N° d'ordre : 889

50376
1981
7

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ème CYCLE

(INFORMATIQUE)

par

Jean Claude MARTI

ETUDE D'ARCHITECTURES MULTIPROCESSEURS POUR LA RESOLUTION DE PROBLEMES A L'AIDE D'ALGORITHMES CHAOTIQUES.

Thèse soutenue le 17 mars 1981 devant la Commission d'Examen

Membres du Jury

C. CARREZ	Président
V. CORDONNIER	Rapporteur
J.C. MIELLOU	Examinateur
M. ANSELME	Examinateur

P R O F E S S E U R S 1ERE C L A S S E

M. BACCHUS Pierre	Mathématiques
M. BEAUFILS Jean Pierre (dét.)	Chimie
M. BIAYS Pierre	G.A.S.
M. BILLARD Jean	Physique
M. BONNOT Ernest	Biologie
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie
M. CELET Paul	Sciences de la Terre
M. COEURE Gérard	Mathématiques
M. CONSTANT Eugène	I.E.E.A.
M. CORDONNIER Vincent	I.E.E.A.
M. DEBOURSE Jean Pierre	S.E.S.
M. DELATTRE Charles	Sciences de la Terre
M. DURCHON Maurice	Biologie
M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOURET René	Physique
M. GABILLARD Robert	I.E.E.A.
M. GRANELLE Jean Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. HEUBEL Joseph	Chimie
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LANSRAUX Guy	Physique
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEBRUN André	C.U.E.E.P.
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MAILLET Pierre	S.E.S.
M. MONTREUIL Jean	Biologie
M. PAQUET Jacques	Sciences de la Terre
M. PARREAU Michel	Mathématiques
M. PROUVOST Jean	Sciences de la Terre

.../...

Professeurs 1ère classe (suite)

M. SALMER Georges	I.E.E.A.
Mme SCHWARTZ Marie Hélène	Mathématiques
M. SEGUIER Guy	I.E.E.A.
M. STANKIEWICZ François	Sciences Economiques
M TILLIEU Jacques	Physique
M. TRIDOT Gabriel	Chimie
M. VIDAL Pierre	I.E.E.A.
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème C L A S S E

M. AL FAKIR Sabah	Mathématiques
M. ANTOINE Philippe	Mathématiques
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BKOUCHE Rudolphe	Mathématiques
M. BOBE Bernard	S.E.S.
M. BODART Marcel	Biologie
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie
M. BOSQ Denis	Mathématiques
M. BREZINSKI Claude	I.E.E.A.
M. BRUYELLE Pierre (Chargé d"enseignement)	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CHAMLEY Hervé	E.U.D.I.L.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
Mle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DEMUNTER Paul	C.U.E.E.P.
M. DE PARIS Jean-Claude	Mathématiques

.../...

M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique
M. DUEE Gérard	Sciences de la Terre
M. DYMENT Arthur	Mathématiques
M. FLAMME Jean Marie	E.U.D.I.L.
M. FONTAINE Hubert	Physique
M. GERVAIS Michel	S.E.S.
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gavriel	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HANGAN Théodor	Mathématiques
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LAURENT François	I.E.E.A.
Mlle LEGRAND Denise	Mathématiques
Mlle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mlle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MIGEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mlle PAUPARDIN Colette	Biologie

Professeurs 2ème classe (suite)

M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
M. TOULOTTE Jean-Marc	I.E.E.A.
M. VANDORPE Bernard	E.U.D.I.L.
M. WALLART Francis	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M. TOP Gérard	S.E.S.
M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jacques	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. OPIGEZ Philippe	S.E.S.

Je suis très heureux d'exprimer ici mes remerciements à

Monsieur Christian CARREZ, Professeur à l'Université de Lille I, qui me fait l'honneur de présider ce jury.

Monsieur Vincent CORDONNIER, Professeur à l'Université de Lille I, qui m'a proposé ce sujet. Ses conseils et ses encouragements m'ont permis de faire progresser et aboutir ce travail.

