1 Cas des systèmes triangulaires

Soit le système linéaire, sous forme triangulaire, suivant :

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ . & & \\ . & & \\ a_{n1} & a_{n2} & \cdots a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix} \qquad (3\text{-}2\text{-}1)$$

On suppose $a_{ii}$ ( i=1 à n) ≠ 0

La résolution de ce système par la *méthode classique de remontée* [3-1] donne

$$x_1 = a_{11}^{-1} b_1$$

$$x_2 = a_{22}^{-1} (b_2 - a_{21} x_1)$$

.....

$$x_n = a_{nn}^{-1} (b_n - a_{n1} x_1 - a_{n2} x_2 \cdots\cdots - a_{n-1} x_{n-1}) \qquad (3\text{-}2\text{-}2)$$

Les variables $x_1$, $x_2$, ....., $x_n$ sont donc calculées en série l'une après l'autre. Cette méthode nécessite au total

$$1+2+ \cdots\cdots +(n-1) = \frac{n(n-1)}{2} \quad \text{additions}$$

$$1+2+ \cdots\cdots +(n-1) = \frac{n(n-1)}{2} \quad \text{multiplications} \qquad (3\text{-}2\text{-}3)$$

$$n \quad \text{division}$$

Cette méthode s'étend aux matrices triangulaires par blocs.

En calcul parallèle, nous pouvons résoudre le système 3-2-1 de la façon suivante :

calculer $x_1 = a_{11}^{-1} b_1$



calculer $b_i^1 = b_i - a_{i1} x_1$ (i=2 à n) en parallèle

calculer $x_2 = a_{22}^{-1} b_2^1$ (3-2-4)

calculer $b_i^2 = b_i^1 - a_{i2} x_2$ (i=3 à n) en parallèle

calculer $x_n = a_{nn}^{-1} b_n^{n-1}$

La résolution du système (3-2-1) s'effectue donc sur au maximum (n-1) processeurs identiques avec un temps nécessaire pour effectuer:

n divisions + (n-1) multiplications +(n-1) additions (3-2-5)

En comparant le nombre d'opérations de 3-2-3 et 3-2-5, le taux d'accélération d'algorithme sur (n-1) processeurs est

$$t_{accel} = \frac{nt_{div} + \frac{n(n-1)}{2} t_{mult} + \frac{n(n-1)}{2} t_{add}}{nt_{div} + (n-1)\, t_{mult} + (n-1)\, t_{add}} \qquad (3\text{-}2\text{-}6)$$

Le graphe correspondant est indiqué à la figure (3-1)

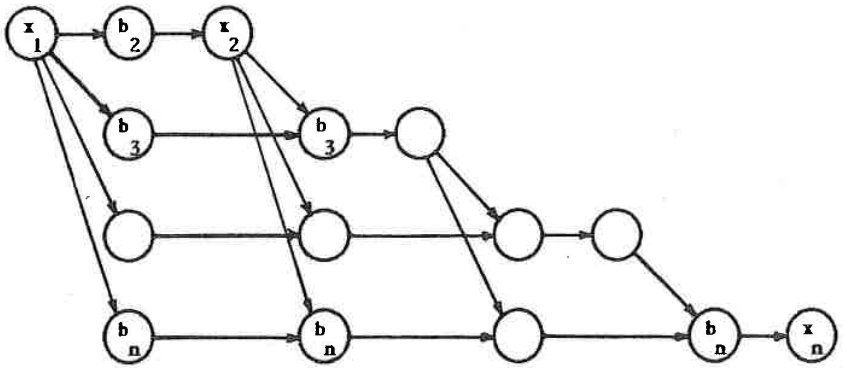



Figure (3-1): graphe de la méthode de remontée

Le taux d'accélération (3-2-6) dépend des processeurs utilisés et nous l'étudions dans trois cas :

1 - $t_{div} = t_{mult} = t_{add}$

2 - $t_{div} = 6\ t_{add}$ ; $t_{mult} = 5\ t_{add}$ qui sont les temps représentatifs des processeurs classiques [3-2], [3-3]

3 - $t_{div} = 34\ t_{add}$ ; $t_{mult} = t_{add}$ pour les processeurs rapides [3-4], [3-5].

Premier cas :



$$t_{accel} = \frac{n + n(n-1)}{n + 2n - 2} = \frac{n^2}{3n - 2} \qquad (3\text{-}2\text{-}7)$$

et sa valeur asympthotique est :

$$t_{accel} = n/3 \qquad (3\text{-}2\text{-}8)$$

Deuxième cas :

$$t_{accel} = \frac{n + \frac{n(n-1)}{2}}{2n - 1} = \frac{n(n+1)}{2(2n-1)} \qquad (3\text{-}2\text{-}9)$$

et sa valeur asympthotique est :

$$t_{accel} = n/4 \qquad (3\text{-}2\text{-}10)$$

Troisième cas :

$$t_{accel} = \frac{34n + n(n-1)}{34n + 2n - 2} = \frac{n^2 + 33n}{36n - 2} \qquad (3\text{-}2\text{-}11)$$

et sa valeur asympthotique est :

$$t_{accel} = n/36 \qquad (3\text{-}2\text{-}12)$$

On peut remarquer que le taux d'accélération dans le dernier cas est 12 fois plus petit que le premier cas.

Pour améliorer le taux d'accélération nous proposons de modifier l'algorithme (3-2-4) de la façon suivante :

calculer $d_{ii} = a_{ii}^{-1}$ (i = 1 à n) en parallèle

calculer $x_1 = d_{11}.b_1$

calculer $b^1_i = b_i - a_{i1}\, x_1$ (i = 2 à n) en parallèle (3-2-13)

calculer $x_2 = d_{22}\, b_2$

calculer $b_i^2 = b_i - ai_2\, n_2$ (i = 3 à n) en parallèle

.....

calculer $x_n = d_{nn}.b^{n-1}_n$

La figure (3-2) donne le graphe correspondant à cet algorithme

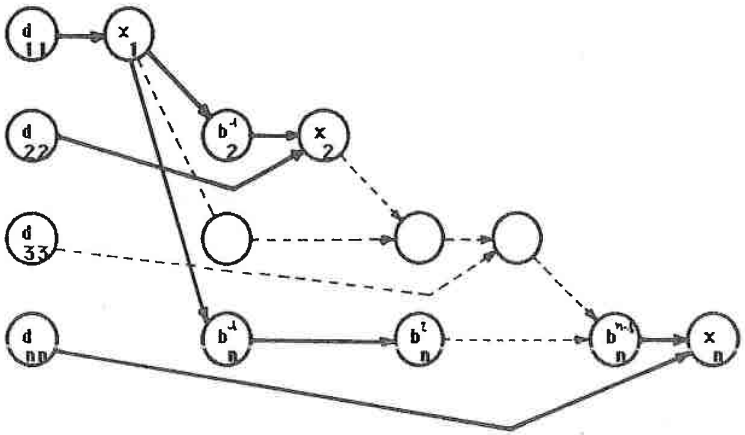



Figure (3-2) : graphe de l'algorithme (3-2-13)



Le chemin critique du graphe (3-2) est composé de :

-1 division + (2n - 1) multiplications + (n - 1) additions (3-2-14)

et le taux d'accélération par rapport à 3-2-3 devient:

$$\tau_{accel}=[n\, t_{div}+n(n-1)/2\, t_{mult}+n(n-1)/2]/[t_{add}+(2n-1)t_{mult}+(n-1)t_{add}] \quad (3\text{-}2\text{-}15)$$

Le taux d'accélération pour les trois cas précédents, est donc :

Premier cas:

$$t_{accel} = \frac{n^2}{3n-1} \quad (3\text{-}2\text{-}16)$$

et sa valeur asymptotique est:

$$t_{accel} = n/3 \quad (3\text{-}2\text{-}17)$$

Deuxième cas :

$$t_{accel} = \frac{6n + \frac{5n\,(n-1)}{2} + \frac{n\,(n-1)}{2}}{6 + 5\,(2n-1) + (n-1)} = \frac{3}{11}\,(n+1) \quad (3\text{-}2\text{-}18)$$

et sa valeur asymptotique est:

$$t_{accel}(asymp) = \frac{3}{11} n = 0{,}273\ n \qquad (3\text{-}2\text{-}19)$$

Troisième cas :

$$t_{accel} = \frac{34n + n(n-1)}{1 + 3n - 2} = \frac{n^2 + 33n}{3n - 1} \qquad (3\text{-}2\text{-}20)$$

et sa valeur asymptotique est :

$$t_{accel} = n/3 \qquad (3\text{-}2\text{-}21)$$

On peut remarquer que dans cette méthode, le taux d'accélération est pratiquement stable. Par rapport à la précédente, il n'est pas amélioré dans les deux premiers cas, mais son amélioration est considérable dans le troisième cas.

SYSTEME TRIANGULAIRE PAR BLOCS

Supposons maintenant le système linéaire :

$$\begin{bmatrix} A_{11} & & & \\ A_{11} & A_{22} & & \\ & & \ddots & \\ A_{N1} & A_{2N} & & A_{NN} \end{bmatrix} \begin{bmatrix} \underline{X}_1 \\ \underline{X}_2 \\ \\ \underline{X}_N \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \\ \underline{b}_N \end{bmatrix} \qquad (3\text{-}2\text{-}22)$$

où $A_{ij}$ sont des sous-matrices généralement d'ordre différents et $\underline{X}_1$,



$\underline{X}_2$, ---, $\underline{X}_n$, $\underline{b}_1$, $\underline{b}_2$, ---, $\underline{b}_n$ sont des vecteurs de même dimension que les matrices correspondantes. De plus les matrices $A_{ii}$ (i = 1 à N) sont inversibles. Dans ce cas l'algorithme (3-2-4) s'applique de la façon suivante

Résoudre le système linéaire $A_{11}\,\underline{X}_1 = \underline{b}_1$

Calculer $\underline{b}^1{}_i = \underline{b}_i - A_{i1}\,\underline{X}_1$ (i = 2 à N) en parallèle

Résoudre le système linéaire $A_{22}\,\underline{X}_2 = \underline{b}_2{}^1$

Calculer $\underline{b}_i{}^2 = \underline{b}^1{}_i - A_{i2}\,\underline{X}_2$ en parallèle (3-2-23)

---

Résoudre le système $A_{NN}\underline{X}_n = \underline{b}_N{}^{N-1}$

Les divisions de l'algorithme (3-2-4) ont été remplacées par les résolutions de systèmes linéaires et les multiplications de deux nombres réels par les multiplications de matrices et de vecteurs. La résolution d'un système linéaire nécessite plus d'opérations élémentaires que la multiplication matrice-vecteur.

Par exemple pour $v = 5\ t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$, on obtient :

- pour la résolution d'un système linéaire par la méthode de Gauss $440\ t_{add}$,
- pour la multiplication matrice-vecteur $145\ t_{add}$.

Les systèmes triangulaires par blocs ressemblent au cas où le temps de calcul de division est supérieur au temps de calcul de multiplication et

d' addition.

Pour avoir un taux d'accélération plus élevé, il est donc préférable d'utiliser l'algorithme (3-2-13). La résolution préalable des systèmes $A_{ii}\ X_i = \underline{b}_i^{i-1}$ n'est pas possible, parce que les $\underline{b}_i^{i-1}$ sont déterminés au cours du calcul. On peut résoudre ce problème de deux façons :

a) factorisation LU préalable des $A_{ii}$

b) inversion préalable des $A_{ii}$

La première méthode nécessite moins d'opérations, mais à la condition que tous les déterminants mineurs de $A_{ii}$ soient non nuls [3-1]. Pour la deuxième méthode, nous avons supposé que les matrices $A_{ii}$ sont inversibles. Cette méthode est donc toujours possible.

Nous avons étudié les deux algorithmes (3-2-4) et (3-2-13) pour un système de l'ordre N = 8, composé de blocs de dimension 6.

Dans le cas où $t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$, les résultats obtenus sont les suivants :



| ALGORITHME | TEMPS DU CHEMIN CRITIQUE | TAUX D'ACCELERATION |
|---|---|---|
| 3-2-4 | $4570\ t_{add}$ | 1,69 |
| 3-2-13 | $2650\ t_{add}$ | 2,9 |

Tableau (3-1) : temps d'exécution pour les algorithmes 3-2-4 et 3-2-13.

ORDONNANCEMENT DU CALCUL

Le temps de calcul correspondant au chemin critique pour la résolution d'un système linéaire triangulaire de l'ordre n est obtenu sur au maximum (n-1) processeurs.

Le taux moyen d'occupation des processeurs dans ce cas est :

$$t_{occup} = \frac{t_{accel}}{(n-1)} \qquad (3\text{-}2\text{-}24)$$

ainsi par exemple, pour le $t_{accel}$ calculé au (3-2-9) on a :

$$t_{occup} = \frac{n\,(n+1)}{2\,(n-1)\,(2n-1)} \qquad (3\text{-}2\text{-}25)$$

et sa valeur asymptotique est:

$$\tau_{asymp} = 0,25. \qquad (3\text{-}2\text{-}26)$$

Les processeurs sont occupés seulement 25% du temps, soit un taux extrêment faible. Pour augmenter ce taux, nous utilisons donc la méthode d'ordonnancement présentée au chapitre 2. Le tableau (3-2) résume les résultats obtenus pour un système d'ordre n = 8.

| Nombre de processeurs | Temps de calcul | Taux d'acceleration | Taux d'occupation |
|---|---|---|---|
| 7 | $90\ t_{add}$ | 2,4 | 0,34 |
| 6 | $90\ t_{add}$ | 2,4 | 0,4 |
| 5 | $90\ t_{add}$ | 2,4 | 0,48 |
| 4 | $90\ t_{add}$ | 2,4 | 0,6 |
| 3 | $90\ t_{add}$ | 2,4 | 0,8 |
| 2 | $120\ t_{add}$ | 1,8 | 0,9 |

Tableau (3-2) : taux d'accélération et d'occupation.

Pour l'algorithme (3-2-4) avec n = 8, $t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$, on peut remarquer que la durée du chemin critique est obtenue avec 3 processeurs au lieu de 7. Le taux d'occupation des processeurs dans ce cas est de 0,8 qui est un taux honnête.

La figure (3-3) donne l'organisation du calcul sur trois processeurs.



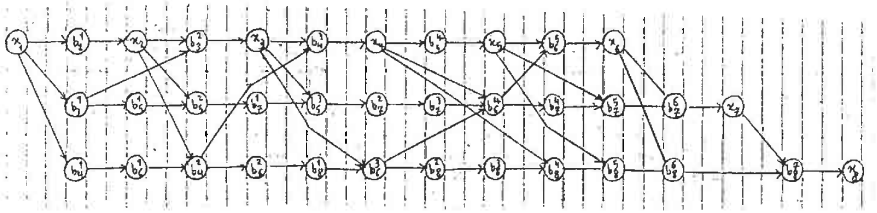


Figure (3-3) : organisation du calcul sur 3 processeurs.

Le tableau (3-3) donne les résultats obtenus pour l'algorithme (3-2-13) dans le cas d'un système de blocs triangulaires d'ordre N=8 et de dimension de blocs v = 5 dans le cas où $t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$.

| Nombre de processeurs | Temps de calcul | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 7 | 2650 $t_{add}$ | 2,91 | 0,42 |
| 6 | 2650 $t_{add}$ | 2,91 | 0,485 |
| 5 | 2650 $t_{add}$ | 2,91 | 0,582 |
| 4 | 2795 $t_{add}$ | 2,76 | 0,69 |
| 3 | 3110 $t_{add}$ | 2,48 | 0,83 |
| 2 | 4585 $t_{add}$ | 1,68 | 0,84 |

Tableau 3-3 : temps de calcul de l'algorithme 3-2-13

Le nombre minimal de processeurs pour exécuter l'algorithme dans le temps du chemin critique est de 5. Ce nombre dépend de l'ordre du système et de la dimension des blocs.

COMPLEXITE DE L'ALGORITHME D'ORDONNANCEMENT.

Le nombre de noeuds dans le graphe 3-1 est:

:

$$N_{noeud}=n(n-1) \qquad (3\text{-}2\text{-}27)$$

La complexité de l'algorithme d'ordonnancement est donc:

$$O(n^2 \log n) \qquad (3\text{-}2\text{-}28)$$



Pour n suffisamment grand, cette valeur dépasse le nombre d'opérations nécessaires pour la résolution du système (3-2-1). L'ordonnancement dynamique de calcul n'est donc pas possible. Mais dans le cas d'un système par blocs avec des blocs d'ordre suffisament grands, ou dans le cas où le même système d'équations doit être résolu plusieurs fois, cet ordonnancement peut être utile.

Exemple :

Le problème d'équilibrage de bilan d'un hydrocyclone [3-6] aboutit à la résolution du système non linéaire suivant :

$$\begin{aligned} f_1(\underline{x}_1) &= \underline{b}_1 \\ f_2(\underline{x}_1,\underline{x}_2) &= \underline{b}_2 \\ f_3(\underline{x}_1,\underline{x}_2,\underline{x}_3) &= \underline{b}_3 \\ f_4(\underline{x}_1,\underline{x}_2,\underline{x}_3,\underline{x}_4) &= \underline{b}_4 \end{aligned} \qquad (3\text{-}2\text{-}29)$$

$\underline{f}_1, \underline{f}_2, \underline{f}_3, \underline{f}_4$, ainsi que $\underline{x}_1,\underline{x}_2,\underline{x}_3,\underline{x}_4$ sont respectivement d'ordre 9, 3, 3, et 4.

Si on utilise une des méthodes classiques de calcul itératif de cette équation, à chaque pas d'itération il faut résoudre un système triangulaire par blocs. Comme la structure du système ne change pas au cours du calcul, le même système est résolu plusieurs fois.

Dans cet exemple, la complexité de l'algorithme d'ordonnancement est d'ordre $2x4^2=32$ tandis que le nombre d'opérations pour la résolutions

du système (3-2-29) dépasse largement le nombre 32.

### 3-2-2 Méthode de Gauss pour la solution d'un système linéaire

Soit le système linéaire suivant:

$$\begin{bmatrix} a_{11} \cdots a_{1n} \\ \cdot \\ \cdot \\ \cdot \\ a_{n1} \cdots a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix} \qquad (3\text{-}2\text{-}30)$$

Nous avons présenté au chapitre 2 l'algorithme parallèle permettant la résolution du système (3-2-30) par la méthode de Gauss. Le tableau (3-4) donne la liste des opérations nécessaires pour n = 4



| N° | OPERATION | N° | OPERATION | N° | OPERATION |
|---|---|---|---|---|---|
| 1 | $d_{21} = -a_{21} \cdot a_{11}^{-1}$ | 10 | $a_{34}^{1} = a_{34} + d_{31} a_{14}$ | 19 | $a_{34}^{2} = a_{34}^{1} + d_{32} a_{24}^{1}$ |
| 2 | $d_{31} = -a_{31} \cdot a_{11}^{-1}$ | 11 | $b_3^{1} = b_3 + d_{31} b_1$ | 20 | $b_3^{2} = b_3^{1} + d_{32} b_2^{1}$ |
| 3 | $d_{41} = -a_{41} \cdot a_{11}^{-1}$ | 12 | $a_{42}^{1} = a_{42} + d_{41} a_{12}$ | 21 | $a_{43}^{2} = a_{43}^{1} + d_{42} a_{23}^{1}$ |
| 4 | $a_{22}^{1} = a_{22} + d_{21} a_{12}$ | 13 | $a_{43}^{1} = a_{43} + d_{41} a_{13}$ | 22 | $a_{44}^{2} = a_{44}^{1} + d_{41} a_{24}^{1}$ |
| 5 | $a_{23}^{1} = a_{23} + d_{21} a_{13}$ | 14 | $a_{44}^{1} = a_{44} + d_{41} a_{14}$ | 23 | $b_4^{2} = b_4^{1} + d_{42} b_2^{1}$ |
| 6 | $a_{24}^{1} = a_{24} + d_{21} a_{14}$ | 15 | $b_4^{1} = b_4 + d_{41} b_1$ | 24 | $d_{43} = -a_{43}^{2} (a_{33}^{2})^{-1}$ |
| 7 | $b_2^{1} = b_2 + d_{21} b_1$ | 16 | $d_{32} = -a_{32}^{1} (a_{22}^{1})^{-1}$ | 25 | $a_{44}^{3} = a_{44}^{2} + d_{43} a_{34}^{2}$ |
| 8 | $a_{32}^{1} = a_{32} + d_{31} a_{12}$ | 17 | $d_{42} = -a_{42}^{1} (a_{22}^{1})^{-1}$ | 26 | $b_4^{3} = b_4^{2} + d_{43} b_3^{2}$ |
| 9 | $a_{33}^{1} = a_{33} + d_{31} a_{13}$ | 18 | $a_{33}^{2} = a_{33}^{1} + d_{32} a_{23}^{1}$ | | |

Tableau (3-4) : Opérations de la méthode de Gauss

La figure (3-4) montre le graphe représentatif pour la partie de triangularisation de la matrice A.

Le chemin critique du graphe (3-4) pour n quelconque est composé de:

(n-1) divisions + (n-1) multiplications + (n-1) additions (3-2-31)

Le nombre d'opérations pour la triangularisation de la matrice A est de:

$(\underline{n^3-n})$ additions + $\underline{n^3-n}$ multiplications + $\underline{n(n-1)}$ divisions (3-2-32)
3 3 2

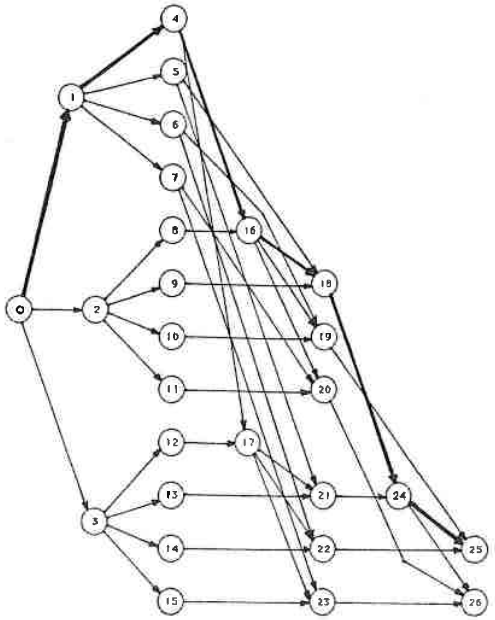



Figure (3-4) : graphe de la méthode de Gauss

Le taux d'accélération dans ce cas est :

$$t_{accel} = \frac{\frac{n(n+1)}{3} t_{mult} + \frac{n(n+1)}{+} t_{add} + \frac{n}{2} t_{div}}{t_{div} + t_{mult} + t_{add}} \qquad (3\text{-}2\text{-}33)$$

Le taux d'accélération pour les trois cas mentionnés au (3-2-1) est :

- Premier cas : ($t_{div} = t_{add} = t_{mult}$)



$$t_{accel} = \frac{4n^2 + 7n}{18} \qquad t_{accel}(asymp) = \frac{2n^2}{9} \qquad (3\text{-}2\text{-}34)$$

- Deuxième cas : ($t_{div} = 6\ t_{add}$ ; $t_{mult} = 5\ t_{add}$)

$$t_{accel} = \frac{2n^2 + 5n}{12} \qquad t_{accel}(asymp) = \frac{n^2}{6} \qquad (3\text{-}2\text{-}35)$$

- Troisième cas : $t_{div} = 34\ t_{add}$ ; $t_{mult} = t_{add}$

$$t_{accel} = \frac{2n^2 + 53n}{108} \qquad t_{accel}(asymp) = \frac{n^2}{54} \qquad (3\text{-}2\text{-}36)$$

METHODE DE GAUSS PAR BLOC

La méthode de Gauss s'étend aux matrices par blocs. Mais dans ce cas, pour le calcul des $d_{ij}$ du tableau (3-4), on calcule d'abord la matrice inverse $A_{jj}^{-1}$ et ensuite on effectue les multiplication $A_{ij} \cdot A_{jj}^{-1}$. Ainsi par exemple pour le tableau (3-4), les trois premières divisions des nombres réels sont remplacées par quatre opérations matricielles $A_{11}^{-1}$, $-A_{21} \cdot A_{11}^{-1}$, $-A_{31} \cdot A_{11}^{-1}$, $-A_{41} \cdot A_{11}^{-1}$

La figure (3-5) montre le graphe correspondant dans ce cas.

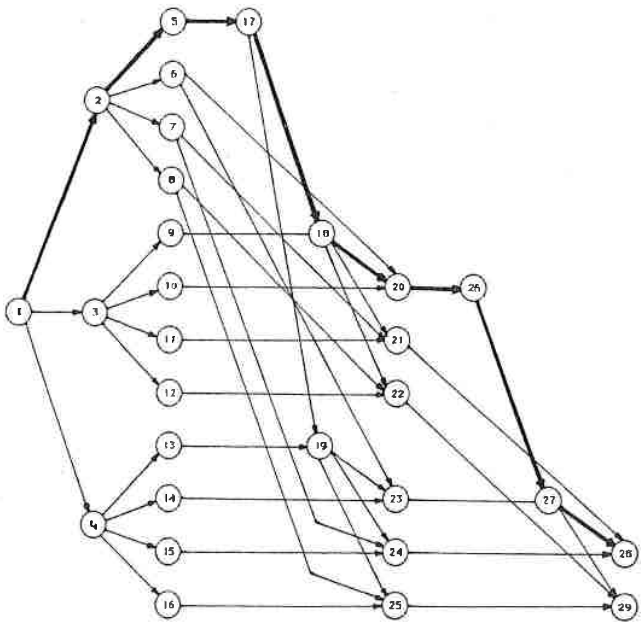


Figure (3-5) : graphe de l'algorithme Gauss par blocs

Le chemin critique du graphe (3-5) est composé alors de :

(N-1) inversions de matrice + 2(N-1) multiplications de deux matrices + (N-1) additions de deux matrices (3-2-37)

Si v est la dimension des blocs, les temps nécessaires pour effectuer ces opérations sont les suivants : (voir Annexe A-1):

multiplication de deux matrices $v^3\, t_{mult} + v^2\,(v-1)\, t_{add}$



addition de deux matrices $v^2\, t_{add}$

inversion d'une matrice $(v^3-1)\, t_{mult} + (v-1)^2\, v\, t_{add} + v t_{div}$

Le taux d'accélération est donc de :

$$t_{accel} = \frac{(N-1)\text{inversion} + \frac{2N^3+3N^2-5N}{6}\text{ mult matr.} + \frac{N^3-N}{3}\text{ Add matr.}}{(N-1)\text{ inversion} + 2(N-1)\text{ mult matr.} + (N-1)\text{ Add. matr.}} \qquad (3\text{-}2\text{-}38)$$

Pour N=4 et v=5 on a

- Premier cas : ($t_{div} = t_{add} = t_{mult}$)

$t_{accel} = 3$

- Deuxième cas : ( $t_{div} = 6\, t_{add}$ $t_{mult} = 5\, t_{add}$)

$t_{accel} = 3{,}3$

- Troisième cas : ( $t_{div} = 34\, t_{add}$ $t_{mult} = t_{add}$)

$t_{accel} = 3{,}1$

On peut constater que le taux d'accélération dans les trois cas reste pratiquement inchangé. En effet le nombre important de multiplications et d'additions pour l'inversion d'une matrice diminue considérablement l'influence du temps des divisions.

ORDONNANCEMENT DU CALCUL

Le taux d'accélération calculé dans (3-2-33) s'obtient sur au

maximum n(n-1) processeurs. Dans ce cas, le taux d'occupation des processeurs est de

$$\tau_{occup}=[n(n+1)/3(t_{add}+t_{mult})+(n/2)t_{div}] / [n(n-1)(t_{div}+t_{mult}+t_{add})] \quad (3\text{-}2\text{-}39)$$

Si $t_{div} = 6\ t_{add}$ ; $t_{mult} = 5\ t_{add}$ on a

$$\tau_{occup}=(4n2+7n)/(18(n(n-1))=(4n+7)/(18(n-1)) \quad (3\text{-}2\text{-}40)$$

et pour n d'ordre elève :

$$t_{occup} = \frac{2}{9} = 0,22$$

Pour augmenter ce taux d'occupation nous utilisons donc la méthode d'ordonnancement présentée au chapitre 2. Les tableaux (3-5) et (3-6) donnent les résultats du calcul pour l'élimination par point et l'élimination par blocs pour n=4, v=5, $t_{div} = 6\ t_{add}$, $t_{mult} = 5\ t_{add}$.



| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 8 | 36 $t_{add}$ | 4,3 | 0,54 |
| 7 | 36 $t_{add}$ | 4,3 | 0,61 |
| 6 | 36 $t_{add}$ | 4,3 | 0,72 |
| 5 | 42 $t_{add}$ | 3,7 | 0,74 |
| 4 | 48 $t_{add}$ | 3,25 | 0,81 |
| 3 | 54 $t_{add}$ | 2,9 | 0,96 |
| 2 | 78 $t_{add}$ | 2 | 1 |

Tableau (3-5) : taux d'acélération pour l'algorithme de Gauss par points

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 8 | 6615 $t_{add}$ | 3,3 | 0,41 |
| 7 | 6615 $t_{add}$ | 3,3 | 0,47 |
| 6 | 6615 $t_{add}$ | 3,3 | 0,55 |
| 5 | 6615 $t_{add}$ | 3,3 | 0,66 |
| 4 | 6680 $t_{add}$ | 3,2 | 0,8 |
| 3 | 8160 $t_{add}$ | 2,6 | 0,87 |
| 2 | 11155 $t_{add}$ | 1,93 | 0,97 |

Tableau (3-6) : taux d'accélération pour l'algorithme de Gauss par blocs

On constate que l'ordonnancement du calcul réduit le nombre de processeurs nécessaires pour le calcul de l'algorithme dans le temps du chemin critique. Pour l'algorithme par point, ce temps s'obtient sur 6 processeurs au lieu de 12 et pour l'algorithme par bloc, sur 5 au lieu de 12. Pour un nombre limité de processeurs, l'ordonnancement du calcul aboutit à un taux d'occupation excellent. La figure (3-6) montre l'organisation du calcul sur 2 processeurs dans le cas de l'algorithme par blocs.

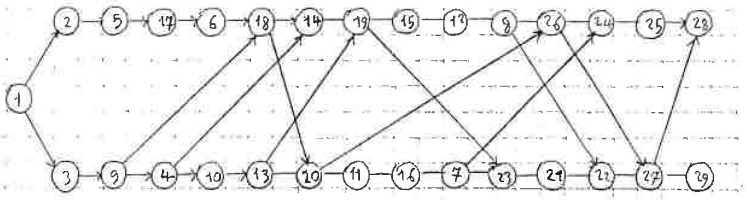


Figure (3-6) : ordonnancement de l'algorithme de Gauss par bloc sur 2 processeurs

STRATEGIE DU PIVOT MAXIMUM

L'algorithme de Gauss tel qu'il a été présenté n'est pas numériquement stable, surtout quand les pivots sont très petits.

Pour rendre cet algorithme stable on choisit donc la stratégie du pivot maximum. On distingue deux stratégies dans la littérature :



- Pivot maximum partiel :

à l'étape K de triangularisation, choisir $a_{ik}$ comme pivot tel que

$$|a_{ik}| = \max_{K< l \leq n} |a_{lm}|$$

- Pivot maximum total :

à l'étape K de triangularisation, choisir $a_{ij}$ comme pivot tel que

$$|a_{ij}| = \max_{\substack{K< l \leq n \\ K< m \leq n}} |a_{lm}|$$

Dans le premier cas, il faut alors permuter les lignes i et k tandis que dans le deuxième cas il faut permuter les lignes i et k et les colonnes j et k en même temps. La stratégie du pivot maximum rend malaisé le calcul parallèle de l'algorithme de Gauss. En effet à chaque étape de triangularisation, il faut comparer k nombres, ce qui peut être effectué avec la méthode de dichotomie en $\log_2 k$ étapes. Le seul choix du pivot maximum nécessite donc :

$$\log_2 n-1 + \log_2 n-2 + \ldots\ldots\ldots + \log_2 1 = \log (n-1)! = n \log_2 n \text{ étapes}$$

Ce temps de calcul du choix du pivot s'ajoute au temps du chemin critique, calculé à (3-2-31). La durée du chemin critique dans ce cas est donc de l'ordre $n \log_2 n$, c'est à dire que le temps de choix du pivot est

prepondérant par rapport au temps de calcul des opérations arithmétiques. Lorsque le nombre de processeurs est limité (inférieur ou égal à n), le temps des opérations mathématiques est de l'ordre $n^2$ ou plus. Le problème de temps du choix de pivot ne se pose pas dans ce cas. Mais le choix dynamique du pivot nous préoccupe davantage en Automatique. En effet, les systemes industriels dans leur majorité ont une structure bien déterminée, ce qui se traduit par des matrices de structure invariantes.Les différentes manipulations sur ces matrices peuvent donc être ordonnancées a priori et d'une façon statique, ce qui n'est pas possible avec la stratégie du pivot maximum. Une autre inconvénient de cette stratégie est qu'elle n'est pas applicable pour les matrices en blocs, souvant utilisées en Automatique. En effet si les dimensions des blocs sont différentes, on peut avoir des blocs rectangulaires qui ne peuvent pas être choisi comme pivot. La méthode de Gauss est donc limitée pour les cas ou l'on peut éviter le choix du pivot maximum. Un de ces cas se présente quand la matrice A est symétrique définie positive.

### 3-2-3 Résolution d'un système symétrique défini positif

Pour la triangularisation d'une matrice symétrique par la méthode de Gauss, on calcule à chaque étape, le triangle supérieur de la matrice. Ainsi par exemple, le tableau (3-4) se simplifie comme le montre le tableau (3-7).



| N° | OPERATION | N° | OPERATION | N° | OPERATION |
|---|---|---|---|---|---|
| 1 | $d_{21} = -a_{21} \cdot a_{11}^{-1}$ | 9 | $a_{34}^{1} = a_{34} + d_{31} a_{14}$ | 17 | $b_3^{2} = b_3^{1} + d_{32} b_2^{1}$ |
| 2 | $d_{31} = -a_{31} \cdot a_{11}^{-1}$ | 10 | $b_3^{1} = b_3 + d_{31} b_1$ | 18 | $a_{44}^{2} = a_{44}^{1} + d_{41} a_{24}^{1}$ |
| 3 | $d_{41} = -a_{41} \cdot a_{11}^{-1}$ | 11 | $a_{44}^{1} = a_{44} + d_{41} a_{14}$ | 19 | $b_4^{2} = b_4^{1} + d_{42} b_2^{1}$ |
| 4 | $a_{22}^{1} = a_{22} + d_{21} a_{12}$ | 12 | $b_4^{1} = b_4 + d_{41} b_1$ | 20 | $d_{43} = -a_{43}^{2} (a_{33}^{2})^{-1}$ |
| 5 | $a_{23}^{1} = a_{23} + d_{21} a_{13}$ | 13 | $d_{32} = -a_{32}^{1} (a_{22}^{1})^{-1}$ | 21 | $a_{44}^{3} = a_{44}^{2} + d_{43} a_{34}^{2}$ |
| 6 | $a_{24}^{1} = a_{24} + d_{21} a_{14}$ | 14 | $d_{42} = -a_{42} (a_{22}^{1})^{-1}$ | 22 | $b_4^{3} = b_4^{2} + d_{43} b_3^{2}$ |
| 7 | $b_2^{1} = b_2 + d_{21} b_1$ | 15 | $a_{33}^{2} = a_{33}^{1} + d_{32} a_{23}^{1}$ | | |
| 8 | $a_{33}^{1} = a_{33} + d_{31} a_{13}$ | 16 | $a_{34}^{2} = a_{34}^{1} + d_{32} a_{24}^{1}$ | | |

Tableau (3-7) : algorithme de Gauss pour un système symétrique

La figure (3-7) montre le graphe représentatif de cet algorithme.

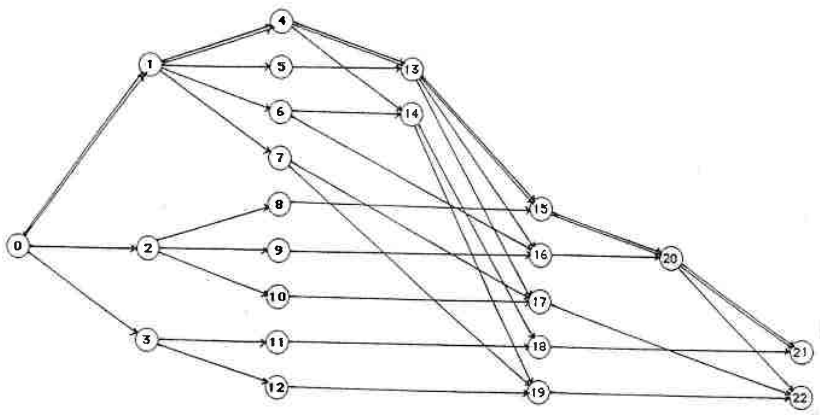



Figure (3-7) : graphe de l'algorithme de Gauss pour un système symétrique

La longueur du chemin critique du graphe (3-7) est le même que le graphe (3-4) mais le nombre total des opérations dans ce cas est de :

$$\frac{n(n-1)(n+4)}{6} \text{ multiplications} + \frac{n(n-1)(n+4)}{6} \text{ additions} + \frac{n(n-1)}{2} \text{ divisions} \qquad (3\text{-}2\text{-}41)$$

Le taux d'accélération de l'algorithme de Gauss dans ce cas est donc :

$$t_{accel} = \frac{\frac{n(n-1)(n+4)}{6} t_{mult} + \frac{n(n-1)(n+4)}{6} t_{add} + \frac{n(n-1)}{2} t_{div}}{(n-1)(t_{div} + t_{mult} + t_{add})} \qquad (3\text{-}2\text{-}41)$$



Le taux d'accélération asymptotique est donc la moitié du taux d'accélération dans le cas des systèmes non symétriques, mais il est réalisable sur un nombre plus petit de processeurs comme le montrent les tableaux (3-8) et (3-9).

| NOMBRE DE PROCESSEURS | DUREE D'EXECUTION | TAUX D'ACCELERATION | TAUX D'OCCUPATION |
|---|---|---|---|
| 6 | 36 | 3,7 | 0,61 |
| 5 | 36 | 3,7 | 0,73 |
| 4 | 42 | 3,1 | 0,79 |
| 3 | 48 | 2,75 | 0,92 |
| 2 | 66 | 2 | 1 |

Tableau (3-8) : temps d'exécution pour la résolution d'un système symétrique.

| Nombre de processeurs | Temps d'execution | Taux d'acceleration | Taux d'occupation |
|---|---|---|---|
| 5 | 6615 | 2,9 | 0,58 |
| 4 | 6615 | 2,9 | 0,71 |
| 3 | 7630 | 2,2 | 0,73 |
| 2 | 9640 | 1,8 | 0,9 |

Tableau (3-9) : temps d'exécution pour un système symétrique par blocs.

Le temps du chemin critique est obtenu dans le premier cas sur 5 processeurs et dans le deuxième cas sur 4 processeurs.

LA METHODE DE CHOLESKY

Dans la méthode de Cholesky, on décompose la matrice symétrique définie positive A en un produit de deux matrices B et $B^T$ où B est triangulaire inférieure.

Les éléments de B sont calculés comme suit :

$$b_{11} = \sqrt{a_{11}},\ b_{21} = a_{12}/b_{11} \qquad b_{n1} = a_{1n}/b_{11},$$

$$b_{ll} = \sqrt{a_{ll} - \sum_{k=1}^{l-1} b_{lk}\, b_{l+1,k}},\ b_{l+1,l} = \frac{a_{l,l+1} - \sum_{k=1}^{l-1} b_{l+1,k}}{b_{ll}},\ b_{nl} = \frac{a_{ln} - \sum_{k=1}^{l-1} b_{lk} b_{nk}}{b_{ll}}$$

La figure (3-8) montre le graphe correspondant de la méthode de Cholesky.

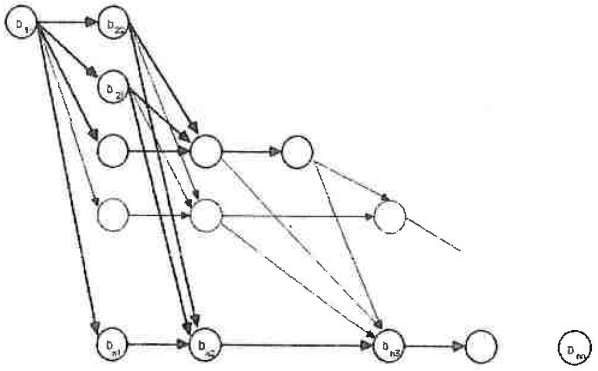


Figure (3-8) : graphe de la méthode de Cholesky



Le chemin critique de ce graphe est composé de :

$b_{11}, b_{22}, b_{32}, b_{43}, \ldots\ldots\ldots, b_{n,\,n-1}, b_{nn}$

Le calcul de $b_{ii}$ nécessite (i-1) multiplications, (i-1) additions et une extraction de racine carrée.

Le calcul de $b_{i+1,i}$ nécessite (i-1) multiplications (i-1) additions et une division. Ainsi, le nombre d'opérations élémentaires du chemin critique est :

n racines carrées + (1 + 2 + ... + n-1) (mult + add) + (1 + 2 + ...+ n-2) (multi + add) + (n-2) div et donc la durée du chemin critique est :

$$(n)\, t_{racine} + (n-2)\, t_{div} + (n-1)^2\, (t_{mult} + t_{add}) \qquad (3\text{-}2\text{-}43)$$

Le nombre d'opérations pour arriver à un système d'équations triangulaire est :

n(n-1)/2 div +(n3-n)/6 add +(n3-n)/6 mult +n racine carées

Le taux d'accélération est donc:

$$t_{accel} = \frac{\frac{n(n-1)}{2} t_{div} + \frac{n^3 - n}{6} t_{add} + \frac{n^3 - n}{-} t_{mult} + n\, t_{racine}}{n\, t_{racine} + (n-2)\, t_{div} + (n-1)^2\, (t_{mult} + t_{add})} \qquad (3\text{-}2\text{-}43)$$

Par exemple, pour $t_{racine} = 40\ t_{add}$, $t_{div} = 34\ t_{add}$, $t_{mult} = t_{add}$ (3-12) on a :

$$t_{accel} = \frac{\frac{n^3}{3} + 17n^2 + 23n}{2n^2 + 70n + 70} \qquad t_{asymp} = \frac{n}{12} \qquad (3\text{-}2\text{-}44)$$

Nous constatons que le taux d'accélération pour la méthode de Cholesky, est beaucoup moins important que pour la méthode de Gauss.

Pour n = 50 on a par exemple :

$t_{accel}$ (Gauss) = 48,6 $\qquad$ $t_{accel}$ (Cholesky) = 10

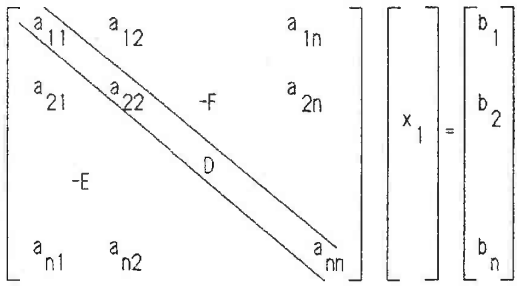
La durée du chemin critique pour les deux cas est respectivement de 1764 $t_{add}$ et 8570. Un autre inconvénient de la méthode de Cholesky, est qu'elle n'est pas applicable à la résolution par blocs. La méthode de Gauss est donc préférable, pour l'identification et le calcul de la méthode des systèmes interconnectés.

## 3-3 METHODE ITERATIVE DE RESOLUTION DES SYSTEMES D'EQUATIONS LINEAIRES :

### 3-3-1 Méthode de Jacobi par points :

Soit à résoudre le système suivant :






La méthode de Jacobi par points consiste à décomposer la matrice A, en trois matrices D, E, F comme nous l'avons indiqué.

Dans ce cas, on a :

$A = D - E - F$ et

$A\underline{x} = (D - E - F)\underline{x} = \underline{b}$

où

$$D\underline{x} = (E+F)\,\underline{x} + \underline{b} \qquad (3\text{-}3\text{-}1)$$

On détermine alors itérativement les valeurs de x de la façon suivante :

$$D\underline{x}^{(k+1)} = (E+F)\,\underline{x}^{(k)} + \underline{b}$$

$$\underline{x}^{(k+1)} = D^{-1}(E+F)\,\underline{x}^{(k)} + D^{-1}\,.\,\underline{b} \qquad (3\text{-}3\text{-}2)$$

La matrice $J = D^{-1}(E+F)$ est appelée *matrice de Jacobi*

La condition de convergence de l'algorithme de Jacobi est alors :

$\rho(J) | < 1$

où $\rho(J)$ est le rayon spectral de la matrice J.

Pour l'implantation parallèle de la méthode de Jacobi, nous supposons qu'elle converge pour les problèmes concernés et nous arrêtons le calcul quand :

$$(x_1^{(k+1)} - x_1^{(k)})^2 + (x_2^{(k+1)} - x_2^{(k)})^2 + \text{---} + (x_n^{(k-1)} - x_n^{(k)})^2 < \varepsilon \qquad (3\text{-}3\text{-}3)$$

où $\varepsilon$ est une valeur positive fixée a priori.

Les calculs effectifs de la méthode de Jacobi se présentent de la façon suivante :

$$x_1^{(k+1)} = a_{11}^{-1} [- a_{12} x_2^{(k)} - a_{13} x_3^{(k)} \text{---} a_{1n} x_n^{(k)} + b_1]$$

$$x_2^{(k+1)} = a_{22}^{-1} [- a_{21} x_1^{(k)} - a_{23} x_3^{(k)} + \text{---} a_{2n} x_n^{(k)} + b_2]$$

$$x_n^{(k+1)} = a_{nn}^{-1} [- a_{n1} x_1^{(k)} - a_{n2} x_2^{(k)} \text{---} a_{nn-1} x_{n-1}^{(k)} + b_n]$$

Si $((x_1^{(k+1)} - x_1^{(k)})^2 + \text{---} + (x_n^{(k+1)} - x_n^{(k)})^2 < \varepsilon)$ Fin

sinon itération suivante.

En se référant à ces calculs, nous présentons donc l'algorithme parallèle de Jacobi comme suit :



Début de l'algorithme

Calculer en parllèle $x_{ii}^{-1}$ , $a_{ij} x_j$ $i = 1$ à $n$; $j = 1$ à $n$ $j \neq i$ sur $n^2$ processeurs;

Calculer $b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j$ sur au maximum $n^2/2$ processeur en $\log_2 n$ étapes

avec la méthode de dichotomie.

Calculer $x_i^{(k+1)}$ sur n processeurs ;

Calculer $(x_i^{(k+1)} - x_i^{(k)})^2$ sur n processeurs ;

Calculer $\sum_{i=1}^{n} (x_i^{(k+1)} - x_i^{(k)})^2$ sur au maximum $n/2$ processeurs en $\log_2^n$ étapes;

Effectuer le test d'arrêt sur un processeur ;

Si le test est vrai fin de calcul ;

Sinon, itération suivante :

Fin de l'algorithme.

Le tableau 3-10 liste une partie des opérations effectuées pour cet algorithme, pour un système d'équations de l'ordre 4. Les autres opérations sont de même type que l'exemple proposé.

| N° | OPERATION | N° | OPERATION | N° | OPERATION |
|---|---|---|---|---|---|
| 1 | $-a_{12}\, x_2^{(k)}$ | 17 | $b_1^1 = b_1 - a_{12}\, x_2^{(k)}$ | 33 | $T_1 = [x_1^{(k+1)} - x_1^{(k)}]^2$ |
| 2 | $-a_{13}\, x_3^{(k)}$ | 18 | $b_1^2 = -a_{13}\, x_3^{(k)} - a_{14}\, x_4^{(k)}$ | 37 | $T_1 + T_2$ |
| 3 | $-a_{14}\, x_4^{(k)}$ | 25 | $b_1^3 = b_1^1 + b_1^2$ | 39 | $(T_1 + T_2) + (T_3 + T_4)$ |
| 4 | $a_{11}^{-1}$ | 29 | $x_1^{(k+1)} = a_{11}^{-1}\, b_1^3$ | 40 | Test |

Tableau (3-10) : liste des opérations de l'algorithme Jacobi

La figure (3-9) montre le graphe correspondant à cet algorithme, pour une itération



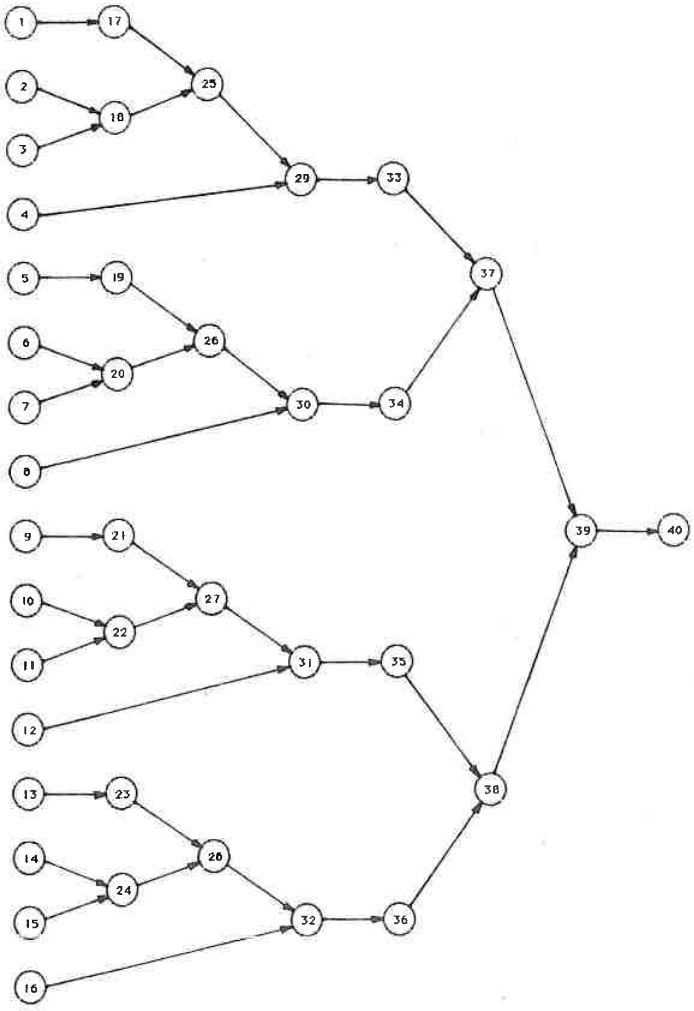



Figure (3-9) Graphe de l'algorithme de Jacobi

Il est à noter que les valeurs $a_{ii}^{-1}$ sont calculées seulement à la première itération ; pour les autres itérations, les arcs 4-29, 8-30, 12-31 et 16-32 n'existent pas.

La durée du chemin critique du graphe (3-9) dépend du temps d'exécution des multiplications, divisions et additions. Nous l'étudions donc pour les trois cas cités auparavant :

Premier cas : $(t_{div} = t_{mult} = t_{add})$

Le chemin critique 1-17-25-29-34-38-40-41, de longueur 10 $t_{add}$ et pour n quelconque, la longueur du chemin critique est :

1 multiplication: $[-a_{ji}x_i^k]$

+ $\log_2 n$ additions: $[b_j - \sum_{\substack{i=1 \\ i\neq j}}^{n} a_{ji}\, x_i^k]$

+1 multiplication: $[a_{jj}^{-1}\, b_j^{n-1}]$

+ 1 addition: $[x_i^{k+1} - x_i^k]$

+ 1 multiplication: $[(x_i^{k+1} - x_i^k)^2]$

+ $\log_2 n$ additions $\sum_{i=1}^{n} (x_i^{k+1} - x_i^k)^2$

+ 1 test



ce qui pour temps de test = $t_{add}$ donne :

$$1 + \log_2 n + 1 + 1 + 1 + \log_2 n + 1 = 5 + 2\log_2 n \qquad (3\text{-}3\text{-}4)$$

Deuxième cas : $(t_{div} = 6\, t_{add},\ t_{mult} = 5\, t_{add},\ t_{test} = t_{add})$

Un des chemins critiques est encore 1-17-25-29-34-38-40-41 et sa durée est :

$$t_{add}(5 + \log_2 n + 5 + 1 + 5 + \log_2 n + 1) = (17 + 2\log_2 n)\, t_{add} \qquad (3\text{-}3\text{-}5)$$

Troisième cas : $(t_{div} = 34\, t_{add},\ t_{mult} = t_{add},\ t_{test} = t_{add})$

Pour la première itération, le chemin critique est 4-29-34-38-40-41 et sa durée est :

$$t_{add}(34 + 1 + 1 + 1 + \log_2 n + 1) = (38 + \log_2 n)\, t_{add} \qquad (3\text{-}3\text{-}6)$$

mais pour les autres itérations, le temps du chemin critique est égal à celui calculé en (3-3-4)

On peut remarquer, surtout dans le premier cas, qu'une partie importante du chemin critique est consacrée aux tests de fin de calcul. Trois solutions pourraient être envisagées pour ce problème :

- La première est d'effectuer le test sur chaque variable. On élimine ainsi la partie $\Sigma(x_i^{k+1} - x_i^k)^2$ et on effectue les tests en parallèle.

On élimine ainsi $\log_2 n$ opérations du chemin critique, mais, en contre-partie, l'exécution des tests individuels sur les variables peut ralentir la convergence.

Si $L_1$ et $L_2$ sont respectivement le nombre d'itérations dans le premier et dans le deuxième cas, la durée minimale d'exécution de l'algorithme est respectivement :

$(5 + 2 \log_2 n) L_1$ (premier cas)

$(5 + \log_2 n) L_2$ (deuxième cas)

Le test individuel sur les variables est donc efficace si :

$(5 + \log_2 n) L_2 < (5 + 2 \log_2 n) L_1$

$$\text{ou } L_2 < L_1 \frac{5 + 2 \log_2^n}{5 + \log_2^n} \qquad (3\text{-}3\text{-}7)$$

on a par exemple pour n = 64

$L_2 < 1{,}55\, L_1$

<u>La deuxième</u> solution est d'effectuer le test, une fois toutes les k itérations (k>1). Dans ce cas on risque de faire au maximum (k-1) itérations



de plus. En effet, si $L_1$ est le nombre d'itérations nécessaires pour la convergence de l'algorithme, le test de fin de calcul peut être effectué au plus tard k-1 itérations après la convergence. Désignons par $L_2$ le nombre d'itérations dans le deuxième cas, on a :

$$L_1 <= L_2 <= L_1 + k\text{-}1$$

La longueur du chemin critique nécessite alors :

- dans le premier cas : un test par itération

soit $L_1 (5 + 2 \log_2^n)$

- dans le deuxième cas : un test par k itérations,

soit

$$(L_1 + k\text{-}1)(5 + \log_2^n + \frac{1}{k} \log_2^n) \qquad (3\text{-}3\text{-}8)$$

La valeur de k qui rend minimum l'expression (3-3-8) est égale à :

$$k = \sqrt{\frac{(L_1 - 1) \log_2^n}{5 + \log_2^n}} \qquad (3\text{-}3\text{-}9)$$

A titre d'exemple, pour n=64 et $L_1$= 30 on trouve k=4. La durée du chemin critique dans les deux cas est alors :

- dans le premier cas :

$30 (5 + 12) = 510$

- dans le deuxième cas :

$32\ (5 + 6 + 1{,}5) = 432$

La durée du chemin critique dans le deuxième cas a diminué de 16%.

Cependant en pratique, on ne connaît pas la valeur de $L_1$. Le calcul de la valeur optimale de k n'est donc pas possible. Dans le cas ou l'on ne peut fournir aucune information, même approximative sur la valeur de $L_1$, on peut toujours se contenter de petites valeurs de k : k = 2 ou 3.

La troisième solution est le chevauchement des itérations. On commence l'itération suivante dès que les nouvelles valeurs $x_i$ (i= 1 à n) sont calculées.

La figure (3-10) montre le graphe de l'algorithme dans ce cas. Sur cette figure les opérations pour le test sont présentées en ligne double.

On peut remarquer que le temps nécessaire pour effectuer le test est pratiquement absorbé par le calcul des opérations de l'itération suivante.

Si l'algorithme converge après L itérations, sa durée d'exécution est

$$2 + \log_2 n + L\ (3 + \log_2 n) \qquad (3\text{-}3\text{-}10)$$

pour n = 64 et L = 30, la durée du chemin critique est :



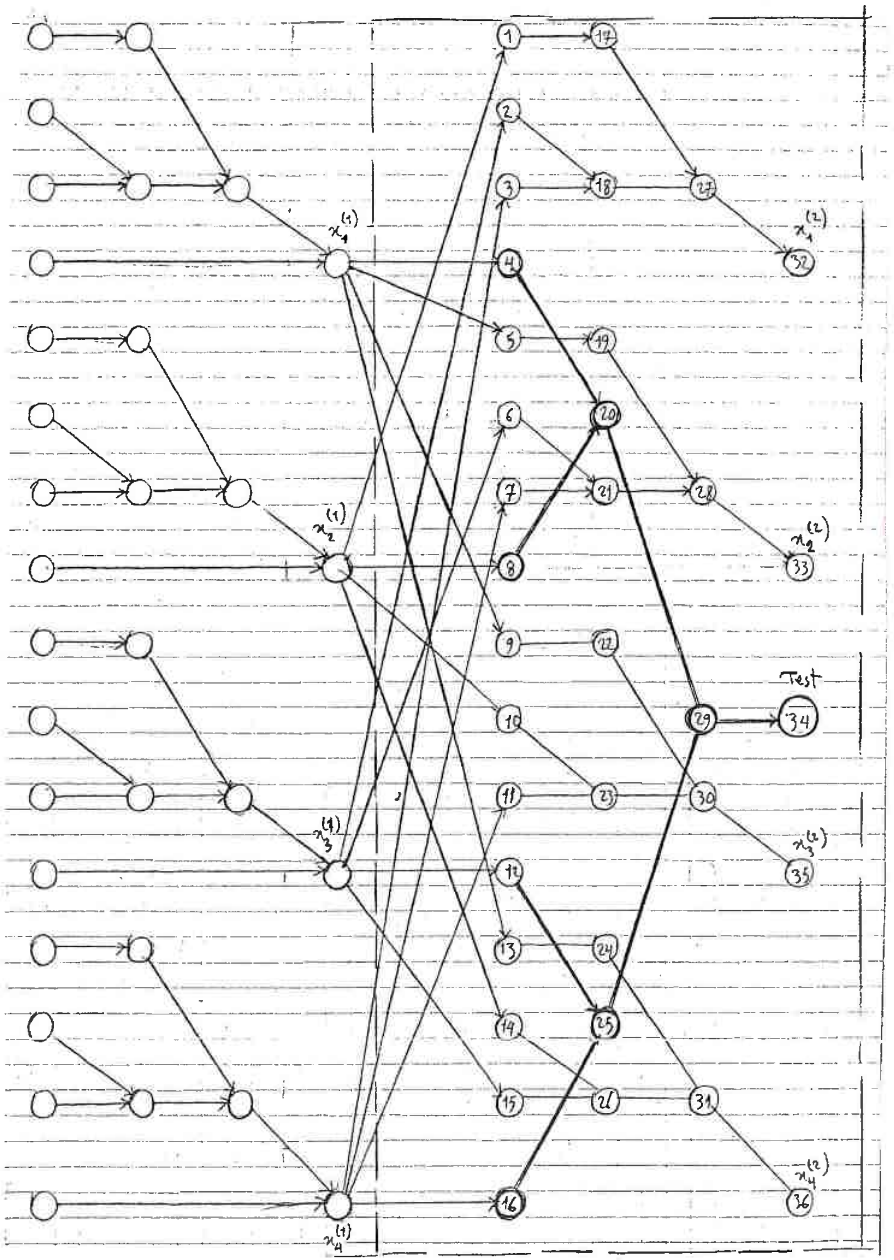



Figure (3-10) Graphe de l'algorithme de Jacobi par chevauchement des it

$2 + 5 + 30\,(3 + 5) = 247$

Cette durée est de 48% de la durée calculée avec (3-3-5), et de 57% de celle de (3-3-9). Pour le cas ou $t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$, la durée du chemin critique pour la troisième solution est de :

$$6 + \log_2 n + L\,(10 + \log_2 n) \qquad (3\text{-}3\text{-}11)$$

Ce qui donne pour n = 64 et L = 30, une durée de 492. La même durée calculée à partir de l'équation (3-3-6) est de 870. On peut donc constater que dans tous les cas, la troisième solution aboutit à une durée du chemin critique considérablement inférieure à celles des équations (3-3-5) et (3-3-6).

ALGORITHME DE JACOBI PAR BLOCS

La décomposition par blocs de la matrice A est identique à la décomposition par points comme la montre la figure (3-11).



| | | | |
|---|---|---|---|
| A11 | 2 x 4 | 2 x 3 | 2 x 3 |
| 4 x 2 | A 22 | 4 x 3 | 4 x 3 |
| 3 x 2 | 3 x 4 | A33 | 3 x 3 |
| 3 x 2 | 3 x 4 | 3 x 3 | A 44 |

Figure (3-11) : décomposition par bloc pour l'algorithme de Jacobi.

Les opérations sont les mêmes que celles présentées dans le tableau (3-10), sauf que les multiplications sont remplacées par des produits matrice-vecteur, les additions par des additions de deux vecteurs et les divisions par des inversions de matrices. Les dimensions de blocs peuvent être différentes.

Le choix des blocs et la méthode de décomposition des matrices ont été abondamment étudiée dans la littérature. Nous renvoyons le lecteur intéressé à [3-6], [3-7], [3-8], [3-9].

D'une façon heuristique, le choix de blocs plus grands améliore la vitesse de convergence de l'algorithme, mais sous certaines conditions

[3-10], [3-11].

Le graphe de l'algorithme par blocs est de la même forme que sur la figure (3-9). Ici encore, les inversions de matrice sont effectuées uniquement à la première itération. Le chemin critique de la première itération dépend des dimension des blocs, mais le chemin critique des itérations suivantes est composé de :

- 2 multiplications d'une matrice par un vecteur de dimension v,
- $\log_2 n$ additions de deux vecteurs de dimension v,
- 1 addition de deux vecteurs de dimension v,
- un produit scalaire de deux vecteurs de dimension v
- $\log_2 n$ additions

où N est le nombre de blocs et v leur dimension.

Dans le cas où les blocs ne sont pas de même dimension, le chemin critique est celui composé des opérations pour le calcul du bloc de plus grande dimension.

Les méthodes présentées dans le cas de l'algorithme par points pour diminuer l'influence de la durée du test, sont valables pour l'algorithme par blocs.

Néanmoins, le temps de test représente un pourcentage de calcul beaucoup moins important que dans le cas de l'algorithme par points. Pour N = 4 et v = 5, par exemple on a :

temps de test = 10% du temps du chemin critique.



ORDONNANCEMENT DU CALCUL

Pour l'ordonnancement d'un algorithme avec la méthode C.P.M. il faut connaître toutes les caractéristiques temporelles des opérations telles que la durée, le début au plus tôt etc...

Dans le cas des algorithmes itératifs, on ne peut pas déterminer toutes ces caractéristiques parce que la durée totale de l'algorithme est indéterminée. L'algorithme de Jacobi en tant qu'algorithme itératif, pose le même problème, mais l'étude de la figure 3-8 nous révèle que les tâches de l'itération suivante ne peuvent commencer qu'après l'exécution du test 41. Les parties de graphes entre deux tests successifs, sont donc les mêmes. On peut ainsi ordonnancer une seule partie et étendre le résultat aux autres. La seule partie différente des autres est la première, qui contient les divisions ou inversions des éléments diagonaux, et doit donc être ordonnancée différemment.

Le tableau (3-11) donne les résultats obtenus pour l'ordonnancement du graphe (3-9) pour l'algorithme de Jacobi par points relatifs à un système d'ordre 4 dans le cas où $t_{div} = 6t_{add}$ et $t_{mult} = 5 t_{add}$.

Le tableau (3-12) donne les mêmes résultats pour les itérations suivantes, où les tâches 4,8,12,16 n'existent plus. Les tableaux (3-13) et (3-14) présentent les résultats obtenus pour l'algorithme de Jacobi par blocs pour la matrice de la figure (3-11).

Tableau (3-11) : temps d'exécution de l'algorithme Jacobi par point pour la première itération.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 12 | 21 | 5,7 | 0,48 |
| 8 | 26 | 4,6 | 0,58 |
| 6 | 27 | 4,4 | 0,74 |
| 4 | 32 | 3,75 | 0,94 |
| 3 | 44 | 2,73 | 0,91 |
| 2 | 61 | 1,97 | 0,98 |

Tableau (3-12) : temps d'exécution de l'algorithme Jacobi par points pour les itérations suivantes.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 5 | 497 | 3,38 | 0,68 |
| 4 | 497 | 3,38 | 0,85 |
| 3 | 565 | 2,97 | 0,99 |
| 2 | 842 | 1,995 | 1 |

Tableau (3-13) : temps d'exécution de l'algorithme de Jacobi par blocs pour la première itération.



| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 7 | 198 | 4,73 | 0,67 |
| 6 | 198 | 4,73 | 0,79 |
| 5 | 215 | 4,36 | 0,87 |
| 4 | 247 | 3,8 | 0,95 |
| 3 | 323 | 2,90 | 0,97 |
| 2 | 472 | 1,98 | 0,99 |

Tableau (3-14) : temps d'exécution de l'algorithme de Jacobi par blocs pour les itérations suivantes.

Pour l'ordonnancement du graphe (3-10), on constate que la partie encadrée du graphe se répète à chaque itération. Il suffit donc d'ordonnancer la partie non encadrée du graphe (3-10) pour la première itératioin et la partie encadrée pour les itérations suivantes.

Les tableaux (3-15) et (3-16) donnent les résultats obtenus pour l'ordonnancement de la partie encadrée du graphe (3-10) pour l'algorithme de jacobi par points et par blocs.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 16 | 12 | 10 | 0,62 |
| 12 | 14 | 8,57 | 0,73 |
| 8 | 18 | 6,7 | 0,84 |
| 6 | 21 | 5,7 | 0,95 |
| 4 | 32 | 3,75 | 0,94 |
| 3 | 42 | 2,96 | 0,95 |
| 2 | 61 | 1,97 | 0,99 |

Tableau (3-15) : temps d'exécution de l'algorithme de Jacobi par points pour le graphe (3-10)

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 6 | 168 | 5,57 | 0,93 |
| 5 | 189 | 4,95 | 0,99 |
| 4 | 243 | 3,85 | 0,96 |
| 3 | 319 | 2,93 | 0,98 |
| 2 | 470 | 1,99 | 0,996 |

Tableau (3-16) : temps d'exécution de l'algorithme de Jacobi par blocs pour le graphe (3-10).



Pour un nombre de processeurs suffisamment grand, les temps d'exécution de l'algorithme présentés au tableau (3-15) sont nettement inférieurs à ceux du tableau (3-12) ; mais pour un petit nombre de processeurs, les deux méthodes donnent les mêmes résultats. Dans le cas de l'algorithme par blocs, l'amélioration du temps d'exécution n'est pas aussi nette, mais on trouve un taux d'occupation excellent partout dans le tableau, ce qui justifie la méthode d'ordonnancement utilisée. La figure (3-12), montre l'organisation de calcul de la partie encadrée du graphe (3-10), sur 6 processeurs dans le cas de la matrice représentée à la figure (3-11).

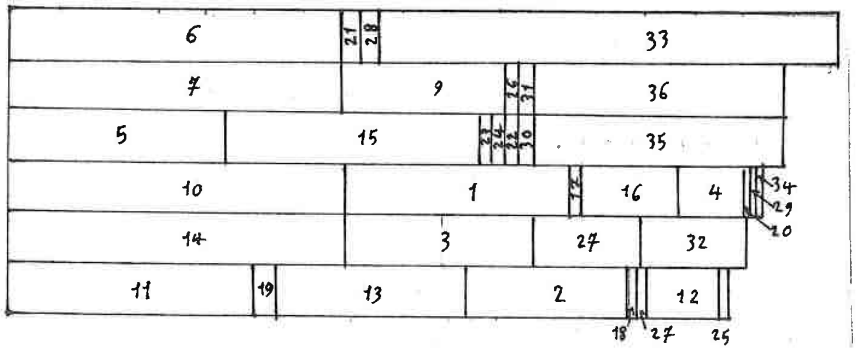



Figure (3-12) : organisation du calcul

### 3-3-2 Méthode de GAUSS SEIDEL

Dans la méthode de Gauss Seidel, la décomposition de la matrice A est la même que pour la méthode de Jacobi, c'est-à-dire :

$$A = D-E-F \qquad (3-3-12)$$

mais on détermine itérativement les valeurs de $\underline{x}$ de la façon suivante :

$$(D-E)x^{(K+1)} = Fx^{(K)} \qquad (3-3-13)$$

le test d'arrêt étant celui énoncé en (3-3-4).

Les calculs de la méthode de Gauss Seidel se représentent donc comme suit :

Début

$x_1^{(K+1)} = a_{11}^{-1} [a_{12}x_2^{(K)} - a_{13}x_3^{(K)} \ldots\ldots - a_{1n}x_n^{(K)} + b_1]$

$x_2^{(K+1)} = a_{22}^{-1} [-a_{21}x_1^{(K+1)} - a_{23}x_3^{(K)} \ldots\ldots a_{2n}x_n^{(K)} + b_2]$

.....

$x_n^{(K+1)} = a_{nn}^{-1} [-a_{n1}x_1^{(K)} - a_{n2}x_2^{(K)} \ldots\ldots - a_{nn-1}x_{n-1}^{(K)} + b_n]$

Si (Test est vrai) FIN

Sinon, itération suivante.

Fin



L'algorithme parallèle correspondant à une itération de la méthode de Gauss Seidel se déroule alors comme ci-dessous :

Début de l'algorithme

Calculer $a_{ii}^{-1}$

Faire la somme $\sum_{j=l+1}^{n} a_{lj}$ i=1 à n; l=1 à n-1 l<j<=n avec la méthode de dichotomie en parallèle.

Calculer $x_1^{(k+1)}$

Calculer $a_{l1}\, x_1^{(k+1)}$ l=2 à n ; $(x_1^{(k+1)} - x_1^{(k)})^2$ ; en parallèle

Calculer $x_2^{(k+1)}$

Calculer $a_{l2}\, x_2^{(k+1)}$ l=3 à n; $(x_2^{(k+1)} - x_2^{(k)})^2$ ; en parallèle

.....

Calculer $x_n^{(k+1)}$

Calculer $(x_n^{(k+1)} - x_n^{(k)})^2$

Si (test est vrai) FIN

Sinon itération suivante

Fin de l'algorithme

Le tableau (3-17) présente une partie des opérations nécessaires pour le calcul de l'algorithme :

| No | Opération | No | Opération | | Operation |
|---|---|---|---|---|---|
| 1 | $-a_{12}x_2^{(k)}$ | 15 | $b_1 - a_{11}x_i^{(k)}$ | 34 | $x_3^{(k+1)}$ |
| 4 | $a_{11}^{-1}$ | 16 | $x_1^{(k+1)}$ | 40 | $x_4^{(k+1)}$ |
| 5 | $-a_{23}x_3^{(k)}$ | 18 | $x_1^{(k+1)} - x_1^{(k)}$ | 43 | $(x_i^{(k+1)} - x_i^{(k)})^2$ |
| 8 | $-a_{34}x_4^{(k)}$ | 22 | $(x_1^{(k+1)} - x_1^{(k)})^2$ | 44 | Test |
| 10 | $a_{44}^{-1}$ | 26 | $X_2^{(K+1)}$ | | |

Tableau (3-17) : liste des opérations de l'algorithme de Gauss Seidel

La figure (3-13) présente le graphe représentatif de cet algorithme. Comme pour l'algorithme de Jacobi, les opérations $a_{ii}^{-1}$ sont effectuées seulement à la première itération, ce qui explique que les opérations 4, 7, 9 et 10 ne sont plus présentes à partir de la deuxième itération.

Le chemin critique de l'algorithme de Gauss-Seidel, à partir de la deuxième itération présentée la ligne double sur la figure (3-13), est composé de :

2 multiplications + $\log_2 n$ additions + [calcul de $x_1^{(k+1)}$ ] (3-3-13)



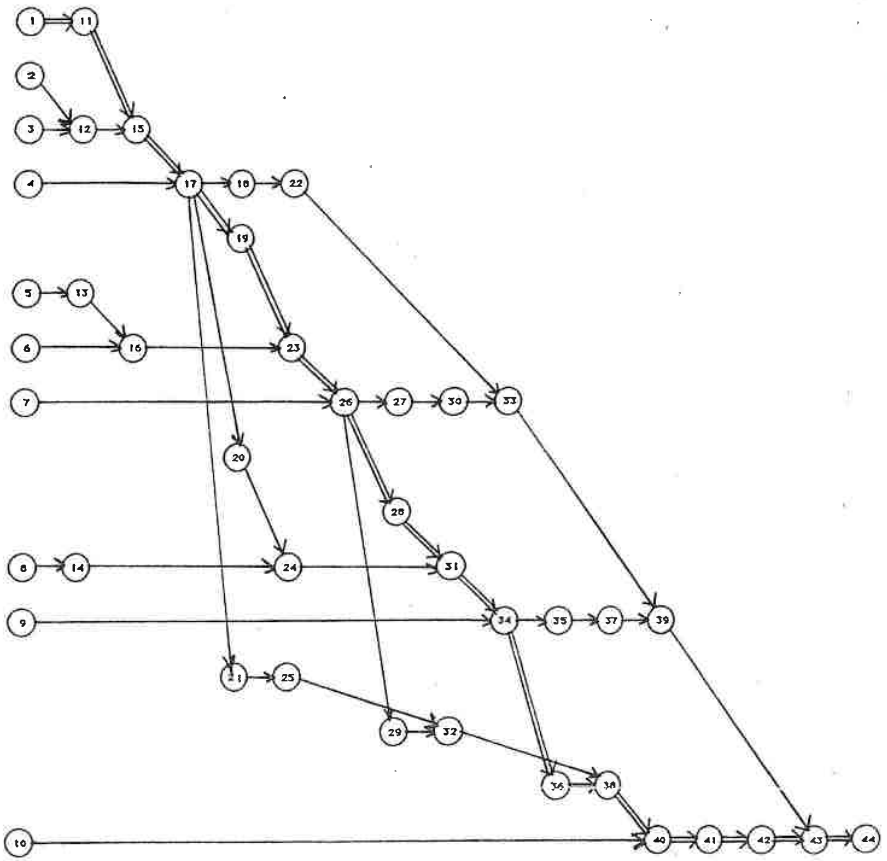


Figure (3-12) Graphe de l'algorithme de Gauss-Seidel

(n-1) (2 multiplications + 1 additions) + [calcul de xi(k+1) 1<i<=n]
2 additions + 1 multiplications + 1 test [test]

Ainsi, pour les trois cas cités précédemment, la durée minimale d'exécution d'une itération de l'algorithme est :

- Premier cas : ($t_{div} = t_{add} = t_{mult} = t_{test}$)

$$(3n + \log_2 n + 3)t_{add} \qquad (3\text{-}3\text{-}14)$$

- Deuxième cas : ($t_{div} = 6t_{add}$ ; $t_{mult} = 5t_{add}$ ; $t_{test} = t_{add}$)

$$(11n + \log_2 n + 7)t_{add} \qquad (3\text{-}3\text{-}15)$$

- Troisième cas : ($t_{div} = 34t_{add}$ ; $t_{mult} = t_{add} = t_{test}$)

Le temps de division influence le chemin critique de la première itération, mais il n'a pas d'effet sur les itérations suivantes. Dans ce cas, la longueur du chemin critique à partir de la deuxième itération est donc la même que celle du premier cas.

La durée du chemin critique de la méthode de Gauss-Seidel est donc beaucoup plus grande que celle de la méthode de Jacobi. Ces deux durées calculées par exemple à partir des équations (3-3-5) et (3-3-14), sont :



- pour l'algorithme de Jacobi ($5 + \log_2 n$ )
- pour l'algorithme de Gauss-Seidel ($3n + \log_2 n + 3$)

En particulier, pour n d'ordre élevé, la durée du chemin critique d'une itération de l'algorithme de Gauss-Seidel est beaucoup plus importante que celle de l'algorithme de Jacobi.

Par exemple, pour n = 64, on a :

- pour l'algorithme de Jacobi 11 $ta_{dd}$
- pour l'algorithme de Gauss-Seidel 201 $ta_{dd}$

Une itération de l'algorithme de Jacobi est 19 fois plus rapide que pour une itération de Gauss-Seidel. La vitesse de convergence de l'algorithme de Gauss-Seidel est normalement plus grande que celle de l'algorithme de Jacobi, mais cette rapidité ne compense pas la différence entre les durées des chemins critiques d'une itération de deux algorithmes. On peut ainsi conclure que pour n d'ordre élevé, l'algorithme de Jacobi est plus rapide que l'algorithme de Gauss-Seidel. Nous démontrerons par la suite qu'il existe des cas où l'algorithme de Gauss-Seidel est avantageux par rapport à l'algorithme de Jacobi en particulier lorsque n est faible (algorithme par bloc par exemple) et lorsque le nombre des processeurs est limité.

## ACCELERATION DE L'ALGORITHME DE GAUSS-SEIDEL

Comme dans le cas de l'algorithme de Jacobi, on peut diminuer la

durée du chemin critique de l'Algorithme de Gauss-Seidel en faisant chevaucher les itérations successives. Ainsi, par exemple, après avoir calculé la valeur de $x_2^{(k+1)}$, on peut commencer le calcul de $x_1^{(k+2)}$ et ainsi de suite. L'algorithme parallèle de Gauss-Seidel dans ce cas se présente de la façon suivante :

Début de l'algorithme

Calculer $a_{ii}^{-1}$, $a_{ij}x_j$, i=1 à n; l=1 à n- ; l<j<=n; en parallèle

Faire la somme $\sum_{j=l+1}^{n} a_{lj}x_j$ avec la méthode de dicotomie

en parallèle

Calculer $x_1^{(k+1)}$

Calculer $a_{l1}x_1^{(k+1)}$ l=2 à n; $\{x_1^{(k+1)}-x_1^{(k)}\}^2$ en parallèle

Calculer $X_2^{(k+1)}$

Calculer $a_{l2}x_2^{(k+1)}$ (l=1 à n l≠2); $\{x_2^{(k+1)}-x_2^{(k)}\}^2$

en parallèle

.....

Calculer $x_n^{(K+1)}$

Calculer $a_{ln}x_n^{(k+1)}$ l=1 à n-1; $\{x_n^{(k+1)}-x_n^{(k)}\}^2$ en parallèle

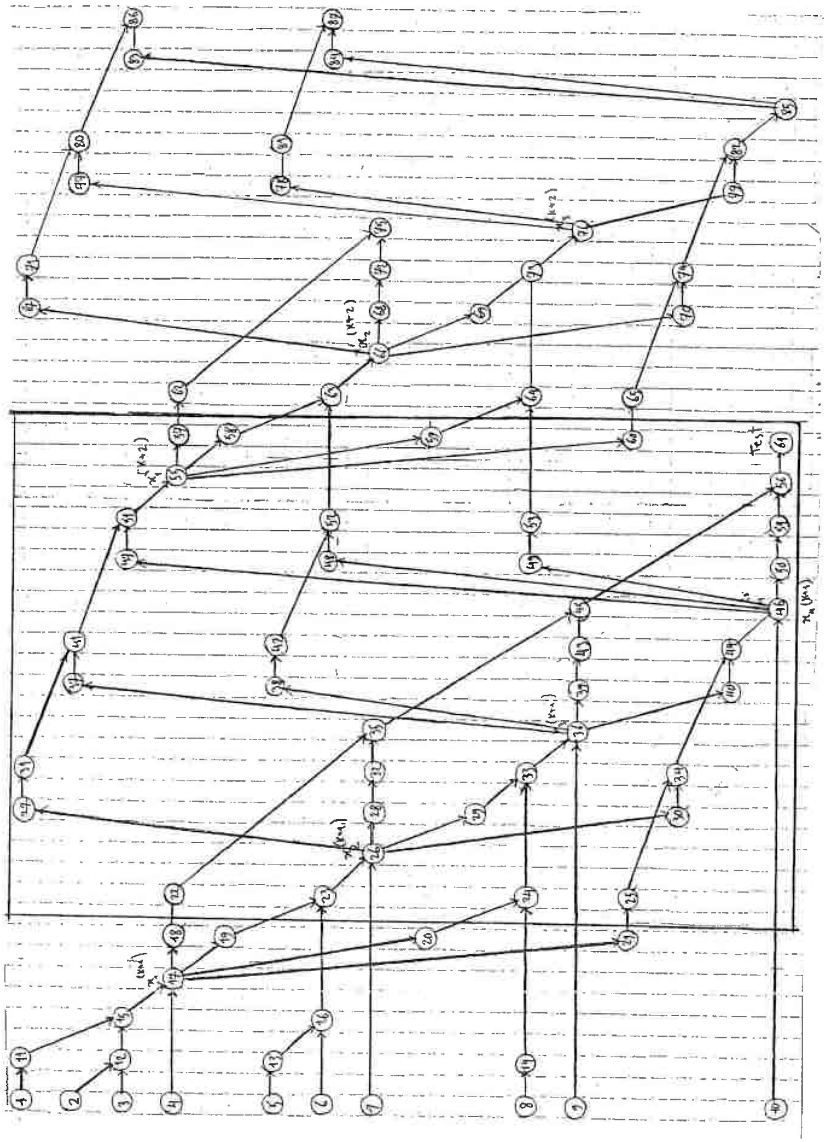
Si (test est vrai) FIN

Sinon itération suivante

Fin de l'algorithme

La figure (3-14) présente le graphe correspondant à cet algorithme. On peut remarquer qu'au moment du test de la première itération, une partie des opérations de l'itération 2 a déjà été exécutée. La partie encadrée du graphe (3-14) se répète ainsi à chaque itération. Cette partie est détaillée sur la figure (3-15).





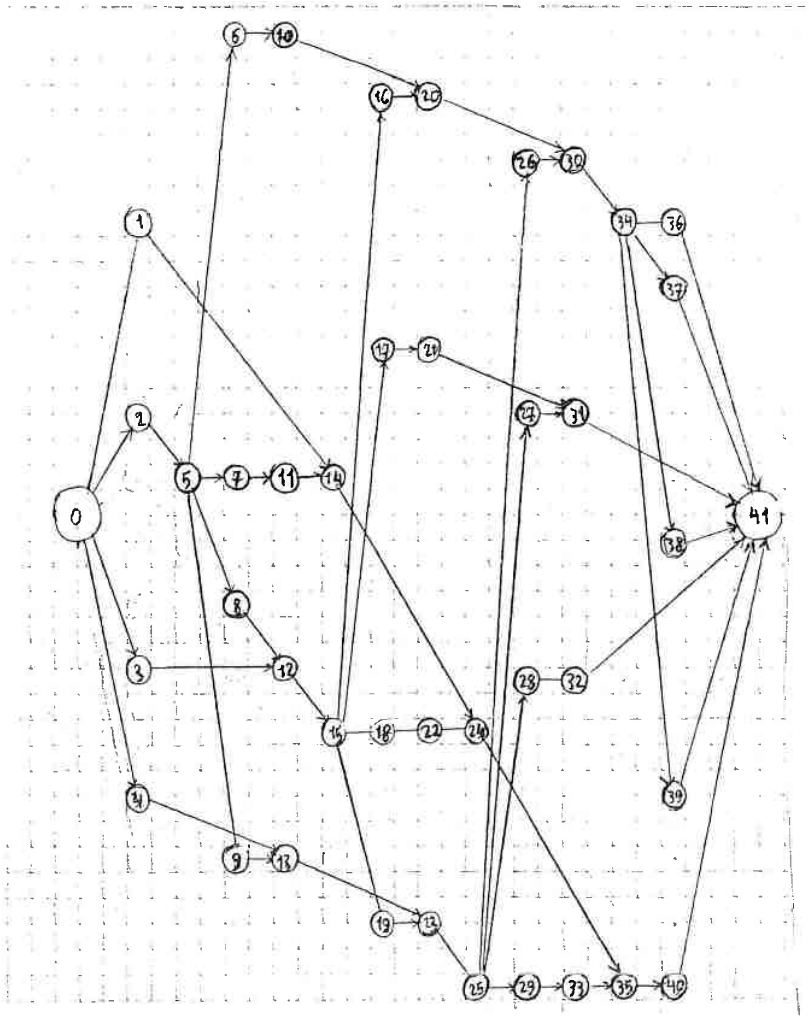



Figure (3-14) : algorithme de Gauss-Seidel par chevauchement des itérations



Le chemin critique du graphe (3-15) est le même que celui du graphe (3-13), sauf que la pratie du calcul de $x_1^{(k+1)}$ a déjà été exécutée. Il est donc composé de :

(2n-1) multiplication, + (n+1) addition, + 1 test

ainsi pour $t_{mult} = t_{add} = t_{test}$, on a :

$$[(2n-1) + (n+1) + 1]\, t_{add} = (3n + 1)\, t_{add} \qquad (3\text{-}3\text{-}16)$$

et pour $t_{mult} = 5\, t_{add}$ ; $t_{test} = t_{add}$

$$[(2n-1) \times 5 + (n+1) + 1]\, t_{add} = (11n-3)\, t_{add} \qquad (3\text{-}3\text{-}17)$$

Les temps calculés à partir des équations (3-3-16) et (3-3-17) sont meilleures que ceux calculés à partir des équations (3-3-14) et (3-3-15), mais ils sont encore très supérieurs au temps obtenus pour la méthode de Jacobi.

ORDONNANCEMENT DU CALCUL

De la même façon que pour l'algorithme de Jacobi, l'ordonnancement de l'algorithme Gauss-Seidel se fait sur une seule itération. Les tableaux (3-18) et (3-19) donnent les résultats de l'ordonnancement dans le cas de l'itération par points pour les graphes (3-12) et (3-14) . Les tableaux (3-20) et (3-21) présentent les mêmes résultats dans le cas d'itérations par blocs pour la matrice de la figure (3-10).

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 4 | 53 | 2,26 | 0,57 |
| 3 | 53 | 2,26 | 0,75 |
| 2 | 61 | 1,97 | 0,98 |

Tableau (3-18) : temps d'exécution pour l'algorithme de Gauss-Seidel par points dans le cas du graphe (3-12)

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 4 | 44 | 2,72 | 0,68 |
| 3 | 45 | 2,67 | 0,89 |
| 2 | 60 | 2 | 1 |

Tableau (3-19) : temps d'exécution pour l'algorithme de Gauss-Seidel par points dans le cas du graphe (3-14)



| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 4 | 462 | 2,02 | 0,51 |
| 3 | 464 | 2,02 | 0,67 |
| 2 | 532 | 1,76 | 0,88 |

Tableau (3-20) : temps d'exécution pour l'algorithme de Gauss-Seidel par blocs dans le cas du graphe (3-12)

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 4 | 426 | 2,2 | 0,55 |
| 3 | 430 | 2,18 | 0,73 |
| 2 | 508 | 1,84 | 0,92 |

Tableau (3-21) : temps d'exécution pour l'algorithme de Gauss-Seidel par blocs dans le cas du graphe (3-14)

Le temps correspondant au chemin critique pour le graphe (3-15) est obtenu avec 4 processeurs. Mais le temps obtenu avec 3 processeurs est pratiquement le même que le temps du chemin critique. Le temps d'exécution en utilisant le chevauchement des itérations a amélioré le temps d'exécution.

La comparaison des tableaux (3-15) et (3-16) avec les tableaux (3-19) et (3-21) montre que si l'on dispose d'un grand nombre de processeurs, le temps d'exécution obtenu par l'algorithme de Jacobi est inférieur à celui de l'algorithme de Gauss-Seidel.

Par contre, pour un nombre limité de processeurs, ces temps ne sont pas très différents l'un de l'autre. Dans ce cas, et à condition que l'algorithme de Gauss-Seidel converge plus vite, il est avantageux de résoudre le problème avec l'algorithme de Gauss-Seidel.

### 3-3-3 Comparaison des méthodes directes et itératives

Nous avons calculé en (3-2), la durée du chemin critique pour la triangularisation de la matrice d'un système d'équations linéaires par la méthode de Gauss La résolution complète du système nécessite ensuite la résolution d'un système triangulaire par la méthode de remontée.

En se référant aux équations (3-2-14) et (3-2-31), on trouve la durée totale du chemin critique de la méthode directe de Gauss , comme suit :

$$n \text{ divisions} + (3n-2) \text{ multiplications} + (2n-2) \text{ additions} \qquad (3\text{-}3\text{-}18)$$

Pour la méthode de Jacobi, avec chevauchement des itérations, cette durée est égale à :

pour le premier cas :



$$t_{div} \gg t_{mult} = t_{add}$$

$$1 \text{ division} + 1 \text{ multiplication} + L\,(2 \text{ multiplications} + \log_2 n \text{ addition}) \qquad (3\text{-}3\text{-}19)$$

et pour le deuxième cas :

$$t_{div} \cong t_{mult}$$

$$(L+1)\,(2 \text{ multiplications} + \log_2 n \text{ additions}) \qquad (3\text{-}3\text{-}20)$$

où L est le nombre d'itérations de l'algorithme de Jacobi, on peut ainsi comparer les deux algorithmes dans les trois cas cités précédemment.

Premier cas : $(t_{div} = t_{add} = t_{mult})$

*Algorithme direct de Gauss*

$$(n + 3n-2 + 2n-2)\, t_{add} = (6n-4)\, t_{add}$$

*Algorithme de Jacobi*

$$(L + 1)\,(2 + \log_2 n)\, t_{add}$$

On constate donc que la méthode de Jacobi est efficace, sous réserve de la condition suivante :

$$(L + 1)\,(2 + \log_2 n) < (6n-4)$$

où $L < (6n-4)/(2\log_2 n)-1$

Pour n = 64, par exemple L < 46

Deuxième cas : ($t_{div} = 6\, t_{add}$ ; $t_{mult} = 5\, t_{add}$)

*Algorithme direct de Gauss*

$$(6n + 15n-10 + 2n-2)\, t_{add} = (23n-12)\, t_{add}$$

*Algorithme de Jacobi*

$$(L + 1)(10 + \log_2 n)$$

on a donc $L < (23n-12)/(10+\log_2 n) - 1$

Pour n = 64 L < 90

Troisème cas : $t_{div} = 34\, t_{add}$ ; $t_{mult} = t_{add}$

*Algorithme direct de Gauss*

$$(34n + 3n-2 + 2n-2)\, t_{add} = (39n-4)$$

*Algorithme de Jacobi*:

$$35\, t\,add + L\,(2 + \log_2 n)\, t\,add$$



on a alors $L < (39n-12)/52+\log_2 n) - 35$

et pour n = 64 L < 276

On peut remarquer l'intérêt de l'algorithme de Jacobi notamment dans le troisième cas.

Dans le cas de l'algorithme par blocs, les deux chemins critiques se présentent de la façon suivante :

*Algorithme direct de Gauss*:

N inversions de matrice
(2N-1) multiplications de deux matrices +
(N-1) additions de deux matrices + (3-3-21)
(N-1) multiplications de matrice par vecteurs +
(N-1) additions de deux vecteurs

*Algorithme de Jacobi*:

1 inversion de matrice + 1 multiplication d'une matrice par un vecteur + L (2 multiplications dune matrice par des vecteurs + $\log_2 N$ additions de deux vecteurs). (3-3-22)

En se référent à l'annexe A1, on peut calculer le temps nécessaire pour effectuer les opérations matricielles, et il est possible de comparer la durée des chemins critiques de deux algorithmes :

Par exemple pour v=5; N=4; $t_{div}=6t_{add}$; $T_{mult}=5t_{add}$

on obtient pour l'Algorithme de Gauss-Seidel : 8085 $t_{add}$

l'Algorithme de Jacobi : ( 875 + 300 L)$t_{add}$

La condition d'efficacité de l'algorithme de Jacobi est alors L < 24.

Il est à noter qu'avec l'ordonnancement du calcul, le temps correspondant au chemin critique de l'algorithme direct de Gauss pour cet exemple, s'obtient sur 5 processeurs (voir tableau 3-6), tandis que le temps de l'algorithme de Jacobi est obtenu avec 16 processeurs.

Dans le cas d'un nombre limité de processeurs, le temps d'exécution dépasse normalement la durée du chemin critique. En ordonnancant le calcul, on peut alors distribuer les opérations de l'algorithme sur les processeurs d'une façon efficace afin d'augmenter le taux d'occupation des processeurs.

Supposons que le taux d'occupation des processeurs pour deux algorithmes directs de Gauss et de Jacobi soit proche de 1 : le temps d'exécution de chaque algorithme est alors proportionnel au nombre total d'opérations. Ce nombre pour les deux algorithmes, est :

*pour l'algorithme de Gauss*:

$((n^3-n)/3+n(n-1)/2)$ multiplications +



$((n^3-n)/3+n(n-1)/2)$ additions + (3-3-23)

$(n(n-1)/2+n)$ divisions

*pour l'algorithme de Jacobi*:

n divisions + L[$(n^2 + n)$ multiplications + $(n^2 + n$ additions] (3-3-24)

On peut ainsi comparer le temps d'exécution de chaque algorithme.

Pour le deuxième cas ($t_{div} = 6\, t_{add}$ $t_{mult} = 5\, t_{add}$) :

la condition d'efficacité de l'algorithme de Jacobi est :

$$L < ( n^2 + 3\, n)/(3n+3) \approx n/3 \qquad (3\text{-}3\text{-}24)$$

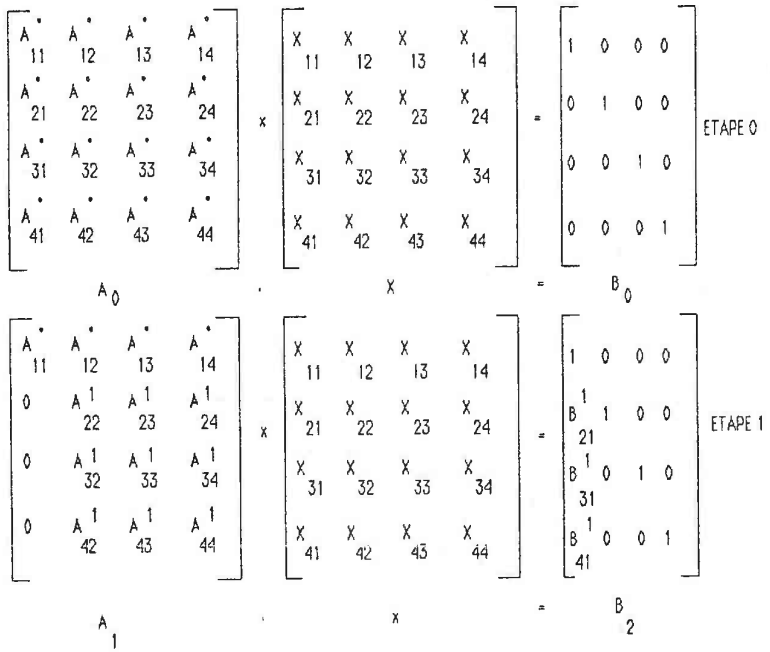
Le nombre total d'opérations pour une itération de l'algorithme de Gauss-Seidel est le même que celui de l'algorithme de Jacobi. Dans le cas d'un nombre limité de processeurs, il est donc nécessaire de comparer le temps de calcul de l'algorithme de Jacobi , de Gauss-Seidel et de la méthode directe de Gauss-Seidel.

## 3-4 INVERSION D'UNE MATRICE

Pour étudier le calcul parallèle de l'inversion d'une matrice, nous considérons le cas d'une matrice symétrique définie positive, matrice que

l'on rencontre fréquemment en Automatique (filtre de Kalman, équation de Riccati ...).

L'approche utilisée pour ce calcul est basée sur la méthode de Gauss pour la résolution des systèmes linéaires. La figure (3-16) montre les différentes étapes de cette méthode pour une matrice de l'ordre 4.





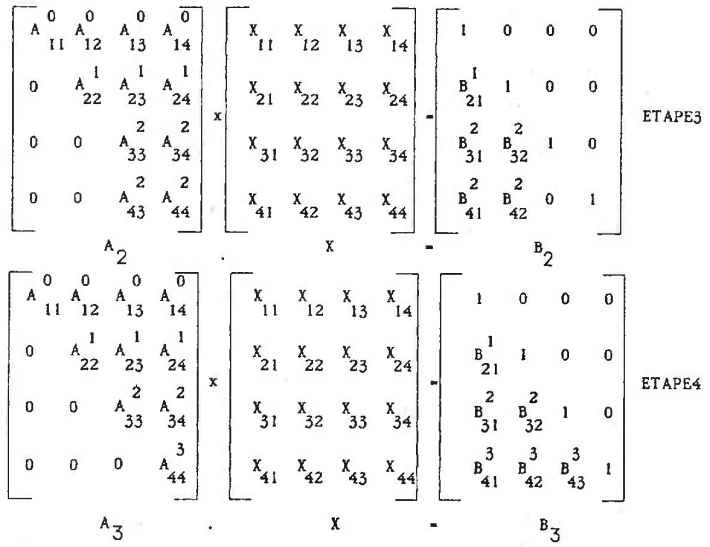




Figure (3-15) : les étapes d'inversion d'une matrice symétrique

A chaque étape, on calcule les éléments du triangle supérieur de la matrice $A_k$ et les éléments du triangle inférieur de la matrice B.

L'algorithme parallèle du calcul l'inverse d'une matrice, ressemble alors, pour la partie triangularisation, à l'algorithme direct de Gauss-Seidel, et pour la partie de calcul des éléments $X_{ij}$ de la matrice inverse, à la résolution de n systèmes linéaire triangulaires. Le tableau (3-22) montre une partie des opérations nécessaires pour cet algorithme, pour une matrice de l'ordre 4.

| N° | OPERATION | N° | OPERATION | N° | OPERATION |
|---|---|---|---|---|---|
| 1 | $(A_{11})^{-1}$ | 30 | $(A_{33})^{-1}$ | 42 | $Y_{22}=A_{24}X_{42}$ |
| 2 | $D_{21}=A_{21}-(A_{11})^{-1}$ | 32 | $C_{44}=D_{43}A_{34}$ | 57 | $Y_{22}=Y_{22}+A_{23}X_{32}$ |
| 5 | $C_{22}=D_{21}A_{12}$ | 37 | $A_{44}=A_{44}+C_{44}$ | 60 | $Y_{22}=1-Y_{22}$ |
| 11 | $A_{22}=A_{22}+C_{22}$ | 38 | $X_{44}=(A_{44})^{-1}$ | 62 | $X_{22}=(A_{22})Y_{22}$ |
| 17 | $(A_{22})^{-1}$ | 40 | $X_{43}=B_{43}X_{44}$ | 68 | Tâche finale |

Tableau (3-22) : liste des opérations pour l'inversion d'une matrice symétrique

La figure (3-17) fournit le graphe correspondant à cet algorithme. Le nombre total d'opérations pour l'inversion d'une matrice symétrique avec cette méthode est égal à :

n divisions

$(n^2-1+n^2(n-1)/2)$ multiplications (3-4-1)

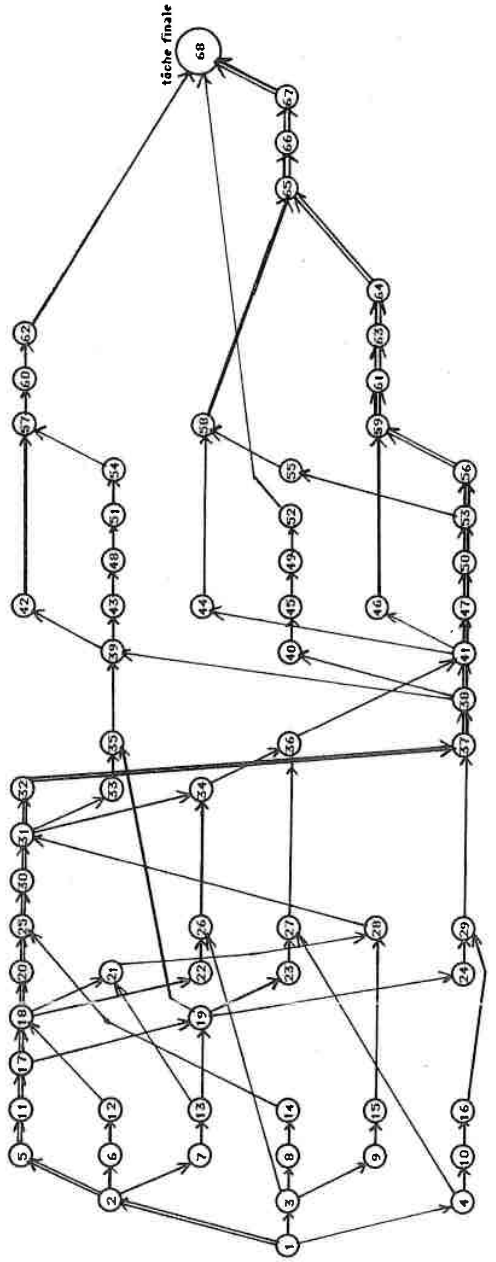
$(n(n-1)+n(n-1)(n-2)/2)$ additions

Cette méthode s'étend à l'inversion par blocs d'une matrice. Dans ce cas, le nombre total d'opérations est égal à :

n inversions de matrices

$(n^2-1)$ multiplications de deux matrices dont le résultat est symétrique






Fig(3-17) Graphe d'inversion d'une matrice symétrique.