Monsieur Jean-Claude MIELLOU, Professeur à l'Université de Besançon, qui a bien voulu s'intéresser à cette étude et m'a apporté une aide fructueuse.

Monsieur Michel ENSELME, Ingénieur à l'ONERA, qui a accepté d'examiner ce travail et de participer au jury.

Je remercie également

Madame Patricia CARON, pour le soin qu'elle a apporté à la mise en forme et à la dactylographie de ces pages.

Madame et Monsieur DEBOCK, qui ont assuré l'impression et la réalisation de ce document.

PLAN

CHAPITRE II : COMPLEXITÉ DES APPLICATIONS ET ARCHITECTURES

1. LE PROBLEME DE LA PREVISION METEOROLOGIQUE
    - 1.1 Formulation
    - 1.2 Facteurs supplémentaires de complexité
    - 1.3 Principe de calcul
    - 1.4 Discrétisation du problème
        - *1.4.1. Le maillage*
        - *1.4.2. Discrétisation des équations*
2. AUTRES PROBLEMES
    - 2.1 Aerodynamique
    - 2.2 Résistance des structures
3. LE POINT SUR LES ARCHITECTURES EXISTANTES
    - 3.1 Introduction
    - 3.2 Architecture pipe-line
    - 3.3 Architectures multiprocesseurs
        - *3.3.1 Définition*
        - *3.3.2 Processeurs en anneau*
        - *3.3.3 Structures bus/multibus*
        - *3.3.4 Structures en réseau*
        - *3.3.5 Conclusion*

CHAPITRE III : ETUDE ET ANALYSE D'UN PROGRAMME TYPE

1. STRUCTURE GENERALE D'UN PROGRAMME
    - 1.1 Objet
    - 1.2 Découpage en éléments fonctionnels
    - 1.3 Existence d'un cycle
    - 1.4 Influence du mode de découpage en sous-domaines
        - *1.4.1 Sous-domaines non convexes*
        - *1.4.2 Sous-domaines réguliers*
2. ETUDE DU CYCLE DE BASE
    - 2.1 Le début du traitement
    - 2.2 Protocole de fin de calcul
    - 2.3 Analyse de la fonction de stockage
        - *2.3.1 Les 3 niveaux*
        - *2.3.2 Le mode de partage*
        - *2.3.3 Volumes des zones de mémoire*

ANNEXE 1

ANNEXE 2

BIBLIOGRAPHIE

page 2, ligne 28 : "..., et de rendre possible ... "

page 4, ligne 11 : "... chaotiques ... "

page 63, figuer 4 : à gauche et à droite de la courbe respectivement : "variables partagées" et "variables privées"

page 88, ligne 6 : lire : "de l'écart moyen" à la place de "de la variance"

page 94, figure 7 : lire : "+ : même w pour tous les processeurs" à la place de : "+ : monoprocesseur"

page 105, ligne 2 : lire "figure 15" et non "figure 12"

pages 131, 132, 133: lire P_G à la place de P_O

Enfin, dans la bibliographie :

[COU 77] : P. COUZOT : rapport de recherche mathématiques appliquées et informatique (1977) Grenoble

INTRODUCTION

Les contraintes économiques qui s'exercent au niveau du développement scientifique et industriel, vont, dans l'avenir, influer considérablement sur les progrès de la science informatique.

En effet, l'augmentation du prix de l'énergie doit conduire, dans les années qui viennent, les industries de pointe à accroître la part de la simulation numérique dans leurs études et leurs essais. Les exigences d'une précision accrue, ressenties tant au niveau d'équipes de recherche qu'à celui de nombreux bureaux d'études, engendrent un besoin d'affiner les modèles utilisés jusqu'à présent. Il y a de plus une forte demande pour une diminution régulière des temps de réponse et, d'une façon générale, des délais réclamés pour une résolution, ceci afin de réduire les coûts et de permettre aux industries de lutter contre une concurrence accrue.