$n^2(n-1)/2$ multiplications de deux matrices avec le résultat non symétrique

$n(n-1)$ additions de deux matrices symétriques

$n(n-1)(n-2)/2$ additions de deux matrices non symétriques (3-4-2)

Le chemin critique du graphe (3-16) est composé de :

n divisions

$[(n+1) + (3n-4)]$ multiplications

$[(n+1) + (2n-5)]$ additions

et dans le cas d'inversion par blocs :

n inversions de matrices

$(n+1)$ multiplications de deux matrices dont le résultat est symétrique

$(3n-4)$ multiplications de deux matrices dont le résultat est non symétrique

$(n+1)$ additions de deux matrices symétriques

$(2n-5)$ additions de deux matrices non symétriques (3-4-3)

Le taux d'accélération de l'algorithme par points est alors :

- Premier cas : $(t_{div} = t_{add} = t_{mult})$

$$t_{acc} = (n^3 + n-1)/(8n-7) \qquad (3-4-3)$$



valeur asynptotique : $n^2/8 = 0,125\ n^2$

- Deuxième cas : $(t_{div} = 6\ t_{add} ; t_{mult} = 5\ t_{add})$

$$t_{acc} = (3n^3 + 2n^2 + 6n - 5)/(29n - 19) \qquad (3-4-5)$$

valeur asynptotique : $3n^2/29 = 0,1n^2$

- Troisème cas : $(t_{div} = 34\ t_{add} ; t_{mult} = t_{add})$

$$t_{acc} = (n^3 + 34n - 1)/(41n - 6) \qquad (3-4-6)$$

valeur asynptotique : $n^2/41 = 0,025\ n^2$

Ici encore, la très grande durée des divions du troisième cas a une influence considérable sur le taux d'accélération.

ORDONNANCEMENT DU CALCUL

Nous avons appliqué les méthodes expliquées au chapitre 2 pour l'ordonnancement du calcul de l'inverse d'une matrice symétrique. Le tableau (3-23) fournit les résultats obtenus dans le cas de l'inversion d'une matrice de l'ordre 4 pour : $t_{div} = 6\ t_{add}$ ; $t_{mult} = 5\ t_{add}$. Le tableau (3-24) donne les mêmes résultats pour l'inversion par blocs d'une matrice de l'ordre 16 avec

des blocs de dimension 4

| Nombre de processeurs | Temps d'exécution | Taux d'acélération | Taux d'occupation |
|---|---|---|---|
| 4 | 97 | 2,5 | 0,63 |
| 3 | 97 | 2,5 | 0,84 |
| 2 | 127 | 1,91 | 0,96 |

Tableau (3-23) : temps d'exécution pour l'inversion d'une matrice symétrique

| Nombre de processeurs | Temps d'exécution | Taux d'acélération | Taux d'occupation |
|---|---|---|---|
| 4 | 97 | 2,5 | 0,63 |
| 3 | 97 | 2,5 | 0,84 |
| 2 | 127 | 1,91 | 0,96 |

Tableau (3-24) : tamps d'exécution pour l'inversion d'une matrice par blocs



On peut remarquer par ordonnancement, que le temps du chemin critique est obtenu sur trois processeurs au lieu de 6 dans le cas de l'inversion par points. Le temps du chemin critique est obtenu sur 4 processeurs au lieu de 6 pour l'inversion par blocs ; mais le temps obtenu sur trois processeurs n'est plus loin du temps du chemin critique.

## 3-4- Conclusion

Nous avons étudié les méthodes directes et ittératives de résolution des sytèmes linaires. Pour chaque algorithme, nous avons déterminé la durée minimum d'exécution et le taux maximum d'accélération. Nous avons calculé ensuite le temps du calcul sur un nombre donné de processeurs en cas d'ordonnancement des tâches. Nous avons démontré, que dans le cas général et avec stratégie du pivot maximum, la méthode de Gauss-Seidel, nécessite un ordonnancement dynamique, qui n'est pas souhaitable en Automatique. Par contre, elle est intéressante pour l'inversion des matrices symétriques définies positives ou les systèmes d'équations linéaires représentées par une telle matrice. Nous avons démontré également, que la méthode de Gauss pour la résolution de ces systèmes en parallèle, convient mieux que la méthode de Cholesky.

Pour les méthodes itératives, nous avons étudié les techniques d'accélération des algorithmes parallèles. Parmi ces techniques, le chevauchement des itérations nous semble le plus efficace. Nous avons comparé les méthodes itératives entre elles et avec les méthodes directes. Pour un grand nombre de processeurs, la méthode de Jacobi est plus rapide

que la méthode de Gauss-Seidel, mais pour un nombre limité de processeurs, la méthode de Gauss-Seidel peut être avantageuse. L'efficacité des méthodes ittératives par rapport aux méthodes directes dépend du nombre d'itérations pour la convergence. Nous avons établi les conditions nécesaires pour cette efficacité.

# - CHAPITRE 4 -

# ALGORITHMES PARALLELES D'IDENTIFICATION

## 4-1 INTRODUCTION

Le domaine aérospatial utilise largement les algorithmes d'identification et d'estimation, mais les techniques employées nécessitent un temps de calcul important : c'est pourquoi nous trouvons peu d'applications dans l'industrie et dans les domaines socio-économiques qui utilisent des systèmes de grande dimension. La décentralisation de ce calcul, en l'effectuant en parallèle sur plusieurs processeurs permet de donner au système la vitesse nécessaire.

La fiabilité des données est d'une très grande importance pour l'identification des systèmes. Nous commençons donc ce chapitre par l'étude des techniques parallèles de validation des données. Nous étudions ensuite l'implantation parallèle des algorithmes d'estimation des paramètres. Nous abordons enfin les algorithmes d'estimation d'état, ce qui nous amène naturellement à étudier le filtre de Kalman.

## 4-2 MULTIPLICATION DE DEUX MATRICES

Dans les algorithmes d'identification, on a souvent besoin de multiplier deux ou plusieurs matrices de grande dimension. Nous étudions donc brièvement les méthodes parallèles de multiplication de deux matrices.

Soit $C = A*B$ (4-1-1)

où A et B sont respectivement de dimensions mxn et nxk.

La matrice C est alors de dimension nxk et ses éléments sont déterminés par :

$$C_{ij} = \sum_{l=1}^{N} a_{il}\, b_{il} \qquad (4\text{-}2\text{-}2)$$

Le nombre de multiplications élémentaires est de mnk. Ces multiplications peuvent être effectuées en une seule étape, sur mnk processeurs. Pour la somme des éléments $a_{il}b_{lj}$, il faut $\log_2 n$ étapes sur au maximum mnk/2 processeurs.

La multiplication de deux matrices se fait donc en $1 + \log_2 n$ étapes sur au maximum mnk processeurs.

Si T est le temps de multiplications de deux matrices, $t_{mult}$ le temps de multiplication de deux nombres réels par les processeurs utilisés et $t_{add}$ le temps de l'addition on a :

$$T = t_{mult} + \log_2 n \cdot t_{add} \qquad (4\text{-}2\text{-}3)$$

La figure (4-1) donne le graphe correspondant pour n=2. Les tâches 1 à 8 représentent les multiplications et les tâches 9 à 12 correspondent aux additions, la tâche 13 étant la tâche finale.



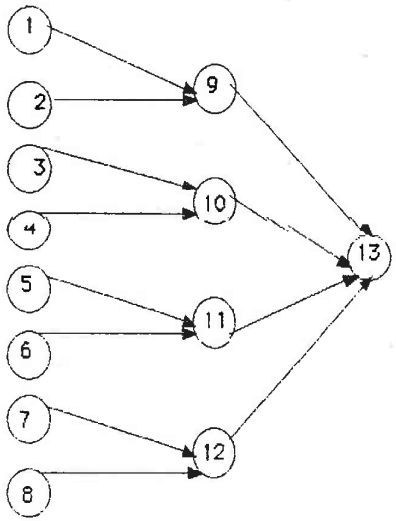



Figure (4-1) : Graphe de multiplication de deux matrices

Pour les matrices creuses, le nombre de processeurs nécessaires pour la multiplication est très inférieur à m.N.K

En effet si $n_j$ et $m_j$ sont respectivement le nombre des éléments non nuls de la colonne j de la matrice A et de la ligne j de la matrice B, le nombre de multiplication élémentaires est donné par :

$$\sum_{j=1}^{n} n_j\, m_j \qquad (4\text{-}2\text{-}4)$$

Ainsi par exemple, pour la multiplication des deux matrices A et B présentées à la figure (4-2), le nombre de multiplications est de

1x2+3x3+2x2+3x2= 21

$$A = \begin{bmatrix} X & 0 & X & X \\ 0 & X & X & 0 \\ 0 & X & 0 & X \\ 0 & X & 0 & X \end{bmatrix} \qquad B = \begin{bmatrix} X & 0 & X & ( \\ 0 & X & X & X \\ X & 0 & X & 0 \\ 0 & X & X & 0 \end{bmatrix}$$

Figure (4-2) : Structure de deux matrices creuses

La figure (4-3) nous présente le graphe représentatif de la multiplication de ces deux matrices.



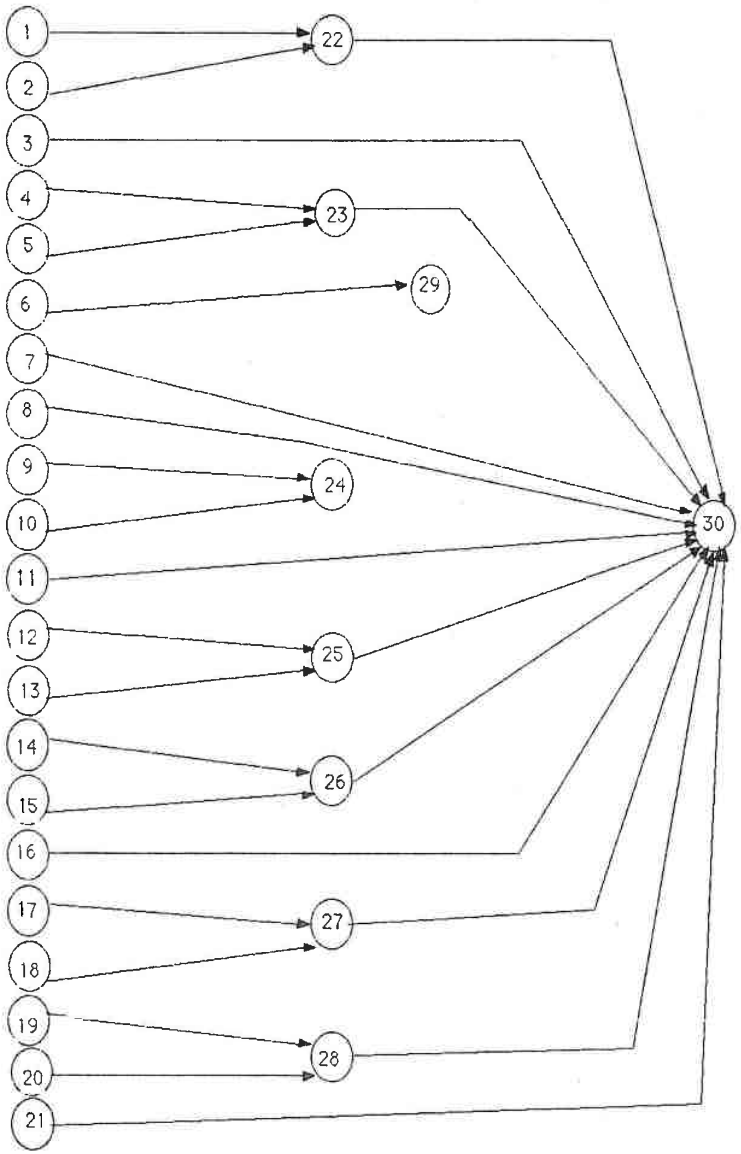



Figure (4-3) Multiplication de deux matrices creuses

MULTIPLICATION DE DEUX MATRICES PAR BLOC

La multiplication par bloc dans son principe est la même que la multiplication par point. La différence est dans la durée d'exécution des tâches qui n'est pas la même si les blocs ont des dimensions différentes.

Dans ce cas, le temps minimum de multiplication de deux matrices est donné par les opérations sur le bloc le plus important.

Considérons les blocs A11, A22, A33, A44 de la matrice A de la figure (4-2), respectivement de dimension 2, 3, 3 et 4. Pour $t_{mult} = 5\ t_{add}$, le chemin critique du graphe 4-3 est composé des tâches 20-28-30 et sa durée est de 288 $t_{add}$. Cette durée sans ordonnancement s'obtient sur 21 processeurs. La durée de multiplication de ces deux matrices sur un seul processeur est 3544 $t_{add}$. Les taux d'accélération et d'occupation pour 21 processeurs sont :

$$\tau_{accel} = \frac{3544}{288} = 12,3$$

$$\tau_{occup} = \frac{12,3}{21} = 0,59$$

Par ordonnancement, on peut multiplier les deux matrices dans un temps de 288 $t_{add}$ sur 15 processeurs. Le taux d'occupation dans ce cas devient :

$$\tau_{occup} = \frac{12,3}{15} = 0,82$$



qui est nettement meilleur que dans le cas de 21 processeurs.

## 4-3 VALIDATION DES DONNEES [4-1], [4-2], [4-3]

Considérons un système représenté par des réseaux de transport en matière, qui peut-être décrit par les modèles linéaires exacts en régime statique, qui découlent des lois de conservation de la matière :

$$M.\,Q^* = 0 \qquad (4\text{-}3\text{-}1)$$

où M. est la matrice d'incidence de dim (n,v) d'éléments :

$m_{ij} = 1$ si la voie j entre dans le noeud i,

$m_{ij} = -1$ si la voie j sort du noeud i,

$m_{ij} = 0$ s'il n'y a pas de lien entre le noeud i et la voie j,

$Q^*$ est le vecteur des vraies valeurs de dim v,

v est le nombre des voies,

n est le nombre de noeuds.

La relation entre le vecteur de mesures Q est les grandeurs vraies sont :

$$Q = Q^* + \underline{\varepsilon} \qquad (4\text{-}3\text{-}2)$$

où $\underline{\varepsilon}$ est le vecteur des erreurs de mesures, qui suit une loi normale centrée de matrice de variance covariance v.

Le problème de la validation de données se formule alors de la façon suivante :

Calculer $\hat{Q}$, le meilleur estimateur de $Q^*$ qui minimise le critère.

$$J = \frac{1}{2} \|\hat{Q}-Q\|^2_{V^{-1}} \qquad (4\text{-}3\text{-}3)$$

sous les contraintes

$$M\hat{Q} = 0 \qquad (4\text{-}3\text{-}4)$$

$\hat{Q}$ est alors le meilleur estimateur de $Q^*$ au sens du maximum de vraisemblance.

La Lagrangien associé à ce problème s'écrit donc :

$$L = \frac{1}{2} \|\hat{Q}-Q\|^2_{V^{-1}} + \underline{\lambda}^T M\hat{Q} \qquad (4\text{-}3\text{-}5)$$

où $\underline{\lambda}$ est le vecteur des paramètres de Lagrange de dimension n.

Les conditions de stationnarité de ce Lagrangien sont :

$$\frac{\partial L}{\partial Q} = V^{-1}(Q-Q) + M^T \underline{\lambda} = 0 \qquad (4\text{-}3\text{-}6)$$

$$\frac{\partial L}{\partial \underline{\lambda}} = M Q = 0$$

ou

$$MVM^T \underline{\lambda} = MQ \qquad (4\text{-}3\text{-}7)$$

$$Q = Q - VM^T \underline{\lambda} \qquad (4\text{-}3\text{-}8)$$



On détermine ainsi le vecteur $\underline{\lambda}$ à partir de la résolution du système 4-3-7 et on calcule la valeur de $\hat{Q}$ en remplaçant $\underline{\lambda}$ dans l'équation (4-3-8).

CALCUL PARALLELE DE L'ALGORITHME

Le calcul de $\hat{Q}$ nécessite les opérations suivantes :

1- multiplication de deux matrices v et $M^T$
2- multiplication de deux matrices M et $(VM^T)$
3- multiplication de la matrice M par le vecteur Q
4- résolution du système linéaire symétrique $(MVM^T)\underline{\lambda} = (MQ)$
5- multiplication de la matrices $(VM^T)$ par le vecteur $\underline{\lambda}$
6- soustraction de deux vecteurs Q et $(VM^T \underline{\lambda})$

Dans les chapitres précédents nous avons étudié les algorithmes parallèles permettant l'exécution de chacune de ces opérations.

Dans ce chapître, nous proposons d'étudier le calcul parallèle de l'estimation $\hat{Q}$ dans trois cas, à partir du système linéaire (4-3-7) :

1- résolution directe
2- résolution itérative par la méthode de Jacobi
3- résolution itérative par la méthode de Gauss-Seidel

Pour mettre en oeuvre les techniques du calcul parallèle, nous décomposons la matrice M en blocs de sous-matrices. Supposons par

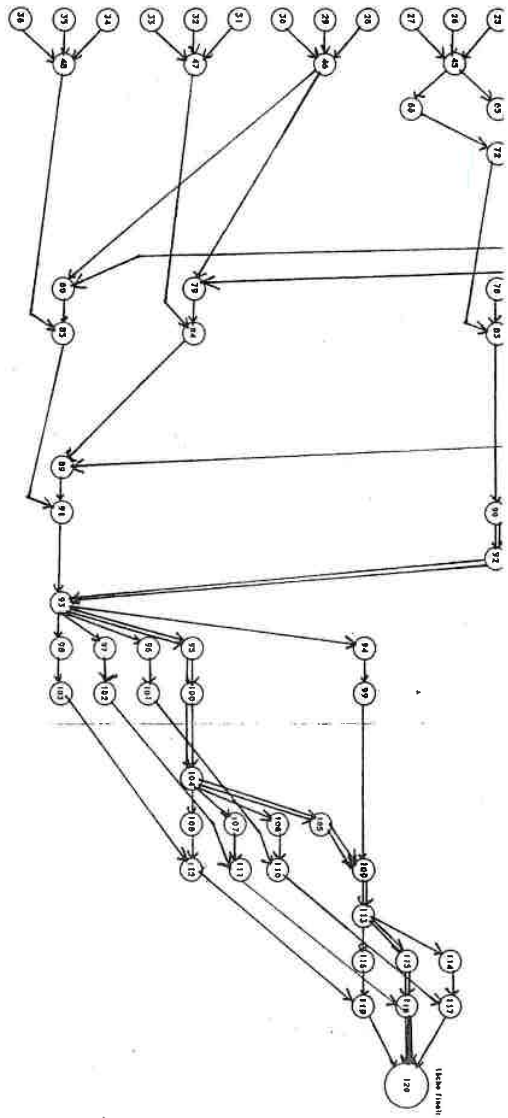
exemple que la matrice M est de dimension 10 . 20 et qu'elle est décomposée comme indiqué sur la figure (4-4). Le vecteur Q dans ce cas est de dimension 20 et est décomposé en trois vecteurs de dimensions 6,8 et 6.

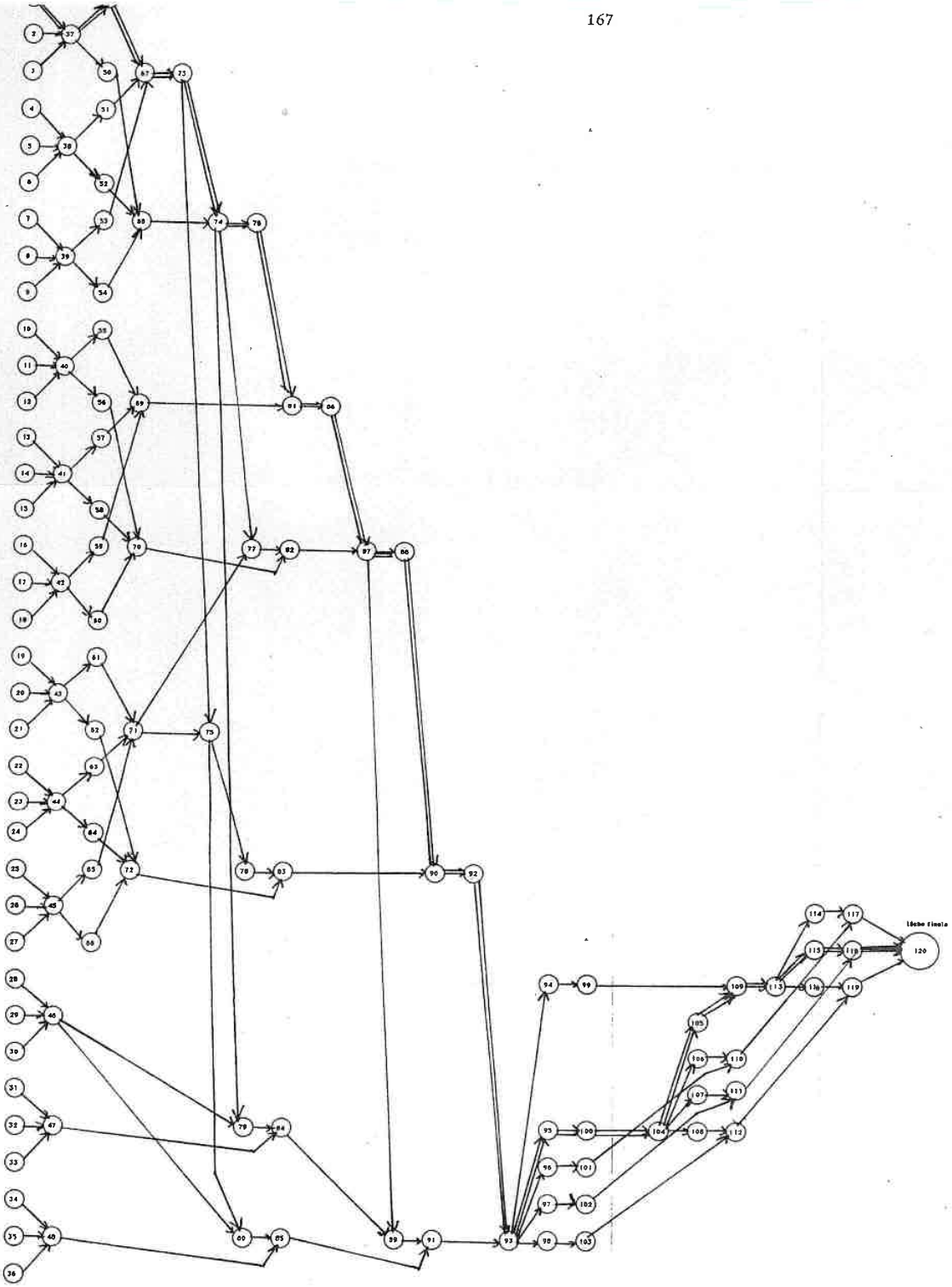
| | | |
|---|---|---|
| 3 x 6<br>M11 | 3 x 8<br>M12 | 3 x 6<br>M13 |
| 4 x 6<br>M21 | 4 x 8<br>M22 | 4 x 6<br>M23 |
| 3 x 6<br>M31 | 3 x 8<br>M32 | 3 x 6<br>M33 |

Figure (4-4) Décomposition de la matrice M

Nous considérons également le cas général où la matrice M est pleine, sachant qu'en pratique cette matrice M est souvent creuse [4-2] .

La figure (4-5) présente le graphe du calcul $\hat{Q}$ dans le cas de la résolution directe du système linéaire 4-2-7. Sur ce graphe, les tâches 1 à 27 correspondent aux multiplications des sous-matrices $L_{ij} = V_{ik}.M_{kj}$ et les tâches 28 à 36 représentent les multiplications des sous-matrices $M_{ij}$ par les vecteurs Qj. Les additions des sous-matrices pour former les blocs de la matrices $L_{ij}$ sont numérotées de 37 à 45. Les tâches 46 à 48 sont les éléments du vecteurs $\underline{b} = MQ$ et les tâches 49 à 66 sont les mutliplications $M_{ik}\ L_{kj}$ . Les tâches 62 à 72 forment les blocs du triangle supérieur de la matrice $H = MVM^T$ ; comme H est symétrique, le calcul du triangle supérieur est suffisant. Enfin, les tâches 73 à 92 correspondent à la partie triangularisation de la matrice symétrique H et les tâches 93 à 119




Idoine finale

représentent la résolution du système triangulaire pour le calcul de $\underline{\lambda}$, le calcul de $VM^T \underline{\lambda}$ et de $\underline{\hat{Q}}$.

Pour le cas où $t_{div}=6t_{add}$ et $t_{mult} = 5t_{add}$, nous avons calculé la durée des tâches 1 à 119 en fonction de $t_{add}$. En fonction de ces durées, le chemin critique du graphe 4-5 est le chemin représenté en ligne double. La longueur du chemin critique dans ce cas est 3697 $t_{add}$ et le taux maximum d'accélération pour cet algorithme est donc

$$t_{accel} = \frac{37074}{3697} = 10$$

Le nombre maximum de processeurs pour exécuter l'algorithme dans le temps du chemin critique est de 36, ce qui donne un taux d'occupation de :

$$t_{occup} = \frac{10}{36} = 0,28$$

Par l'ordonnancement du calcul, on peut exécuter cet algorithme dans le temps du chemin critique sur 17 processeurs. Le taux d'occupation dans ce cas devient :

$$t_{occup} = \frac{10}{17} = 0,59$$

qui n'est pas encore très élevé. De toute façon, si l'on veut exécuter cet

algorithme dans le temps du chemin critique on ne peut pas espérer un taux d'occupation élevés.

Le tableau 4-1 donne le temps d'exécution de l'algorithme pour différents nombres de processeurs.

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 17 | 3697 | 10,00 | 0,59 |
| 15 | 3844 | 9,64 | 0,64 |
| 12 | 4225 | 8,77 | 0,73 |
| 10 | 4587 | 8,08 | 0,81 |
| 8 | 5295 | 7,00 | 0,88 |
| 6 | 6531 | 5,68 | 0,95 |
| 4 | 9385 | 3,95 | 0,99 |
| 3 | 12438 | 2,98 | 0,99 |

Tableau (4-1) Temps d'exécution pour l'algorithme de validation des données.

On peut remarquer qu'à partir de 6 processeurs le taux d'occupation des processeurs est très élevé. Le tableau 4-2 donne l'organisation du calcul sur 3 processeurs.

3 processeurs

| activite | duree | debut | fin |
|---|---|---|---|
| 14 | 1504 | 0 | 1504 |
| 5 | 1128 | 0 | 1128 |
| 13 | 1120 | 0 | 1120 |
| 15 | 1120 | 1120 | 2240 |
| 6 | 840 | 1128 | 1968 |
| 4 | 840 | 1504 | 2344 |
| 8 | 846 | 1968 | 2814 |
| 2 | 846 | 2240 | 3086 |
| 11 | 1128 | 2344 | 3472 |
| 17 | 1128 | 2814 | 3942 |
| 7 | 630 | 3086 | 3716 |
| 3 | 630 | 3472 | 4102 |
| 9 | 630 | 3716 | 4346 |
| 1 | 630 | 3942 | 4572 |
| 23 | 1128 | 4102 | 5230 |
| 16 | 840 | 4346 | 5186 |
| 12 | 840 | 4572 | 5412 |
| 18 | 840 | 5186 | 6026 |
| 10 | 840 | 5230 | 6070 |
| 22 | 840 | 5412 | 6252 |
| 24 | 840 | 6026 | 6866 |
| 20 | 846 | 6070 | 6916 |
| 26 | 846 | 6252 | 7098 |
| 38 | 48 | 6866 | 6914 |
| 52 | 564 | 6914 | 7478 |
| 51 | 423 | 6916 | 7339 |

Tableau (4-2) Organisation du calcul sur trois processeurs

pour l'algorithme de validation de données



| | | | |
|---|---|---|---|
| 39 | 36 | 7098 | 7134 |
| 37 | 36 | 7134 | 7170 |
| 25 | 630 | 7170 | 7800 |
| 21 | 630 | 7339 | 7969 |
| 27 | 630 | 7478 | 8108 |
| 19 | 630 | 7800 | 8430 |
| 53 | 315 | 7969 | 8284 |
| 49 | 315 | 8108 | 8423 |
| 54 | 420 | 8284 | 8704 |
| 50 | 420 | 8423 | 8843 |
| 41 | 64 | 8430 | 8494 |
| 57 | 752 | 8494 | 9246 |
| 40 | 48 | 8704 | 8752 |
| 42 | 48 | 8752 | 8800 |
| 67 | 18 | 8800 | 8818 |
| 55 | 560 | 8818 | 9378 |
| 59 | 560 | 8843 | 9403 |
| 73 | 112 | 9246 | 9358 |
| 68 | 24 | 9358 | 9382 |
| 44 | 48 | 9378 | 9426 |
| 74 | 204 | 9382 | 9586 |
| 58 | 564 | 9403 | 9967 |
| 63 | 423 | 9426 | 9849 |
| 45 | 36 | 9586 | 9622 |
| 43 | 36 | 9622 | 9658 |

| | | | |
|---|---|---|---|
| 68 | 315 | 9658 | 9973 |
| 61 | 315 | 9849 | 10164 |
| 76 | 272 | 9967 | 10239 |
| 60 | 420 | 9973 | 10393 |
| 56 | 420 | 10164 | 10584 |
| 69 | 32 | 10239 | 10271 |
| 81 | 16 | 10271 | 10287 |
| 86 | 243 | 10287 | 10530 |
| 71 | 18 | 10393 | 10411 |
| 77 | 204 | 10411 | 10615 |
| 64 | 564 | 10530 | 11094 |
| 70 | 24 | 10584 | 10608 |
| 62 | 420 | 10608 | 11028 |
| 82 | 12 | 10615 | 10627 |
| 87 | 276 | 10627 | 10903 |
| 66 | 420 | 10903 | 11323 |
| 75 | 153 | 11028 | 11181 |
| 88 | 207 | 11094 | 11301 |
| 29 | 141 | 11181 | 11322 |
| 78 | 153 | 11301 | 11454 |
| 32 | 188 | 11322 | 11510 |
| 28 | 102 | 11323 | 11425 |
| 30 | 102 | 11425 | 11527 |
| 33 | 140 | 11454 | 11594 |
| 31 | 140 | 11510 | 11650 |



| | | | |
|---|---|---|---|
| 35 | 141 | 11527 | 11668 |
| 46 | 6 | 11594 | 11600 |
| 72 | 18 | 11600 | 11618 |
| 79 | 68 | 11618 | 11686 |
| 83 | 9 | 11650 | 11659 |
| 90 | 9 | 11659 | 11668 |
| 34 | 102 | 11668 | 11770 |
| 36 | 102 | 11668 | 11770 |
| 92 | 112 | 11686 | 11798 |
| 47 | 8 | 11770 | 11778 |
| 80 | 51 | 11770 | 11821 |
| 84 | 4 | 11778 | 11782 |
| 89 | 69 | 11782 | 11851 |
| 48 | 6 | 11798 | 11804 |
| 85 | 3 | 11821 | 11824 |
| 91 | 3 | 11851 | 11854 |
| 93 | 51 | 11854 | 11905 |
| 95 | 68 | 11905 | 11973 |
| 94 | 51 | 11905 | 11956 |
| 97 | 136 | 11905 | 12041 |
| 99 | 3 | 11956 | 11959 |
| 98 | 102 | 11959 | 12061 |
| 100 | 4 | 11973 | 11977 |
| 104 | 92 | 11977 | 12069 |
| 96 | 102 | 12041 | 12143 |

| | | | |
|---|---|---|---|
| 102 | 8 | 12061 | 12069 |
| 105 | 69 | 12069 | 12138 |
| 107 | 184 | 12069 | 12253 |
| 109 | 3 | 12138 | 12141 |
| 113 | 51 | 12141 | 12192 |
| 106 | 138 | 12143 | 12281 |
| 108 | 138 | 12192 | 12330 |
| 115 | 136 | 12253 | 12389 |
| 114 | 102 | 12281 | 12383 |
| 116 | 102 | 12330 | 12432 |
| 101 | 6 | 12383 | 12389 |
| 103 | 6 | 12389 | 12395 |
| 111 | 8 | 12389 | 12397 |
| 110 | 6 | 12395 | 12401 |
| 112 | 6 | 12397 | 12403 |
| 118 | 8 | 12401 | 12409 |
| 117 | 6 | 12403 | 12409 |
| 119 | 6 | 12432 | 12438 |
| 120 | 0 | 12438 | 12438 |



## METHODES ITERATIVES

Considérons les système linéaires de la forme

$$(H^T H)\underline{\theta} = H^T \underline{Z} \qquad (4\text{-}3\text{-}9)$$

Pour les résoudre, plusieurs méthodes itératives ont été proposées [4-4], [4-5], [4-6]. La méthode proposée dans [4-4] consiste à décomposer les matrices H et $H^T H$ de la manière suivante :

$$H = H_D + H_{HORS} \qquad (4\text{-}3\text{-}10)$$

$$H^T H = (H^T H)_D + (H^T H)_{HORS}$$

où les indices D et HORS désignent respectivement les parties blocs-diagonales et hors-diagonales des matrices.

L'algorithme de moindre carré est alors déterminé comme suit :

$$\underline{\theta}^{(k+1)} = \underline{\theta}^{(k)} - (H_D{}^T H_D)^{-1} (H^T H_D)^{-1} (H^T H)_D \underline{\theta}^{(k+1)} - (H_D{}^T H_D)^{-1} (H^T H)_{HORS} \underline{\theta}^{(k)} + (H^T H_D)^{-1} H^T Z \; ; \; K \geq 0 \qquad (4\text{-}3\text{-}11)$$

Cet algorithme utilisé efficacement pour l'estimation des paramètres des systèmes de grande dimension, pose des problèmes pour l'étude de la validation des données. Son inconvénient majeur dans ce cas, réside dans les inversions de matrices $H_D{}^T H_D$ ($M_D VM_D{}^T$ en validation des données) qui peuvent être singulières. Nous utilisons donc la méthode

décentralisée de validation de données proposée dans [4-1]. Dans cette méthode, on décompose le réseau initial en N sous-réseaux faiblement couplés. Cette décomposition permet de formuler le problème de validation des données de la façon suivante :

**minimiser**

$$J = \frac{1}{2} \sum_{i=1}^{N} \| \hat{\underline{Q}}_i - \underline{Q}_i \|^2_{V_i^{-1}} + \frac{1}{2} \| \hat{\underline{q}} - \underline{q} \|^2_{V_q^{-1}} \qquad (4\text{-}3\text{-}12)$$

sous la contraintes $M_i \underline{Q}_i + m_i \underline{q} = \underline{0}$ $\quad i=1, N$ $\qquad (4\text{-}3\text{-}13)$

avec $\underline{Q}_i$ : vecteur des débits du sous-système i,

$V_i$ : matrice de variance-covariance des mesures du sous-système i,

$\underline{q}$ : vecteur des débits de couplage entre les sous-sytèmes,

$m_i$ : matrice d'incidence de couplage du sous-système i avec les autres sous-systèmes,

$V_q$ : matrice de variance-covariance des mesures des voies de couplage.

Les conditions d'optimalité de ce problème sont alors :

$$V_i^{-1} (\hat{\underline{Q}}_i - \underline{Q}_i) + M_i^T \underline{\lambda}_i = \underline{0} \qquad i = 1, N \qquad (4\text{-}3\text{-}14)$$

$$V_i^{-1} (\hat{\underline{q}} - \underline{q}) + \sum_{i=1}^{N} m_i^T \underline{\lambda} = \underline{0} \qquad (4\text{-}3\text{-}15)$$



$$M_i \hat{\underline{Q}}_i + m_i \hat{\underline{q}} = \underline{0} \qquad (4\text{-}3\text{-}16)$$

où $\underline{\lambda}_i$ est le vecteur des paramètres de Lagrange du sous-système i.

Des équations (4-3-14), (4-3-15) et (4-3-16), on peut tirer $\lambda_i$ :

$$\underline{\lambda}_i = R_i^{-1} (\underline{r}_i - m_i V_q \sum_{\substack{j=1 \\ j \neq 1}}^{N} m_j^T \underline{\lambda}_j) \qquad i = 1, N \qquad (4\text{-}3\text{-}17)$$

avec $R_i = M_i V_i M_i^T + m_i V_q m_i T \qquad i = 1,N \qquad (4\text{-}3\text{-}18)$

$$\underline{r}_i = M_i \underline{Q}_i + m_i \underline{q} \qquad i = 1.N \qquad (4\text{-}3\text{-}19)$$

La résolution itérative du système (4-2-15) est alors effectuée :

a) *par la méthode de Jacobi* :

$$\underline{\lambda}_i^{(k+1)} = R_i^{-1} (\underline{r}_i - m_i V_q \sum_{\substack{j=1 \\ j \neq 1}}^{N} m_j^T \underline{\lambda}_j^{(k)}) \qquad i=1,N \qquad (4\text{-}3\text{-}20)$$

b) *par la méthode de Gauss-Seidel* :

$$\underline{\lambda}_i^{(k+1)} = R_i^{-1} (\underline{r}_i - m_i V_q \sum_{j=1}^{i-1} m_j^T \underline{\lambda}_j^{(k+1)} + \sum_{j=i+1}^{N} m_j^T \underline{\lambda}_j^{k}) \quad i=1,N \qquad (4\text{-}3\text{-}21)$$

Pour l'implantation parallèle de ces deux algorithmes, nous

étudions le réseau de la figure (4-6) [3-1] décomposé en trois sous-sytèmes.

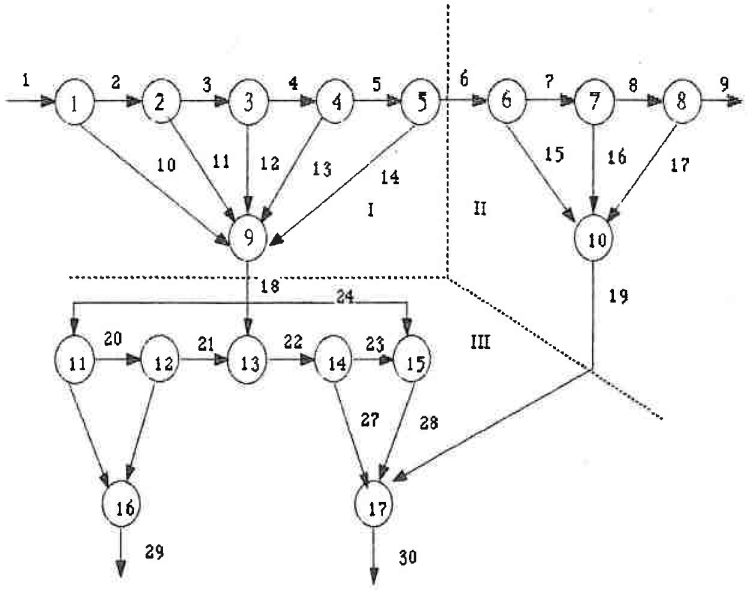



Figure (4-6) : Réseau du transport de matières.

La matrice d'incidence de ce réseau a la structure suivante : Figure (4-7)



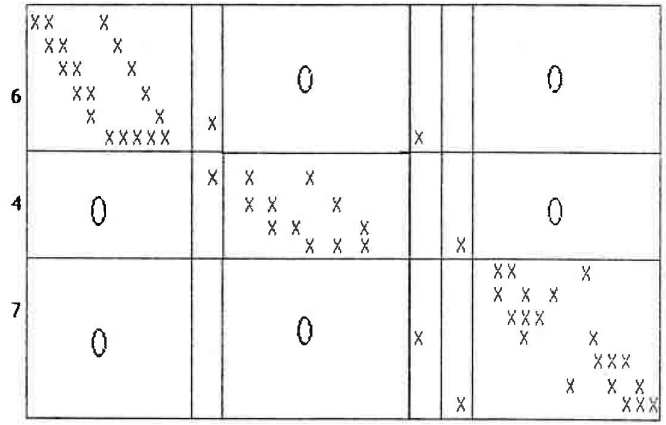



Figure (4-7) Matrice d'incidence du réseau (4-6)

METHODE GLOBALE

La figure (4-8) fournit le graphe pour une méthode globale. Pour $t_{div} = t_{add}$ et $t_{mult} = 5\ t_{add}$, la durée du chemin critique est de 3150 $t_{add}$. Cette durée nous sert pour la comparaison des méthodes globales et itératives.

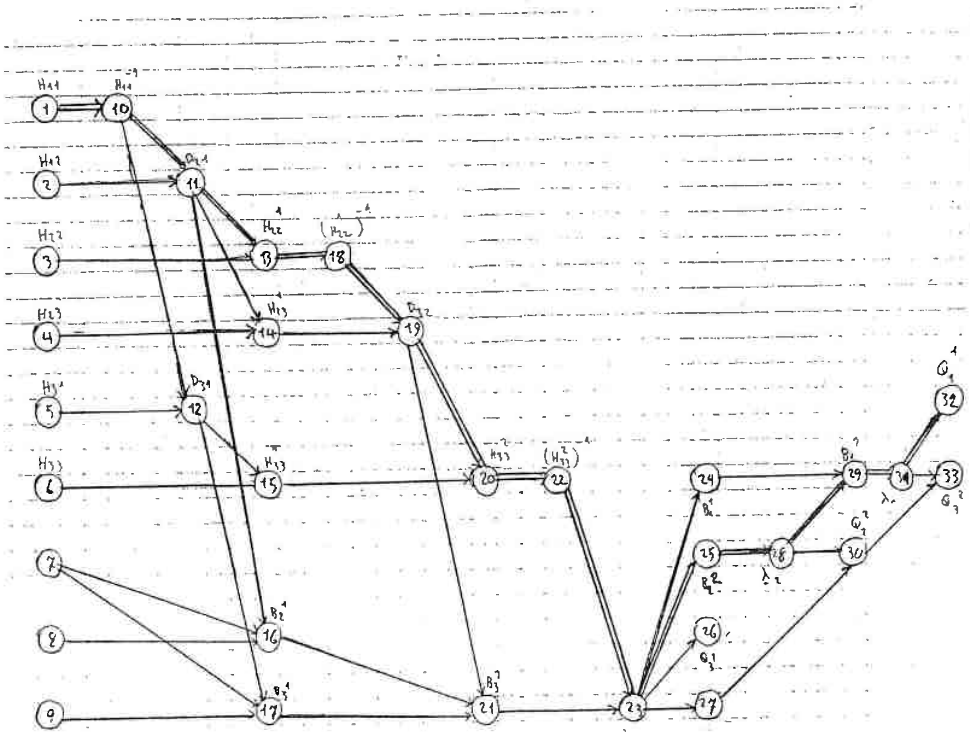


Figure (4-8) Graphe de la méthode globale pour le réseau (4-6)

METHODE DE JACOBI

La figure (4-9) montre la partie d'initialisation de l'algorithme de Jacobi.

Le rôle des différentes tâches est le suivant :



- de 1 à 6 : calcul de $R_i$ et $\underline{r}_i$.
- de 7 à 9 : calcul de $R_i^{-1}$.
- de 10 à 18 : calcul de $R_i^{-1}\,\underline{r}_i$, et $Ri^{-1}\, m_i V_q mj^T$
- de 19 à 20 : calcul de $R_i^{-1}\, m_i m_j^T \underline{\lambda}j$
- de 25 à 27 : calcul de $\underline{\lambda}_i = R_i^{-1} V_i - R_i^{-1} \sum m_i V_q m_j^T \underline{\lambda}_j$

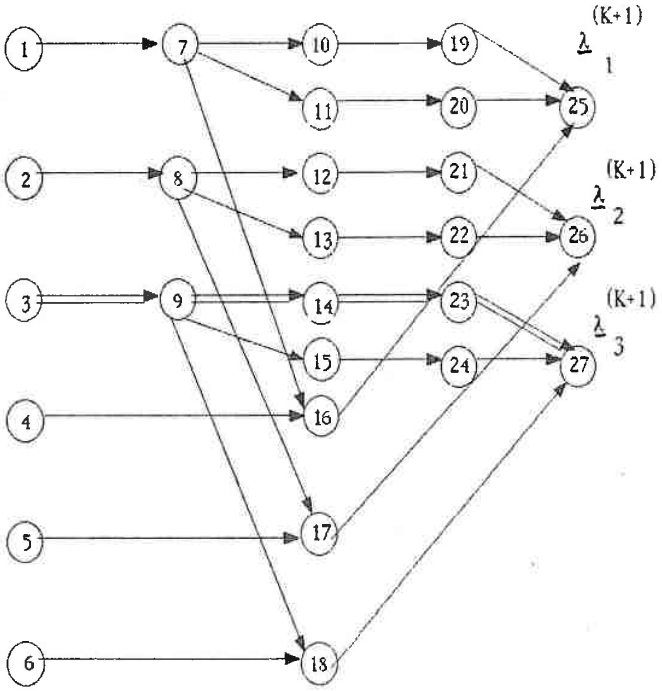



Figure (4-9) : Graphe de la partie d'initialisation pour le réseaux (4-6)

La durée du chemin critique pour ce graphe est de 1443 $t_{add}$ .

La figure (4-10) présente le graphe pour la partie itérative de l'algorithme. En plus de calcul de $\underline{\lambda}_i^{(k+2)}$, ce graphe comporte la partie test de fin de calcul de l'itération (k+2). Ces tâches correspondent aux numéros 3, 6, 9, 11, 13, 15, 18, 20 et 21. La durée du chemin critique pour ce graphe est de 49 $t_{add}$ .

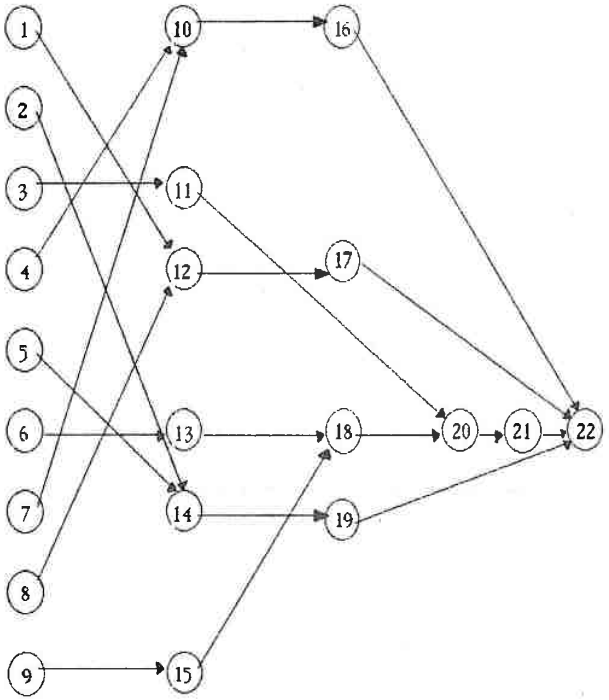



Figure (4-10) Le graphe de la partie itérative de l'algorithme de Jacobi



Si L est le nombre d'itérations nécessaires à la convergence de l'algorithme, la durée minimale d'exécution est :

$$1443+49L$$

Pour que la méthode de Jacobi soit efficace il faut que l'on ait :

$$1433+49L<3150 \quad \text{ou} \quad L<35$$

L'algorithme de Jacobi pour cet algorithme converge après 34 itérations [4-1]. Les deux méthodes globale et de Jacobi ont donc à peu près les mêmes vitesses de calcul parallèle.

ALGORITHME DE GAUSS- SEIDEL

La figure 4-11 présente la partie initialisation jusqu'à la fin du test de la première itération . On peut remarquer que le calcul de $\underline{\lambda}_2^{(k+1)}$ (tâche 28) est retardé jusqu'au calcul de $\underline{\lambda}_1^{(k+1)}$ (la tâche 24) et pour calculer $\underline{\lambda}_3^{(k+1)}$ (la tâche 35) les valeurs de $\underline{\lambda}_1^{(k+1)}$ et $\underline{\lambda}_2^{(k+1)}$ ont été déjà calculées. Pour accélérer l'algorithme on a commencé les calculs de l'itération suivant juste après le calcul de $\underline{\lambda}_2^{(k+1)}$ (les tâches 30,33,36,37,39,40,42,43,45). La durée du chemin critique du graphe 3-11 est 1530 $t_{add}$ .

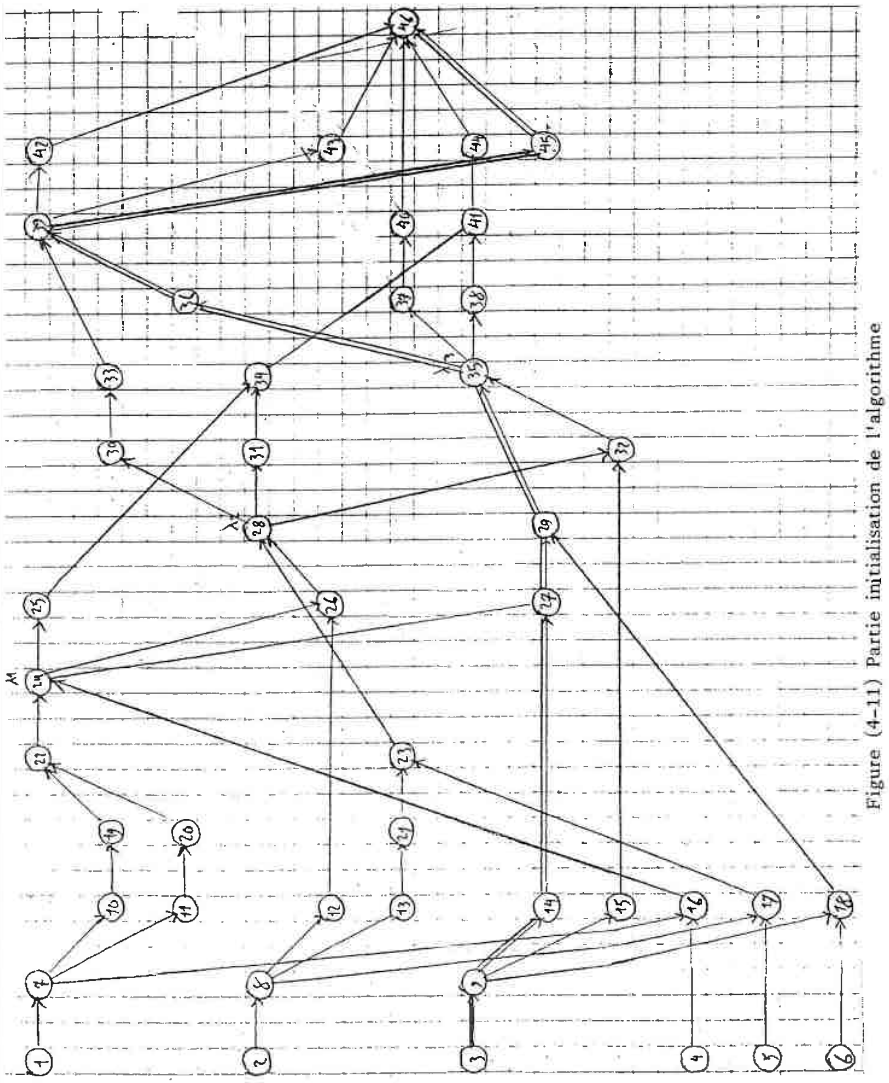


Figure (4-11) Partie initialisation de l'algorithme de Gauss Seidel



La figure 4-12 montre la partie itérative de l'algorithme de Gauss-Seidel. La durée de chemin critique pour ce graphe est de 119 t add .

Alors, si L est le nombre d'itérations nécessaires pour la convergence de l'algorithme, la durée totale d'exécution est :

$$[1530 + 119\,(L-1)]\,t_{add}$$

Pour étudier l'efficacité de l'algorithme par rapport à la méthode globale, il faut que l'on ait :

$$1530 + 119\,(L-1) < 3150 \quad \text{ou} \quad L<15$$

Le nombre d'itération de l'algorithme Gauss-Seidel pour cette exemple est de 20 [4-1]. L'algorithme de Gauss-Seidel est donc moins rapide que la méthode globale pour le calcul parallèle.

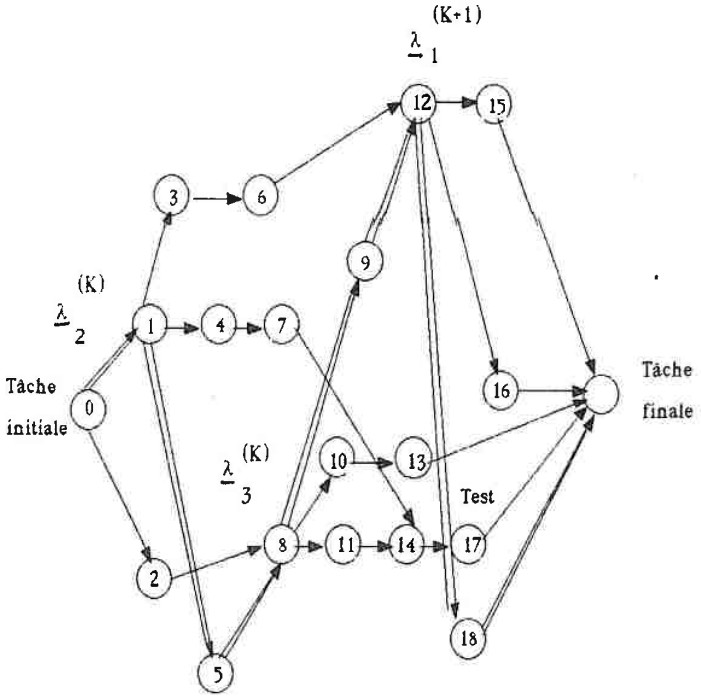



Figure (4-12) Partie itérative de l'algorithme de Gauss-Seidel

## 4-4 ESTIMATION DES PARAMETRES

### 4-4-1 Méthode directe

Considérons le modèle d'observation suivant :

$$Z = H\theta + V \qquad (4\text{-}4\text{-}1)$$

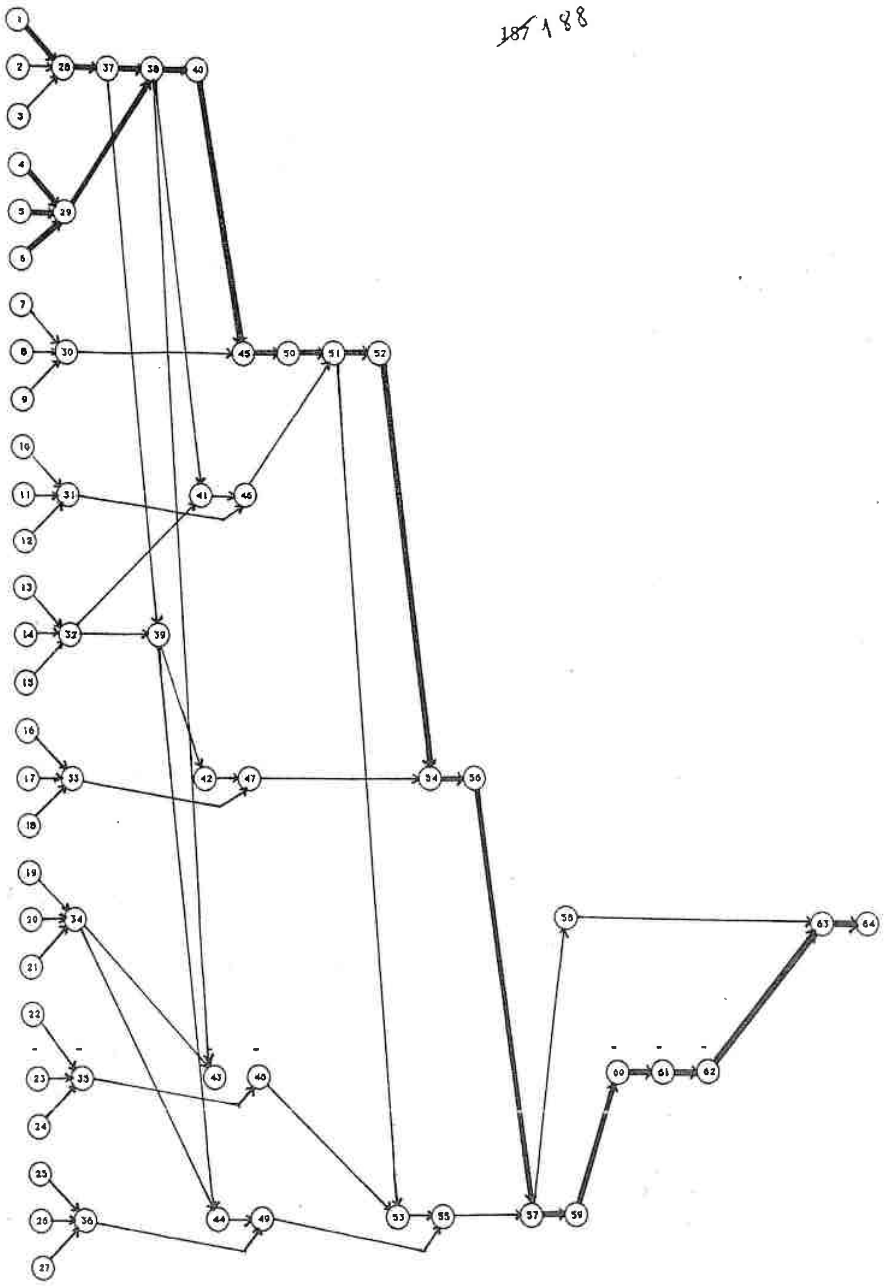


Figure (4-13) Graphe de l'algorithme des moindres carrés



Si N est le nombre de sous-systèmes et si les blocs ont tous les mêmes dimensions d.v avec d>>v, alors le chemin critique du graphe 4-13 est composé de :

1 mutiplication de deux matrices v.d et d.v

$Log_2N$ additions de deux matrices v.v

N-1 multiplication de deux matrices vxv

N-1 inversion de matrices symètriques vxv

N-1 multiplication de deux matrices vxv dont le résultat est symètrique

N-1 addition des matrices v.v symètriques

2N-1 multiplication de deux matrices vxv et vxr

2N-3 addition de matrices vxr

Pour d=25, v=4, r=1,$t_{div}=6\ t_{add}$ et $t_{mult}=5\ t_{add}$, la durée du chemin critique est de 4582 $t_{add}$.

Le tableau 4-2 présente les résultats obtenus par l'ordonnancement du calcul, en fonction du nombre de processeurs.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 14 | 4582 | 9,95 | 0,71 |
| 12 | 4809 | 9,48 | 0,79 |
| 10 | 5462 | 8,35 | 0,83 |
| 8 | 6150 | 7,41 | 0,93 |
| 6 | 7835 | 5,82 | 0,97 |
| 4 | 11417 | 3,99 | 1 |
| 3 | 15197 | 3 | 1 |

Tableau 4-2 Temps d'exécution pour l'algorithme des moindres carrés dans le cas de blocs de mêmes dimensions

Le tableau 4-3 donne les mêmes résultats pour $d_1$=20, $v_1$=3, $d_2$=25, $v_2$=4 et $d_3$=30, $v_3$=5. La durée du chemin critique dans ce cas est de 5388 $t_{add}$.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 12 | 5388 | 8,46 | 0,71 |
| 10 | 5586 | 8,16 | 0,82 |
| 8 | 6295 | 7,24 | 0,91 |
| 6 | 7977 | 5,72 | 0,95 |
| 4 | 11438 | 3,99 | 1 |
| 3 | 15094 | 3 | 1 |

Tableau 4-3 Temps d'exécution pour l'algorithme des moindres carrés dans le cas de blocs de dimensions différentes

### 4-4-2 Méthode Itérative

Pour la résolution iterative de l'équation 4-4-2, nous utilisons la méthode proposée dans [4-4], que nous avons étudiée en 4-3-2.

Pour simplifier la notation, posons

$$T = [I_n + (H_D{}^T H_D)^{-1}(H^T H)_D]^{-1} [I_n - (H_D{}^T H_D)^{-1} (H^T H)_{hors}]$$



où $Z \in R^{mxr}$ : matrice des mesures

$\theta \in R^{nxr}$ : matrice des paramètres constantes à estimer

$H \in R^{mxn}$ : matrice de relation entre les paramètres et les mesures.

$V \in R^{mxr} = [v_1, v_2, \ldots\ldots v_r]$ : matrices représentant le bruit et/ou l'incertitude des mesures, composée de vecteurs aléatoires normaux indépendants, de moyennes nulles et de variances $Z\ I_m$

La solution à partir de la méthode des moindres carrés du systéme 4-3-1 est donnée par le système

$$(H^T H)\hat{\theta} = H^T Z \qquad (4\text{-}4\text{-}2)$$

où $\hat{\theta}$ est la matrice des meilleurs estimateurs des paramètres $\theta$.

Le système (4-4-2) est à peu près de la même forme que celui de l'équation 4-3-7 obtenu pour la validation des données. La différence réside dans l'absence de la matrice V dans 4-4-2, ce qui simplifie la solution du problème dans ce cas.

La figure 4-13 présente le graphe correspondant à la résolution du système (4-3-2), dans le cas où la matrice H, de dimension 75x12, est décomposée en blocs de dimensions 25x4.

$$t = [I_n + (H_D^T H_D)^{-1} (H^T H)_D]^{-1} (H^T H_D)^{-1} H^T Z$$

alors l'équation 4-3-11 s'écrit sous la forme

$$\hat{\theta}^{(k+1)} = T \hat{\theta}^{(k)} + t ; \quad k \geq 0 \qquad (4\text{-}4\text{-}3)$$

et si on utilise un paramètre de relaxation on a :

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \Omega (T \hat{\theta}^{(k)} + t - \hat{\theta}^{(k)}) ; \quad k \geq 0 \qquad (4\text{-}4\text{-}4)$$

où $\Omega = \omega I_n$ est le paramètre de relaxation.

On peut écrire alors

$$\hat{\theta}^{(k+1)} = T_\omega \hat{\theta}^{(k)} + t_\omega \qquad k \geq 0 \qquad (4\text{-}4\text{-}5)$$

où $T_\omega = I_n + \Omega (T - I_n) ; \; t_\omega = \Omega t$

Si on décompose les matrices Z et θ de la façon suivante :

$$Z = [Z_1^T, Z_2^T, \ldots\ldots\ldots, Z_N^T]$$

$$\theta = [\theta_1^T, \theta_2^T, \ldots\ldots\ldots, \theta_N^T]$$

où

$Z_i \in R^{mi \times r}$ $\qquad \theta_i \in R^{n_i \cdot r}$ $\qquad i = 1,2, \ldots\ldots, N$



$$\sum_{i=1}^{N} m_i = m \qquad \sum_{i=1}^{N} n_i = n$$

l'équation 4-4-4 s'écrit sous la forme :

$$\hat{\theta}_i^{(k+1)} = \hat{\theta}_i^{(k)} - \Omega_i [I_{ni} + D_{ii}^{-1} (H^T H)_{ii}]^{-1} D_{ii}^{-1} [\sum_{j=1}^{N} (H^T H)_{ij} \theta_j^{(k)} - \sum_{j=1}^{N} H_{ji}^T Z_j] \qquad (4\text{-}4\text{-}6)$$

où $\Omega_i = \omega I_{ni}$, $I_{ni}$ est la matrice d'identité de dimension $n_i$ et $D_{ii} = H_{ii}^T H_{ii}$

Posons :

$$M_{ij} = (H^T H)_{ij}, \; U_i = \Omega_i [I_{ni} + D_{ii}^{-1} (H^T H)_{ii}]^{-1} D_{ii}^{-1} ; \; q_i = \sum H_{ji}^T Z_j$$

$$i \text{ et } j = 1 \text{ à } N$$

on a alors :

$$\hat{\theta}_i^{(K+1)} = \hat{\theta}_i^{(K)} - U_i M_{i1} \hat{\theta}_1^{(K)} - U_i M_{i2} \hat{\theta}_2^{(K)} - \cdots\cdots\cdots - U_i M_{iN} \hat{\theta}_N^{(K)} + U_i q_i \qquad (4\text{-}4\text{-}7)$$

On calcule donc hors itération les matrices $u_i M_{i1}$, $u_i M_{i2}$, ....., $u_i M_{iN}$ et $u_i q_i$.

Chaque itération comprend alors :

N multiplications de deux matrices.

N additions de deux matrices.

Pour le calcul parallèle de cet algorithme, nous utilisons les méthodes présentées précedemment pour la multiplication des matrices et la résolution itératives des systèmes d'équations linéaires. La figure 4-14 présente le graphe correspondant à la première itération de cet algorithme. La signification des différentes tâches du graphe 4-14 est la suivante :

| | |
|---|---|
| 1 à 18 | $H_{Ki}^T H_{Kj}$ $i = 1$ à $N$ $K = 1$ à $i$ |
| 19 à 27 | $H_{ij}^T Z_j$ $i,j = 1$ à $N$ |
| 28,31,35 | $D_{ii}^{-1}$ $i = 1$ à $N$ |
| 29,30,32,33,34,36 | $M_{ij} = \sum_j H_{Ki}^T H_{Kj}$ $i = 1$ à $N$ $j = 1$ à $N$ |
| 37,38,39 | $\sum_j H_{ij}^T Z_j$ $i = 1$ à $N$ |
| 40,42,45 | $\Omega_i D_{ii}^{-1}$ $i = 1$ à $N$ |
| 41,43,44 | $Q_i = I_{ni} + D_{ii}^{-1} M_{ii}$ $i = 1$ à $N$ |
| 46,47,48 | $Q_i^{-1}$ $i = 1$ à $N$ |
| 49,50,51 | $u_{ii} = Q_i^{-1} \Omega_i D_{ii}^{-1}$ $i = 1$ à $N$ |
| 52 à 60 | $A_{ij} = u_{ii} M_{ij}$ $i, j = 1$ à $N$ |
| 61 à 63 | $q_i = u_{ii} \sum_j H_{ij}^T Z_j$ $i = 1$ à $N$ |
| 64 à 71 | $A_{ij} \theta_j^{(0)}$ $i, j = 1$ à $N$ |
| 72,73,74 | $\theta_i^{(1)} = \sum_j A_{ij} \theta_j^{(0)} + q_i$ $i, j = 1$ à $N$ |

Le chemin critique du graphe 4-14 dans le cas général est composé de :



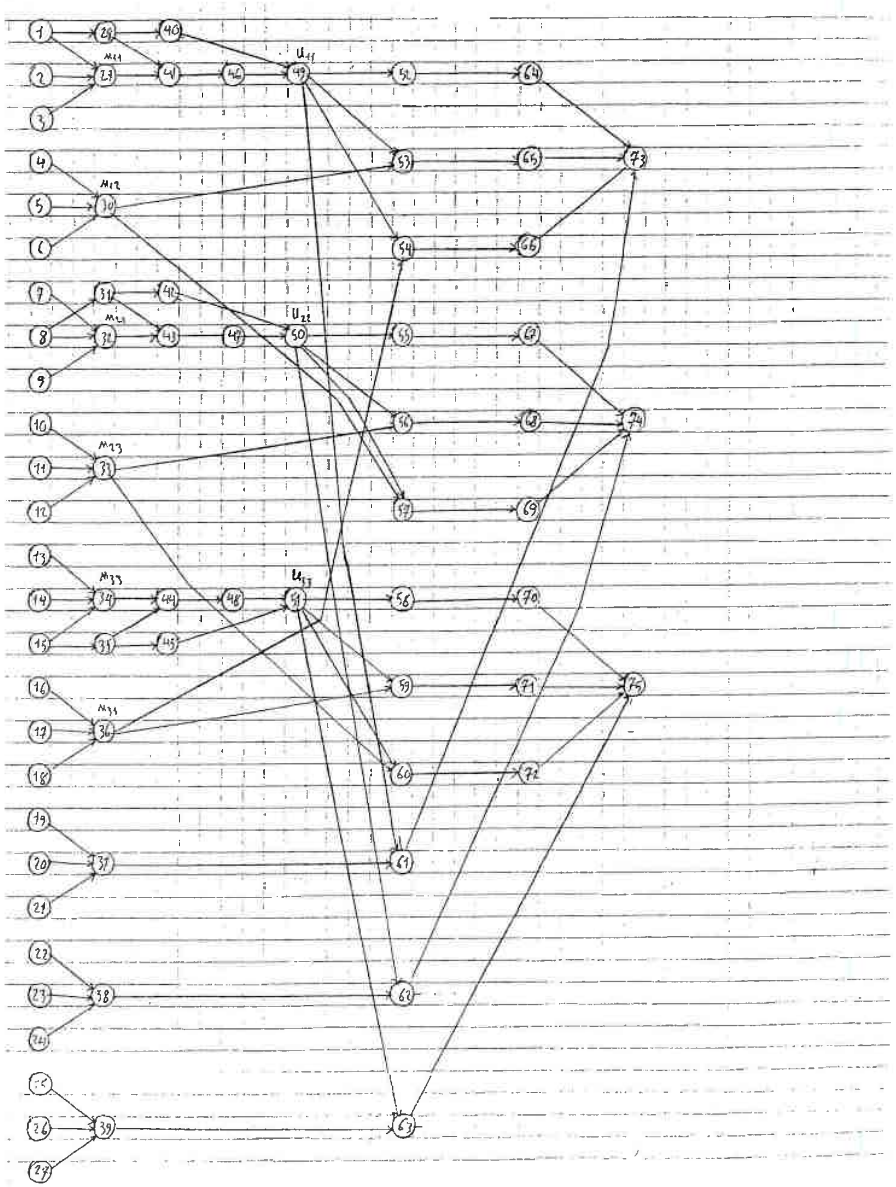


Figure (4-14) Graphe de la première itération de l'algorithme (4-4-7)

1 multiplication de deux matrices v . d et d . v dont le resultat est symètrique.

1 inversion de matrices symètriques v . v

1 inversion de matrice v . v

3 multiplications de deux matrices v . v
1 addition de deux matrices v . v

1 multiplication de deux matrices v . v et v . r

$1+\log_2 N$ additions de deux matrices v . r

où N est le nombre de décompositions, d et r sont les dimensions des sous-matrices z et v et r sont les dimensions des sous-matrices $\theta_j$.

La figure 4-15 présente le graphe correspondant à l'itération k (k>1) de l'algorithme 4-4-7,



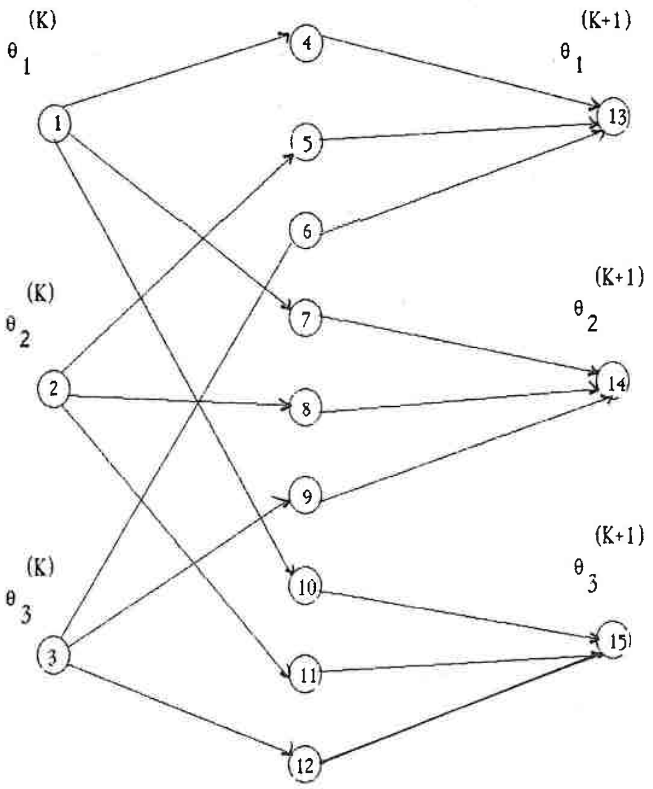



Figure (4-15) Graphe de l'itération k de l'algorithme 4-4-7

Le chemin critique du graphe 4-4-7 est composé de :

1 multiplication de deux matrices v . v

$\text{Log}_2 N$ +1 additions de deux matrices v . r

Pour d=25, v=4, r=1, $t_{div}$ = 6 et $t_{add}$ = 5, la durée du chemin critique du graphe 4-4-7 est de 100. Si L est le nombre d'itérations

nécessaires à la convergence de l'algorithme 4-4-7 , la durée totale du chemin critique est :

$$3318 + (L-1)100$$

Pour que l'algorithme itérative soit plus rapide que la méthode directe il faut que l'on ait :

$$3318+(L-1)100<4582, \text{ soit } L <-13$$

On peut trouver des relation semblables entre le nombre d'itération L et le nombre de décompositions N en connaissant les dimensions des sous-matrices θ. Par exemple, pour d = 20, v = 4, r = 1, $t_{div} = 6t_{add}$, $t_{mult} = 5t_{add}$, la durée du chemin critique du graphe 4-13 est

$$(1441+1039N+16 \log_2 N) T_{add}$$

et la durée du chemin critique de l'algorithme itératif est :

$$(3318+(L-1)(92+4(1+\log_2 N))) t_{add}$$

Pour que l'algorithme itératif soit plus rapide que l'algorithme direct, il faut que l'on ait :

$$3318 + (L-1) (92 + 4(1 + \log N)) <1441 + 1039N +16 \log N$$



ou encore

$$L< 1039 N + 20 \log_2 N + 1781 / 96 + 4 \log_2 N \qquad (4\text{-}4\text{-}8)$$

Les tableaux 4-4 et 4-5 donnent respectivement le temps d'exécution par ordonnancement de la première itération, représentée par le graphe 4-14 et celui des itérations suivantes resprésentées par le graphes 4-15.

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 18 | 3318 | 14,8 | 0,82 |
| 14 | 4374 | 11,2 | 0,8 |
| 12 | 4374 | 11,2 | 0,94 |
| 10 | 4944 | 9,93 | 0,99 |
| 8 | 6758 | 7,26 | 0,91 |
| 6 | 8208 | 5,98 | 1 |
| 4 | 12302 | 3,99 | 1 |
| 3 | 16360 | 3 | 1 |

Tableau 4-4 Temps d'exécution pour la première itération de l'algorithme 4-4-7

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 10 | 100 | 8,52 | 0,85 |
| 9 | 100 | 8,52 | 0,95 |
| 8 | 192 | 4,44 | 0,55 |
| 6 | 192 | 4,44 | 0,74 |
| 4 | 284 | 3 | 0,75 |
| 3 | 284 | 3 | 1 |

Tableau 4-5 Temps d'exécution de l'itération k(k>1) de l'algorihtme 4-4-7

La comparaison des deux tableaux (4-4, 4-5) et (4-3) donne le nombre d'itérations maximum de l'algorithme itératif en fonction du nombre de processeurs. Ainsi par exemple, pour un nombre de processeurs égal à 18 :

$$3318+(L-1)100<4582 \quad \text{ou} \quad L<=13$$

Pour un nombre de processeurs égal à 10 :

$$4944+(L-1)100<5462 \quad \text{soit } L<-6$$

Pour un nombre de processeurs égal à 6, la durée d'exécution de la première itération de l'algorithme 4-4-7 dépasse la durée de l'algorithme global. On peut constater ainsi que, pour cet exemple, l'algorithme global est plus rapide que l'algorithme itératif.

### 4-4-3 Algorithme récurssif

Considérons l'équation matricielle suivante :

$$Z = H\,\theta + e \qquad (4\text{-}4\text{-}9)$$

Si N observations de la matrice des mesures Z sont disponibles, on peut estimer la matrice $\theta_N$ avec une des méthodes précédentes. L'algorithme récursif des moindres carrés nous permet alors de calculer les paramètres d'une façon récurssive, dès que de nouvelles mesures sont disponibles. Supposons les relations suivantes :



$$\begin{bmatrix} Z_N \\ z_{N+1} \end{bmatrix} = \begin{bmatrix} H_N \\ h^T_{N+1} \end{bmatrix} \theta_{N+1} + \begin{bmatrix} E_N \\ e_{N+1} \end{bmatrix} \qquad (4\text{-}4\text{-}10)$$

On a alors [4-7]

$$\theta_{N+1} = (H_N{}^T H_N + h_{N+1} h_{N+1}{}^T)^{-1} (H^T Z_N + h_{N+1} z_{N+1}) \qquad (4\text{-}4\text{-}11)$$

qui après quelques manipulations aboutit à l'algorithme suivant :

$$P_N = (H^T H)^{-1} \; ; P_{N+1} = (H_N{}^T H_N + h_{N+1} h_{N+1}{}^T)^{-1}$$

$$K_N = P_N h_{N+1} (1 + h_{N+1}{}^T P_N h_{N+1})^{-1} \qquad (4\text{-}4\text{-}12)$$

$$\hat{\theta}_{N+1} = \hat{\theta}_N + K_N (z_{N+1} - h^T_{N+1} \theta_N). \qquad (4\text{-}4\text{-}13)$$

$$P_{N+1} = P_N - P_N h_{N+1} (1 + h_{N+1}{}^T P_N h_{N+1})^{-1} h_{N+1} P_N \qquad (4\text{-}4\text{-}14)$$

On peut partitionner alors $\theta$ en M . R sous-matrices et utiliser les méthodes présentées précédemment pour le calcul parallèle de l'algorithme. La figure 4-16 présente le graphe correspondant à cet algorithme pour M=3 et R=2. La signification des différentes tâches de ce graphe est le suivante :

1 à 9 $\qquad (P_N)_{ij} (h_N)_j \quad i = 1 \text{ à } M \quad j = 1 \text{ à } M$

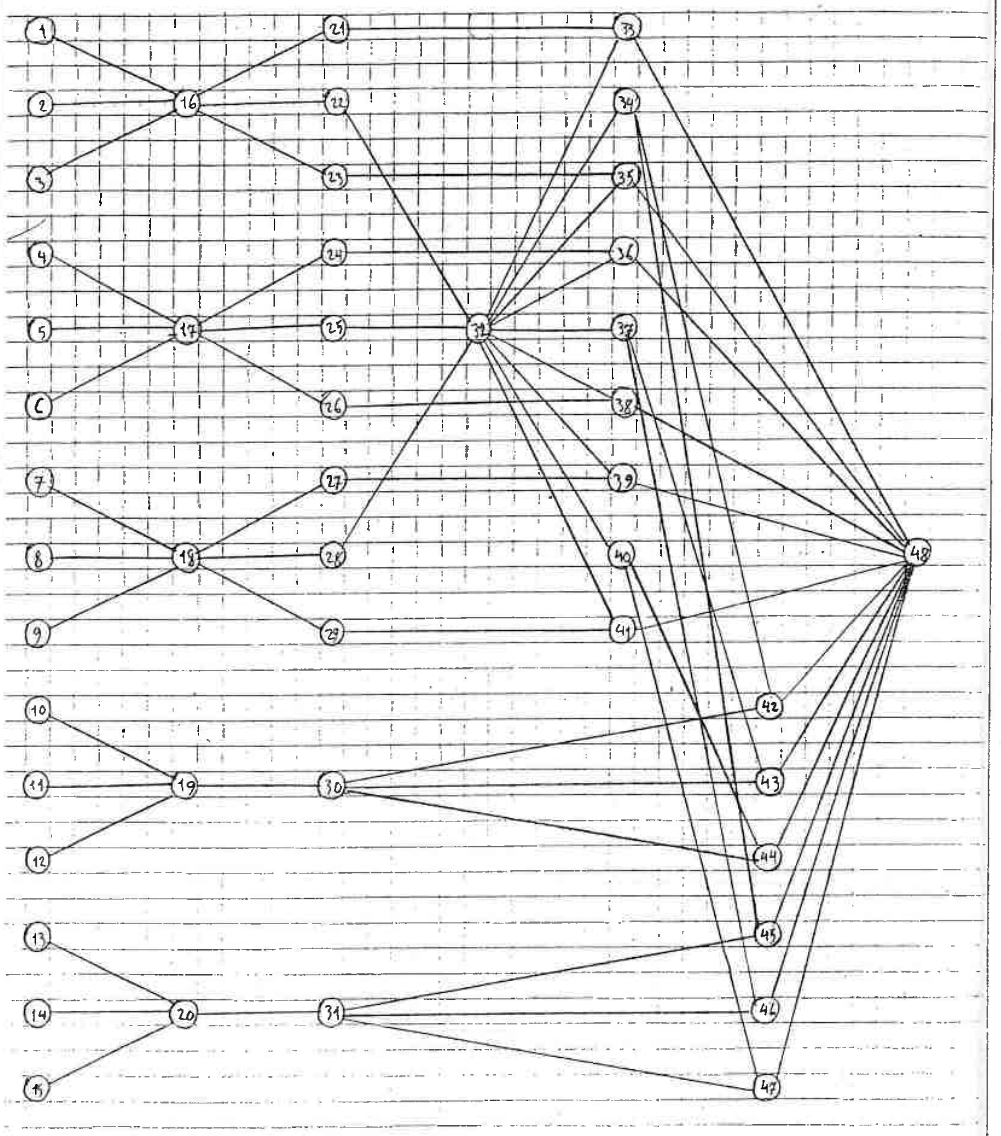
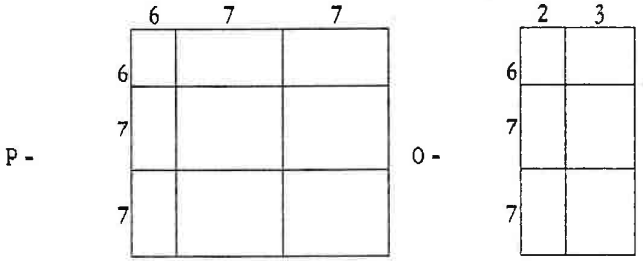


Figure (4-16) Graphe de l'algorithme récursif des moindres carrés



| | |
|---|---|
| 10 à 15 | $(h_N^T)_i\,\theta_{ij}$ i = 1 à M j = 1 à R |
| 16 à 18 | $L_i = \sum_{j=1}^{M} (P_N)_{ij}\,(h_N)_j$ i = 1 à M |
| 19,20 | $y_i = \sum_{j=1}^{M} (h_N^T)_i\,\theta_{ij}$ j = 1 à R |
| 21,23,24,26,27,29 | $L_i\,L_j^T$ i = 1 à M j = i à M |
| 22,26,28 | $h_i^T\,L_i$ i = 1 à M |
| 30,31 | $Z_j = z_j - y_j$ |
| 32 | $(1 + h_{N+1}^T\,P_N\,h_{N+1})^{-1}$ |
| 33,35,36,38,39,41 | $(P_{N+1})_{ij}$ i = 1 à M j = i à M |
| 34,37,40 | $(K_N)_i$ i = 1 à M |
| 42 à 47 | $(\theta_{N+1})_{ij}$ i = 1 à M j = 1 à R |
| 48 tâche finale | |

Considérons une matrice de paramètres de dimensions 20.5 décomposée de la façon suivante :




dans le cas où $t_{mult} = 5\ t_{add}$, la durée du chemin critique du graphe 4-16 est de 840 $t_{add}$. Cette durée est réalisable sur au maximum 15 processeurs. Par ordonnancement du calcul, on peut réduire à 10 le nombre de processeurs pour exécuter l'algorithme dans le temps du chemin critique. Le Tableau 4-6 donne le temps d'exécution de l'algorithme par ordonnancement en fonction du nombre de processeurs.

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 10 | 840 | 8,03 | 0,8 |
| 8 | 929 | 7,25 | 0,91 |
| 6 | 1133 | 5,95 | 0,99 |
| 4 | 1720 | 3,92 | 0,98 |
| 3 | 2251 | 2,99 | 1 |
| 2 | 3372 | 2 | 1 |

Figure (4-6) Temps d'exécution pour l'algorithme récursif des moindres carrés



## 4-5 FILTRE DE KALMAN

### 4-5-1 Filtre de Kalmann global

Considérons un système linéaire, sous forme discrète défini par:

$$\underline{x}(K+1) = A(K)\,\underline{x}(K) + \underline{w}(K)$$

$$\underline{Z}(K) = H(K)\,\underline{x}(K) + \underline{v}(K) \qquad (4\text{-}5\text{-}1)$$

où $\underline{x}(k+1)$ est le vecteur d'état de dimension n,

$\underline{z}(k)$ est le vecteur d'observation de dimension m,

k est l'instant d'échantillonage,

A (k) et H (k) sont des matrices d'élément connus,

$\underline{w}$ (k) et $\underline{v}$ (k) sont des bruits blancs Gaussiens, non correlés, de moyennes nulles et de matrices de variance-covariance respectives Q et R

Le problème de filtre de Kalman est d'estimer le vecteur d'état tout en minimisant le critère quadriatique de l'erreur : [4-8]

$$E\left[(\underline{x}(k) - \hat{\underline{x}}(k))^T S\,(\underline{x}(k) - \hat{\underline{x}}(k)\right]$$

où S est une matrice symétrique définie positive

et $\underline{x}(k)$ est le meilleur estimateur de $\underline{x}(k)$.

Les équations de filtre de Kalman sont définie alors comme suit : [4-9]) ,[4-10], [4-11] :

- prédiction à un pas :

$$\hat{\underline{x}}(K \mid K-1) = A(K-1)\,\hat{\underline{x}}(K-1 \mid K-1) \qquad (4\text{-}5\text{-}2)$$

$$\hat{\underline{P}}(K \mid K-1) = A(K-1)\,\hat{\underline{P}}(K-1 \mid K-1)\,A^{T}(K-1) + Q \qquad (4\text{-}5\text{-}3)$$

où $\hat{\underline{x}}(K-1 \mid K-1)$ est le meilleur estimateur de $\hat{\underline{x}}(K-1)$, avec $\underline{Z}(K-1)$

$\underline{P}(K-1 \mid K-1)$ et $\underline{P}(K \mid K-1)$ sont les covariences de $[\hat{\underline{x}}(K-1) - \underline{x}(K-1 \mid K-1)]$ et $[\hat{\underline{x}}(K) - \underline{x}(K \mid K-1)]$ respectivement.

Quand la valeur de Z (K) est acquise, il faut modifier ces valeurs. On a alors pour les prévisions à un pas :

$$\hat{\underline{x}}(K \mid K) = \hat{\underline{x}}(K \mid K-1) + K(K)\,[\underline{Z}(K) - H(K)\,\hat{\underline{x}}(K \mid K-1)] \qquad (4\text{-}5\text{-}4)$$

$$\underline{P}(K \mid K) = (I - K(K)\,h(K))\,\underline{P}(K \mid K-1) \qquad (4\text{-}5\text{-}5)$$

$$K(K) = P(K \mid K-1)\,H^{T}(K)\,[H(K)\,P(K \mid K-1)\,H^{T}(K) + R]^{-1} \qquad (4\text{-}5\text{-}6)$$

Pour l'implantation parallèle du filtre de Kalman, nous décomposons le vecteur $\underline{x}$ en N vecteurs de dimension $n_j$ (j=1 ...., N) de la facon suivante :



$$\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \\ \underline{x}_N \end{bmatrix}_{(K+1)} = \begin{bmatrix} & & & \\ & A(K) & & \\ & & & \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \\ \underline{x}_N \end{bmatrix}_{(K)} + \begin{bmatrix} \underline{w}_1 \\ \underline{w}_1 \\ \\ \underline{w}_N \end{bmatrix}_{(K)}$$

$$\begin{bmatrix} \underline{Z}_1 \\ \underline{Z}_2 \\ \\ \underline{Z}_N \end{bmatrix}_{(K+1)} = \begin{bmatrix} & & & \\ & H(K) & & \\ & & & \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \\ \underline{x}_N \end{bmatrix}_{(K)} + \begin{bmatrix} \underline{v}_1 \\ \underline{v}_1 \\ \\ \underline{v}_N \end{bmatrix}_{(K)}$$

Les équations 4-4-2 et 4-4-3 sont alors de la forme :

$$\underline{x}_i(K \mid K-1) = \sum_{j=1}^{N} A_{ij}(K-1)\,\underline{x}_j(K-1 \mid K-1) \quad i = 1 \text{ à } N \qquad (4\text{-}5\text{-}7)$$

$$P_{ij}(K \mid K-1) = \sum_{f=1}^{N} A_{ir} \sum \{ P_{rm}(K-1 \mid K-1)\,A_{jm}{}^{T}) + Q_{ij} \; i, j = 1 \text{ à } N \qquad (4\text{-}5\text{-}8)$$

et elles peuvent alors être résolues en parallèle.

Pour les équations 4-4-4 à 4-4-6, la même décomposition donne la possibilité de calculer parallèlement les sous-matrices. En effet, le calcul de K(k) est composé de multiplications de matrices et de l'inversion d'une matrices symétrique : nous avons déjà étudié les méthodes de calcul

parallèle de ces operations.

La figure 4-17 présente le graphe associé, pour N=2 et pour la partie correspondant au calcul de P (K | K).

Sur cette figure on peut distinguer les calculs suivants :

- tâches 1 à 21 : calcul de P ((K+1 | K) = A (K) P (K | K) $A^T$ (P (K),

- les tâches 19, 20, 21 correspondent respectivement aux sous-matrices de covariance $P_{11}$ (K +1 | K), $P_{12}$ (K+1 | K) et $P_{22}$ (K+1 | K). La matrice P(K+1 | K) étant symétrique, le calcul de $P_{21}$ (K+1 | K) n'est pas nécessaire.

- tâches 22 à 33 : calcul de M (K) = P (K | K-1).

On doit stocker les sous-matrices $M_{ij}$ (K) dans la mémoire pour la suite des calculs.

- tâches 39 à 42 : calcul de H (K) M (K),

Là encore, la matrice résultante est symétrique. On calcule dont les blocs triangulaires supérieurs.

- tâches 43 à 45 : calcul de u (K) : H (K) M (K) + R,



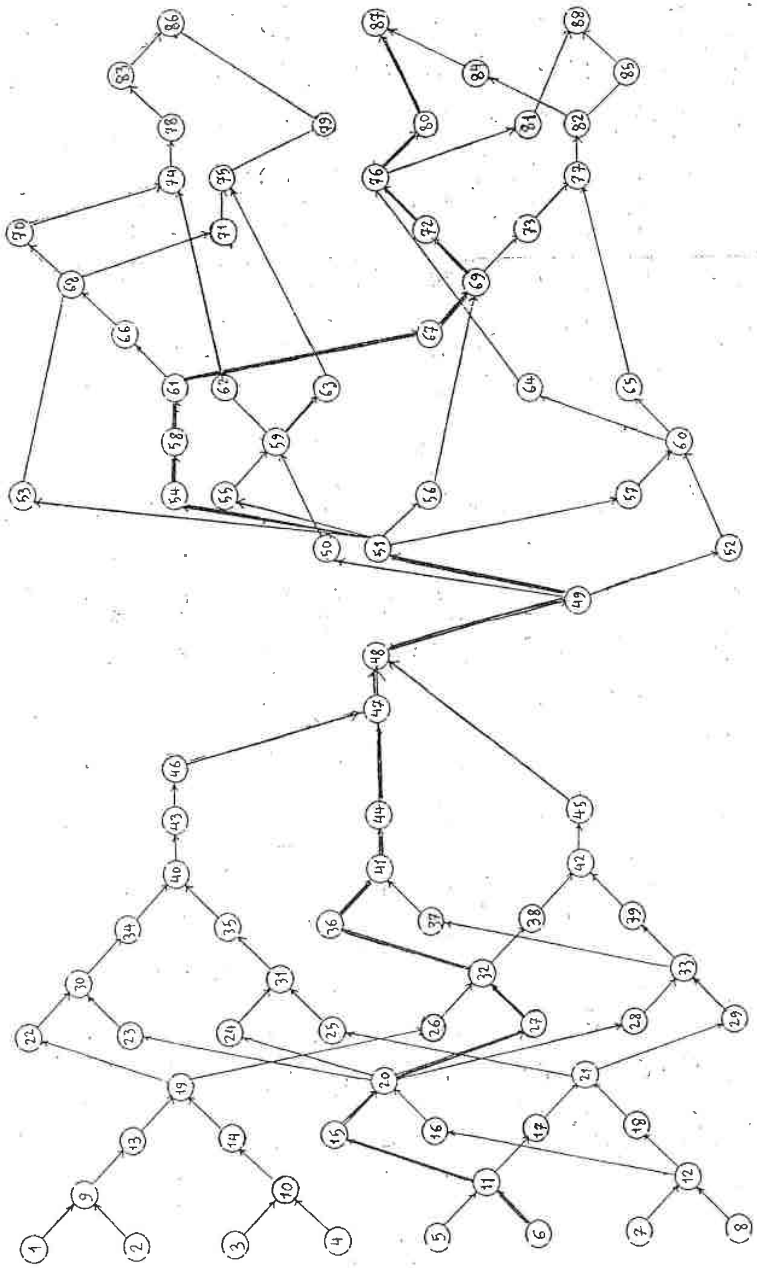


Figure (4-17) Graphe du Filtre de Kalman décentralisé pour le système 4-4-25

- tâches 46 à 49, 51, 54, 58, 61 : calcul de $[u(K)]^{-1}$,

- tâches 50, 55, 59, 53, 66, 68, 57, 52, 60, 56, 67, 69 : calcul de $K(k) = M(K)[u(K)]^{-1}$

- tâches 62, 63, 64, 65, 70, 71, 72, 73, 74, 75, 76, 77, 78, 82 : calcul de $I - K(K) H(K)$

- dernières tâches : calcul de $\underline{P}(K|K) - (I - K(K) H(K)) \underline{P}(K|K-1)$.

On peut remarquer que, pour accélérer le calcul, nous avons effectué le chevauchement du calcul de $[u(K)]^{-1}$, $K(K)$ et $I - K(K) H(K)$. A chaque fois qu'une sous-matrice est calculée, nous l'avons utilisé pour le calcul suivant.

Dans le cas où les blocs de A, P, H ont les mêmes dimensions v, le chemin critique du graphe 4-7 est composé de :

- $3N + 3$ multiplications de deux matrices v.v,
- N inversions de matrices symétriques v.v,
- $(N+1)$ multiplications de deux matrices v.v dont le résultat est symétrique,
- $(N+2)$ additions de deux matrices v.v dont le résultat est symétrique,
- $5 \log_2 N + (2N - 4)$ additions de deux matrices v.v.

Le nombre total d'opérations pour le calcul d'un filtre de Kalman



est de :

- N inversions de matrices symétriques v.v,
- $(6N^3 - 2N^2)$ multiplications de deux matrices v.v,
- $(N-1)(6N^2 - 4N)$ additions de deux matrices v.v,
- $(4N^2 - 1)$ multiplications de deux matrices v.v dont le résultat est symétrique,
- $(5N^2 - 3N)$ additions de deux matrices v.v dont le résultat est symétrique,
- $3N^2$ multiplications d'une matrice v.v par un vecteur de dimension v,
- $3N^2 - N$ additions de deux vecteurs de dimension v.

Connaissant le nombre total d'opérations et la durée du chemin critique, on peut calculer le taux d'accélération pour N et v donnés.

Pour N et v d'ordre élevé, on peut se contenter de prendre en compte uniquement le nombre de multiplications dans le nombre total d'opérations ; mais pour l'étude du chemin critique, il faut tenir compte du nombre d'inversions des matrices.

Pour v suffisamment grand, on peut écrire :

- temps d'inversion d'une matrice symétrique : $\frac{1}{2}$ (temps de multiplication de deux matrices),
- temps de multiplication de deux matrices dont le résultat est symétrique = $\frac{1}{2}$ (temps de multiplication de deux matrices).

Le taux d'accélération asymptotique pour le filtre de Kalman est donc de :

$$\tau_{accel} = \frac{6\,N^3 - 2\,N^2 + 2\,N^2}{3\,N + N/2 + N/2} = 1{,}5\,N^2 \qquad (4\text{-}5\text{-}9)$$

### 4-4-2 Filtre de Kalman décentralisé [4-12]

Pour les systèmes interconnectés, des capteurs implantés sur chaque sous-système mesurent les variables qui lui sont propres. Dans ce cas, la matrice H des équations 4-4-1 est sous forme de bloc diagonal. On peut donc écrire les équations 4-4-1 sous la forme suivante :

$$\underline{x}_j(K+1) = A_{jj}(K)\,\underline{x}_j(K) + \sum_{\substack{m=1 \\ m\neq j}}^{N} A_{jm}(K) + \underline{x}_m(K) + \underline{w}_j(K) \qquad j = 1 \text{ à } N$$

$$\underline{z}_j(K+1) = H_j(K)\,\underline{x}_j(K) + \underline{v}_j(K) \qquad j = 1 \text{ à } N \qquad (4\text{-}5\text{-}10)$$

où $\underline{x}_j$ est le vecteur d'état du sous-système j de dimension $n_j$,

$\underline{z}_j$ est le vecteur d'observation de dimension $v_j$,

$\underline{w}_j(K)$ et $\underline{v}_j(K)$ sont des bruits blancs gaussiens non correlés de moyennes nulles et de covariances respectives $Q_j$ et $R_j$.

Supposons :

$$\tilde{Z}_1(K+1 \mid K) = Z_1 - E[Z_1(K+1) \mid Z(K)]$$



$$Z_2(K+1 \mid K) = Z_2 - E[Z_2(K+1) \mid Z(K), Z_1(K+1)]$$

$$\tilde{Z}^{N-1}{}_N(K+1 \mid K+1) = Z_N - E[Z_N(K+1) \mid Z(K), Z_1(K+1), \ldots\ldots Z_{N-1}(K+1)]$$

Nous pouvons décomposer l'espace H* de la variable aléatoire $\underline{Z}$ en (N+1) sous-espaces orthogonaux [4-13], [4-14] :

$$Z_K \oplus \tilde{Z}_1(K+1 \mid K) \oplus \tilde{Z}_2{}^1(K+1 \mid K+1) \oplus \ldots\ldots\ldots\ldots \oplus \tilde{Z}_N{}^{N-1}(K+1 \mid K+1)$$

Utilisons la propriété géométrique suivante :

- soit $\underline{\beta}$ un élément de l'espace H* qui est un sous-espace fermé de $L_2$ ;
- soit $\hat{\underline{\beta}}_1$ sa projection sur le sous-espace fermé $Z_1$ de H* ($\hat{\underline{\beta}}_1$ est donc la meilleure estimation de $\underline{\beta}$ dans $Z_1$) ;
- soit $\hat{Z}_2$ le vecteur de dimension m des projections de $Z_2$ dans $Z_1$ ($Z_2$ est donc la meilleure estimation de $Z_2$ dans $Z_1$) ;
- soit enfin $\tilde{Z}_1 = Z_2 - \tilde{Z}_2$

$\hat{\underline{\beta}}$, projection de $\beta$ dans le sous-espace $Z_1 \oplus Z_2$, s'écrit alors :

$$\hat{\underline{\beta}} = \beta_1 + E[\underline{\beta}\,\tilde{Z}_2{}^T]\,\{E[\tilde{Z}_2 \cdot \tilde{Z}_2{}^T]\}^{-1}\,\tilde{Z}_2 \qquad (4\text{-}5\text{-}11)$$

En écrivant les équations 4-4-10 sous la forme :

$$\underline{x}_j(K+1) = \sum_{m=1}^{N} A_{jm}(K)\;\underline{x}_m(K) + \underline{w}_j(K) \qquad (4\text{-}5\text{-}12)$$

nous obtenons alors pour la prédiction à un pas :

$$\underline{x}_j (K+1 \mid K) = \sum_{m=1}^{N} A_{jm} (K) \hat{\underline{x}}_m (K \mid K) \text{ et} \qquad (4\text{-}5\text{-}13)$$

$$\tilde{\underline{x}}_j (K+1 \mid K) = \underline{x}_j (K+1) - \hat{\underline{x}}_j (K+1 \mid K) = \sum_{m=1}^{N} A_{jm} (K) [\underline{x}_m (K) - \hat{\underline{x}}_m (K \mid K)] \qquad (4\text{-}5\text{-}14)$$

La covariance de la prédiction à un pas est donc :

$$P_{jj} (K+1 \mid K) = E [\tilde{\underline{x}}_j (K+1 \mid K) \; \tilde{\underline{x}}_j^T (K+1 \mid K)]$$

$$= \sum_{m=1}^{N} A_{jm} (K) \{ \sum_{f=1}^{N} P_{mr} (K \mid K) A_{jr}^T (K) \} + Q_j (K) \qquad (4\text{-}5\text{-}15)$$

$$P_{jm} (K+1 \mid K) = E [\tilde{\underline{x}}_j (K+1 \mid K) \; \tilde{\underline{x}}_j^T (K+1 \mid K)]$$

$$= \sum_{f=1}^{N} A_{jr} (K) \{ \sum_{s=1}^{N} P_{rs} (K \mid K) A_{ms}^T (K)) \qquad (4\text{-}5\text{-}16)$$

L'application de 4-4-11 pour les sous-espaces $\tilde{Z}_1, \tilde{Z}_2, \ldots\ldots\tilde{Z}_N$ aboutit alors à [4-13] d'où :

$$\hat{\underline{x}}_j (K+1 \mid K+1)_s = \hat{\underline{x}}_j (K+1 \mid K+1)_{s-1} + (K_j)^{s-1} (K+1) \tilde{\underline{z}}_s^{s-1} (K+1 \mid K+1) \qquad (4\text{-}5\text{-}17)$$

$$P_{jm} (K+1 \mid K+1) = P_{jm} (K+1 \mid K+1)_{s-1} - (K_j)_s^{s-1} (K+1) H_s (K+1) P_{sm} (K+1 \mid K+1)_{s-1} \qquad (4\text{-}5\text{-}18)$$

$$(K_j)_s^{s-1} (K+1) = P_{js} (K+1 \mid K+1)_{s-1} H_s^T (K+1) [H_s^T (K+1) P_{ss} (K+1 \mid K+1) H_s^T (K+1) + R_s (K+1)]^{-1} \qquad (4\text{-}5\text{-}19)$$



L'algorithme décentralisé du filtre de Kalman peut donc se résumer ainsi :

1) à partir des équations 4-5-13 à 4-5-16, on calcule les prédictions à un pas
2) posons s=1 et utilisons les conditions suivantes :

$$\underline{x}_j (K+1 \mid K+1)_0 = \underline{x}_j (K+1) \text{ et } P_{jm} (K+1 \mid K+1)_0 = P_{jm} (K+1 \mid K+1)$$

3) on calcule alors :

$(K_j)^{s-1}$ ; $\underline{x}_j (K+1 \mid K+1)_s$ et $P_{jm} (K+1 \mid K+1)_s$

4) on pose s = s+1 et on boucle les calculs en retournant à l'étape 3.

Quand s = N+1, le calcul est terminé.

Si les sous-matrices ont les mêmes dimensions v, le nombre total des opérations pour le filtre décentralisé est :

- N inversions de matrices symétriques de dimension v.v,
- $(2N^3 + N^2)$ multiplications de matrices non symétriques de dimension v.v,
- $(2N^2 + N)$ multiplications de matrices de dimension v.v dont le résultat est symétrique,
- N (N-1) (4 N-1) additions de matrices non symétriques de dimension v.v,

- $2N^2$ additions de matrices symétriques v.v par un vecteur de dimension v,
- $2N^2$ additions de vecteurs de dimension v.

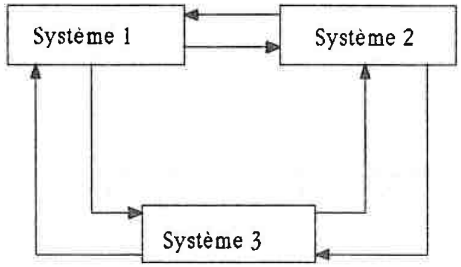
Le calcul parallèle du filtre décentralisé se fait donc en (N+1) étapes :

Etape 1 : - calcul des prédictions à un pas. Cette étape est la même que celle du filtre centralisé

Etape 2 : - calcul de $\underline{x}_j (K+1 \mid K+1)_1$ et $P_{jm} (K+1 \mid K+1)_1$

Etape N+1 : - calcul de $\underline{x}_j (K+1 \mid K+1)_N$ et $P_{jm} (K+1 \mid K+1)_N$

*Exemple* : Considérons le système interconnecté suivant :




La représentation mathématique de ce système est :



$$\begin{bmatrix} \underline{X}_1(K+1) \\ \underline{X}_2(K+1) \\ \underline{X}_3(K+1) \end{bmatrix} = \begin{bmatrix} A_{11}(K) & A_{12}(K) & A_{13}(K) \\ A_{21}(K) & A_{22}(K) & A_{23}(K) \\ A_{31}(K) & A_{32}(K) & A_{33}(K) \end{bmatrix} \begin{bmatrix} \underline{X}_1(K) \\ \underline{X}_2(K) \\ \underline{X}_3(K) \end{bmatrix} \begin{bmatrix} \underline{W}_1(K) \\ \underline{W}_2(K) \\ \underline{W}_3(K) \end{bmatrix} \quad (4\text{-}5\text{-}20)$$

$$\begin{bmatrix} \underline{Z}_1(K+1) \\ \underline{Z}_2(K+1) \\ \underline{Z}_3(K+1) \end{bmatrix} = \begin{bmatrix} H_1^{(K+1)} & 0 & 0 \\ 0 & H_2^{(K+1)} & 0 \\ 0 & 0 & H_3^{(K+1)} \end{bmatrix} \begin{bmatrix} \underline{X}_1(K+1) \\ \underline{X}_2(K+1) \\ \underline{X}_3(K+1) \end{bmatrix} \begin{bmatrix} \underline{V}_1(K) \\ \underline{V}_2(K) \\ \underline{V}_3(K) \end{bmatrix} \quad (4\text{-}5\text{-}21)$$

La figure 4-18 présente le graphe correspondant de l'algorithme décentralisé du Filtre de Kalman pour le calcul des sous-matrices de covariance. Sur cette figure, les tâches sont numérotées de la façon suivante :

- tâches 1 à 60 : calcul de $\sum_{m=1}^{N} A_{jm} \sum_{f=1}^{N} (P_{mr}(K|K) A_{jr}^T(K))$
- tâches 61 à 80 : calcul des $P_{jm} (K+1 \mid K+1)$
- tâches 81 à 100 : calcul des $P_{jm} (K+1 \mid K+1)_2$
- tâches 101 à 120 : calcul des $P_{jm} (K+1 \mid K+1)_3$

A une étape donnée (étape de calcul des $P_{jm} (K+1 \mid K+1)_1$ par exemple), on distingue notamment :

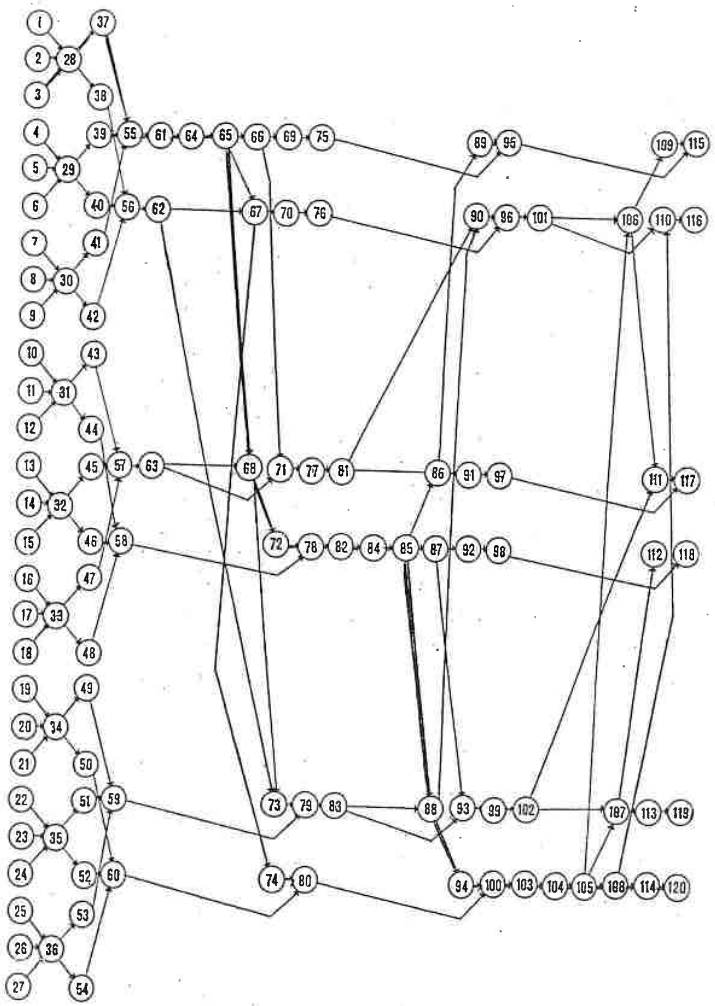


Figure (4-18) Filtre de Kalman décentralisé



- tâche 61 : $M_{11} = P_{11}(K+1 \mid K)\, H_1^T$
- tâche 65 : $u_1 = (H_1 M_{11} + R_1)^{-1}$
- tâche 68 : $(K_{21})_1^0 = M_{21} - u_1$
- tâches 75 à 80 : $P_{ij}(K+1 \mid K+1)$ $i = 1$ à $3$ $j = i$ à $3$

Dans le cas où les sous-matrices $A_{ij}$ et $H_i$ ont les mêmes dimensions v, le chemin critique du graphe 4-18 est composé de :

- (2N + 1) multiplications de deux matrices de dimensions v.v,
- (2N +1) multiplications de deux matrices de dimensions v.v dont le résultat est symétrique,
- (3N - 1) additions de deux matrices de dimensions v.v symétriques
- (N-1) additions de deux matrices de dimensions v.v non symétriques;
- N inversions de matrices symétriques de dimensions v.v.

En connaissant le nombre total d'opérations et la durée du chemin critique, on peut calculer le taux d'accélération de l'algorithme. Pour v suffisamment grand, on peut négliger le temps de calcul des additions sachant que le temps de calcul correspondant aux opérations sur les matrices symétriques est la moitié de celui correspondant au temps des opérations sur les matrices non symétriques, ce taux d'accélération asymptotique de l'algorithme est donc :

$$\tau_{accel} = \frac{2N^3}{2N + N + N/2} = \frac{4N^2}{7} \qquad (4\text{-}5\text{-}22)$$

Le taux d'accélération asymptotique du filtre décentralisé est inférieur à celui du filtre non décentralisé, mais le nombre d'opérations total du filtre décentralisé est beaucoup moins important : en conséquence, la durée du chemin critique du filtre décentralisé est plus courte que celle du filtre centralisé.

La figure 4-19 présente le graphe correspondant au calcul du filtre centralisé pour le système (4-5-20), (4-5-21). Le nombre de tâches à exécuter est de 179 au lieu de 120 pour le filtre décentralisé.

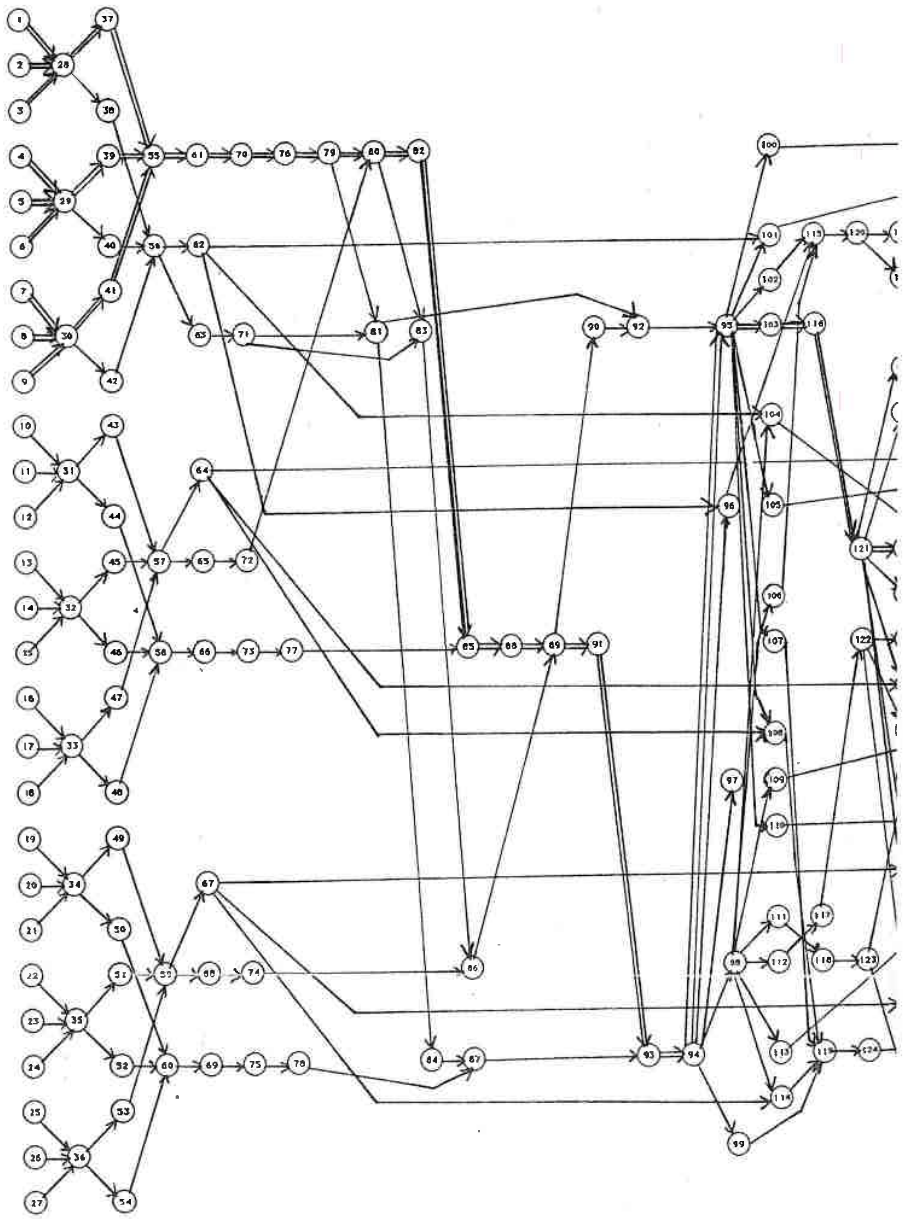
La durée du chemin critique du graphe (4-19) correspond à :

- 3N multiplications de deux matrices non symétriques,
- (N+4) multiplications de deux matrices symétriques,
- (3N - 4) additions de deux matrices non symétriques,
- (2N + 2) additions de deux matrices symétriques,
- N inversions de matrices symétriques.

Par rapport au filtre décentralisé, le nombre de multiplications et additions non symétriques a été augmenté, ce qui par conséquent augmente la durée du chemin critique.

Le tableau (4-5) compare le temps du chemin critique des deux filtres, pour N = 3 et dans le cas où $t_{div} = 6\ t_{add}$, $t_{mult} = 5\ t_{add}$.





220

| v | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| Filtre centralisé | 2511 | 5799 | 11150 | 19068 |
| Filtre décentralisé | 2139 | 5015 | 9625 | 16440 |
| Rapport = R | 0,85 | 0,86 | 0,861 | 0,86 |

Tableau (4-5) Comparaison du temps du chemin critique
des filtres centralisés et décentralisés

Pour la valeur asymptotique de R, en supposant toujours que le temps de calcul des opérations sur les matrices symétriques est la moitié de celui des matrices non symétriques, on peut écrire :

$$R = \frac{(2N+1) + \frac{2N+1}{2} + \frac{N}{2}}{3N + \frac{N+4}{2} + \frac{N}{2}} \approx 0{,}875 \qquad (4\text{-}5\text{-}23)$$

Le temps du chemin critique du filtre décentralisé est donc de 12,5 % inférieur à celui du filtre centralisé.

ORDONNANCEMENT DU CALCUL

Nous avons calculé le taux d'accélération maximum pour le filtre décentralisé dans l'équation (4-5-22). Ce taux est réalisable sur au maximum de $N^3$ processeurs. Le taux d'occupation des processeurs est dans ce cas de :

$$\tau_{occup} = \frac{\frac{4}{7}N^2}{N^3} = \frac{4}{7N} \qquad (4\text{-}5\text{-}24)$$

Pour N suffisamment grand, le taux d'occupation des processeurs est très faible. L'ordonnancement du calcul pour diminuer le nombre de processeurs est donc indispensable dans ce cas. Nous avons étudié le nombre de processeurs nécessaires pour effectuer le calcul dans le temps du chemin critique : ce nombre minimal est de $N^2$, le taux d'occupation des processeurs est :

$$\tau_{occup} = \frac{4N^2}{7N^2} = 0{,}55 \qquad (4\text{-}5\text{-}25)$$

Ce taux d'occupation n'est pas excellent. Nous pouvons trouver un compromis qui, tout en améliorant ce taux, augmente légèrement la durée du calcul et illustré dans le tableau (4-6) qui donne le temps d'exécution de l'algorithme pour le système (4-4-19) par l'ordonnancement du calcul dans le cas où V = 3, $t_{div} = 6\ t_{add}$ et $t_{mult} = 5\ t_{add}$. On peut remarquer que le temps du chemin critique est obtenu sur 11 processeurs, avec un taux d'occupation de 0,53. Pour 6 processeurs, la durée d'exécution est de 2400, soit 13 % de plus que le temps du chemin critique, mais le taux d'occupation passe de 0,53 à 0,86, ce qui représente une amélioration considérable. Dans ce cas, le choix de 6 processeurs nous semble un bon compromis entre le taux d'occupation et le temps d'exécution.



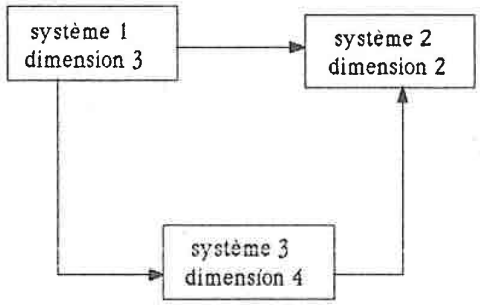
| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 11 | 2139 | 5,8 | 0,53 |
| 10 | 2226 | 5,6 | 0,56 |
| 8 | 2292 | 5,4 | 0,68 |
| 6 | 2400 | 5,2 | 0,86 |
| 4 | 3126 | 3,97 | 0,99 |
| 3 | 4173 | 2,98 | 0,99 |

Tableau (3-6) Temps d'exécution pour le filtre décentralisé pour le système 4-4-19

ORDONNANCEMENT DANS LE CAS GENERAL

Nous avons étudié le cas où les sous-systèmes sont tous de même dimension et où les sous-matrices $A_{ij}$ sont toutes non nulles.

Généralement, les systèmes interconnectés n'ont pas tous la même dimension et nous pouvons trouver $A_{ij} = 0$. Considérons alors le système suivant :




La représentation mathématique de ce système est :

$$\begin{bmatrix} \underline{x}_1(K+1) \\ \underline{x}_2(K+1) \\ \underline{x}_3(K+1) \end{bmatrix} = \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & A_{23} \\ A_{31} & 0 & A_{33}(K) \end{bmatrix} \begin{bmatrix} \underline{x}_1(K) \\ \underline{x}_2(K) \\ \underline{x}_3(K) \end{bmatrix} \begin{bmatrix} \underline{w}_1(K) \\ \underline{w}_2(K) \\ \underline{w}_3(K) \end{bmatrix}$$

$$\begin{bmatrix} \underline{Z}_1(K+1) \\ \underline{Z}_2(K+1) \\ \underline{Z}_3(K+1) \end{bmatrix} = \begin{bmatrix} H_1 & 0 & 0 \\ 0 & H_2 & 0 \\ 0 & 0 & H_3 \end{bmatrix} \begin{bmatrix} \underline{X}_1(K+1) \\ \underline{X}_2(K+1) \\ \underline{X}_3(K+1) \end{bmatrix} \quad (4\text{-}4\text{-}25)$$

La figure (4-19) présente le graphe correspondant pour le filtre centralisé du système (4-4-25). Sur cette figure, nous avons représenté également le calcul des vecteurs d'état (tâches en haut de la figure). La durée du chemin critique dans ce cas est de 2685 $t_{add}$, c'est-à-dire une valeur supérieure à celle du système (4-4-19), mais réalisable sur 6 processeurs au lieu de 11. Les tâches critiques de la figure (4-19) sont légérement différentes de celles de la figure (4-18). Le tableau (4-7) donne le temps d'exécution de l'algorithme pour ls systèmes (4-4-25) dans le cas d'ordonnancement du calcul.

Enfin, lorsque dans le cas général, on ne peut pas déterminer le chemin critique a priori, il est alors possible d'utiliser les techniques de la méthodes P.E.R.T. présentées au chapître 2.



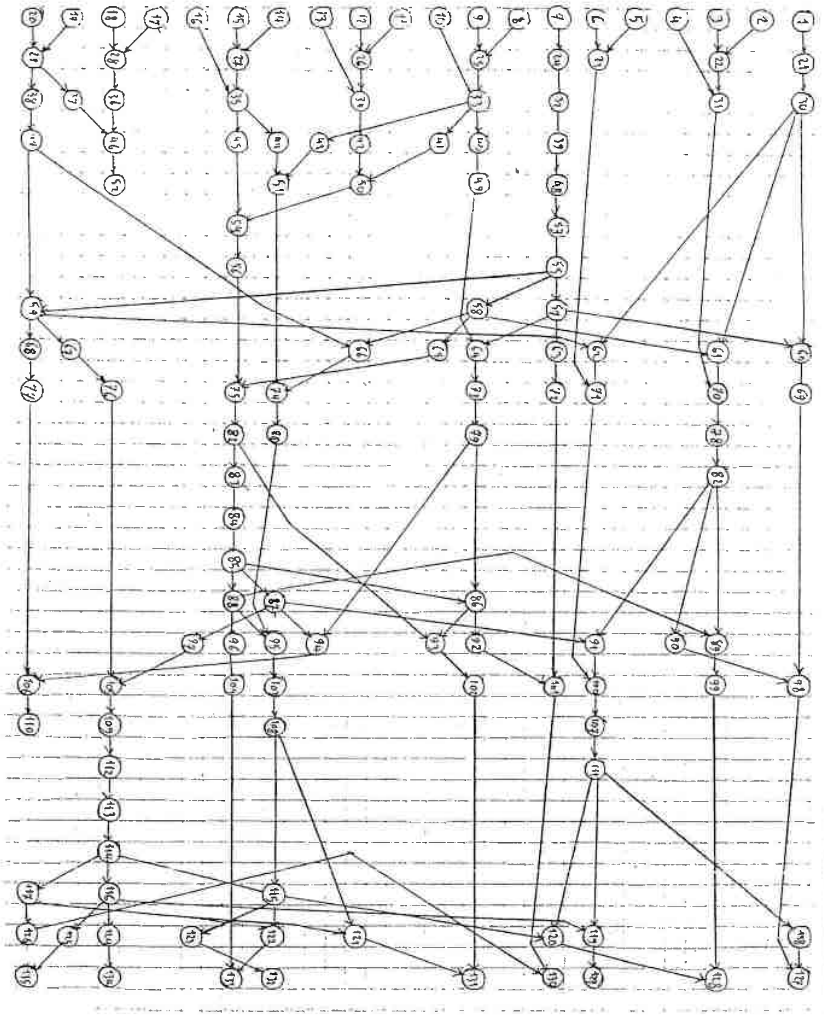


Figure (4-20) Filtre de Kalman non décentralisé

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 11 | 2139 | 5,8 | 0,53 |
| 9 | 2226 | 5,6 | 0,62 |
| 6 | 2400 | 5,2 | 0,86 |
| 5 | 2652 | 4,7 | 0,94 |
| 4 | 3126 | 3,97 | 0,99 |
| 3 | 4173 | 2,98 | 0,99 |

Tableau (4-9) Temps d'exécution pour le filtre de Kalman décentralisé

## 4-6 CONCLUSION

Nous avons étudié les techniques de calcul parallèle pour la multiplication de deux matrices et appliqué les résultats de cette étude pour le calcul parallèle des algorithmes de validation des données et d'estimation des paramètres.

Dans chaque cas, nous avons comparé les algorithmes directs et irératifs et nous avons établi les conditions d'efficacité des algorithmes itératifs pour améliorer le temps de calcul.

Nous avons égallement appliqué les méthodes du calcul parallèle dans le cas du filtre de Kalman centralisé et décentralisé. Dans tous les cas, nous avons démontré que les méthodes d'ordonnancement de calcul présentées au chapitre 2 aboutissent à un taux d'occupation des processeurs plus élevé.

# - CHAPITRE 5 -

# CALCUL PARALLELE

# DE LA COMMANDE OPTIMALE

## 5-1 INTRODUCTION

Le problème de la commande optimale a suscité beaucoup d'attention ces dernières années. Les raisons principales sont le besoin croissant de systèmes de hautes performances d'une part et la disponibilité des ordinateurs d'autre part. Mais l'application des méthodes de commande optimale pose deux problèmes essentiels : le temps de calcul et la capacité de mémoire, deux problèmes qui sont en pratique liés.

Pour un problème discret par exemple, la solution du problème linéaire-quadratique peut être obtenue par la résolution d'une équation de Riccati. Mais l'équation de Riccati dans ce cas est un système d'équations récursives non linéaires de $\frac{n(n+1)}{2}$ éléments et il faut en plus stocker les valeurs de la matrice p solution de l'équation de Riccati, pour toute la période d'optimisation : le temps de calcul et la place mémoire nécessaire augmentent rapidement avec la dimension du système. Les techniques de décomposition permettent de surmonter en partie ces difficultés et fournissent également une base assez solide pour l'utilisation des architectures parallèles.

Nous avons donc consacré ce chapitre à l'étude de la commande optimale. Nous étudierons d'abord les méthodes de résolution en parallèle des systèmes d'équations différentielles . Nous étudierons ensuite les possibilités d'application des techniques parallèles du calcul de la commande optimale des systèmes continus ainsi que des systèmes discrets. Nous aborderons enfin la résolution parallèle de l'équation de Riccati.



## 5-2 RESOLUTION DES SYSTEMES D'EQUATIONS DIFFERENTIELLES

La résolution en parallèle des systèmes d'équations différentielles peut être effectuée par décomposition de ces équations dans l'espace et dans le temps. Nous étudierons la décomposition spatiale en même temps que le calcul parallèle de la commande optimale. Nous considérons donc dans un premier temps la décomposition temporelle.

### 5-2-1 Equations différentielles non linéaires

Soit à résoudre le système d'équations différentielles suivant :

$$\dot{\underline{Z}} = f(\underline{Z}, t) \qquad t_0 < t < t_f$$

$$\underline{C}[\underline{Z}(t_0), \underline{Z}(t_f)] = 0 \qquad (5\text{-}2\text{-}1)$$

où $\underline{Z}$ est un vecteur de dimension n, $t_0$ est l'instant initial, $t_f$ est l'instant final et $\underline{C}$ est le vecteur des conditions aux deux bouts. Pour partitionner cette équation dans le temps, nous divisons l'intervalle $[t_0, t_f]$ en m intervalles $[t_0, t_1], [t_1, t_2], \ldots\ldots\ldots, [t_{m-1}, t_m]$.

Supposons $t_0 < t_1 < t_2 \ldots\ldots\ldots < t_m = t_f$ (5-2-2)

Pour $t_0, t_1, \ldots\ldots\ldots t_{m-1}$, nous estimons les valeurs de Z que nous désignons par $\underline{Z}_0, \underline{Z}_1, \ldots\ldots\ldots \underline{Z}_{m-1}$.

A partir de ces valeurs initiales, nous pouvons intégrer 5-2-1 sur

chaque intervalle par une méthode classique, par exemple par la méthode de Kutta-Runge.

Nous estimons ainsi les nouvelles valeurs de $Z^{(1)}_1, Z^{(1)}_2, \ldots\ldots Z_m{}^{(1)}$.

Nous essayons ensuite d'améliorer les premières estimations afin de diminuer d'itération en itération la différence entre les estimations successives.

Supposons donc qu'à l'itération K, $Z_0{}^{(K+1)}$, $Z_1{}^{(K+1)}$, ..... $Z_{m-1}{}^{(K+1)}$ sont des valeurs estimées à partir des conditions initiales $Z_0{}^{(K)}, Z_1{}^{(K)}, \ldots\ldots Z_{m-1}{}^{(K)}$.

Définissons les vecteurs $\underline{h}^{(K)}$ comme :

$$\underline{h}_j{}^{(K)} = Z_{j+1}{}^{(K+1)} - Z_{j+1}{}^{(K)} \quad j = 0 \text{ à } m-2 \qquad (5\text{-}2\text{-}3)$$

$$\underline{h}_{m-1}{}^{(K)} = \underline{C}\,[Z_0{}^{(K)}, Z_M{}^{(K+1)}]$$

or $$Z_{j+1}{}^{(K+1)} = \underline{\Phi}\,(Z_j{}^{(K)}, t) \quad j = 0 \text{ à } m-1 \qquad (5\text{-}2\text{-}4)$$

où $\underline{\Phi}_j$ est une fonction obtenue par l'intégration de l'équation 5-2-1 de l'intervalle $[t_{j-1}, t_j]$. On peut donc écrire :

$$\underline{h}_j{}^{(K)} = \underline{\Phi}_j\,(Z_j{}^{(K)}, t) - Z_{j+1}{}^{(K)} \quad j = 0 \text{ à } m-2$$

$$\underline{h}_{m-1}{}^{(K)} = \underline{C}\,[Z_0{}^{(K)}, \underline{\Phi}_{m-1}\,(Z_{m-1}{}^{(K)}, t)] \qquad (5\text{-}2\text{-}5)$$



En écrivant le développement limité de $h_j{}^{(K)}$ et $h_{m-1}{}^{(K)}$, nous obtenons :

$$h_j{}^{(K)} = (Z_j{}^{(K)} + \Delta Z_j{}^{(K)}, Z_{j+1}{}^{(K)} + \Delta Z_{j+1}{}^{(K)}) = h_j{}^{(K)}\,(Z_j{}^{(K)}, Z_{j+1}{}^{(K)}) + \frac{\partial h_j{}^{(K)}}{\partial Z_j}\Delta Z_j{}^{(K)} + \frac{\partial h_j{}^{(K)}}{\partial Z_{j+1}}\Delta Z_{j+1}{}^{(K)}$$

$$h_{m-1}{}^{(K)}\,(Z_0{}^{(K)} + \Delta Z_0{}^{(K)}, Z_{m-1} + \Delta Z_{m-1}{}^{(K)}) = C\,[Z_0{}^{(K)}, Z_{m-1}{}^{(K)}] + \frac{\partial C}{\partial Z_0}\Delta Z_0{}^{(K)} + \frac{\partial C}{\partial Z_{m-1}}\Delta Z_{m-1}{}^{(K)}$$

(5-2-6)

et en appliquant la méthode de Newton, on peut écrire :

$$h_j{}^{(K)}\,(Z_j{}^{(K)}, Z_{j+1}{}^{(K)}) + \frac{\partial h_j{}^{(K)}}{\partial Z_j}\Delta Z_j{}^{(K)} + \frac{\partial h_j{}^{(K)}}{\partial Z_{j+1}}\Delta Z_{j+1}{}^{(K)} = 0$$

$$C\,[Z_0{}^{(K)}, Z_{m-1}{}^{(K)}] + \frac{\partial C}{\partial Z_0}\Delta\ Z_0{}^{(K)} + \frac{\partial C}{\partial Z_{m-1}}\Delta Z_{m-1}{}^{(K)} = 0$$

(5-2-7)

or

$$\frac{\partial h_j{}^{(K)}}{\partial Z_j} = \frac{\partial \varphi_j\,(Z_j{}^{(K)}, t)}{\partial Z_j} = M_j\,(t, Z_j{}^{(K)}) \text{ et } \frac{\partial h_j{}^{(K)}}{\partial Z_{j+1}} = -\,I_n \quad j = 0 \text{ à } m-2$$

(5-2-8)

où $M_j\,(t, Z_j)$ sont des matrices de dimension n et $I_n$ est la matrice d'identité de l'ordre n.

On a donc :

$$\frac{\partial C}{\partial Z_{m-1}} = \frac{\partial C}{\partial Z_m} = \frac{\partial Z_m}{\partial Z_{m-1}} = \frac{\partial Z}{\partial Z_m} \cdot M_{m-1}\,(t, Z_{m-1})$$

(5-2-9)

En posant :

$$\frac{\partial \underline{C}}{\partial \underline{Z}_0} = T \text{ et } \frac{\partial \underline{C}}{\partial \underline{Z}_m} = R$$

on a alors les équations suivantes :

$$M_j \Delta \underline{Z}_j^{(K)} - \Delta \underline{Z}_{j+1}^{(K)} = -\underline{h}_j^{(K)} \qquad j = 0,2,\ldots\ldots\ldots, m-2$$

$$T \Delta \underline{Z}_0^{(K)} + R\, M_{m-1} \Delta \underline{Z}_{m-1}^{(K)} = \underline{h}_{m-1}^{(K)} \qquad (5\text{-}2\text{-}10)$$

A partir des équations (5-2-10), on peut donc calculer les valeurs de $\Delta \underline{Z}_j$ (j = 0 à m-1) et appliquer l'algorithme de Newton :

$$\underline{Z}_j^{(K+1)} = \underline{Z}_j^{(K)} + \alpha\, \Delta \underline{Z}_j^{(K)} \qquad j = 1 \text{ à } m-1$$

$$0 < \alpha < 1 \qquad (5\text{-}2\text{-}11)$$

La résolution de l'équation différentielle 5-2-1 par la partition temporelle, comprend donc les étapes suivantes :

a) Calcul des $\underline{Z}_{j+1}^{(K+1)}$ à partir des $\underline{Z}_j^{(K)}$

b) Calcul des $M_j(t)$, $T(t)$, $R(t)$

c) Résolution du système d'équations linéaires (5-2-10)

Les étapes a et b peuvent être exécutées en parallèle sur au maximum (m+2) processeurs, tandis que pour l'étape trois, on peut appliquer les méthodes itératives de résolution des équations linéaires.



Dans la théorie de la commande optimale, on rencontre souvent des équations différentielles avec des conditions aux deux bouts sous la forme :

$$\dot{\underline{x}} = f_1(\underline{x}, \underline{y}, t) \quad \underline{x}(t_0) = \underline{x}_0 \quad t_0 < t < t_f \qquad (5\text{-}2\text{-}12)$$

$$\dot{\underline{y}} = f_2(\underline{x}, \underline{y}, t) \quad \underline{y}(t_f) = \underline{y}_{tf} \qquad (5\text{-}2\text{-}13)$$

où $\underline{x}$ et $\underline{y}$ sont des vecteurs de même dimension n.

Si on pose :

$\dot{\underline{Z}} = \begin{bmatrix} \dot{\underline{x}} \\ \dot{\underline{y}} \end{bmatrix}$, on a :

$$\dot{\underline{Z}} = f(\underline{Z}) \qquad \underline{C} = \begin{bmatrix} \underline{x}(t_0) - \underline{x}_0 \\ \underline{y}(t_f) - \underline{y}_{t_0} \end{bmatrix} \qquad (5\text{-}2\text{-}14)$$

On peut donc utiliser la méthode présentée ci-dessus, pour pouvoir utiliser au mieux les deux conditions aux bouts, on peut modifier légèrement la méthode précédente, à savoir :

- estimation des valeurs $\underline{y}_0, \underline{x}_1, \underline{y}_1, \ldots\ldots\ldots \underline{x}_{m-1}, \underline{y}_{m-1}$,
- intégration de l'équation 5-2-12 dans chaque intervalle dans le sens direct,
- intégration de l'équation 5-2-13 dans chaque intervalle à rebours ;

On définit alors

$$h_{j1}^{(K)} = x_{j+1}^{(k+1)} - x_j^{(K)}$$
$$h_{j2}^{(K)} = y_{j-1}^{(k+1)} - y_j^{(K)}$$

et on applique la méthode de Newton comme dans le cas précédent. Les deux méthodes sont exécutables sur 2m processeurs.

La méthode de partition temporelle est applicable également dans le cas des équations différentielles avec des conditions à un seul bout. Dans ce cas, $\Delta Z_0$ est nul et la dernière équation du système 5-2-10 disparaît. La résolution du système 5-2-10 est donc plus simple.

### 5-2-2 Equations différentielles linéaires

Soit à résoudre l'équation différentielle :

$$\dot{Z} = A(t)\,Z + b(t) \qquad t_0 \le t \le t_f \qquad (5\text{-}2\text{-}15)$$
$$T\,Z(t_0) + R\,Z(t_f) = d$$

La méthode de partition temporelle présentée en 5-2-1 est toujours applicable. Dans ce cas, si on utilise une méthode récursive comme par exemple celle de Kutta-Runge, les matrices $M_j$ sont indépendantes des $Z_j$. Elles sont donc calculées uniquement pour la première itération.

Pour les équations différentielles avec des conditions à un seul bout, on peut utiliser d'autres méthodes numériques pour la résolution de



ces équations. Nous étudierons deux méthodes ; la méthode de Kurra-Runge et la méthode de prédiction-correction.

a) Méthode de Kutta-Runge à deux paramètres : [5-1]

Soit à résoudre l'équation différentielle suivante :

$$\dot{x} = A(t)\,x + b(t) \qquad x(0) = x_0 \qquad t_0 \le t \le t_f \qquad (5\text{-}2\text{-}16)$$

Dans la méthode de Kutta-Runge, on divise l'intervalle $[t_0, t_f]$ en m sous-intervalles $[t_0, t_1]$, $[t_1, t_2]$, ....., $[t_{n-1}, t_f]$ de longueur h, ou h est le pas d'intégration. On calcule alors les valeurs de $x_1$, $x_2$, ....., $x_n$, récursivement à partir des équations suivantes :

$$K_1 = h\,[A(t_j)\cdot x_j + b(tj)]$$
$$K_2 = h\,[A(t_{j+1})\,(x_j + K_1) + b(t_{j+1}) \qquad (5\text{-}2\text{-}17)$$
$$x_{j+1} = x_j + 1/2\,(K_1 + K_2) \qquad j = 0 \text{ à } m-1$$

En remplaçant $K_1$ et $K_2$ dans la troisième équation, on obtient :

$$x_{j+1} = 1/2\,[I + (I + h\,A(t_j))\,(I + h\,A(t_{j+1}))]\,x_j + h\,[(I + hA(t_{j+1}))\,b_j + b_{j+1}$$
$$j = 0 \text{ à } n-1 \qquad (5\text{-}2\text{-}18)$$

où $$x_{j+1} = A_j\,x_j + P_j \qquad (5\text{-}2\text{-}19)$$

Le calcul des $A_j$ et $P_j$ peut être effectué en parallèle : pour la résolution de l'équation (5-2-8), on peut utiliser des méthodes itératives de type Jacobi. Au niveau du calcul des $A_j$, on peut donc calculer : $[I + h A (t_j)]$ et $[I + h_1 (t_{K+1})]$ en parallèle.

On diminue ainsi les étapes de calcul à effectuer. Dans le cas où, pour une raison quelconque (convergence très lente de la méthode irérative ou nombre limité de processeurs), on doit mettre en oeuvre la méthode directe, on peut toujours utiliser l'algorithme suivant :

$$\begin{aligned} \underline{K}_1 &= h\,[A(t_j)\ \underline{x} + b_j] \\ \underline{K}_2 &= h\,[A(t_{j+1})\ \underline{x}_j + \underline{b}_{j+1}) \\ \underline{x}_{K+1} &= \underline{x}_j + 1/2\,[(\underline{K}_1 + \underline{K}_2) + hA(t_{j+1})\,\underline{K}_1] \end{aligned} \qquad (5\text{-}2\text{-}20)$$

Le calcul de $\underline{K}1$ et $\underline{K}_2$ peut être effectué en parallèle, mais pour le calcul de $\underline{x}_{j+1}$, il faut attendre la fin du calcul de $\underline{K}_1$ et $\underline{K}_2$. Le schéma du calcul est ainsi le suivant :

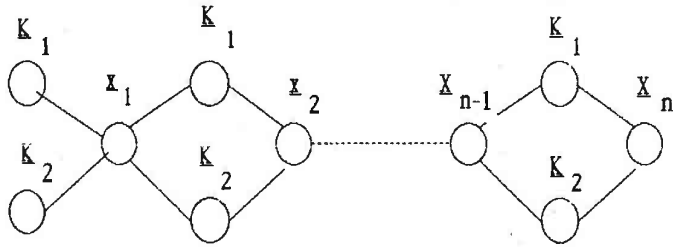



Figure (5-1) Schéma du calcul parallèle pour le système 5-2-20



b) Méthode de prédiction-correction [5-2]

Dans cette méthode, on utilise le schéma suivant :

$$\begin{aligned} \underline{x}_{j+1}^{(P)} &= \underline{x}_j^{(C)} + h/2\,(3\underline{f}_j^{(C)} - \underline{f}_{j-1}^{(C)}) \\ \underline{x}_{j+1}^{(C)} &= \underline{x}_j^{(C)} + h/2\,(\underline{f}_{j+1}^{(P)} + \underline{f}t_j^{(C)}) \end{aligned} \qquad (5\text{-}2\text{-}21)$$

où $\underline{f} = A(t)\,\underline{x} + b(t)$, $\underline{x}_{j+1}^{(P)}$ est la valeur prédite de $\underline{x}_{j+1}$ et $\underline{x}_{j+1}^{(C)}$ est sa valeur corrigée.

Déjà à ce niveau, on peut calculer $\underline{f}_j^{(C)}$ et $\underline{f}_{j-1}^{(C)}$ en parallèle sur deux processeurs. Le schéma du calcul est le même que celui présenté à la figure 5-1. Mais dans le cas du système (5-2-6), l'algorithme peut avoir un parallélisme beaucoup plus grand. En effet, l'application des équations (5-2-20) pour le système (5-2-6) donne :

$$\underline{x}_{K+1}^{(P)} = \underline{x}_K^{(C)} + 3\,h/2\,(A(t_K)\,\underline{x}_K^{(C)} + \underline{b}_K) - h/2\,(A(t_{K-1})^{(C)} + \underline{b}_{K-1}) \qquad (5\text{-}2\text{-}22)$$

$$\underline{x}_{K+1}^{(C)} = \underline{x}_K^{(C)} + h/2\,(A(t_{K+1})\,\underline{x}_{K+1}^{(P)} + \underline{b}_{K+1}) + h/2\,(A(t_K\,X_K)^{(C)} + \underline{b}_K) \qquad (5\text{-}2\text{-}23)$$

En tirant $\underline{x}_{K+1}^{(P)}$ de l'équation (5-2-11) reportant dans l'équation (5-2-12) on obtient une équation de la forme :

$$\underline{x}_{K+1}^{(C)} = A_K\,\underline{x}_K^{(C)} + A_{K-1}\,\underline{x}^{(C)}{}_{K-1} + P_K \qquad (5\text{-}2\text{-}24)$$

Le calcul des $A_K$, $A_{K-1}$ et $P_K$ peut être effectué en parallèle et pour la résolution de l'équation 5-2-24, on peut utiliser des méthodes itératives de type Jacobi.

## 5-3 COMMANDE OPTIMALE DES SYSMTEMES LINEAIRES CONTINUS

Considérons le problème d'optimisation :

$$\min_{u(t)} J = \int_{t_0}^{t_f} (\| \underline{x}(t) \|_Q^2 + \| \underline{u}(t) \|_R^2)\, dt \qquad (5\text{-}3\text{-}1)$$

sous $\dot{\underline{x}}(t) = A(t)\ \underline{x}(t) + B(t)\,\underline{U}(t)$

où $\underline{x}(t)$ est le vecteur d'état de dimension n, $\underline{U}(t)$ le vecteur de commande de dimension m, A (t), B (t), Q (t), R (t) sont des matrices de dimension apropriée dont les coefficients sont des fonctions continues du temps sur l'intervalle $[t_0, t_f]$.

Nous supposons de plus que Q (t) est symétrique semi-défini positif, et R (t) est symétrique défini positif, $\forall\ t \in [t_0, t_f]$.

Les conditions d'optimalité pour le problème 5-3-1 sont alors : [5-3]

$$\dot{\underline{x}}(t) = A(t)\ \underline{x}(t) + B(t)\,\underline{U}(t) \qquad \underline{x}(t_0) = x_0$$



$$\dot{\underline{\lambda}}(t) = Q(t)\ \underline{x}(t) + A^T(t)\,\underline{\lambda}(t) \qquad \underline{\lambda}(t)(t_f) = 0$$
$$\underline{u}(t) = -R^{-1}\,B^T(t)\,\underline{\lambda}(t) \qquad (5\text{-}3\text{-}2)$$

où $\underline{\lambda}(t)$ est le vecteur des paramètres de Lagrange associé. En remplaçant $\underline{U}(t)$ dans la première équations, on obtient :

$$\dot{\underline{x}}(t) = A(t)\ \underline{x}(t) - B(t)\ R(t)^{-1}\,B(t)^T\ \underline{\lambda}(t) \qquad \underline{x}(t_0) = x_0$$
$$\dot{\underline{\lambda}}(t) = -Q(t)\ \underline{x}(t) - A^T(t)\,\underline{\lambda}(t) \qquad \underline{\lambda}(t)(t_f) = 0$$
$$(5\text{-}3\text{-}3)$$

Le système d'équations différentielles 5-3-3 est un système avec des conditions aux deux bouts.

Pour la résolution de ce problème, plusieurs partitionnements sont envisageables, partitionnement spatial et partitionnement temporel. Nous étudierons deux partitionnements spatiaux, le premier avec conditions en un seul bout et la deuxième avec conditions aux deux bouts.

Dans chaque cas, nous appliquons également le partitionnement temporel.

### 5-3-1 Décomposition en sous-problèmes avec conditions en un seul bout [5-4]

On peut décomposer les vecteurs $\underline{x}$ et $\underline{\lambda}$ en N sous vecteurs.

Le système 5-3-3 est alors partitionné de la façon suivante :

$$
\begin{array}{|c|}\hline \dot{\underline{X}}_1 \\ \dot{\underline{X}}_2 \\ \dot{\underline{X}}_N \\ \hline \dot{\underline{\lambda}}_1 \\ \dot{\underline{\lambda}}_2 \\ \dot{\underline{\lambda}}_N \\ \hline \end{array}
\begin{array}{|cc|cc|c|}\hline
A_{11} & A_{12} & A_{1N} & -D_{11} & -D_{12} & -D_{1N} \\
A_{11} & A_{12} & & -D_{11} & -D_{12} & \\
A_{N1} & & A_{NN} & -D_{N1} & & -D_{NN} \\ \hline
-Q_{11} & -Q_{12} & -Q_{1N} & -A_{11}^T & -A_{12}^T & -A_{1N}^T \\
-Q_{11} & -Q_{12} & & -A_{11}^T & -A_{12} & \\
-Q_{N1} & & -Q_{NN} & -A_{N1}^T & & -A_{NN}^T \\ \hline
\end{array}
\begin{array}{|c|}\hline \underline{X}_1 \\ \underline{X}_2 \\ \underline{X}_N \\ \hline \underline{\lambda}_1 \\ \underline{\lambda}_2 \\ \underline{\lambda}_N \\ \hline \end{array}
$$

où $D = B\,R^{-1}\,B^T$

On décompose alors le système en 2 N sous-systèmes par la décomposition de la matrice correspondante en parties bloc-diagonales et hors-diagonales. Par la relaxation de la partie hors-diagonale, on obtient ainsi 2 N équations différentielles avec conditions en un seul bout.

Le schéma de décomposition parallèle sur deux nniveaux est alors le suivant :



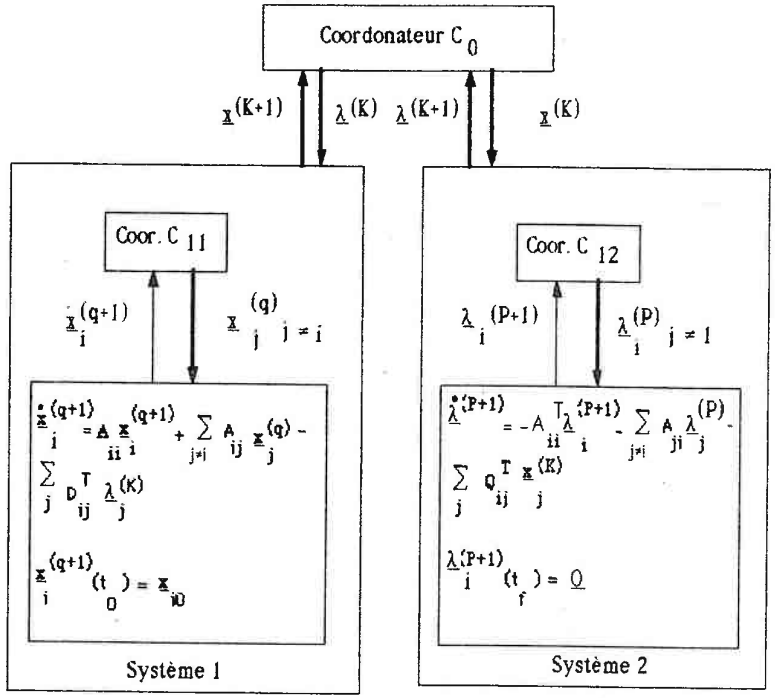



Figure (5-2) Décomposition sur deux niveaux

Dans cette méthode, les équations d'état et les équations adjointes sont découplées au niveau le plus haut. Les coordonateurs $C_{11}$ et $C_{12}$ ont respectivement la tâche de coordination parallèle des sous-systèmes des systèmes 1 et 2. Le coordonateur $C_0$ assure la suite des coordinations des systèmes 1 et 2.

Au niveau de chaque système 1 ou 2, la coordination des sous-systèmes peut être faire d'une façon synchrone ou asynchrone. Dans

le schéma 5-2, la coordination est synchrone. Le coordinateur $C_{11}$ par exemple effectue un test global à la fin de chaque itération, et selon le résultat du test, il arrête le calcul ou il déclenche l'itération suivante sur tous les sous-systèmes. Dans une coordination asynchrone par contre, à chaque fois que le coordinateur $C_{11}$ reçoit une nouvelle valeur de $\underline{x}_j$ du sous-système j, il effectue le test de la fin du calcul et si le test est faux, il met $\underline{x}_j$ à la disposition des autres sous-systèmes.

Le sous-système j commence son itération suivante avec la nouvelle valeur de $\underline{x}_j$ et les autres sous-systèmes utilisent cette nouvelle valeur quand ils auront fini leur itération en cours. Les équations itératives du calcul dans ce cas sont les suivantes :

$$\dot{\underline{x}}_i^{(q+1)} = A_{ii}\,\underline{x}_i^{(q+1)} + \sum A_{ij}\,\underline{x}_j^{(q-s_j)} - \sum D_{ij}^T\,\underline{\lambda}_j^{(K)}$$

$$\dot{\underline{\lambda}}^{(P+1)} = -A_{ii}\,\underline{\lambda}_i^{(P+1)} + \sum A_{ji}^T\,\underline{\lambda}_j^{(P-r_j)} - \sum Q_{ij}^T\,\underline{x}_j^{(K)} \qquad (5\text{-}3\text{-}4)$$

où $s_j$ et $r_j$ sont des retards éventuels des sous-systèmes j. La convergence de cet algorithme appelé algorithme de ralaxation cahotique à retard à été étudié par différents auteurs. Voir par exemple [ 5-5 ], [ 5-6 ], [ 5-7 ].

L'intérêt de la coordination asynchrone est de réduire le temps d'attente des processeurs et d'augmenter ainsi leur taux d'occupation. Cette méthode est surtout efficace pour le cas où la durée d'exécution des tâches n'est pas prévisible. Dans le cas où l'on peut prévoir la durée des tâches, on peut augmenter le taux d'occupation des processeurs par l'ordonnancement du calcul. La méthode que nous proposons est donc la méthode synchrone



ordonnancée. Les équations itératives d'état et adjointes restent donc les mêmes que dans le schéma 5-2. Ainsi, les sous-systèmes dans chacun des deux systèmes d'état et adjoints, sont synchronisés à chaque itération. Nous démontrons un peu plus loin, sur des exemples concrets, que cette synchronisation peut être effectuée au milieu des itérations à la fin de certaines opérations.

La figure 5-3 montre le graphe correspondant au calcul synchronisé des sous-systèmes au niveau des coordinateurs $C_{11}$ et $C_{12}$. On peut remarquer sur cette figure que les suites des itérations q et p dans les systèmes d'état et adjointes sont indépendantes l'une de l'autre. Les deux systèmes 1 et 2 ne sont donc pas synchronisés entre eux. En effet, si à l'itération K du coordinateur $C_0$, les équations d'un des systèmes 1 ou 2 convergent plus vite, les processeurs du système concerné doivent attendre la convergence des équations de l'autre système pour pouvoir commencer l'itération suivante, ce qui se traduit par une perte de temps de calcul. Pour surmonter ce problème et donc pour synchroniser les deux systèmes, on peut effectuer un test global sur les variables d'état et adjointes à la fin de chaque itération. L'indice d'itération des deux systèmes 1 et 2 dans ce cas est la même et les équations d'état et adjointes convergent en même temps. La figure 5-4 montre le graphe du calcul dans le cas où les itérations des systèmes 1 et 2 sont synchronisées.

On peut ajouter au partitionnement spatial le partitionnement temporel. Ainsi on augmente le degré de parallélisme de l'algorithme. La méthode de partitionnement temporel peut être une des méthodes expliquées auparavant.

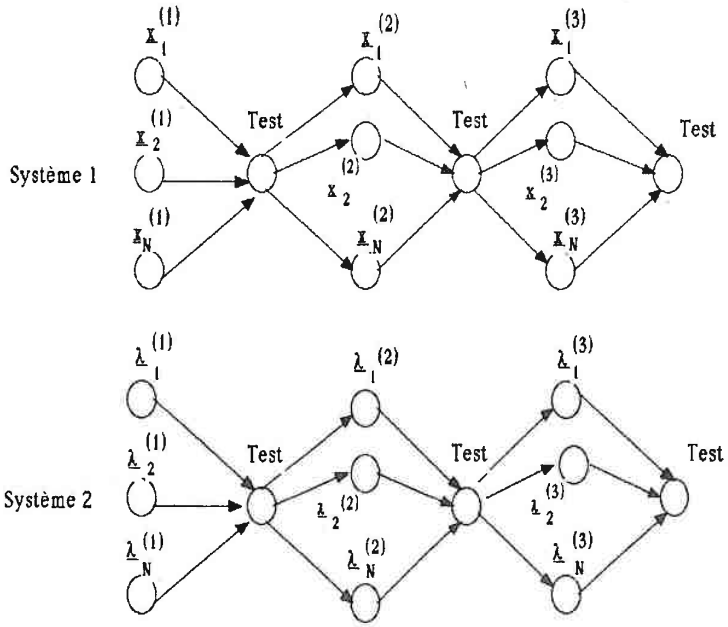



Figure (5-3) Graphe du calcul sans synchronisation des systèmes 1 et 2



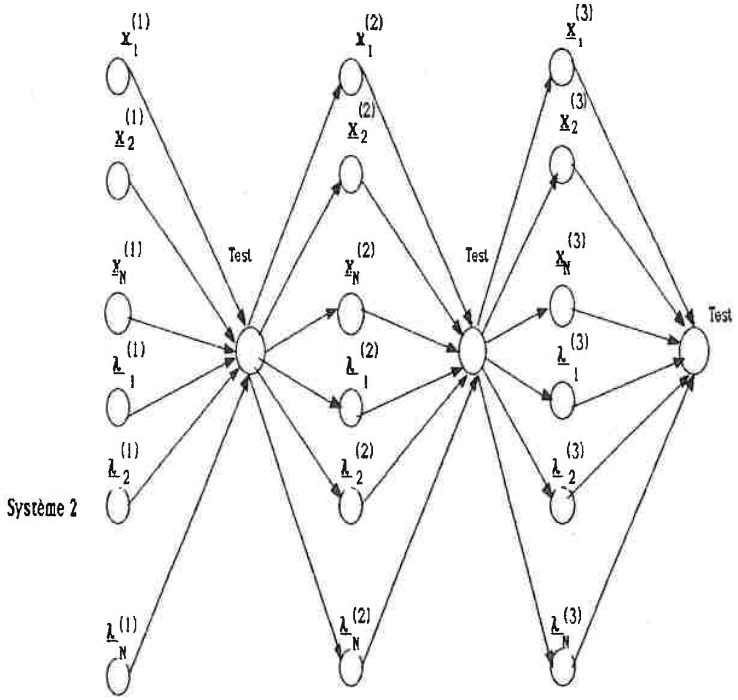



Figure (5-4) Graphe du calcul avec synchronisation des systèmes 1 et 2

Nous allons montrer le partitionnement spatial sur un exemple.

Soit un système de grande dimension composé des sous-systèmes interconnectés de dimensions différentes représenté dans le schéma suivant Figure (5-5) :

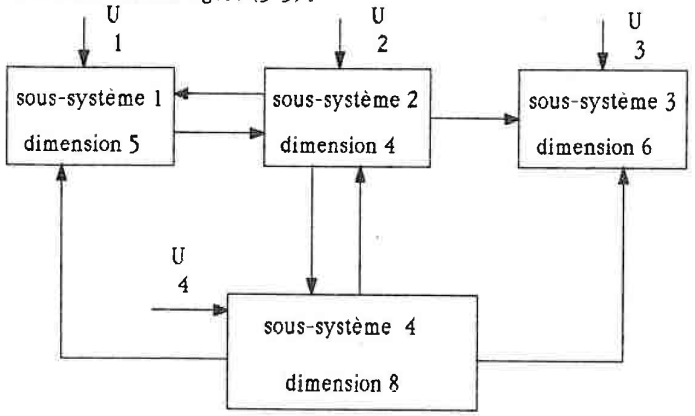



Figure (5-5) Schéma d'un système interconnecté

Supposons que la dynamique du système est décrite par l'équation suivante :

$$\dot{\underline{X}} = A\,\underline{X} + B\,\underline{U} \qquad \underline{X}_0\,(t_0) = \underline{X}_0 \qquad (5\text{-}3\text{-}5)$$

où A et B ont la forme suivante :



$$A = \begin{bmatrix} A_{11} & A_{12} & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ 0 & A_{32} & A_{33} & A_{34} \\ A_{41} & 0 & 0 & A_{44} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & 0 & 0 & 0 \\ 0 & B_{22} & 0 & 0 \\ 0 & 0 & B_{33} & 0 \\ 0 & 0 & 0 & B_{44} \end{bmatrix}$$

$A_{11}$, $A_{22}$, $A_{33}$ et $A_{44}$ sont respectivement de dimension 5, 4, 6 et 8 ;

$B_{11}$, $B_{22}$, $B_{33}$ et $B_{44}$ sont respectivement de dimension 5x2, 4x1, 6x2, 8x3.

Le critère à minimiser est de la forme :

$$\min_{\underline{u}\,(t)} J = \int_{t_0}^{t_f} \frac{1}{2}\left[ \| \underline{x}\,(t) \|_Q^2 + \| \underline{u}\,(t) \|_R^2 \right] dt$$

où Q et R sont des matrices blocs diagonales de dimensions appropriées.

Supposons $D = B\,R^{-1}\,B^T$ ; D est alors une matrice bloc diagonale.

Les équations d'optimalité sont alors de la forme :

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \\ \dot{X}_3 \\ \dot{X}_4 \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \\ \dot{\lambda}_4 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & 0 & 0 & -D_{11} & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 & -D_{22} & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 & 0 & D_{33} & 0 \\ A_{41} & 0 & 0 & A_{44} & 0 & 0 & 0 & -D_{44} \\ -Q_{11} & 0 & 0 & 0 & -A_{11}^T & -A_{21}^T & 0 & -A_{41}^T \\ 0 & -Q_{22} & 0 & 0 & -A_{12}^T & -A_{22}^T & -A_{32}^T & 0 \\ 0 & 0 & -Q_{33} & 0 & 0 & -A_{23}^T & -A_{33}^T & 0 \\ 0 & 0 & 0 & -Q_{44} & 0 & 0 & -A_{34}^T & -A_{44}^T \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} \qquad (5\text{-}3\text{-}6)$$

$$X(t_0) = X_0 \qquad \lambda(t_f) = 0$$

La décomposition du schéma 5-2 pour le système 5-3-6 donne alors au niveau des coordonateurs $C_{11}$ et $C_{12}$ les équations itératives suivantes :

$$\begin{aligned} \dot{X}_1^{(q+1)} &= A_{11} X_1^{(q+1)} + A_{12} X_2^{(q)} - D_{11} \lambda_1^{(K)} \\ \dot{X}_2^{(q+1)} &= A_{22} X_2^{(q+1)} + A_{21} X_1^{(q)} - A_{23} X_3^{(q)} - D_{22} \lambda_2^{(K)} \\ \dot{X}_3^{(q+1)} &= A_{33} X_3^{(q+1)} + A_{32} X_2^{(q)} + A_{34} X_4 - D_{33} \lambda_3^{(K)} \\ \dot{X}_4^{(q+1)} &= A_{44} X_4^{(q+1)} + A_{41} X_1^{(q)} - D_{44} \lambda_4^{(K)} \end{aligned} \qquad (5\text{-}3\text{-}7)$$

et dans le cas où la résolution des systèmes d'équations d'état est adjointe



et synchronisé, on a :

$$\begin{aligned} \dot{\lambda}_1^{(q+1)} &= -A_{11}^T \lambda_1^{(q+1)} - A_{21}^T \lambda_2^{(q)} - A_{41}^T \lambda_4^{(q)} - Q_{11} X_1^{(K)} \\ \dot{\lambda}_2^{(q+1)} &= -A_{22}^T \lambda_2^{(q+1)} - A_{12}^T \lambda_1^{(q)} - A_{32}^T \lambda_3^{(q)} - Q_{22} X_2^{(K)} \\ \dot{\lambda}_3^{(q+1)} &= -A_{33}^T \lambda_3^{(q+1)} - A_{23}^T \lambda_2^{(q)} - Q_{33} X_3^{(K)} \\ \dot{\lambda}_4^{(q+1)} &= -A_{44}^T \lambda_4^{(q+1)} - A_{34}^T \lambda_1^{(q)} - Q_{44} X_4^{(K)} \end{aligned} \qquad (5\text{-}3\text{-}8)$$

Pour l'intégration de chaque équation différentielle, nous utilisons la méthode de Kutta-Runge à deux paramètres. Le graphe correspondant à une itération est le suivant :

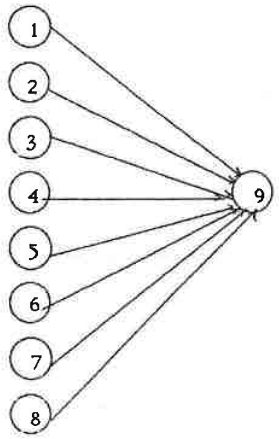



Figure (5-6) Graphe du calcul des systèmes (5-3-7) et (5-3-8)

Dans ce graphe, les tâches 1 à 8 correspondent à l'intégration

des équations différentielles (5-3-7) et (5-3-8). La tâche 9 est la tâche finale qui peut être constituée par les tests ou par une simple tâche de synchronisation de durée zéro.

Pour l'intégration des équations, nous procédons comme suit : supposons que l'on divise l'intervalle $[t_0, t_f]$ en 64 pas d'intégration. En utilisant les processeurs pour lesquels $t_{mult} = t_{add}$, la durée du chemin critique du graphe (5-6) est de 43584 $t_{add}$. Cette durée s'obtient sur au maximum 8 processeurs, mais par ordonnancement du calcul, on peut l'obtenir sur 6 processeurs.

Le tableau 5-1 donne le temps d'exécution par l'ordonnancement de cet algorithme en fonction du nombre de processeurs.

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 6 | 43584 | 5,17 | 0,86 |
| 5 | 53760 | 4,19 | 0,84 |
| 4 | 60608 | 3,72 | 0,93 |
| 3 | 80384 | 2,8 | 0,93 |
| 2 | 113536 | 1,98 | 0,99 |

Tableau (5-1) Temps d'exécution pour les systèmes (5-3-7) et (5-3-8)

On peut constater que le taux d'occupation pour 5 processeurs est inférieur à celui pour 6 processeurs. La raison en est que d'une part, le nombre des tâches à ordonnancer n'est pas important et d'autre part, la durée des tâches est très grande. Nous avons divisé les tâches 1 à 8 en tâches de durées sensiblement équivalentes comme le montre la figure



5-7 :

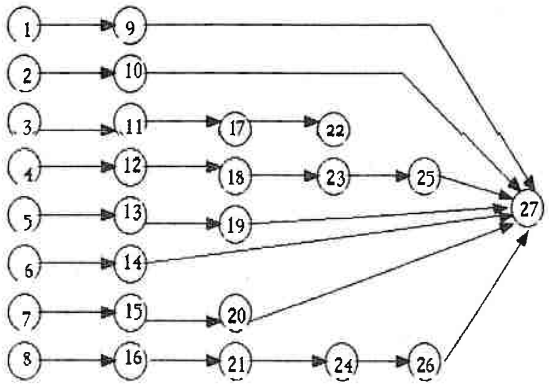



Figure (5-7) Graphe des systèmes (5-3-7) et (5-3-8)

Le tableau 5-2 donne le temps d'exécution par l'ordonnancement de l'algorithme dans ce cas :

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 6 | 43584 | 5,22 | 0,87 |
| 5 | 50376 | 4,47 | 0,9 |
| 4 | 59629 | 3,78 | 0,95 |
| 3 | 77749 | 2,9 | 0,97 |
| 2 | 112768 | 2 | 1 |

Tableau (5-2) Temps d'exécution pour les systèmes (5-3-7) et (5-3-8) dans le cas de tâches partitionnées

On peut remarquer que les taux d'accélération et d'occupation sont améliorés par rapport au tableau 5-1. Les tableaux 5-3 et 5-4

donnent les mêmes résultats dans le cas où $t_{mult} = 5\ t_{add}$.

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 6 | 133952 | 5,19 | 0,86 |
| 5 | 162112 | 4,29 | 0,86 |
| 4 | 187072 | 3,71 | 0,93 |
| 3 | 248768 | 2,79 | 0,93 |
| 2 | 347456 | 2 | 1 |

Tableau (5-3) Temps d'exécution dans le cas où $t_{mult} = 5\ t_{add}$ pour les tâches non partitionnées

| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 6 | 133952 | 5,19 | 0,86 |
| 5 | 155008 | 4,48 | 0,9 |
| 4 | 183872 | 3,78 | 0,95 |
| 3 | 239360 | 2,9 | 0,97 |
| 2 | 347392 | 2 | 1 |

Tableau (5-4) Temps d'exécution dans le cas où $t_{mult} = 5\ t_{add}$ pour les tâches partitionnées

Les taux d'accélération et d'occupation dans les tableaux 5-3 et 5-4 sont les mêmes que ceux des tableaux 5-1 et 5-2. On constate donc que l'ordonnancement du calcul est le même pour les deux types de processeurs .

Nous avons démontré au chapitre 2 que l'agrégation des tâches diminue le temps (overhead) du à l'organisation du calcul, mais également le taux d'accélération. Le graphe 5-6 est un exemple de ce type



d'agrégation. En effet, en supposant que l'intégration de l'équation différentielle constitue la tâche 1 :

$$\dot{X}_1^{(q+1)} = A_{11}\ X_1^{(q+1)} + A_{12}\ X_2^{(q)} - D_{11}\ \lambda_1^{(K)}$$

Nous avons éliminé un nombre important de possibilités du parallélisme qui existent dans l'exécution de cette tâche. Si l'intégration de cette équation se fait par la méthode de Kutta-Runge avec deux paramètres, on peut écrire :

$$K_{11} = h\,(A_{11} X_1^{(q+1)}(0) + A_{12}\ X_2^{(q)} - D_{11}\lambda_1^{(K)}(0)$$

$$K_{12} = h\,(A_{11} X_1^{(q+1)}(0) + A_{12}\ X_2^{(q)}(1) - D_{11}\lambda_1^{(K)}(1)$$

$$K_{13} = K_{12} + h\,A_{11}\,K_{11}$$

$$X_1(1) = X_1(0) + 1/2\,(K_{11} + K_{13})$$

On peut calculer $A_{11} X_1^{(q+1)}(0)$, $A_{12} X_2^{(q)}(0)$, $D_{11}\lambda_1^{(K)}(0)$, $A_{12} X_2^{(q)}$, $D_{11}\lambda_1^{(K)}(0)$ en parallèle.

Le graphe correspondant est le suivant :

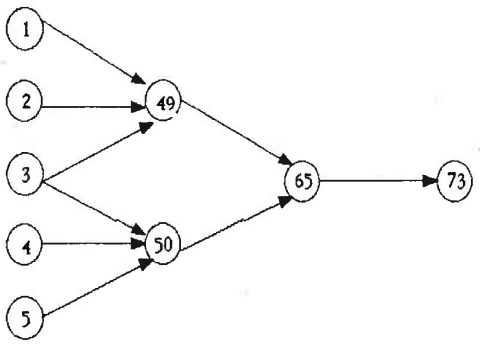



Figure (5-8) Graphe décomposé de la tâche 1

Chaque étape de la méthode de Kutta-Runge pour l'intégration des systèmes d'équations différentielles (5-3-7) et (5-3-8) est donc composée de graphes semblables au graphe (5-8) et disposés en parallèle. Le tableau 5-5 montre le résultat de l'ordonnancement du calcul d'une étape d'intégration des systèmes (5-3-7) et (5-3-8) par la méthode de Kutta-Runge pour le graphe composé des graphes 5-8. La durée du chemin critique dans le cas où $t_{mult}$ est 312 $t_{add}$. Cette durée s'obtient sur au maximum 48 processeurs, mais par l'ordonnancement du calcul, on peut l'obtenir sur 14 processeurs.



| Nombre de Processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 15 | 312 | 12,5 | 0,83 |
| 14 | 313 | 12,5 | 0,89 |
| 12 | 370 | 10,5 | 0,88 |
| 10 | 425 | 9,18 | 0,92 |
| 8 | 521 | 7,49 | 0,94 |
| 6 | 661 | 5,9 | 0,98 |
| 5 | 788 | 4,95 | 0,99 |
| 4 | 977 | 3,99 | 1 |
| 3 | 1310 | 2,98 | 0,99 |
| 2 | 1951 | 2 | 1 |

Tableau (5-5) Temps d'exécution pour le graphe 5-8

Une autre possibilité d'augmenter le parallélisme de l'algorithme est comme nous l'avons déjà expliqué le partitionnement temporel. Le parallélisme offre de grandes possibilités sur ce type de partitionnement.

En revanche, l'intégration des équations dans ce cas se fait par les méthodes itératives. Si le nombre d'itérations est très élevé, on risque alors de ne pas pouvoir augmenter le taux d'accélération, malgré l'utilisation d'un nombre plus grand de processeurs.

La décomposition en sous-problèmes avec conditions en un seul bout peut être effectué sur un seul niveau. Dans ce cas, le schéma de calcul est le suivant :

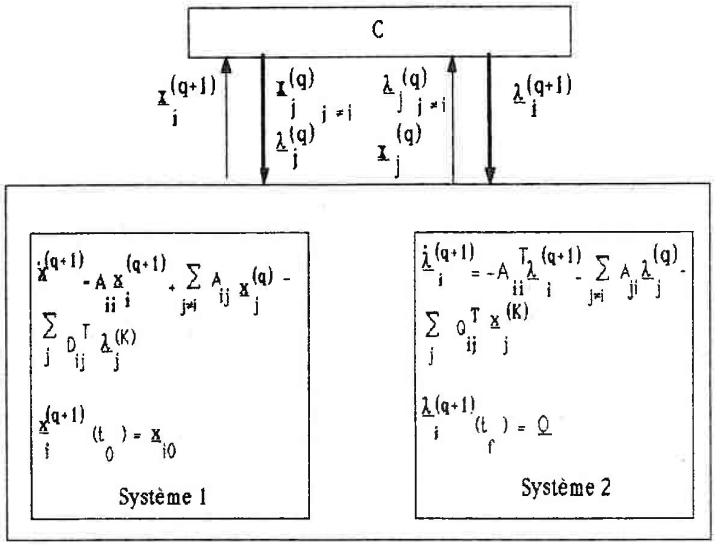



Figure (5-9) Schéma de décomposition à un niveau

Dans ce schéma, les valeurs de $\underline{\lambda}$ dans le système 1 et de $\underline{x}$ dans le système 2 sont renouvellées à chaque itération, l'organisation parallèle du calcul étant la même que dans le cas de la décomposition à deux niveaux. L'avantage de cette décomposition est la suppression du niveau 2, mais son inconvénient dans le calcul parallèle est l'augmentation du nombre d'échanges entre les sous-systèmes. En effet, dans la décomposition à deux niveaux, l'échange de $\underline{x}$ et de $\underline{\lambda}$ entre les systèmes 1 et 2 se fait une fois par itération dans le niveau supérieur. Par contre, dans le cas d'une décomposition à un niveau, cet échange se fait à chaque itération. Le choix du mode de décomposition doit tenir compte des conditions et de la rapidité de convergence, du temps nécessaire pour les calculs du niveau 2 et des possibilités du réseau



d'interconnexion entre les processeurs.

### 5-3-2 Décomposition en sous-problèmes avec conditions aux deux bouts

Ecrivons $\underline{Z}$ sous la forme :

Supposons $\underline{Z}^T = [\underline{x}_1^T, \underline{\lambda}_1^T, \ldots\ldots, \underline{x}_N^T, \underline{\lambda}_N^T]$

On peut alors décomposer le problème de commande optimale décrit par les équations (5-3-3) en N sous-problèmes d'optimisation locale appartenant à la classe des *"problèmes avec conditions aux deux bouts"*.

Le schéma de la décomposition est alors le suivant :

$$\begin{bmatrix} \dot{\underline{x}}_1 \\ \dot{\underline{\lambda}}_1 \\ \dot{\underline{x}}_2 \\ \dot{\underline{\lambda}}_2 \\ \\ \dot{\underline{x}}_N \\ \dot{\underline{\lambda}}_N \end{bmatrix} = \begin{bmatrix} A_{11} & -D_{11} & A_{12} & -D_{12} & & & A_{1N} & -D_{1N} \\ -Q_{11} & -A_{11} & -Q_{12} & -A_{12} & & & -Q_{1N} & -A_{N1}^T \\ A_{21} & -D_{21} & A_{22} & -D_{22} & & & & \\ -Q_{21} & -A_{21}^T & -Q_{22} & -A_{22}^T & & & & \\ & & & & & & & \\ A_{N1} & -D_{N1} & & & & & A_{NN} & -D_{NN} \\ -Q_{N1} & -A_{1N}^T & & & & & -Q_{NN} & -A_{NN}^T \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{\lambda}_1 \\ \underline{x}_2 \\ \underline{\lambda}_2 \\ \\ \underline{x}_N \\ \underline{\lambda}_N \end{bmatrix}$$

La décomposition parallèle du système s'effectue alors par la relaxation de la partie hors diagonale de la matrice concernée, à partir des schémas suivants : [5-4]

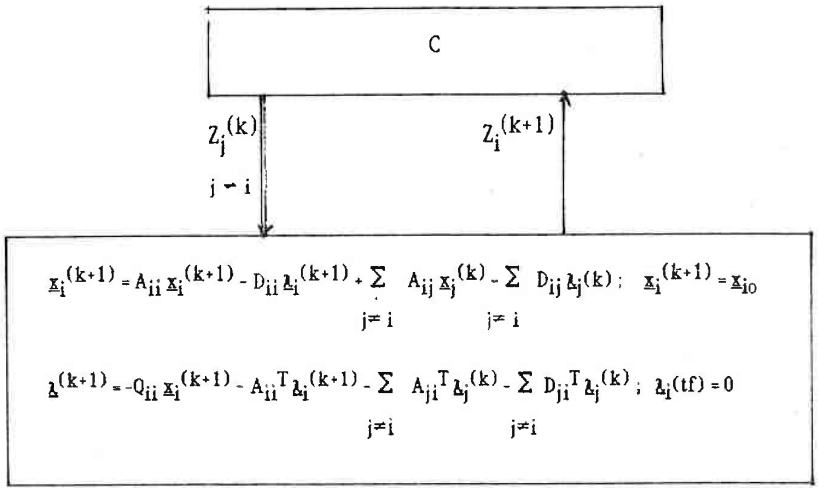



Figure (5-10) Décomposition parallèle avec conditions aux deux bouts

Le nombre de sous-systèmes dans ce schéma est la moitié de celui des sous-systèmes du schéma 5-2. Cette décomposition offre donc moins de parallélisme que la décomposition précédente. En plus, au niveau des sous-systèmes, l'optimisation consiste à résoudre un sous-problème local avec conditions aux deux bouts, qui nécessite plus de temps de calcul qu'un sous-problème avec conditions à un seul bout. En revanche, la décomposition 5-10 converge généralement plus vite que la décomposition 5-2. Le choix entre les deux décompositions doit donc tenir compte de tous ces éléments.

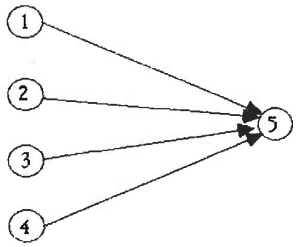
Exemple : Les équations d'optimalité du système de la figure (5-5) dans le cas d'une décomposition selon le schéma (5-10), sont les suivantes :



$$\begin{bmatrix} \dot{\underline{x}}_1 \\ \dot{\underline{\lambda}}_1 \\ \dot{\underline{x}}_2 \\ \dot{\underline{\lambda}}_2 \\ \dot{\underline{x}}_3 \\ \dot{\underline{\lambda}}_3 \\ \dot{\underline{x}}_4 \\ \dot{\underline{\lambda}}_4 \end{bmatrix} = \begin{bmatrix} A_{11} & -D_{11} & A_{12} & 0 & 0 & 0 & 0 & 0 \\ -Q_{11} & -A_{11}^T & 0 & -A_{21}^T & 0 & 0 & 0 & -A_{41}^T \\ A_{21} & 0 & A_{22} & -D_{22} & A_{23} & & 0 & 0 \\ 0 & -A_{12}^T & -Q_{22} & -A_{22}^T & & -A_{32}^T & 0 & 0 \\ 0 & 0 & A_{32} & 0 & A_{33} & -D_{33} & 0 & 0 \\ 0 & 0 & 0 & -A_{23}^T & -Q_{33} & -A_{33}^T & 0 & 0 \\ A_{41} & 0 & 0 & 0 & 0 & 0 & A_{44} & -D_{44} \\ 0 & 0 & 0 & 0 & 0 & -A_{34}^T & -Q_{44} & -A_{44}^T \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{\lambda}_1 \\ \underline{x}_2 \\ \underline{\lambda}_2 \\ \underline{x}_3 \\ \underline{\lambda}_3 \\ \underline{x}_4 \\ \underline{\lambda}_4 \end{bmatrix}$$

$$\underline{x}(t_0) = \underline{x}0 \qquad \underline{\lambda}(t_f) = \underline{0} \qquad (5\text{-}3\text{-}9)$$

Le graphe correspondant pour une itération de l'algorithme (5-3-9) est le suivant :




où les tâches 1 à 4 représentent le calcul de $\underline{z}i = \begin{pmatrix} \dot{\underline{x}}_i \\ \dot{\underline{\lambda}}_i \end{pmatrix}$ = 1 à 4 par relaxation des éléments hors-diagonale de la matrice du système (5-3-9).

Le temps minimum d'exécution de l'algorithme (5-3-9) dans le cas où tmult = $t_{add}$ est de 850 000 $t_{add}$ ; il est réalisable sur deux processeurs. En effet, 850 000 $t_{add}$ est le temps d'exécution de la tâche 4 pour un système d'équations différentielles à deux bouts d'ordre 16. Un processeur s'occupe

du calcul de cette tâche pendant qu'un autre processeur peut exécuter les tâches 1,2 et 3 l'une après l'autre.

Le temps minimum d'exécution dans le cas de l'algorithme avec les conditions à un bout est de 43 584 $t_{add}$ (voir tableau 5-1). Pour avoir le même temps d'exécution, la convergence du schéma 5-10 doit être 20 fois plus rapide que la convergence totale du schéma 5-2.

On peut accroître le parallélisme du schéma 5-10 par la résolution itérative des équations différentielles appropriées. Plusieurs schémas d'itérations sont alors envisageables :

a) Relaxation de $\lambda_i^{(K+1)}$ dans l'équation d'état et de $x_j^{(K+1)}$ dans l'équation adjointe, puis renouvellement de ces deux valeurs à la fin du calcul de $x$ et $\lambda$. C'est le cas de la décomposition à deux niveaux représentée par le schéma 5-2.

b) Même type de relaxation, mais renouvellement de ces valeurs à chaque itération. C'est le cas de la décomposition à un niveau représentée par le schéma (5-9).

c) Décomposition temporelle de chaque équation, intégration de la première équation dans le sens direct et de la deuxième équation à rebours, puis mise en oeuvre de l'algorithme présenté en (5-2-2).

Le choix d'une méthode itérative dépend naturellement du nombre de processeurs disponibles et de la vitesse de convergence. Avec les méthodes d'ordonnancement présentées au chapitre 2, on peut calculer le



temps d'exécution de chaque algorithme sur les processeurs disponibles.

On peut choisir ainsi la méthode itérative qui convient le mieux à un problème donné.

### 5-3-3 Intégration en parallèle de l'équation de Riccati

Dans le cas de la commande optimale en boucle fermée, la résolution du système (5-3-3) passe par l'intégration d'un système quadratique de dimension n(n+1)/2 correspondant à une équation différentielle avec condition finale de type Riccati, soit :

$$\dot{P}(t) = P(t)\,B(t)\,R^{-1}(t)\,B^{T}(t) - 1^{T}(t)\,P(t) - P(t)\,A(t) - Q(t) \qquad P(t_f) = 0 \qquad (5\text{-}3\text{-}10)$$

La loi de commande optimale est alors déterminée par :

$$U(t) = -\,R^{-1}(t)\,B^{T}(t)\,P(t)\,x(t) \qquad t \in [t_0, t_f] \qquad (5\text{-}3\text{-}11)$$

Pour la résolution parallèle du système (5-3-10), plusieurs schémas sont envisageables et sont étudiées par la suite.

a) Partitionnement de la matrice P(t)

Si on divise l'intervalle $[t_0, t_f]$ en n pas d'intégration, alors en intégrant à rebours de l'équation (5-3-10), on peut calculer la matrice P(t) par une méthode classique. Les opérations à effectuer à chaque pas

d'intégration sont des multiplications matricielles. On peut donc partitionner la matrice P(t) en blocs de sous-matrices et utiliser la méthode de multiplication parallèle de matrices présentée au chapitre 4. Les sous-systèmes dans ce cas doivent être synchronisés à la fin de chaque pas d'intégration.

Supposons par exemple qu'on utilise la méthode la méthode de Kutta-Runge présentée précédemment. En divisant l'intervalle $[t_0, t_f]$ en n pas d'intégration, on a :

$$k_1 = h\,[P_j B_j R_j R_j^{-1} B_j^T P_j - A_j^T P_j - P_j A_j - Q_j] \qquad j = 0 \text{ à } n-1 \qquad P_n = 0$$

$$k_2 = h\,[(P_j+k_1) B_{j-1} R_{j-1}^{-1} B_{j-1}^T (P_j+k_1) - A_{j-1}^T (P_j+k_1) - (P_j+k_1) A_{j-1} - Q_{j-1}]$$

$$P_{j-1} = P_j + \frac{1}{2}(k_1 + k_2)$$

à chaque pas d'intégration, on peut donc partitionner les matrices concernées et effectuer le calcul de $K_1$, $K_2$ et $P_j$ en parallèle.

La figure 5-11 présente le graphe de calcul dans le cas où l'on a partitionné les matrices de la façon suivante :



$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \qquad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix} \qquad B$$

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \qquad K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \qquad R = \begin{bmatrix} R_{11} & 0 \\ 0 & R_{22} \end{bmatrix}$$

On suppose de plus que les éléments des matrices A, B, R et Q sont constants.

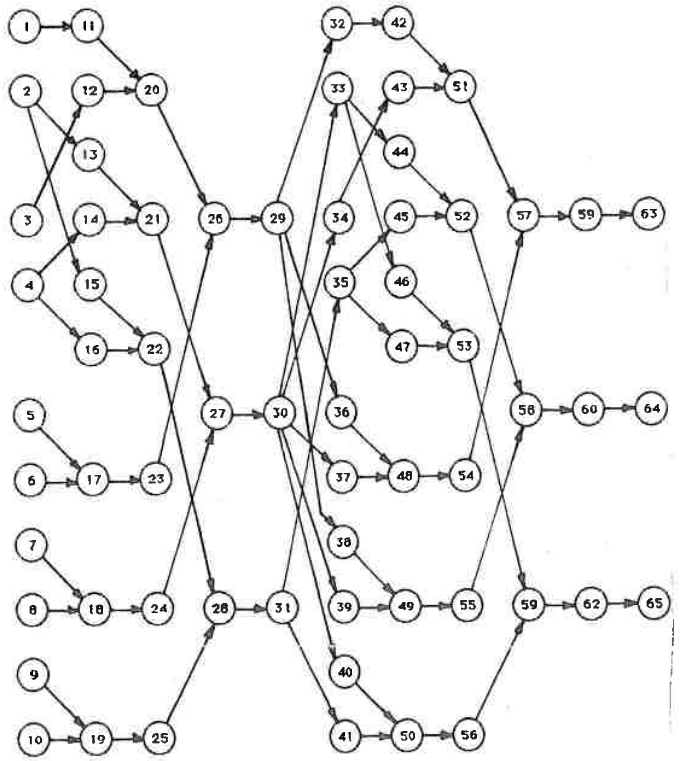



Figure (5-11) Graphe de l'intégration de l'équation de Riccati

Les calculs correspondants aux différentes tâches de ce graphe, sont les suivantes :

tâches 1 à 4 : calcul de $P_{ij} . (BR^{-1}B^T)jj$ i, j = 1 à 2

tâches 5 à 10 : calcul de $A_{ik}{}^T . P_{Kj}$ i = 1 à 2 j = i à 2 K = 1 à 2

tâches 11 à 28 : sommes de différentes sous-matrices pour former les sous-matrices $(K_1)ij$

tâches 29 à 31 : calcul de $(P_j + K_1)$

tâches 32 à 62 : calcul de $K_2$, effectué de façon analogue à celui de $K_1$

tâches 63 à 65 : calcul de $P_{j-1}$

Dans le cas général où la matrice P est décomposée en $N^2$ sous-matrices, la durée du chemin critique du graphe 5-11 correspond à :

- 4 multiplications de sous-matrices
- 2N+3 additions de sous-matrices

Cette durée du chemin critique s'obtient sur au maximum

$$N^2 + \frac{N^2(N+1)}{2} \quad \text{processeurs}$$

EQUATION DE RICCATI DANS LE CAS D'HORIZON INFINI

Si la paire (A,B) est commandable, alors la solution P(t) de l'intégration rétrograde (5-3-10) converge pour (t-tf) ----------> $\infty$, vers la solution de l'équation algébrique associée [ 5-8] :



$$P B R^{-1} B^T P - A^T P - P A - Q = 0 \qquad (5\text{-}3\text{-}12)$$

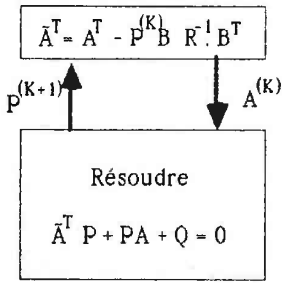
Pour résoudre cette équation, on peut utiliser la méthode de Newton qui consiste à l'écrire sous la forme suivante :

$$(-A + P B R^{-1} B^T)P - PA - Q = 0$$

En supposant que $-\tilde{A}T = -A^T + P B R^{-1} B^T$, on a :

$$\tilde{A}^T P + PA + Q = 0 \qquad (5\text{-}3\text{-}13)$$

On peut donc relaxer la matrice P pour le calcul de A et résoudre l'équation de Lyapunov (5-3-13). Le schéma du calcul est alors le suivant :




Pour la solution en parallèle de (5-3-13), on peut utiliser le produit de Krohnecker défini par :

$$A_{mxn} \otimes B_{ixp} = (A \otimes B)_{mixnp} = \begin{matrix} a_{11}B & \ldots & a_{1n}B \\ a_{m1}B & \ldots & a_{mn}B \end{matrix}$$

Le système (5-3-13) s'écrit alors sous la forme suivante :

$$[\check{A}^T \otimes I|_n + I|_n \otimes A^T] \quad \underline{p} = -\underline{q} \qquad (5\text{-}3\text{-}14)$$

où $\underline{P} = (\underline{P}_{11}\ldots\underline{P}_{1n}, \underline{P}_{12}\ldots\underline{P}_{2n}, \ldots\ldots\ldots\ldots \underline{P}_{1n}\ldots\underline{P}_{nn})^T$

$\underline{q} = (\underline{q}_{11}\ldots\underline{q}_{1n}, \underline{q}_{12}\ldots\underline{q}_{2n}, \ldots\ldots\ldots\ldots \underline{q}_{1n}\ldots\underline{q}_{nn})^T$

L'équation (5-3-14) est donc une équation linéaire d'ordre $n^2$ que l'on peut résoudre en parallèle avec les méthodes présentées au chapitre 3.

En posant :

$D =$

| | | | |
|---|---|---|---|
| $\tilde{A}_{11} \times I_n + A$ | $\tilde{A}_{12} \times I_n$ | | $\tilde{A}_{1n} \times I_n$ |
| $\tilde{A}_{21} \times I_n$ | $\tilde{A}_{22} \cdot I_n + A$ | | |
| | | $\ddots$ | |
| $\tilde{A}_{n1} \cdot I_n$ | | | $\tilde{A}_{nn} \cdot I_n + A$ |

la matrice D est déjà décomposée en $n^2$ sous-matrices d'ordre $n^2$. Si l'on



utilise la méthode de Jacobi par bloc pour la solution de cette équation, on a alors :

$$\underline{P}_1^{(l+1)} = D_{11}^{-1} (-\underline{q}_1 - A_{12}^{(K)} \underline{P}_2^{(l)} \ldots\ldots - A_{1n}^{(K)} \underline{P}_n^{(l)})$$

$$\underline{P}_2^{(l+1)} = D_{22}^{-1} (-\underline{q}_2 - A_{21}^{(K)} \underline{P}_1^{(l)} \ldots\ldots - A_{2n}^{(K)} \underline{P}_n^{(l)})$$

---

---

$$\underline{P}_n^{(l+1)} = D_{nn}^{-1} (-\underline{q}_n - A_{n1}^{(k)} \underline{P}_1^{(l)} \ldots\ldots\ldots - A_{n\,n-1}^{(k)} \underline{P}_{n-1}^{(l)})$$

Le calcul parallèle de la méthode de Jacobi se déroule comme suit :

Première itération :

- calcul de $D_{ii}^{-1}$ $\quad i = 1$ à n en parallèle sur n processeurs
- calcul de $D_{ii}^{-1} A_{ij}$ $\quad i = 1$ à n, $j = 1$ à n $\quad j \neq i$ et $D_{ii}^{-1} Q_i$ en parallèle sur $n^2$ processeurs

Pour chaque itération suivante :

- calcul de $(D_{ii}^{-1} A_{ij}) P_j$ sur $n(n-1)$ processeurs
- $\Sigma\ (D_{ii}^{-1} A_{ij}) P_j^{(l)}$ sur $n^2/2$ processeurs en $(\log_2 n)$ étapes

La figure (5-12) présente le graphe de calcul pour n = 4

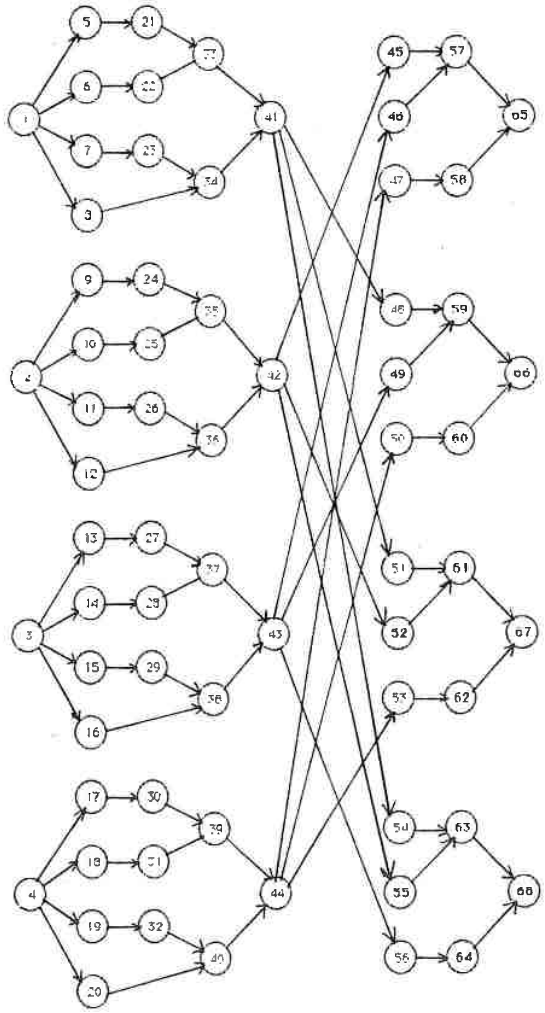


Figure (5-12) Graphe correspondant à la résolution du système (5-3-14) par la méthode de Jacobi par bloc.

Le tableau (5-6) présente la définition d'une partie des tâches du



graphe 5-12. Les autres tâches sont semblables aux tâches définies dans le tableau (5-6).

| N° | Nature de la tâche | N° | Nature de la tâche | N° | Nature de la tâche |
|---|---|---|---|---|---|
| 1 | $D_{11}^{-1}$ | 22 | $-m_{13}\, p_3^{(l)}$ | 46 | $-m_{13}\, p_3^{(l+1)}$ |
| 5 | $m_{12} = D_{11}^{-1} A_{12}$ | 23 | $-m_{14}\, p_4^{(l)}$ | 47 | $-m_{14}\, p_4^{(l+1)}$ |
| 6 | $m_{13} = D_{11}^{-1} A_{13}$ | 33 | $-m_{12}\, p_3^{(l)} - m_{13}\, p_3^{(l)}$ | 57 | $-m_{12}\, p_2^{(l+1)} - m_{13}\, p_3^{(l+1)}$ |
| 7 | $m_{14} = D_{11}^{-1} A_{14}$ | 34 | $b_1 - m_{14}\, p_4^{(l)}$ | 58 | $b_1 - m_{14}\, p_4^{(l+1)}$ |
| 8 | $b_1 = -D_{11}^{-1} q_1$ | 41 | $p_1^{(l+1)}$ | 65 | $p_1^{(l+2)}$ |
| 21 | $m_{12}\, p_2^{(l)}$ | 45 | $-m_{12}\, p_2^{(l+1)}$ | | |

Tableau (5-6) : Définition des tâches du graphe (5-12)

Si L est le nombre d'itérations pour la convergence de la méthode de Jacobi, la durée d'exécution pour la solution du système 5-3-14 est le temps nécessaire pour :

- l'inversion d'une matrice n. n +
- la multiplication d'une matrice par un nombre réel +
- L multiplications d'une matrice nxn par un vecteur n + $\log_2 n$ additions de deux vecteurs de dimension n.

Dans le cas où les matrices Q et R sont symétriques, la matrice P de l'équation de Riccati est symétrique. Le nombre d'inconnues dans l'équation 5-3-14 est donc $\frac{n(n+1)}{2}$. Dans ce cas la dimension de la matrice D est

également $\frac{n(n+1)}{2}$. On peut alors partitionner cette matrice et utiliser les méthodes de résolution parallèle d'équations linéaires expliquées au chapitre 3.

## 5-4 COMMANDE OPTIMALE DES SYSTEMES LINEAIRES DISCRETS

Considérons le problème d'optimisation :

Calculer le minimum de J par rapport à $\underline{U}_K$ où :

$$J = \tfrac{1}{2}\Sigma\,(\underline{x}^T_{K+1}\, Q_{K+1}\, \underline{x}_{K+1} + \underline{U}_K{}^T\, R_K\, \underline{U}_K) \qquad (5\text{-}4\text{-}1)$$

sous

$$\underline{x}_{K+1} = A_K\, \underline{x}_K + B_K\, \underline{U}_K \;;\; K = o, 1, \text{---}, K_f\text{-}1 \;;\; x_0 \text{ fixé}$$

où $xK \in Rn$ caractérise l'état global du système à l'instant discret K, $\underline{U}_K \in Rm$, le vecteur de commande et où AK, BK, RK, QK sont des matrices de dimensions appropriées.

Nous supposons de plus que les matrices $Q_{K+1}$ sont définies semi-positives et les matrices $R_{K+1}$ sont définies positives pour K=0....., $K_f$-1.

Les conditions de stationnarité du problème 5-4-1 sont alors les suivantes :



$$\begin{aligned}
&\underline{x}_K - A_{K-1}\underline{x}_{K-1} - B_{K-1}\,\underline{U}_{K-1} = 0 && K = 1,\ldots\ldots, K_f \qquad x0 \text{ donné}\\
&\underline{\lambda}_K - Q_K\,\underline{x}_K - A^T_K\,\underline{\lambda}_{K+1} = 0 && K = 1,\ldots\ldots, K_f\text{-}1\\
&\underline{\lambda}_{Kf} - Q_{Kf}\,\underline{x}_{Kf} = 0 && (5\text{-}4\text{-}2)\\
&R_K\,\underline{U}_K + B^T_K\,\underline{\lambda}_{K+1} = 0 && K = 0,\ldots\ldots, K_f\text{-}1
\end{aligned}$$

En remplaçant $\underline{U}_K = R^{-1}_K\, B^T_K\, \underline{\lambda}_{K+1}$ dans la première équation , on obtient :

$$\begin{aligned}
&\underline{x}_K - A_{K-1}\underline{x}_{K-1} + B_{K-1}\,R^{-1}_{K-1}\,B^T_{K-1}\,\underline{\lambda}_K = 0 && K=1,\ldots\ldots,K_f \qquad x_0 \text{ fixe}\\
&\underline{\lambda}_K - Q_K\,\underline{x}_K - A_K\,\underline{\lambda}_{K+1} = 0 && K=1, \ldots\ldots, K_f\text{-}1\\
&\underline{\lambda}_{Kf} - Q_{Kf}\,\underline{x}_{Kf} = 0 && (5\text{-}4\text{-}3)
\end{aligned}$$

Pour la résolution parallèle du système (5-4-3), plusieurs décompositions sont envisageables.

### 5-4-1 Décomposition spatiale

Comme dans le cas des systèmes continus, on peut décomposer les matrices A, B, Q, et R par blocs et décomposer de la même manière les vecteurs $\underline{x}_K$ et $\underline{\lambda}_K$. Dans la première équation du système (5-4-3,) on relaxe les variables $\underline{\lambda}_K$ et on calcule $\underline{x}_K$. Dans la deuxième et troisième équations, on fixe les variables $\underline{x}_K$ et on calcule $\underline{\lambda}_K$. Une itération de cet algorithme est exprimée alors de la façon suivante :

$$\underline{x}_K^{(q+1)} = A_{K-1}\,\underline{x}^{(q+1)}_{K-1} - B_{K-1}\,R_{K-1}^{-1}\,B^T_{K-1}\,\underline{\lambda}_K^{(q)} \qquad \underline{x}_0^{(q+1)} = \underline{x}_0$$

$$\underline{\lambda}_K^{(q+1)} = Q_K\,\underline{x}_K^{(q)} + A K^T\,\underline{\lambda}_{K+1}^{(q+1)} \qquad \underline{\lambda}_{Kf}^{(q+1)} = Q_{Kf}\,\underline{x}_{Kf}^{(q)} \qquad (5\text{-}4\text{-}4)$$

Cette décomposition ressemble essentiellement à la décomposition présentée en 5-3-1.

Exemple :

$$\text{Soit } x_K = A_{K-1}x_{K-1} - D_{K-1}\,\underline{\lambda}_K$$

$$\underline{\lambda}_K = Q_K\,x_K + A_K\,\underline{\lambda}_{K+1} \qquad (5\text{-}4\text{-}5)$$

où A, D et Q ont les formes suivantes :



| | | | | |
|---|---|---|---|---|
| $A_{11}$ 4x4 | $A_{12}$ 4x6 | $A_{13}$ 4x6 | 0 | 0 |
| 0 | $A_{22}$ 6x6 | 0 | 0 | |
| $A_{31}$ | 0 | $A_{33}$ 4x4 | 0 | 0 |
| 0 | $A_{42}$ | 0 | $A_{44}$ 4x4 | 0 |
| $A_{51}$ | 0 | 0 | 0 | $A_{55}$ 8x8 |

$Q_K$ et $D_K$ Bloc diagonale

On peut alors écrire :

$$\underline{x}_{1(K)}^{(q+1)} = A_{11}\,\underline{x}_{1(K-1)}^{(q+1)} + A_{12}\,\underline{x}_{2(K-1)}^{(q+1)} + A_{13}\,\underline{x}_{3(K-1)}^{(q+1)} - D_{11}\,\underline{\lambda}_{1(K)}^{(q)}$$

---

$$\underline{x}_5^{(q+1)} = A_{41}\,\underline{x}_{1(K-1)}^{(q+1)} + A_{55}\,\underline{x}_{5(K)}^{(q+1)} - D_{55}\,\underline{\lambda}_{1(K)}^{(q)}$$

$$\underline{\lambda}_{1(K)}^{(q+1)} = Q_{11}\,\underline{x}_{1\,(K)}^{(q)} + A_{11}^T\,\underline{\lambda}_{1\,(K+1)}^{(q+1)} - A_{31}^T\,\underline{\lambda}_{3\,(K+1)}^{(q+1)} - A_{51}^T\,\underline{\lambda}_{5\,(K+1)}^{(q+1)}$$

---

$$\underline{\lambda}_{5(K)}^{(q+1)} = Q_{55}\,\underline{x}_{5(K)}^{(q)} + A_{24}^T\,\underline{\lambda}_{2(K)}^{(q+1)} + A_{55}^T\,\underline{\lambda}_{5(K)}^{(q+1)}$$

La figure 5-13 présente la graphe correspondant à ce calcul.

Les tâches 1 à 5 représentent les calculs de $\underline{x}_{1(K)}$.....$\underline{x}_{5(K)}$ et les tâches 5 à 6 ceux de $\underline{\lambda}_{1(K)}$..... $\underline{\lambda}_{5(K)}$ La tâche 11 est une tâche de synchronisation de calcul : en effet, tous les vecteurs $\underline{x}_{i(K)}$ ou de $\underline{\lambda}_{j(K)}$ doivent être calculés avant le calcul de $\underline{x}_{i(K+1)}$ ou $\underline{\lambda}_{j(K+1)}$.

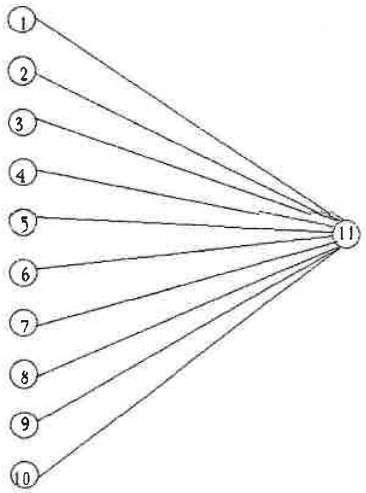



Figure (5-13) Graphe du calcul du système (5-4-6)

La durée du chemin critique du graphe 5-13 dans le cas où $t_{mult} = t_{add}$ est de 344 $t_{add}$. Cette durée s'obtient sur au maximum 10 processeurs. Le tableau 5-7 donne le temps d'exécution par ordonnancement de l'algorithme (5-4-6- en fonction du nombre de processeurs.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 7 | 344 | 5,8 | 0,83 |
| 6 | 358 | 5,56 | 0,93 |
| 5 | 420 | 4,74 | 0,95 |
| 4 | 528 | 3,78 | 0,95 |
| 3 | 698 | 2,85 | 0,95 |
| 2 | 1012 | 1,97 | 0,98 |

Tableau (5-7) Temps d'exécution pour le graphe (5-13)



Si L est le nombre d'itérations nécessaires à la convergence de l'algorithme (5-4-5), le temps total pour la résolution du système (5-4-5) est de :

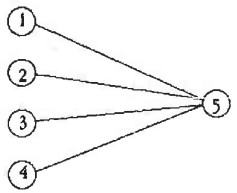
$$L \cdot K_f \cdot T_{exec}$$

où $K_f$ est l'horizon d'optimisation et $T_{exec}$ est le temps calculé dans le tableau 5-7. Ainsi par exemple, pour L = 50 et $K_f$ = 40, le temps minimum d'exécution est de 688 000 $t_{add}$.

On peut augmenter le degré de parallélisme de cet algorithme par le calcul parallèle des composants de chaque vecteur. On calcule par exemple :

$A_{11}\underline{x}_{1\,K-1}^{(q+1)}$, $A_{12}\underline{x}_{2\,K-1}^{(q+1)}$, $A_{13}\underline{x}_{3\,K-1}^{(q+1)}$ et $D_{11}\underline{\lambda}_{1K}^{(q)}$ en parallèle et on fait la somme de ces 4 vecteur pour le calcul de $x_{iK}^{(q+1)}$

Ainsi la tâche 1 du graphe 5-13 se décompose sous la forme suivante :




On peut ainsi décomposer chaque tâche du graphe 5-13 de la même façon.

Le résultat donne un graphe comportant 32 branches parallèles dont la durée du chemin critique est 136 $t_{add}$. Le tableau 5-8 donne le temps d'exécution de l'algorithme (5-4-6) en fonction du nombre de processeurs dans le cas de cette décomposition des tâches.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 16 | 136 | 14,7 | 0,92 |
| 14 | 152 | 13,1 | 0,94 |
| 12 | 174 | 11,5 | 0,96 |
| 10 | 206 | 9,7 | 0,97 |
| 8 | 254 | 7,86 | 0,98 |
| 6 | 336 | 5,94 | 0,99 |
| 5 | 402 | 4,96 | 0,99 |
| 4 | 500 | 3,99 | 1 |
| 3 | 668 | 2,99 | 1 |
| 2 | 998 | 2 | 1 |

Tableau 5-8 Temps d'exécution dans le cas de décomposition des tâches pour le graphe 5-13

Le temps minimum d'exécution donné dans le tableau 5-8 est 2,5 fois plus petit que celui donné dans le tableau 5-7. On peut remarquer également que pour un même nombre de processeurs, le temps d'exécution représenté dans le tableau 5-8 est inférieur à ceux présentés dans le tableau 5-7. La décomposition des tâches augmente donc le rendement du système, mais comme nous l'avons déjà expliqué au chapitre 2, le temps de transfert et l'effet des conflits peuvent diminuer ce rendement. Nous

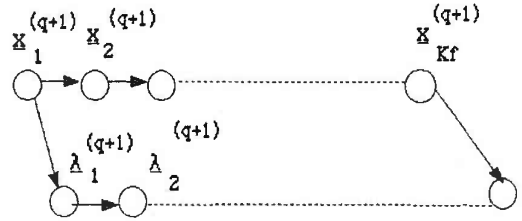


étudierons cet effet au chapitre 6.

Dans le cas où l'on dispose d'une structure pipe-line, on peut modifier l'algorithme (5-4-4) de la façon suivante :

$$x_K^{(q+1)} = A_{K-1}\, x_{K-1}^{(q+1)} - B_{K-1}\, R^{-1}{}_{K-1}\, B_{K+1}{}^T \lambda_K^{(q)}$$
$$\lambda_K^{(q+1)} = A_K\, x_K^{(q+1)} + A_K{}^T\, \lambda_{K+1}^{(q)} \qquad (5\text{-}4\text{-}7)$$

Le parallélisme de l'algorithme( 5-4-7) est le même que celui de (5-4-4). En effet, le graphe de calcul pour l'algorithme (5-4-7) est le suivant :




Si l'horizon d'optimisation est assez grand, le temps de démarrage (start-up) de l'algorithme est négligeable par rapport au temps total d'exécution. Les deux algorithmes (5-4-4) et (5-4-7) ont donc pratiquement les mêmes temps d'exécution pour une itération, mais leur taux de convergence peut être différent pour un problème donné. Dans le cas du système représenté par le système (5-4-5) l'utilisation de la structure pipe-line conduit au graphe de la figure 5-14.

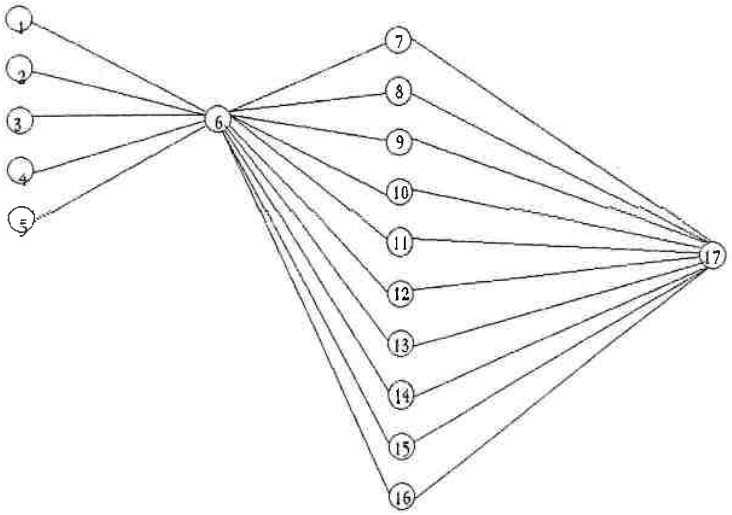



Figure (5-14) Graphe de l'algorithme (5-4-7)

Sur cette figure, les tâches 1 à 5 représentent le calcul de $x_{iK}$, les tâches 7 à 11 le calcul de $\underline{x}_{iK+1}$ et les tâches 12 à 16 le calcul de $\underline{\lambda}_{iK}$. Les tâches 6 et 17 sont des tâches de synchronisation.La durée d'exécution du graphe 5-14 entre deux tâches de synchronisation 6 et 17 est la même que la durée présentée au tableau 5-6.

A la fin de chaque itération, on peut inverser la structure pipe-line; on calcule $\underline{\lambda}_K$ de la deuxième équation et on le remplace dans la première équation pour le calcul de $\underline{x}_K$. Cette alternance de calculs équilibre les itérations et peut améliorer la convergence de l'algorithme.

Une autre méthode qui peut aboutir à une meilleure convergence de l'algorithme (5-4-4) consiste à relaxer les valeurs $\underline{\lambda}_K$ dans la première

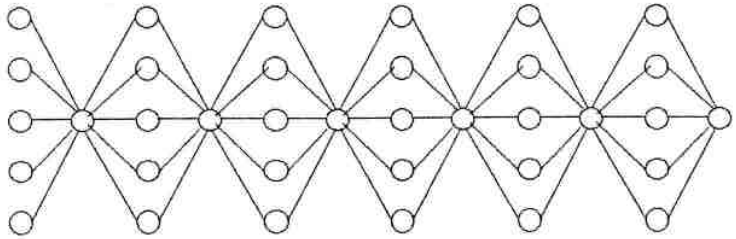


équation et de calculer successivement $\underline{\lambda}_K$ à partir de la deuxième équation en utilisant les $\underline{x}_K$ calculés dans la première équation. Le schéma de calcul est alors le suivant :

$$\underline{x}_{K+1}^{(q+1)} = A_{K-1}\,\underline{x}_{K-1}^{(q+1)} - B_{K-1}\,R_{K-1}^{-1}\,B_{K-1}^{T}\,\underline{\lambda}_K^{(q)} \quad x_0 \text{ fixé} \quad K=1 \text{ à } K_f$$

$$\underline{\lambda}_K^{(q+1)} = Q_{K-1}\,\underline{x}_K^{(q+1)} + A_K^{T}\,\underline{\lambda}_{K+1}^{(q+1)} \qquad \underline{\lambda}_{Kf}^{(q+1)} = Q_{Kf}\,\underline{x}_{Kf}$$

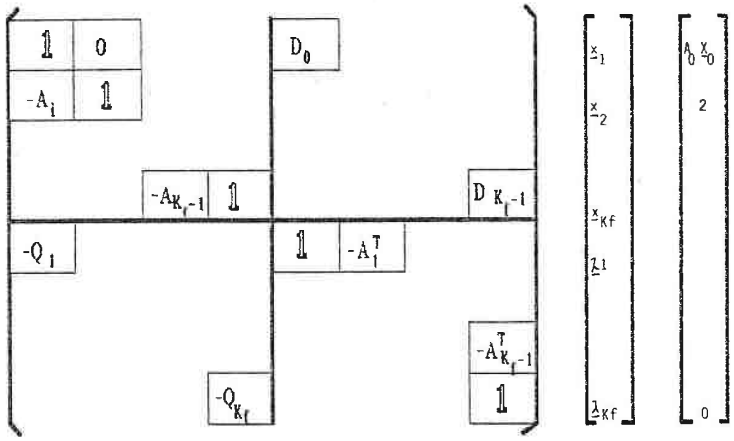
Dans le cas du système (5-4-5,) le graphe correspondant à cet algorithme est le suivant :



Cette méthode offre moins de possibilités de parallélisme, mais dans le cas où l'on dispose d'un nombre limité de processeurs et à condition qu'elle converge plus vite que l'algorithme 5-4-4, il est avantageux d'utiliser cette méthode à la place de l'algorithme 5-4-4.

### 5-4-2 Décomposition temporelle

On peut écrire le système (5-4-3) sous la forme suivante :



(5-4-9)

Le système 5-4-9 est alors un système d'équations linéaires d'ordre $K_f$. n. On peut donc utiliser les méthodes de calcul parallèle appliquées à la résolution des systèmes linéaires présentées au chapître 3. Etant donnée la structure du système (5-4-9) qui présente notamment une matrice quasi bloc-diagonale, les méthodes directes de résolution des systèmes linéaires ne dégagent que peu de possibilités de parallélisme. On doit donc résoudre ce système par des méthodes itératives.

a) Méthode de Gauss-seidel

L'application de la méthode de Gauss-seidel pour la résolution du système (5-4-9) donne :

$$\underline{x}_1^{(q+1)} = A_0\ \underline{x}_0 - D_0\ \underline{\lambda}_1^{(q)}$$

---

$$\underline{x}_{Kf}^{(q+1)} = A_{Kf-1}\ \underline{x}_{Kf-1}^{(q+1)} - D_{Kf}\ \underline{\lambda}_{Kf}$$



$$\underline{\lambda}_1^{(q+1)} = Q_1\ \underline{x}_1^{(q+1)} + A_1^T\ \underline{\lambda}_2^{(q)}$$

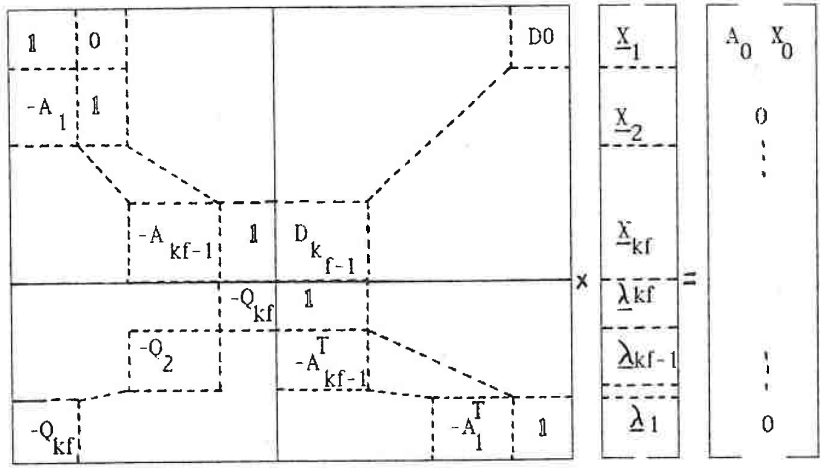
---

$$\underline{\lambda}_{Kf}^{(q+1)} = Q_{Kf}\ \underline{x}_{Kf}^{(q+1)}$$

(5-4-10)

Cette formulation est la meme que celle de l'algorithme (5-4-7) étudié précédemment.

L'algorithme 5-4-8 correspond également à l'application de la méthode de Gauss-Seidel, mais pour la formulation suivante :



(5-4-10

b) Méthode de Jacobi par blocs

On peut formuler de différentes façons la méthode de Jacobi par blocs, selon la matrice bloc-diagonale choisie pour la résolution des systèmes (5-4-10). D'une manière générale, supposons qu'on partitionne les $[\underline{x}_1,\underline{x}_2,\ldots\underline{x}_{Kf}]^T$, $[\underline{\lambda}_1,\underline{\lambda}_2,\ldots\underline{\lambda}_{Kf}]$ en S sous-vecteurs

$[\underline{x}_1,\underline{x}_2,\ldots\underline{x}_{K1}, \underline{x}_{K+1},\ldots, \underline{x}_{K1}\underline{x}_{Kf}]$ $[\underline{\lambda}_1, \underline{\lambda}_2,\ldots \underline{\lambda}_{K1}\underline{\lambda}_{K+1},\ldots \underline{\lambda}_{K2},\ldots \underline{\lambda}_{Kf}]$

On peut alors partitionner les matrices des systèmes (5-4-9) et (5-4-10) de la même façon et appliquer la méthode de Jacobi. L'algorithme de Jacobi dans ce cas offre 25 voies parallèles. Si on applique également la décomposition spatiale, ce nombre est multiplié par le nombre des voies parallèles offertes par la décomposition parallèle. On peut remarquer qu'il existe un très grand choix pour la résolution parallèle des systèmes (5-4-9) et (5-4-10). Le choix d'un nombre S élevé augmente la possibilité de parallélisme, mais ralentit la convergence de l'algorithme. Ce choix est donc fonction du nombre de processeurs disponibles et du taux de convergence de l'algorithme. Nous étudions ici deux cas particuliers :

*b-1) cas où S = 1*

Cet algorithme correspond à l'algorithme 5-4-4 présenté précédemment. On applique uniquement la décomposition spatiale.

*b-2) cas où S = $K_f$*

Ca cas offre un plus grand degré de parallélisme.

Par exemple, Pour $K_f = 40$ et dans le cas d'une décomposition spatiale complète, on obtient 1280 voies parallèles pour la résolution du système (5-4-5).

### 5-4-3 Décomposition de l'équation de Riccati

Dans le cas discret, l'équation réccurente de Riccati s'écrit sous la forme :



$$P_K = A_K^T [P_{K+1} - P_{K+1} B_K B_K^T P_{K+1} B_K + P_K]^{-1} A_K + Q_K \qquad (5\text{-}4\text{-}11)$$

Cette forme est trés proche de celle du filtre de Kalman. On peut donc utiliser les mêmes méthodes de calcul parallèle.

Si Kf — ∞, la matrice P est la solution de l'équation algébrique de Riccati suivante : [5-9]

$$P = A^T PA - A^T PB (B^T PB + R)^{-1} B^T PA + Q \qquad (5\text{-}4\text{-}12)$$

La résolution parallèle de cette équation peut être effectuée de différentes manières :

a) Mise en oeuvre de l'algorithme de Hewer [ 5-10], [5-11]

Cet algorithme est défini comme suit :

$$W_N = \sum_{I=1}^{N} A^I B R^{-1} B^T (A^I)^T \qquad N > \dim(A) - \mathrm{rang}(B) + 1$$

$$G_0 = R^{-1} B^T (A^T)^N W_N^{-1} A^{N+1}$$

$$\tilde{A}_j = A - B G_j$$

$$M_j = \tilde{A}_j^T M_j \tilde{A}_j^T = G_j^T R G_j + Q \qquad j = 0,1$$

$$G_j = (B^T M_{j-1} B + R)^{-1} B^T M_{j-1} A \qquad j = 1,2 \ldots$$

Pour le calcul de $W_N$, nous supposons $N = 2^K$ où K est un nombre entier. On peut alors écrire :

$$W_N = W_{N/2} + \sum_{l=N/2+1}^{N} A^l B R^{-1} B^T (A^l) = W_{N/2} + A^{N/2} \left( \sum_{l=1}^{N/2} A^l B R^{-1} B^T (A^l) \right) T (A^{N/2})^T$$

d'où

$$W_N = W_{N/2} + A^{N/2} W_{N/2} (A^{N/2})^T = W_{2K-1} + A^{2K-1} (A^{K-1})^T$$

En continuant cette procédure, on obtient la formulation suivante :

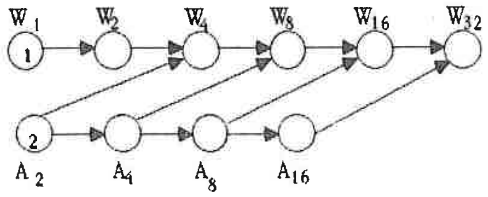
$$W_2 = W_2 + A\ W_1\ A^T$$

$$W_4 = W_2 + A^2\ W_2\ (A^2)^T$$

---

$$W_2^K = W_{K-1} + A^{K+1}\ W_{K-1}\ (A^{K-1})^T$$

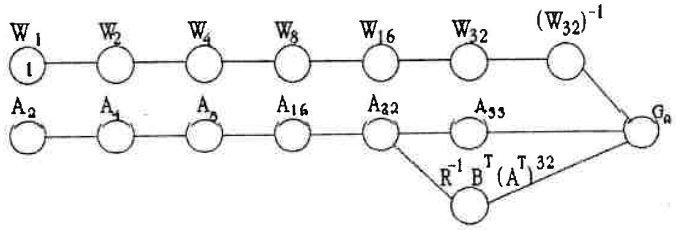
On calcule ainsi la matrice $W_N$ en $K = \log_2 N$ étapes. Par exemple, le schéma de calcul pour N = 32 est le suivant :




Pour le calcul des W et $A^n$, on peut utiliser les méthodes de multiplication parallèle des matrices présentées au chapitre 4.



Pour le calcul de $G_0$, on peut utiliser le graphe suivant :




Là encore, pour l'inversion de la matrice symétrique $(W_{32})^{-1}$, on peut utiliser les méthodes d'inversion parallèle d'une matrice présentée au chapitre 3.

Pour le calcul de $M_j$, il faut résoudre l'équation de Lyapunov 5-4-13d. Nous avons étudié en 5-3 la résolution parallèle d'une équation de Lyapunov. Dans ce cas, même si la formulation de l'équation 5-4-13d est un peu différente de celle de l'équation 5-3-13, la méthode de résolution en parallèle ne change pas essentiellement. Enfin le calcul de $G_j$ pour l'itération suivante comprend l'inversion d'une matrice symétrique et des multiplications matricielles que l'on peut effectuer en parallèle.

b) Algorithme de Kailath [5-12]

Cet algorithme est exprimé de la façon suivante :

$$P_0 = G_0 = 0 \qquad j = 0, 1, \ldots\ldots a$$

$$\tilde{A}_j = A - B G_j \qquad j = 0, 1 \ldots\ldots b$$

$$P_{j+1} = \tilde{A}_j^T P_j A + G_j^T R G_j + Q \qquad j = 0, 1 \ldots\ldots c$$

$$G_{j+1} = (B^T P_{j+1} B + R)^{-1} B^T P_{j+1} A$$

Dans cette formulation, on remplace l'équation de Lyapunov 5-4-13d par l'équation récursive 5-4-14 b. Le coût de calcul par itération de cet algorithme est trés inférieur à celui de l'algorithme de Hewer, mais son taux de convergence est beaucoup plus élevé [5-1 ].

Le calcul parallèle de cet algorithme comprend des multiplications et des inversions des matrices, qui peuvent être effectuées avec les méthodes déjà présentées. La figure 5-4-15 présente le graphe correspondant à une itération de cet algorithme pour un système utilisant les matrices suivantes :

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \begin{bmatrix} B_{11} & 0 \\ C & B_{22} \end{bmatrix} \quad \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \quad \begin{bmatrix} R_{11} & 0 \\ 0 & R_{22} \end{bmatrix} \qquad (5\text{-}4\text{-}15)$$

Voici les significations des tâches du graphe 5-15 :

| | | |
|---|---|---|
| 1 à 4 | $B G^j$ | |
| 5 à 12 | $(P_j)_{iK} \cdot A_{Km}$ | i, K, m = 1 à 2 |
| 13 à 16 | $M_{im} = R_{ii} G_{im}$ | i = 1 à 2 m = 1 à 2 |
| 17 à 20 | $\tilde{A}_{im} = A_{im} - (B G_j)_{im}$ | |
| 21 à 24 | $N_{im} = \sum_K (P_j)_{iK} \cdot A_{Km}$ | |
| 25 à 30 | $(G_j)_{iK} \cdot M_{Km}$ | i, K = 1 à 2 m = 1 à 2 |



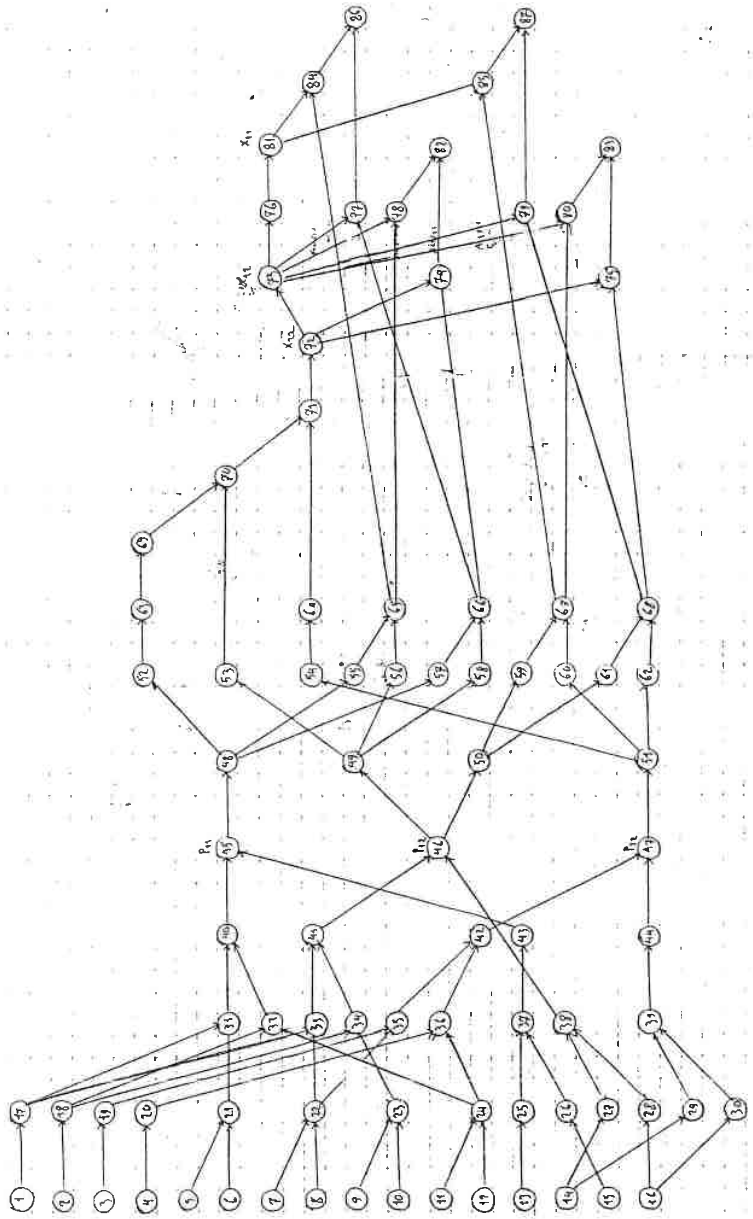


Figure (5-15) Graphe de l'algorithme de Kailath

31 à 36 $\tilde{A}_{iK}\,\tilde{N}_{Km}$

37 à 39 $(G_j\ R\ G_j)_{im} = \sum_K (G_j)_{iK} \cdot M_{Km}$

40 à 42 $(\tilde{A}_j)^T P_j A$

43 à 44 $(G_j R G_j) + Q$

45 à 47 $P_{j+1}$

48 à 51 $L = B^T P_{j+1}$

52 à 53 $L B$

54 à 62 $L_{iK} A_{Km}$

63 à 64 $L B + R$

65 à 68 $T = B^T P_{j+1} A = \Sigma L_{iK} A_{Km}$

69-70-71-72-73-76-81 inversion de $(L B + R)$

autres tâches : $G_{j+1} = (L B + R)^{-1} T$

Dans le cas général, si la matrice A est décomposée en $N^2$ sous-matrices, le chemin critique du graphe 5-15 est composé des opérations suivantes :

- N inversions de matrices
- N+1 multiplications de deux matrices dont le résultat est symétrique
- (N+3) additions de matrices symétriques
- $2\log_2 N$ additions de deux matrices
- (3N-1) multiplications de deux matrices
- (2N-4) additions de deux matrices

Dans le cas du système 5-4-15, supposons que l'on ait dim $A_{11}$ =



dim $A_{22}$ = 5 et 6 $t_{div} = t_{add}$ ; $t_{mult} = 5\ t_{add}$. La durée du chemin critique est alors de 7093 $t_{add}$.

Le tableau 5-9 donne le temps d'exécution par ordonnancement de l'algorithme( 5-4-14) en fonction du nombre de processeurs.

| Nombre de processeurs | Temps d'exécution | Taux d'accélération | Taux d'occupation |
|---|---|---|---|
| 14 | 7093 | 5,73 | 0,41 |
| 12 | 7629 | 5,33 | 0,44 |
| 10 | 7654 | 5,31 | 0,53 |
| 8 | 7808 | 5,20 | 0,65 |
| 6 | 8558 | 4,75 | 0,79 |
| 5 | 9228 | 4,40 | 0,88 |
| 4 | 10325 | 3,94 | 0,98 |
| 3 | 13760 | 2,95 | 0,98 |
| 2 | 20415 | 1,99 | 0,995 |

Tableau (5-9) Temps d'exécution pour l'algorithme 5-4-14

On peut remarquer que le taux d'occupation des processeurs pour l'exécution de l'algorithme dans le temps du chemin critique est insuffisant jusqu'à 4 processeurs, seuil à partir duquel le taux d'occupation est excellent.

## 5-5 CONCLUSION

Nous avons étudié les possibilités d'application des techniques

parallèles au calcul de la commande optimale des systèmes interconnectés. Deux types fondamentaux de décompositions pour le calcul parallèle de la commande ont été étudiés : la décomposition temporelle et la décomposition spatiale.

Ces deux décompositions peuvent être appliquées simultanément, ce qui conduit à un degré de parallélisme trés élevé, mais une décomposition en un nombre élevé de sous-systèmes peut ralentir la convergence de l'algorithme de calcul de la commande optimale. Le choix du nombre de sous-systèmes se fait donc par un compromis entre le taux de convergence de l'algorithme, le taux d'accélération et le nombre de processeurs disponibles. Nous avons étudié également les méthodes de synchronisation de calcul de sous-systèmes tout en maximisant le taux d'occupation des processeurs par l'ordonnancement de calcul. Nous avons démontré enfin la possibilité de rendre parallèle l'équation de Riccati par la partition des différentes matrices présentes dans l'équation.

# - CHAPITRE 6 -

# ORDONNANCEMENT DU TRANSFERT

## 6-1 INTRODUCTION

Dans les chapitres précédents, nous nous sommes limités à l'étude du parallélisme basé sur la logique des algorithmes. Nous avons donc supposé que le temps de transfert des données des mémoires vers les processeurs est négligeable.

Dans la pratique, si la dimension de blocs à traiter n'est pas grande, ou si la capacité de transfert du réseau d'interconnexion est limitée, ce temps de transfert n'est plus négligeable et il faut en tenir compte pour évaluer le temps d'exécution d'un algorithme, il faut tenir compte également du temps de transfert. Un autre problème apparaît également lié à la structure du réseau d'interconnexion : c'est celui du nombre de blocs mémoires disponibles. Aux chapitres précédents, nous avons supposé que le nombre de blocs mémoires est illimité, mais dans la réalité ce nombre est toujours limité et entraîne des conflits entre les différents processeurs.

Nous avons donc consacré ce chapître à l'étude des effets du temps de transfert et du nombre de blocs mémoires sur le temps d'exécution d'un algorithme. Pour celà, nous étudierons d'abord le cas d'une architecture multiprocesseurs avec un bus commun et une mémoire globale, puis le cas d'une architecture à multiprocesseurs reliés à un nombre $N > 1$ de blocs mémoires à travers un réseau d'interconnection sans conflit.

## 6-2 ARCHITECTURE MULTIPROCESSEURS AVEC UN BUS COMMUN

Dans cette architecture, les processeurs sont reliés entre eux et avec une mémoire globale par un bus commun qu'ils utilisent en temps partagé pour communiquer entre eux et avec la mémoire (voir Figure 1-5). Afin de libérer le bus pour les autres processeurs, chaque processeur transfère les données nécessaires pour ces opérations à sa mémoire locale. Pendant le temps d'exécution de ces opérations, les autres processeurs peuvent utiliser le bus commun. Si le temps d'exécution des opérations à l'intérieur de chaque processeur est trés supérieur au temps de transfert des opérandes, le temps moyen d'occupation des bus par processeur est faible. On peut ainsi relier un nombre plus important de processeurs sur un seul bus. Nous avons démontré au chapitre 2 que la fusion des tâches et l'exécution de l'algorithme par opérations sur les blocs de données peuvent augmenter le rapport entre le temps d'exécution d'une opération et le temps de transfert de ces opérandes. La fusion des tâches constitue donc une solution pour diminuer la part du temps de transfert dans le temps total d'exécution et de permettre d'utiliser un nombre plus grand de processeurs. Mais en contre partie, pour un problème de taille donnée, l'augmentation de la taille des tâches réduit le degré du parallélisme de l'algorithme et diminue ainsi le nombre de processeurs utilisables pour l'exécution de cet algorithme. La taille des blocs est donc déterminée par un compromis entre le temps de transfert et le degré de parallélisme de l'algorithme. Illustrons cette situation par un exemple :

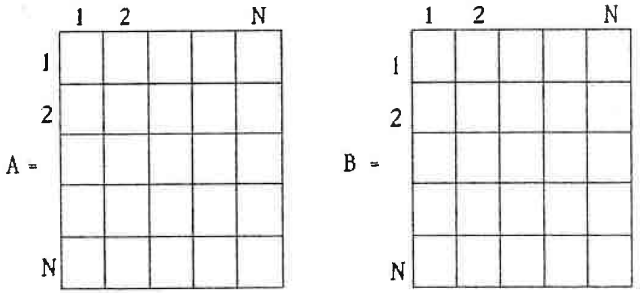


soit à multiplier deux matrices A et B de dimension n x n

$$C = A \cdot B$$

Considérons le cas où $t_{mult} = t_{add} = t_{transfert}$, où $t_{transfert}$ est le temps de transfert d'un nombre réel de la mémoire locale d'un processeur à la mémoire globale ou de la mémoire globale à la mémoire locale, et $t_{mult}$ et $t_{add}$ sont respectivement le temps de multiplication et d'addition de deux nombres réels dans les processeurs.

Si on décompose les matrices A et B en $N^2$ blocs de la façon suivante :




Ce nombre d'opérations pour le calcul de C est alors de :

$N^3$ multiplications de matrices

$N^2(N-1)$ additions de matrices

Le nombre de transferts pour chacune de ces opérations est

de 3 : deux transferts pour les opérandes et un transfert pour le résultat. Le nombre total des transferts est ainsi de

$$3\,[N^3 + N^2(N-1)] = 6N^3 - 3N^2$$

Si la capacité du réseau d'interconnection n'est pas limitée, le temps minimum de multiplication des deux matrices est :

1 multiplication de deux sous-matrices + $\log_2 N$ additions de deux sous-matrices (6-1-1)

Dans le cas d'une architecture à bus commun, les transferts des opérandes se font en série. Ce temps peut facilement dépasser le temps minimum calculé en 6-1-1. Les tâches de transfert peuvent alors constituer le chemin critique du graphe de multiplication des deux matrices. Dans ce cas, l'augmentation du nombre de processeurs n'a aucun effet sur le temps minimum d'exécution de l'algorithme. Cette contrainte de blocage limite donc le nombre de partitions des matrices A et B. Si Tt, Tm et Ta sont respectivement le temps de transfert, de multiplication et d'addition des sous-matrices, la condition de non-blocage est :

$$(6N^3 - 3N^2)\,T_t \leq T_m + (\log^2 N)T_{add} + 3T_t(1 + \log^2 N) \qquad (6\text{-}1\text{-}2)$$

Si V est la dimension des sous-matrices, on a :

$$T_t = V^2 t_t \; ; \; T_m = V^3 t_{mult} + V^2(V-1)t_{add} \; ; \; T_a = V^2\, t_{add}$$



Pour $t_{mult} = t_t = t_{add}$, la condition 6-1-2 s'écrit sous la forme :

$$(6N^3 - 3N^2)V^2 \leq V^2 + V^2(V-1) + (\log_2 2N)V^2 + 3V^2(1+\log_2 N)$$

$$\text{ou } V \geq \frac{6N^3 - 3N^2 - 2 - 4\log_2 N}{2}$$

mais en même temps $V = n/N$, on peut donc écrire

$$6N^4 - 3N^3 - 2N - 4N\log_2 N \leq 2n \qquad (6\text{-}1\text{-}3)$$

L'inégalité 6-1-3 permet de calculer la valeur maximale de N à partir duquel le temps de transfert peut, à lui seul, constituer le chemin critique. Pour n = 30 par exemple, on doit avoir $N \leq 2$ et donc la dimension des blocs doit être supérieure à 15.

Dans le cas général d'un algorithme quelconque, on peut décomposer chaque tâche en trois ou quatre tâches selon le cas : deux tâches de transfert des opérandes (une tâche dans le cas d'opérations comme l'inversion d'une matrice ou la multiplication d'une matrice par elle-même ou par sa conjuguée), une tâche pour les opérations mathématiques ou autres sur les données et une tâche pour le transfert des résultats. La durée d'une opération est donc la somme des durées de ces quatres tâches. On peut former ainsi un graphe de calcul pour ces tâches composées, basé sur la simple logique de l'algorithme comme nous l'avons fait aux chapitres précédents. On détermine ainsi la durée du chemin critique du graphe. Cette durée, dans tous les algorithmes que nous avons étudiés auparavant, est fonction du nombre N de décompositions des matrices concernées. Le temps de transfert des opérandes et les résultats de toutes les tâches sont aussi fonction de N. Si le temps de transfert

dépasse la durée du chemin critique, l'exécution de l'algorithme dans le temps du chemin critique est impossible. Dans ce cas, l'augmentation du nombre N de décompositions non seulement n'est pas utile, mais risque d'augmenter le temps d'exécution de l'algorithme à cause du temps de transfert très élevé. La limite de N est donc déterminée par la relation suivante

$$t_{\text{chemin critique}} < t_{\text{tranfert}} \qquad (6\text{-}1\text{-}3)$$

Si on dispose d'un nombre P fixe de processeurs, le problème se résout différemment : pour un algorithme donné, on calcule le nombre departitionnements N, pour lequel la durée du chemin critique de l'algorithme s'obtient sur P processeurs.

Si le nombre N vérifie l'inégalité (6-1-3), on l'utilise pour le partitionnement, sinon on calcule N à partir de la relation 6-1-3 et on partitionnel'algorithmeselonlanouvellevaleur.

Exemple :

Pour l'inversion d'une matrice symétrique définie positive, nous décomposons celle-ci en $N^2$ sous-matrices de dimension V. Alors, si l'on applique la méthode de GAUSS, la durée du chemin critique dans le cas où

$t_{div} = 34t_{add}$, $t_{mult} = t_{add} = t_{trans}$ correspond à la somme de :

N transferts de matrices symétriques = N . V(V+1)/2



N inversions de matrices symétriques = N [34 V + (V-1)($V^2$+V+1)]

N transferts de matrices symétriques = N V(V+1)/2

2(N+1) tranferts de matrices = 2(N+1)$V^2$

(N+1) multiplications de matrices dont le résultat est symétrique =

$$(N+1)\left[\frac{V^2(V+1)}{2} + V\frac{(V^2+1)}{2}\right]$$

(N+1) transferts de matrices symétriques = (N+1)V(V+1)/2

2(N+1) transferts de matrices symétriques = 2(N+1)V(V+1)/2

(N+1) additions de matrices symétriques = (N+1)V(V+1)/2

(N+1) transferts de matrices symétriques = (N+1)V(V+1)/2

2(3N-4) transferts de matrices = 2(3N-4)$V^2$

(3N-4) multiplications de matrices = (3N-4) [$V^3$ +$V^2$ (V-1)]

(3N-4) transferts de matrices = (3N-4)$V^2$

2(2N-5) transferts de matrices = 2(2N-5)$V^2$

(2N-5) additions de matrices = (2N-5)$V^2$

(2N-5) transferts de matrices = (2N-5)$V^2$

Le temps total de transfert est la somme de :

$$2N \text{ transferts de matrices symétriques} = 2N\frac{V(V+1)}{2}$$

$$2(N^2-1) \text{ transferts de matrices} = 2(N^2-1)V^2$$

$$(N^2-1) \text{ transferts de matrices symétriques} = (N^2-1)\frac{V(V+1)}{2}$$

$$3N(N-1) \text{ transfert de matrices symétriques} = 3N(N-1)\frac{V(V+1)}{2}$$

$$\frac{3N^2(N-1)}{2} \text{ transferts de matrices} = \frac{3N^2(N-1)}{2}V^2$$

$$\frac{3N(N-1)(N-2)}{2} \text{ transferts de matrices} = \frac{3N(N-1)(N-2)}{2} V^2$$

La condition (6-1-3) s'écrit alors sous la forme :

$$\begin{cases} V^3(8N-7) + V^2(20N-23) + V(37N+2) - N \geq \dfrac{V^2(6N^3-4N^2+7N-5)}{2} + V\left(\dfrac{4N^2+N-1}{2}\right) \\ n = N \cdot V \end{cases} \qquad (6\text{-}1\text{-}5)$$

Pour chaque valeur de n, on peut ainsi déterminer le nombre de partitionnements N.

Pour n = 16 par exemple, on trouve N < 4 et V > 4

On peut donc partitionner les matrices en blocs de dimensions 4.

D'autres problèmes importants dans le cas d'une architecture à bus commun sont l'ordonnancement des transferts. Considérons le problème de multiplication des deux matrices suivantes :

$$A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}$$

$A_{11}$, $B_{11}$ et $A_{22}$, $B_{22}$ sont respectivement de dimensions 3 et 5. Pour la multiplication de ces deux matrices, supposons deux modes de transfert différents :



Pour $t_{mult} = t_t = t_{add}$, la condition 6-1-2 s'écrit sous la forme :

$$(6N^3 - 3N^2)V^2 \leq V^2 + V^2(V-1) + (\log_2 2N)V^2 + 3V^2(1+\log_2 N)$$

$$\text{ou } V \geq \frac{6N^3-3N^2-2-4\log_2 N}{2}$$

mais en même temps V = n/N , on peut donc écrire

$$6N^4 - 3N^3 - 2N - 4N\log_2 N \quad 2n \qquad (6\text{-}1\text{-}3)$$

L'inégalité 6-1-3 permet de calculer la valeur maximale de N à partir duquel le temps de transfert peut, à lui seul, constituer le chemin critique. Pour n = 30 par exemple, on doit avoir N 2 et donc la dimension des blocs doit être supérieure à 15.

Dans le cas général d'un algorithme quelconque, on peut décomposer chaque tâche en trois ou quatre tâches selon le cas : deux tâches de transfert des opérandes (une tâche dans le cas d'opérations comme l'inversion d'une matrice ou la multiplication d'une matrice par elle-même ou par sa conjuguée), une tâche pour les opérations mathématiques ou autres sur les données et une tâche pour le transfert des résultats. La durée d'une opération est donc la somme des durées de ces quatres tâches. On peut former ainsi un graphe de calcul pour ces tâches composées, basé sur la simple logique de l'algorithme comme nous l'avons fait aux chapitres précédents. On détermine ainsi la durée du chemin critique du graphe. Cette durée, dans tous les algorithmes que nous avons étudiés auparavant, est fonction du nombre N de décompositions des matrices concernées. Le temps de transfert des opérandes et les résultats de toutes les tâches sont aussi fonction de N. Si le temps de transfert

a) On organise le transfert comme suit :

$A_{11}$ - $B_{11}$, $A_{11}$ - $B_{12}$, $A_{12}$-$B_{22}$ , $A_{22}$ -$B_{22}$,....

Le graphe de calcul est alors le suivant fig (6-1)

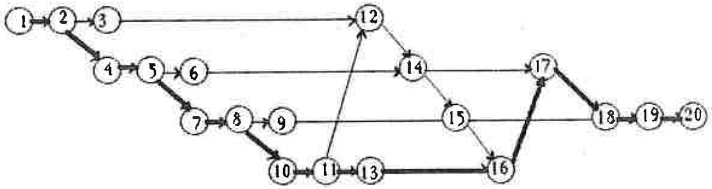



Figure (6-1) Graphe de calcul dans le cas a

La signification des différentes tâches du graphe 6-1 est la suivante :

1 - transfert de $A_{11}$ de la mémoire globale à la mémoire locale

2 - transfert de $B_{11}$ de la mémoire globale à la mémoire locale

3 - $A_{11}$ -$B_{11}$

4 - transfert de $A_{11}$ à la mémoire locale

5 - transfers de $B_{12}$ à la mémoire mocale

6 - $A_{11}$-$B_{12}$

7 - transfert de $A_{12}$ à la mémoire locale

8 - transfert de $B_{12}$ à la mémoire locale

9 - $A_{12}$ -$B_{22}$



10 - transfert de $A_{22}$ à la mémoire locale

11 - transfert de $B_{22}$ à la mémoire locale

12 - transfert de $A_{11}$-$B_{11}$ à la mémoire globale

13 - $A_{22}$ -$B_{22}$

14 - transfert de $A_{11}$-$B_{12}$ à la mémoire globale

15 - transfert de $A_{12}$ -$B_{22}$ à la mémoire globale

16 - transfert de $A_{22}$ -$B_{22}$ à al mémoire globale

17 - transfert de $A_{11}$-$B_{12}$ à la mémoire locale

18 - transfert de $A_{12}$ -$B_{22}$ à la mémoire locale

19 - $A_{11}$-$B_{12}$ + $A_{12}$ -$B_{22}$

20 - transfert de ($A_{11}$-$B_{12}$ + $A_{11}$-$B_{22}$ ) à la mémoire globale

Pour $t_{mult}$ = $t_{add}$ = $t_{trans}$, la durée du chemin critique du graphe 6-1 est de 442 $t_{add}$.

b) On organise le transfert de la façon suivante :

$A_{22}$ -$B_{22}$, $A_{12}$ -$B_{22}$, $A_{11}$ -$B_{12}$, $A_{11}$ -$B_{11}$, ...

Le graphe de calcul dans ce cas est : (Fig 6-2)

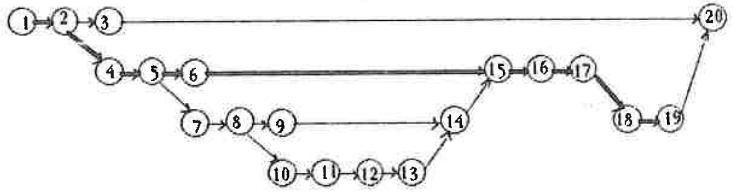



Figure (6-2) Graphe de calcul dans le cas b

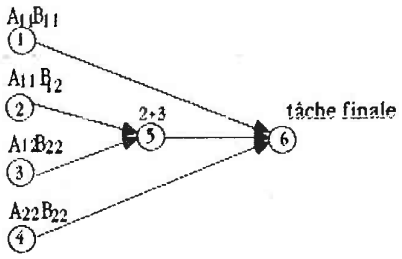
La durée du chemin critique du graphe 6-2 est de 325 $t_{add}$. Elle est donc de l'ordre de 73 % de la durée calculée pour le graphe (6-1). Cette grande différence vient du fait que nous avons avancé le transfert de blocs pour les tâches les plus grandes.

La méthode que nous proposons pour l'ordonnancement du transfert de données pour un algorithme quelconque est le suivant :

- construire le graphe de calcul de l'algorithme comme indiqué dans les chapitres précédents
- décomposer chaque tâche en tâches de transfert et de calcul et former un nouveau graphe
- appliquer la méthode d'ordonnancement présentée au chapitre 2 en imposant un nombre de lignes de transferts égal à 1.

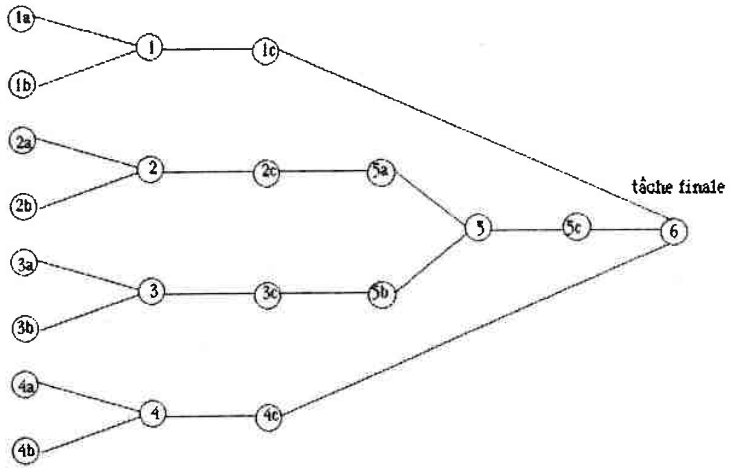
Dans la cas de la multiplication de deux matrices A et B de l'exemple précédent, la méthode s'applique comme suit :

a) Graphe de calcul :






b) Formation du graphe par décomposition des tâches




c) Application de la méthode d'ordonnancement sur ce graphe en imposant une ligne de transfert : on aboutit à l'organisation du calcul présenté sur le graphe 6-2.

Nous avons appliqué cette méthode à l'algorithme de moindres carrés mis en oeuvre pour l'estimation des paramètres.

$$(H^T H)\underline{\theta} = H^T Z$$

La figure 6-3 fournit le graphe de calcul dans le cas d'une décomposition de $\underline{\theta}$ et $\underline{Z}$ en deux sous-vecteurs.

Le tableau 6-1 donne le temps d'exécution pour ce graphe

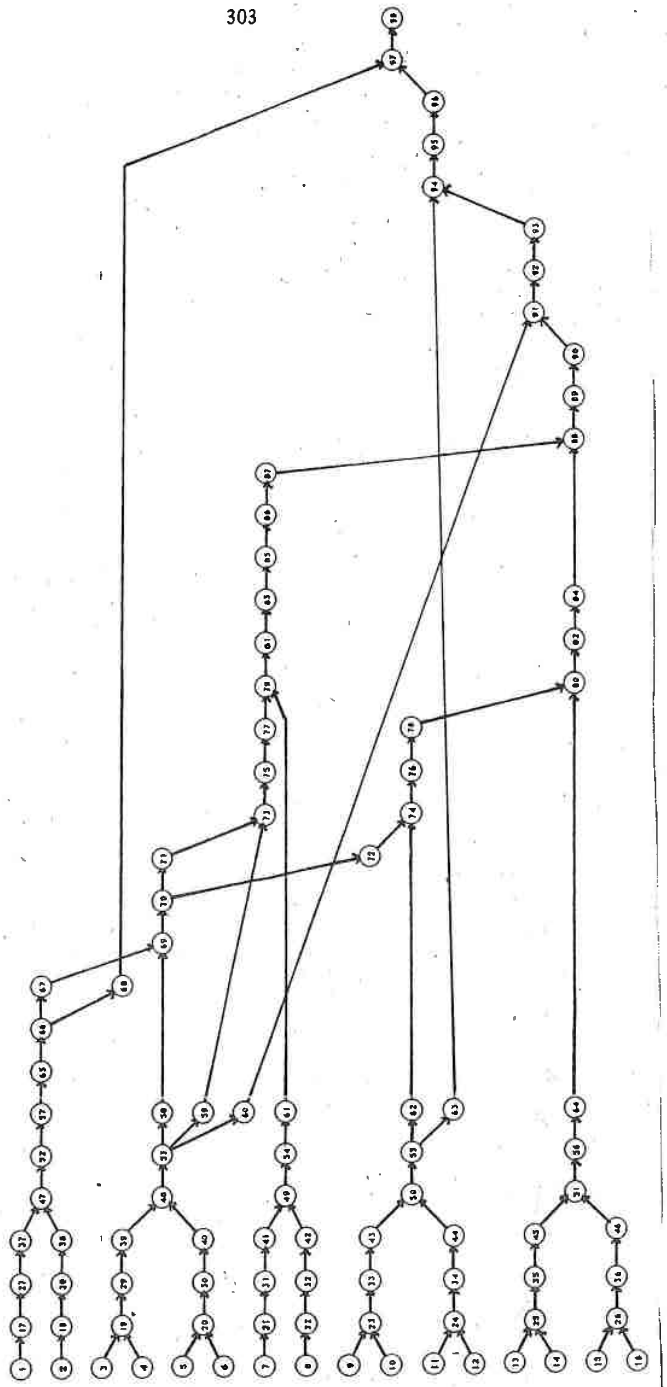


Figure (6-3) Graphe de l'algorithme des moindres carrés



6-3 en fonction du nombre de processeurs pour différentes dimensions des vecteurs O(n) et Z(m).

| Nombre de processeurs | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| temps d'execution pour m=30, n=10 | 3584 | 2759 | 2429 | 2324 | 2314 |
| temps d'execution pour m=18, n=6 | 1112 | 836 | 750 | 749 | 749 |
| temps d'execution pour m=12, n=6 | 905 | 651 | 623 | 632 | 632 |
| temps d'execution pour m=60, n=20 | 21500 | 15739 | 13329 | 13304 | 13143 |

Tableau 6-1 Temps d'exécution pour l'algorithme de moindres carrés pour l'estimation de paramètres

On peut remarquer que pour les petites dimensions de Z et O, le temps d'exécution de l'algorithme à partir de 4 processeurs ne diminue pas. Même pour m = 12 et n = 6, le temps d'exécution de l'algorithme est légèrement supérieur à celui de 4 processeurs. En effet comme nous l'avons expliqué auparavant, c'est le temps total du transfert qui, dans ce cas, empêche le temps d'exécution de diminuer.

Le tableau 6-2 donne le séquencement de calcul pour m= 30 et un nombre de processeurs égal à 5.

Tableau (6-2) Organisation du calcul pour l'algorithme des moindres carrés

6 processeurs

| activité | durée | début | fin |
|---|---|---|---|
| 3 | 75 | 0 | 75 |
| 4 | 75 | 75 | 150 |
| 5 | 75 | 150 | 225 |
| 19 | 725 | 150 | 875 |
| 6 | 75 | 225 | 300 |
| 1 | 75 | 300 | 375 |
| 20 | 725 | 300 | 1025 |
| 2 | 75 | 375 | 450 |
| 17 | 435 | 375 | 810 |
| 18 | 435 | 450 | 885 |
| 7 | 75 | 450 | 525 |
| 8 | 75 | 525 | 600 |
| 21 | 435 | 525 | 960 |
| 22 | 435 | 600 | 1035 |
| 27 | 15 | 810 | 825 |
| 37 | 15 | 825 | 840 |
| 9 | 75 | 840 | 915 |



| | | | |
|---|---|---|---|
| 28 | 15 | 915 | 930 |
| 38 | 15 | 930 | 945 |
| 47 | 15 | 945 | 960 |
| 29 | 25 | 945 | 970 |
| 52 | 15 | 970 | 985 |
| 57 | 15 | 985 | 1000 |
| 65 | 294 | 1000 | 1294 |
| 39 | 25 | 1000 | 1025 |
| 30 | 25 | 1025 | 1050 |
| 40 | 25 | 1050 | 1075 |
| 48 | 25 | 1075 | 1100 |
| 31 | 15 | 1075 | 1090 |
| 32 | 15 | 1090 | 1105 |
| 53 | 25 | 1105 | 1130 |
| 58 | 25 | 1130 | 1155 |
| 59 | 25 | 1155 | 1180 |
| 41 | 15 | 1180 | 1195 |
| 42 | 15 | 1195 | 1210 |
| 49 | 15 | 1210 | 1225 |
| 11 | 75 | 1210 | 1285 |

| | | | |
|---|---|---|---|
| 54 | 15 | 1285 | 1300 |
| 66 | 15 | 1300 | 1315 |
| 67 | 15 | 1315 | 1330 |
| 69 | 225 | 1330 | 1555 |
| 61 | 15 | 1330 | 1345 |
| 13 | 75 | 1345 | 1420 |
| 15 | 75 | 1420 | 1495 |
| 10 | 15 | 1495 | 1510 |
| 12 | 15 | 1510 | 1525 |
| 23 | 145 | 1510 | 1655 |
| 24 | 145 | 1525 | 1670 |
| 14 | 15 | 1525 | 1540 |
| 16 | 15 | 1540 | 1555 |
| 25 | 145 | 1540 | 1685 |
| 70 | 25 | 1555 | 1580 |
| 26 | 145 | 1555 | 1700 |
| 71 | 25 | 1580 | 1605 |
| 73 | 135 | 1605 | 1740 |
| 72 | 25 | 1605 | 1630 |



| | | | |
|---|---|---|---|
| 60 | 25 | 1630 | 1655 |
| 33 | 5 | 1655 | 1660 |
| 43 | 5 | 1660 | 1665 |
| 68 | 15 | 1665 | 1680 |
| 34 | 5 | 1680 | 1685 |
| 44 | 5 | 1685 | 1690 |
| 50 | 5 | 1690 | 1695 |
| 35 | 5 | 1690 | 1695 |
| 55 | 5 | 1695 | 1700 |
| 62 | 5 | 1700 | 1705 |
| 74 | 45 | 1705 | 1750 |
| 36 | 5 | 1705 | 1710 |
| 45 | 5 | 1710 | 1715 |
| 46 | 5 | 1715 | 1720 |
| 51 | 5 | 1720 | 1725 |
| 63 | 5 | 1720 | 1725 |
| 56 | 5 | 1725 | 1730 |
| 64 | 5 | 1730 | 1735 |
| 75 | 15 | 1740 | 1755 |
| 77 | 15 | 1755 | 1770 |

| | | | |
|---|---|---|---|
| 79 | 15 | 1770 | 1785 |
| 76 | 5 | 1770 | 1775 |
| 78 | 5 | 1775 | 1780 |
| 80 | 5 | 1780 | 1785 |
| 81 | 15 | 1785 | 1800 |
| 83 | 15 | 1800 | 1815 |
| 85 | 294 | 1815 | 2109 |
| 82 | 5 | 1815 | 1820 |
| 84 | 5 | 1820 | 1825 |
| 86 | 15 | 2109 | 2124 |
| 87 | 15 | 2124 | 2139 |
| 88 | 45 | 2139 | 2184 |
| 89 | 5 | 2184 | 2189 |
| 90 | 5 | 2189 | 2194 |
| 91 | 45 | 2194 | 2239 |
| 92 | 5 | 2239 | 2244 |
| 93 | 5 | 2244 | 2249 |
| 94 | 5 | 2249 | 2254 |
| 95 | 2 | 2254 | 2259 |



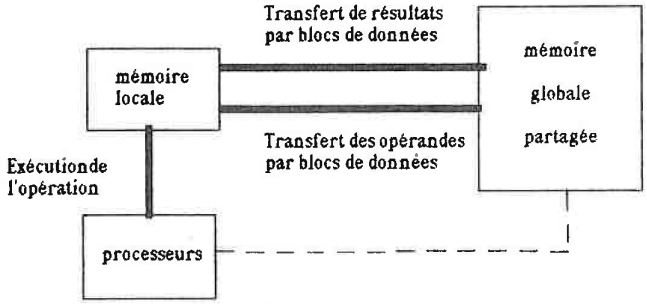
| | | | |
|---|---|---|---|
| 96 | 5 | 2259 | 2264 |
| 97 | 45 | 2264 | 2309 |
| 98 | 5 | 2309 | 2314 |
| 99 | 0 | 2314 | 2314 |

## 6-3 ARCHITECTURE AVEC RESEAU D'INTERCONNEXION

Dans cette architecture, nous supposons que P processeurs sont liés à n mémoires par un réseau d'interconnexion sans blocage. Si P > M, les processeurs transfèrent les données nécessaires pour leurs opérations respectives dans leurs mémoires locales et libèrent les mémoires pour les autres processeurs. Par contre, si M > P, et surtout quand M = 2P [6-1], les processeurs peuvent exécuter leurs opérations tout en communiquant avec les mémoires globales, sans être obligés de transférer les données à la mémoire locale.

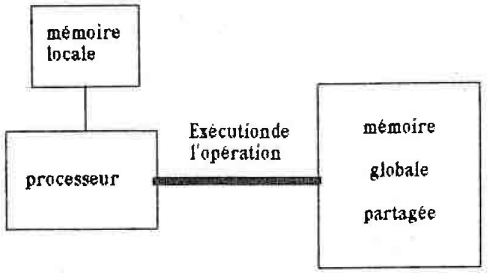
Le schéma de calcul dans les deux cas est alors le suivant :

a) Premier cas




b) Deuxième cas






Dans le deuxième cas, la mémoire locale sert à stocker les éventuels résultats intermédiaires.

Pour une architecture multiprocesseurs et multimémoires, deuxproblèmesessentielsseposent:

1) Déterminer la source et la destination de données

2) Ordonnancer le transfert

Le premier problème consiste à organiser les données dans les mémoires de telle façon que les conflits d'accès à mémoire soient réduitsauminimum.Ledeuxièmeproblèmeconsisteensuiteàorganiserle transfert des données chronologiquement pour minimiser le temps total d'exécution de l'algorithme. Les deux problèmes sont liés l'un à l'autre et il faut les traiter ensemble.

Le problème d'organisation de données dans le cas des architectures S.I.M.D. a été étudié par différents auteurs. [6-2], [6-3], [6-4], [6-5]. Mais à notre connaissance, il n'y a aucune étude dans le cas des architectures M.I.M.D. Pour positionner le problème, supposons qu'on ait à multiplier deux matrices A et B de la forme suivante :

| | |
|---|---|
| $A_{11}$ | $A_{12}$ |
| $A_{21}$ | $A_{22}$ |

| | |
|---|---|
| $B_{11}$ | $B_{12}$ |
| $B_{21}$ | $B_{22}$ |

$A_{11}$ et $B_{11}$ ne sont pas de même dimension. Supposons d'une part que l'on dispose de deux blocs mémoires et que l'on stocke A dans la première et B dans la deuxième et d'autre part que l'on exécute la multiplication sur deux processeurs. Alors, si le premier processeur exécute le produit $A_{11}B_{11}$ en communiquant directement avec les mémoires, il bloque les deux mémoires pendant toute la durée d'exécution du programme. L'autre processeur ne peut qu'attendre la fin du calcul. Deux solutions peuvent être envisagées :

a) On stocke les matrices de la manière suivante

| | |
|---|---|
| $A_{11}$ | $B_{12}$ |
| $A_{21}$ | $B_{12}$ |

| | |
|---|---|
| $A_{12}$ | $B_{21}$ |
| $A_{22}$ | $B_{22}$ |

bloc 1

On calcule ainsi $A_{11}B_{11}$, $A_{11}B_{12}$, $A_{21}B_{11}$, $A_{21}B_{12}$ dans le premier bloc et on stocke les résultats dans le même bloc. On calcule également $A_{12}B_{21}$, $A_{12}B_{22}$, $A_{22}B_{21}$, $A_{22}B_{22}$ dans le deuxième bloc et on stocke les résultats dans le même bloc. Le problème de blocage de mémoire ne se pose pas à cette étape. Mais au moment de calculer les sommes $A_{12}B_{21}+A_{11}B_{11}$..... les données sont dans les deux mémoires et l'exécution



de cette opération bloque ces deux mémoires.

b) On transfere les données nécessaires à chaque opération aux mémoires locales des processeurs. On libère ainsi les mémoires pour que les autres processeurs puissent les utiliser. L'avantage de cette méthode est qu'elle minimise les conflits d'accès aux mémoires, mais elle introduit des transferts supplémentaires qui peuvent représenter dans certains cas un pourcentage assez important du temps de calcul. On peut également mettre en oeuvre ces méthodes simultanément.

Dans le cas où le nombre de processeurs est supérieur au nombre de blocs mémoires, le transfert des données dans la mémoire locale est indispensable. Mais là encore, le problème d'organisation des données dans les mémoires se pose. En effet, pendant le transfert des données, la mémoire concernée est bloquée. Si le nombre de transferts est important, ce blocage peut être la source de conflits et par conséquent d'augmentation du temps de calcul. Pour organiser les données dans les mémoires, nous proposons une méthodologie qui, dans certains cas, réduit les conflits d'accès et diminue ainsi le temps de calcul. Nous croyons qu'en approfondissant cet algorithme, il est éventuellement possible de résoudre le problème d'organisation des données dans la mémoire.

Cette méthodologie s'exprime de la manière suivante :

1- Former le graphe de calcul en tenant compte des tâches de transfert (exemple du graphe 6-3)

2- Déterminer l'ordonnancement du calcul sur un nombre de processeurs donné en supposant le nombre de blocs mémoires

illimit

3- Faire une liste ordonnée des tâches de transfert dans la liste 2

4- Affecter dans l'ordre les données concernées aux blocs mémoires dans cette liste

Cette organisation a l'avantage de distribuer les tâches urgentes sur les différents blocs mémoires et de rendre possible l'accès simultané à ces tâches pour les processeurs. Nous avons essayé cet algorithme pour le calcul des multiplications matricielles suivantes :

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \qquad A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$$

$$C = A P A^T$$

La figure 6-4 présente le graphe correspondant à ce calcul.

Nous avons supposé que l'on dispose de deux blocs mémoires et de six processeurs.

Dans le premier cas, nous avons stocké les données comme suit :

mémoire 1 $P, L = PA^T$

mémoire 2 $A, C = A L$

Dans le deuxième cas, nous avons utilisé l'algorithme proposé précédemment.



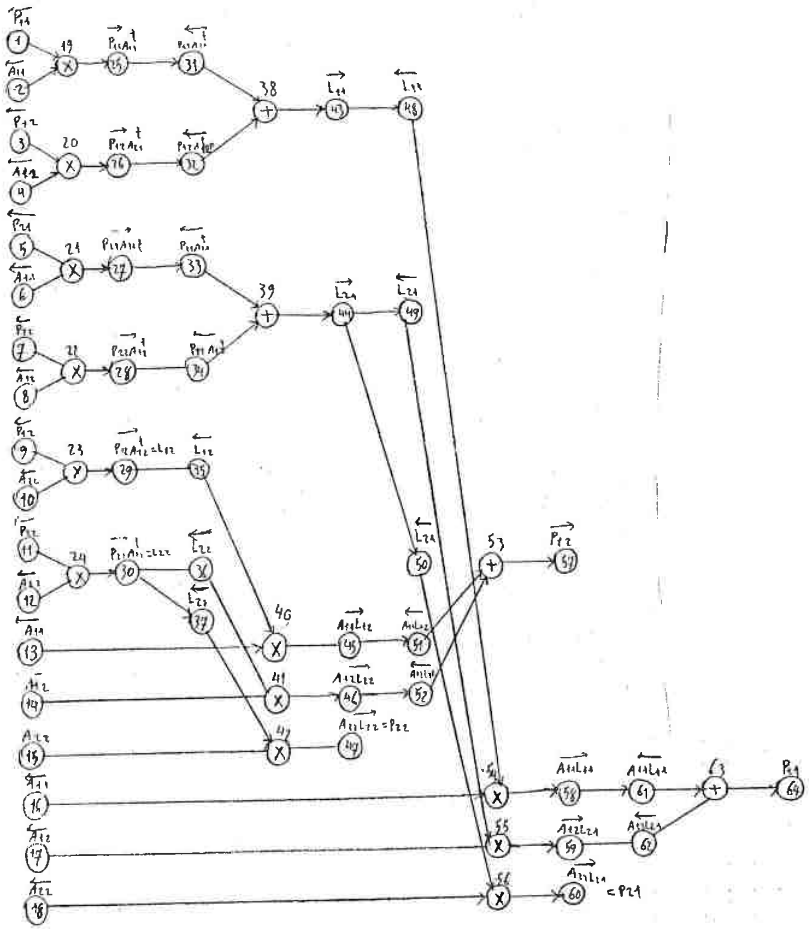


Figure (6-4) Graphe du calcul de APAT

Le tableau 6-3 compare le temps d'exécution de l'algorithme pour les différentes dimensions des sous-matrices $P_{11}$, $P_{12}$, ....

| Dimension des sous-matrices | 5 | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|---|
| cas 1 | 1050 | 6200 | 18450 | 40800 | 76250 | 127800 |
| cas 2 | 875 | 5500 | 16875 | 38000 | 71875 | 121500 |

Tableau (6-3) Temps d'exécution de $APA^T$

On peut remarquer que le temps d'exécution dans le deuxième cas est meilleur. Le tableau 6-5 donne l'ordonnancement de calcul dans ce cas.

Nous avons essayé le même algorithme pour le graphe 6-3. Dans le premier cas, nous avons organisé les données de manière suivante :

mémoire 1 $H_{11}, H_{12}, z_1, H_{11}H_{11}, H_{11}H_{12}, H_{12}H_{12}, H_{11}z_2, H_{12}z_2, O_1$

mémoire 2 $H_{21}, H_{22}, z_2, H_{21}H_{21}, H_{11}H_{22}, H_{22}H_{22}, H_{22}z_2, H_{21}z_1, O_2$

Dans le second cas, l'organisation des données est faite en utilisant l'algorithme précédent. Le tableau 6-4 donne le temps d'exécution de l'algorithme en fonction du nombre de processeurs pour m=40 et n=10.



| Nombre de processeurs | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| cas 1 | 4224 | 3049 | 2434 | 2484 |
| cas 2 | 4224 | 2969 | 2408 | 2423 |

Tableau 6-4 Temps d'exécution du graphe 6-3

On peut constater que le temps d'exécution est pratiquement le même. Cela est probablement dû au fait que les deux organisations de données pour les grandes tâches sont identiques.

Tableau (6-4) Organisation du calcul de APAT

6 processeurs

| activité | durée | début | fin |
|---|---|---|---|
| 1 | 25 | 0 | 25 |
| 2 | 25 | 0 | 25 |
| 3 | 25 | 25 | 50 |
| 4 | 25 | 25 | 50 |
| 19 | 225 | 25 | 250 |
| 5 | 25 | 50 | 75 |
| 6 | 25 | 50 | 75 |
| 20 | 225 | 50 | 275 |
| 7 | 25 | 75 | 100 |
| 8 | 25 | 75 | 100 |

| | | | |
|---|---|---|---|
| 21 | 225 | 75 | 300 |
| 22 | 225 | 100 | 325 |
| 9 | 25 | 100 | 125 |
| 10 | 25 | 100 | 125 |
| 11 | 25 | 125 | 150 |
| 12 | 25 | 125 | 150 |
| 23 | 225 | 150 | 375 |
| 24 | 225 | 150 | 375 |
| 25 | 25 | 250 | 275 |
| 26 | 25 | 275 | 300 |
| 31 | 25 | 275 | 300 |
| 27 | 25 | 300 | 325 |
| 32 | 25 | 300 | 325 |
| 28 | 25 | 325 | 350 |
| 33 | 25 | 325 | 350 |
| 38 | 25 | 325 | 350 |
| 34 | 25 | 350 | 375 |
| 43 | 25 | 350 | 375 |
| 39 | 25 | 375 | 400 |
| 30 | 25 | 375 | 400 |



| | | | |
|---|---|---|---|
| 17 | 25 | 375 | 400 |
| 29 | 25 | 400 | 425 |
| 44 | 25 | 400 | 425 |
| 14 | 25 | 425 | 450 |
| 48 | 25 | 425 | 450 |
| 16 | 25 | 450 | 475 |
| 36 | 25 | 450 | 475 |
| 13 | 25 | 475 | 500 |
| 35 | 25 | 475 | 500 |
| 54 | 225 | 475 | 700 |
| 41 | 225 | 475 | 700 |
| 49 | 25 | 500 | 525 |
| 40 | 225 | 500 | 725 |
| 37 | 25 | 500 | 525 |
| 55 | 225 | 525 | 750 |
| 15 | 25 | 525 | 550 |
| 18 | 25 | 550 | 575 |
| 42 | 225 | 550 | 775 |
| 50 | 25 | 575 | 600 |

| | | | |
|---|---|---|---|
| 56 | 225 | 600 | 825 |
| 58 | 25 | 700 | 725 |
| 46 | 25 | 700 | 725 |
| 45 | 25 | 725 | 750 |
| 61 | 25 | 725 | 750 |
| 59 | 25 | 750 | 775 |
| 52 | 25 | 750 | 775 |
| 51 | 25 | 775 | 800 |
| 62 | 25 | 775 | 800 |
| 53 | 25 | 800 | 825 |
| 63 | 25 | 800 | 825 |
| 47 | 25 | 800 | 825 |
| 60 | 25 | 825 | 850 |
| 64 | 25 | 825 | 850 |
| 57 | 25 | 850 | 875 |
| 65 | 0 | 875 | 875 |

## 6-4 CONCLUSION

Nous avons étudié le problème du transfert de données entre les différents modules de calcul d'un système multiprocesseurs.



Nous avons démontré que pour un système en bus commun, l'application des techniques d'ordonnancement réduit considérablement les conflits d'accès au bus et à la mémoire communs.

Dans le cas de systèmes avec plusieurs mémoires, nous avons démontré l'importance de l'organisation des données dans les mémoires. Nous avons proposé un algorithme pour stocker des données dans les différentes mémoires. Nous avons ensuite démontré par des exemples que cet algorithme réduisait le temps de transfert des données.

# CONCLUSION GENERALE

# ET PERSPECTIVES

Dans les ouvrages existants, les algorithmes de calcul parallèle sont essentiellement orientés vers la résolution de grands systèmes d'équations linéaires ou vers la résolution par discrétisation des équations différentielles. [C-1], [C-2], [C-3]. On ne trouve que trés peu de développements sur les problèmes d'identification de paramètres ou de vecteurs d'état de systèmes industriels de grande dimension. Nous avons donc étudié au cours de ce mémoire les possibilités de résolution sous forme parallèle de ces problèmes. Ayant constaté que les ouvrages spécialisés traitaient abondamment de méthodes de décomposition et de coordination (voir [C-4], [C-5], [C-6], [C-7]) appliquées à la commande optimale des systèmes interconnectés, nous avons considéré que ces méthodes offrent une base solide pour l'application des techniques de calcul parallèle : nous avons donc orienté ces méthodes vers des applications parallèles.

Dans cet esprit, nous avons présenté dans le chapitre 1 les différentes architectures des calculateurs parallèles, ainsi que leurs avantages et leurs inconvénients. Chacune de ces architectures offre des possibilités d'application de calcul parallèle pour des problèmes d'automatique. L'étude et la réalisation des architectures convenables pour ces problèmes, en se référant aux algorithmes concrets présentés dans ce mémoire, constituent une perspective de recherche appliquée trés intéressante, en la menant parallèlement aux études théoriques. L'étude des problèmes logiciels ou matériels, tels que les systèmes d'exploitation, les techniques de synchronisation, les problèmes de gestion de mémoire ont une importance primordiale :l a réalisation d'un système parallèle nous donne également la possibilité de tester pratiquement les algorithmes parallèles proposés pour les problèmes d'automatique. La réalisation d'un

système parallèle sophistiqué étant trés coûteuse, nous pensons que l'on peut commencer par les architectures assez simples et dans notre cas par une architecture avec un bus commun. Cet architecture a déjà été réalisée en utilisant des processeurs 6809 pour les modules de calcul [ C-7].

Il est évident que l'utilisation des processeurs rapides pour les modules de calcul ne peut que rendre le système plus applicable. En effet, puisque la performance de ces processeurs est comprise entre 6 et 10 M flops, la performance d'un système parallèle composé de ces processeurs atteint un niveau trés intéressant et tout à fait comparable aux besoins des sytèmes à moyenne, voire à grande dimension.

Au deuxième chapître, nous avons étudié les méthodes de mise en évidence du parallèlisme des algorithmes, ainsi que la création des ces algorithmes parallèles. Nous avons démontré l'efficacité de l'application des graphes pour clarifier le parallèlisme et le séquencement des différentes tâches d'un algorithme. Nous avons étudié ensuite l'application des techniques de la méthode c.p.m. pour l'ordonnancement du calcul, afin de minimiser soit le temps de calcul, soit le nombre de processeurs. La complexité de ces algorithmes nous contraint à appliquer ces méthodes aux systèmes non évolutifs ou d'une évolution trés lente. La diminution de la complexité de ces algorithmes, la création de nouveaux algorithmes optimaux ou heuristiques dans le cadre des problèmes liés à l'automatisme des sytèmes interconnectés peut donc être envisagée comme une des perspectives de recherche dans ce domaine. D'autres problèmes à étudier dans ce domaine sont le choix optimal de la taille des blocs de données, le nombre de processeurs, le nombre et la capacité de blocs de mémoire, les entrées sorties. Ces problèmes rejoignent le problème de choix



d'architecture expliqué auparavant.

Le troisième chapître est consacré à la résolution parallèle des systèmes d'équations linéaires. Nous avons étudié les différents algorithmes parallèles directs ou itératifs pour la résolution de ces systèmes et nous avons comparé ces algorithmes dans le contexte des problèmes liés à l'automatisme des systèmes interconnectés. Dans ce domaine, les perspectives de recherche dans le cas d'une application en automatique sont envisageables essentiellement dans une étude approfondie et comparative des différentes méthodes itératives, surtout dans le cas de systèmes avec des matrices à structure bande.

Dans le quatrième chapître, nous avons abordé le problème de l'estimation parallèle des paramètres et du vecteur d'état de systèmes interconnectés. Nous avons notamment étudié le calcul parallèle des algorithmes classiques d'estimation tel que la méthode de moindres carrés et le filtre de Kalman. Etant donné l'énorme variété des algorithmes d'estimation, la perspective de recherche dans ce domaine est trés vaste. L'étude du parallèlisme des différents algorithmes d'estimation existants, la création de nouveaux algorithmes parallèles, l'extension de ces études dans le cas des systèmes non linéaires, en sont quelques exemples.

Dans le cinquième chapître, nous avons étudié les méthodes parallèles de calcul de commande optimale des systèmes interconnectés. Nous avons abordé les méthodes de résolution parallèle des systèmes d'équations différentielles par décomposition spatiale et temporelle. Nous avons appliqué ces méthodes au calcul de la commande optimale des systèmes continus. Nous avons également étudié le calcul parallèle de la

commande optimale dans le cas de systèmes discrets. Dans ce domaine, comme dans celui de l'estimation, la perspective de recherche est trés vaste. L'étude comparative du calcul parallèle de toutes les méthodes de décomposition-coordination pour le calcul de la commande optimale, l'extension de ces méthodes au cas des systèmes non linéaires, l'application des méthodes de calcul parallèle dans d'autres domaines de l'automatisme telle que la commande adaptative nous donnent une idée du travail qui reste à faire.

Au sixième chapitre, nous avons étudié le problème de la gestion des transferts dans le cas d'une architecture avec bus commun, ainsi que pour les architectures avec un réseau d'interconnection. Dans le premier cas, nous avons mis en évidence l'importance du séquencement de transfert des données. Nous avons démontré que ce problème peut être résolu par l'application de la méthode de c.p.m. Dans le cas des architectures, avec nombre de blocs mémoires supérieur à 1, le problème d'organisation des données dans les mémoires devient primordial. Nous avons proposé un algorithme pour l'organisation de données, mais dans ce domaine il reste à répondre à de nombreuses interrogations. Parmi celles-ci, on peut citer notamment le choix du nombre de blocs de mémoire, le choix entre le transfert de données aux mémoires locales et l'exécution des opérations en communication directe avec la mémoire globale, la taille des mémoires, le problème de diffusion de données, la distribution des données initiales, des résultats intermédiaires et des résultats finaux sur les blocs mémoires de façon à minimiser le blocage. La perspective de recherche dans ce domaine est de donner des réponses à ces problèmes, tout en sachant que ceux-ci sont fortement liés entre eux ainsi qu'à l'architecture du calculateur parallèle utilisé.

## A-1 Nombre d'opérations élementaires des algorithmes d'algèbre linéaire

1- Addition de deux vecteurs de dimension v :

v additions

2- Produit scalaire de deux vecteurs de dimension v :

v multiplications

v-1 additions

3- Multiplication d'une matrice m.v par un vecteur de dimension v :

mv multiplications

m(v-1) additions

4- Addition de deux matrices m.n :

mn additions

5- Addition de deux matrices symétriques n.n :

n(n+1)/2 additions

6- Multiplication de deux matrices m.n et n.k :

mnk multiplications

m(n-1)k additions

7- Multiplication de deux matrices m.n et n.m dont le resultat est symétrique :

nm(m+1)/2 multiplications

(n-1)m(m+1)/2 additions

8- Inversion d'une matrice n.n par la méthode de Gauss :

n divisions

$n^3$-1 multiplications

$n(n-1)^2$ additions

9- Inversion d'une matrice symétrique n.n

n divisions

$(n-1)(n^2+2n+2)/2$ multiplications

$n(n-1)^2/2$ additions

10- Résolution d'un système d'équations linéaires triangulaire par la méthode de remontée :

n divisions

$n(n-1)/2$ multiplications

$n(n-1)/2$ additions

11- Triangularisation d'un système d'équations linéaires par la méthode de Gauss :

$n(n-1)/2$ divisions

$(n^3-n)/3$ multiplications

$(n^3-n)/3$ additions

12- Résolution d'un système d'équations linéaire par la méthode de Cholesky :

n extractions de racine carrées

$n(n-1)/2$ divisions

$(n^3-n)/6$ additions

$(n^3-n)/6$ multiplications

## BIBLIOGRAPHIE


[I-1] MESAROVIC M.D, MAKO D., TAKAHARA Y. - 1970 : Theory of hierarchical multilevel systems, Academic press

[I-2] TITLI A. Editeur - 1979 : Analyse et commande des systèmes complexes

[I-3] SINGH M.G., TITLI A. - 1978 : Systems Decomposition, Optimisation and Control

[I-4] FINDEISEN W., BAILEY F.N., BRDYS M., MALINOWSKI K., WOZNIAK A. - 1980 : Control and coordination in hierarchical systems. II ASA, wiley

[I-5] LAUB A.J. - 1974 : Decentralization of optimisation problems by iterative coordination, Ph. D., Univ. of Minnesota

[I-6] HASSAN M.F, SALUT G., SING M.G., TITLI A. : A decentralized computional algorithm for the global Kalman Filter - IEEE Trans. Autom. Contr. vol. AC-23, pp 262-267

## CHAPITRE 1


[1-1] LENFANT J. - 1985 Quelle architecture pour les gigaflops ? Séminaire INRIA, 14-18 Octobre 1985 Saint Cyprien, pp (1-9)

[1-2] Advanced signal processor (ASP) MPD 77230 - 1987 publié par Nec Electronics

[1-3] TS 68930,1 (Programmable signal processor) - 1986 Thomson

[1-4] DSP 56000 (Digital signal processor) - 1977, MOTOROLA

[1-5] MC 68020 32 Bit Microprocessor - 1985 MOTOROLA

[1-6] FLYNN. M.J. : Some computer organizations and their effectiveness - Septembre 1972 - IEEE Transactions on computers, pp 948-960

[1-7] PURCEL C.J. : The control data STAR - 100 Performance measurments - 1977, National Computer Conférence , pp 385-387

[1-8] WATSON W.J., CARR H.M. : Operational experience with the TI advanced scientific computer - 1974 National Computer Conference , pp 289-297

[1-9] SITES R.L : An analysis of the CRAY-1 computer - The 5 th annual symposium on computer architecture - 1968. IEEE Press, pp 101-106

[1-10] MPD 77 230, MPD 77 220 : Relocatabe assembleur user's manual - 1977 publié par Nec Electronics

[1-11] RAMAMOORTHY C.V., L.I H.H : Pipe-line architecture computing surveys - 1977 Vol. 9, pp 61-102

[1-12] MAHMOUD M.S. SINGH M.G. : Discrete systems analysis, control and optimization - 1984, Springer-verlag

[1-13] PAPAGERGIOU M., SCHMIDT G. : On the hierarchical solutions of non linear optimal control problems, LSSTA, VOL. 1, n° 4, pp 265-271

[1-14] BUDNICK P., KUCK D.J. : The organization and use of parallel memories - 1971, IEEE Transaction on computers, Vol. C-20, pp 1566-1569

[1-15] JALBY W., FRAILONG J.M., LENFANT J. , "Diamond schemas : on organization of parallel memories for efficient array processing" - Oct. 1984, Rapport de recherche INRIA 342, INRIA

[1-16] LEFANT J. : Mémoires parallèles et réseau d'interconnexion. Technique et science d'informatique, 1982 Vol. 1, pp 135-142

[1-17] BATCHER K.E. : The multidimensional acces memory in STRAN - Fév. 1977, IEEE Transaction on computers C-26, pp 174-177

[1-18] MIKLOSKO J., KOTOV V.E. : Algorithms, software and hardware of parallel computers - 1984, Springer-verlay

[1-19] LICHNEWSKY A. : Solving some linear system arising in finite element methods on parallel processors - 1982, SIAM National Meeting, Standford

[1-20] BARNES J.H., BROWN E.M., KATO M., KUCK D.J., SLOTNICK D.L., STOKES R.A. : The ILLIAIV computer - 1968, IEEE Transactions on computers C-17, pp 746-757

[1-21] BATCHER K.E. : STARAN parallel processor system hardware - 1974, National Computer Conference, pp 405-416

[1-22] RON A. Spanke : Architectures for guided-wave optical space swithing systems - 1987, IEEE Communications Magazine, Vol. 25, n° 5, pp 42-48

[1-23] ABBOTT G.F. : Digital space division, a technique for swithing high-speed data signals - 1984, IEEE Communications Magazine, Vol. 22, n° 4, pp 32-41

[1-24] LEFANT J., CLAUD M. : Les réseaux d'interconnexion - Oct. 1985, Seminaire INRIA, Saint Cyprien, pp 9-21

[1-25] BENES V.E : Mathematical theory of connecting networks and telephone traffic - 1965, Newy-York : Academic Press

## CHAPITRE 2


[2-1] MIKLOSKO J., KOTOV V.E. : Algorithms, software and hardware of parallel computers - 1984. Springer-verlay

[2-2] MPD 77 230 R-002 : Program library systems - 1987, Nec Electronics

[2-3] JOHNSON O.G., MICHELLI C.A., PAUL G. : Polynomial preconditioners for conjugate gradient calculations - 1983, SIAM, J. Numer Anal. 20, pp 362-376

[2-4] JOLY P. : Présentation de synthèse des méthodes de gradient conjugé - 1986, RAIRO, Vol. 20, pp 639-665

[2-5] LEGRAS J. : Méthodes et techniques de l'analyse numérique - 1971, DUNOP I

[2-6] CSANKY L. : On the parallel complexity of some computational problems - 1974, Dissertation Univ. of California, Berkley

[2-7] LIFERMANN J. : Les méthodes rapides de transformation du signal - 1980, MASSON

[2-8] Advanced signal processor (ASP) MPD 77230 - 1987 publié par Nec Electronics

[2-9] PERRIN G.R. : Thèse de Doctorat d'Etat - 1985, Université de Nancy I

[2-10] LIONS J.L., MARCHOUK G.I. sur les méthodes numériques en sciences physiques et économiques - 1974, DUNOD

[2-11] KUNG H.T. : Synchronized and asynchronous parallel algorithms for multiprocessors : In algorithms and complexity - 1976, Academic Press, New-York

[2-12] CHAZAN D., MIRANKERO W. : Chaotic relaxation - 1969, Linear algebra and its application, Vol. 2, pp 199-222

[2-13] MEILLO J.C. : Asynchronous iterations and order of intervals - 1986, International workshop, Centre National de Rencontres Mathématiques, Luminy, France

[2-14] GIEN D. : Calculateur parallèle pour la commande optimale en ligne de processus dynamique - 1981, Thèse de Docteur Ingénieur, Université de Franche Comté

[2-15] FRABOUL CH. HIFDIN - 1985 : Expressing ans exploiting MIMD parallelism on a multi array processors architecture array, Meeting New Orleans

[2-16] PIERROT P., Mise en oeuvre des processeurs de traitement rapide du signal pour la conception de systèmes dédiés à l'Automatisme - 1988, Thèse de Doctorat

[2-17] DSP 56000 (Digital signal processor) - 1977, MOTOROLA

[2-18] MC 68020 32 Bit Microprocessor - 1985, MOTOROLA

[2-19] CIARLET P.G. : Introduction à l'analyse numérique matricielle et à l'optimisation - 1982, MASSON

[2-20] MC 68020 32 Bit Microprocessor - 1985, MOTOROLA

[2-21] MODER J., PHILIPPS C. : Project management with cpm and PERT - 1970, Van Nostrand Reinhold Company

[2-22] PRANGISHVILI I.V., IGNATUSHCHENKO V.V. : A uniform array-based rearrangeable control system - 1975, 6th trieanal world Congres Boston, p 133

## CHAPITRE 3


[3-1] CIARLET P.G. : Introduction à l'analyse numérique matricielle et à l'optimisation - 1982, MASSON

[3-2] Microprocesseurs 68000 MOTOROLA - 1980

[3-3] Microprocesseurs 8086 INTEL - 1982

[3-4] DSP 56000 (Digital signal processor) - 1977, MOTOROLA

[3-5] MC 68020 32 Bit Microprocessor - 1985 MOTOROLA

[3-6] KRZAKALA G. : Contribution à l'extension du calcul hiérarchisé. Application a l'équilibrage des bilans de mesures et à la commande des systèmes cascades. Thèse de Docteur ès-Sciences - 1981, Université de Nancy I

[3-7] VARGA R.S.: Matrix iterative analysis - 1962, Prentice Hall

[3-8] LAUB A.J., BAILEY F.N. : An iterative coordination approach to decentralized decision problems - 1978, IEEE Trans. Autom. control, Vol. AC-23, n° 6, pp 1031-1036

[3-9] MIELLOU J.C. : Algorithme de relaxation chaotique à retard - 1975 RAIRO, 9ème année, R.1, pp 55-82

[3-10] CALVET J.L.: Méthodes itératives et partitionnement. Application aux problèmes de commande, Thèse de Doctorat ès-Sciences, Université Paul Sabatier, Toulouse - 1984

[3-11] WINKLER C.: Basics factorization for bloc-angular linear programs : Unified theory partitioning and decomposition using the simplex method - 1974, Report Sol. 74-19, Systems optimization Laboratory : Departement of operational research, Standford Univ. of California


## CHAPITRE 4


[4-1] ZASADZINSKI M., DAROUACH M., MAQUIN D., PIERROT P., KRZAKALA G. : Validation des données des systèmes linéaires de très grande dimension à l'aide du calcul décentralisé - Févr.1988, Colloque International de IASTED Grindelwald, Suisse.

[4-2] DAROUACH M. : Observabilité et validation des données. Application à l'équilibrage des bilans de mesures. Thèse de Docteur ès-Sciences, Université de Nancy I

[4-3] MAQUIN D. : Observabilité, diagnostic et validation des procédés industriels. Thèse de Doctorat de l'Université de Nancy I - 1987

[4-4] SULTAN M.A., HASSAN M.F., CALVET J.L., BILAL A.Y. : A first-order parallel decomposition method for parameter estimation in large scale systems - 1985. North-Hollan Large Scale Systems 9

[4-5] SULTAN M.A., CALVET J.L., HASSAN M.F. : An iterative overlapping decomposition approach to parameter estimation. IEEE, 25 th Conference on decision and control Athen - Déc. 1985

[4-6] SULTAN M.A., BILAL A.Y., HASSAN M.F. : A decentralized parameter estimation. Algorithm for large scale systems - 1982, Proceeding of the modeling and simulation Conférence Piltsburgh, USA, April, pp 249-255

[4-7] YOUNG P. : Recursive estimation and time-series analysis - 1984, Springer-verlag.

[4-8] RUYMGOART P.A., SAONG T.T. : Mathematics of Kalman-Bucy Filtering - 1985, Springer-verlag

[4-9] CARLSON A. : Federated square root filter for decentralized parallel processes - 1987, IEEE, National Aerospace and Electronics, Dayton, Ohaio, USA, 18-22 Mai, IEEE, 4 Vol., pp 1540

[4-10] DEASI V.O., DAS B. : Parallel algorithms for Kalman filtering, 1985, Proceding of 1985 American Control, 19-21 Juin, IEEE

[4-11] MAHMOUD M.S. SINGH M.G. : Discrete systems analysis, control and optimization - 1984 Springer-verlag

[4-12] FOULADGAR J. : Calcul parallèle du filtre de Kalman - Fév. 1988, Colloque International de IASTED, Grindelwald, Suisse.

## CHAPITRE 5

[5-1] CESCHINO F., KUNTZMANN J. : Méthodes numériques, Problèmes différentiels de conditions initiales - 1963, Ed. Ounod, Paris

[5-2] MIRANKER W.K., LINIGER W.M. : Parallel methodes for the numerical integration of ordinary differential equations, Academic Press - 1974

[5-3] of CALVET J.L., TITLI A. : Hierarchical optimization and control large scale system with dynamical interconnection system - 1980, 2nd IFAC Symposium on large scals systems : Theory and applications, June 26-58, Toulouse

[5-4] CALVET J.L. : Méthodes itératives et partitionnement. Application aux problèmes de commande - 1984, Université Paul Sabatier, Toulouse

[5-5] LHOTE J.C., MIELLOU J.C. : Algorithmes de décentralisation et de coordination par relaxation et commande optimale : Analyse et commande des systèmes somplexes - 1979, CEPADUES

[5-6] KUNG H.T. : Synchronized and asynchronous parallel algorithms for multiprocessors in Algorithms and complexity. JF TRAUB (Editor) Academic Press, New-York - 1976

[5-7] CHAZAN D., MIRANKER W. : Chaotic relaxation - 1969, Linear algebra and its application, pp 199-222

[5-8] BARNETT S. : Matrices in control theory - 1971, Van Nostrans Reinhold-Company

[5-9] BARRAUD A.Y. : Produit étoile en fonction signe de matrice. Application à l'équation de Riccati dans le cas discret - 1980, RAIRO Vol. 14, n° 1- 1980

[5-10] HEWER G.A. : An iterative technique for the computation of the steady state gain for the discrete optimal regulator, IEEE Trans Aut. Cont., Vol. AC-16, n° 4, Août 1971, pp 382-384

[5-11] MAHMOUD M.S., SINGH M.G. : Discret systems analysis, Control and optimization - 1984, Springer-verlag

[5-12] KAILATH T., MORF M. : Square root algorithms for steady state Riccati equations, Int. J. Control, Vol. 28 n° 2 - 1978, pp 295-306

## CHAPITRE 6

[6-1] LAWRIE D.H. : Memory processor connection networks. tech. report 557, Dep. Comp. Sci. Univ. of Illinois Urbana - 1973

[6-2] BATCHER K.E. : the multidimensional acces memory in STARAN - IEEE Trans. on Computer, C-26 - 1977, 2, pp 174-177

[6-3] FRAILONG J.M., JALBY W., LENFANT J. XOR Schemes : A flexible data organization in parallel memories

[6-4] LAWRIE D.H., VARA C.R. : The prime memory system for array acces, IEEE Trans. on Computers, Vol C-24, 12, pp 1145-1155 - Déc. 1975

[6-5] LAWRIE D.H. : Access and alignment of data in an array processor, IEEE Trans. on Computers, Vol. C-31, pp 435-442 - Mai 1982

## CONCLUSION


[C-1] DUFF I.S. : Parallel implementation of multifrontal schemes CSS 174 Computer science and system division Harwell Laboratory - May 1985

[C-2] DONGARRA J.J., GUSTAVSON F.G., KARP A. : - 1984 : Implementing linear algebra algorithms for dense matrices on a vector pipeline machine 1984, SIAM Review 26, pp 91-112

[C-3] SAMEH A.H., KUCK D.J. : On stable parallel linear system salvers J. ACM, 25 - 1978, pp 81-91

[C-4] CALVET J.L., LABIE P., MANO A. : Partitionnement et méthodes itératives. Application aux problèmes de simulation et commande, Ecole d'automne du PR IV RIREN : "Modélisation et simulation des systèmes naturels " CNUSC, 4-15 Oct., Montpellier

[C-5] BERNUSSOU J., TITLI A., AUTHIE G., CALVET J.L. : Interconnected dynamical systems : stability, decomposition and decentralization - 1982, North Holland

[C-6] COHEN G. : Optimization by decomposition and coordination : a unified approach - 1978, IEEE Trans. Autom. Control, Vol. AC-23, n° 2, pp 222-232

[C-7] CALVET J.L : Méthodes itératives et partitionnement. Application aux problèmes de commande - 1984, Thèse de Docteur ès-Sciences, Univ. Paul Sabatier Toulouse.

NOM DE L'ETUDIANT : FOULADGAR Javad

NATURE DE LA THESE : Doctorat d'Etat ès sciences

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 21 JUIL. 1988 1979

LE PRESIDENT DE L'UNIVERSITE DE NANCY I