Pour toutes ces raisons, des puissances de calcul toujours plus élevées vont être exigées par des utilisateurs de plus en plus nombreux, et les progrès de l'informatique doivent favoriser cette mutation indispensable. Des développements technologiques (tels que les circuits josephson, à arsenide de Gallium, les fibres optiques) permettent de prévoir un accroissement sensible de la vitesse de traitement des unités centrales jusqu'au milieu des années 90.

Cependant, de nombreuses applications réclament une capacité bien supérieure à celle des ordinateurs de haut de gamme actuels (par exemple le CDC 7600, ou même CRAY 1). Elles concernent de multiples aspects de la recherche fondamentale et du développement de technologies avancées (avionique, génie civil).

Face à cette demande accrue de puissance de calcul, les machines parallèles sont la seule solution actuellement envisageable. La technologie intégrée, en plein essort, a eu pour corollaire de réduire fortement les coûts des composants, et à rendre possible la conception de calculateurs constitués de plusieurs centaines, voire de plusieurs milliers de processeurs. Mais cette évolution suscite à son tour des besoins de développement d'algorithmes parallèles efficaces. Ce parallélisme peut être envisagé de plusieurs façons, comme le montre la grande variété des architectures parallèles déjà existantes. Historiquement, le passage du monoprocesseur au multiprocesseur s'est traduit par la réalisation de machines SIMD. Ainsi, ILLIAC IV,

conçue explicitement pour résoudre des équations différentielles par des méthodes aux différences finies, est composée de 64 processeurs qui exécutent de façon synchrone les mêmes instructions sur différentes données. D'autres machines, CDC Star 100, CRAY 1 fonctionnent aussi en mode SIMD. Ce sont des machines-vecteur, et leur efficacité provient d'instructions vectorielles, capables d'éxécuter en parallèle la même opération sur tous les éléments d'un vecteur.

Etant composé d'un ensemble de processeurs indépendants, un multiprocesseur asynchrone offre une plus grande flexibilité dans sa programmation qu'une machine SIMD. Ce sont Chazan et Miranker qui ont proposé une nouvelle classe d'algorithmes, qui prend en compte la nature asynchrone de tels multiprocesseurs : les algorithmes chaotiques.

Leur intérêt est multiple :

- Ils présentent une grande simplicité de formulation et de mise en oeuvre ; ceci est du principalement à la minimisation des contraintes, réalisée tant au niveau de la fonction de communication qu'au niveau de celle de contrôle de la machine.

- Ils présentent, en comparaison avec d'autres familles d'algorithmes, d'excellentes performances, selon ce que l'on peut en juger à partir des expériences menées à ce jour.

Ce travail se situe dans le cadre de l'utilisation d'algorithmes chaotiques appliqués à la résolution de problèmes discrétisés de grande taille, décrits par des systèmes d'équations qui servent de modèles à divers problèmes de la physique. A travers une série de simulation, il en précise le comportement lorsque les matrices des opérateurs ne présentent pas les caractéristiques de symétrie et de "régularité" que l'on envisage habituellement. Il aboutit, à partir de l'analyse de ces applications, à la définition d'une architecture qui nous semble bien adaptée aux principales caractéristiques des méthodes de calcul envisagées.

Le chapitre I, après avoir rapidement rappelé les recherches, menées en algorithmique parallèle, expose le principe des méthodes d'itérations chaotiques, ainsi que les principales expériences menées à leur sujet.

Dans le chapitre II, nous présentons, afin d'illustrer les difficultés rencontrées avec la plupart des grands problèmes discrétisés, l'exemple du problème de la prévision météorologique. Puis, nous faisons brièvement le point sur les architectures des machines parallèles susceptibles de les traiter.

Le chapitre III est consacré à l'étude d'une application type, ce qui nous permet d'en dégager les caractéristiques en ce qui concerne le volume des données et le taux des échanges entre les processeurs.

Dans le chapitre IV, nous présentons les résultats de simulations effectuées sur un exemple précis, ce qui nous amène à développer un modèle analytique pouvant servir à prévoir le comportement d'algorithmes choatiques, qui seraient implémentés sur diverses architectures. Un exemple est traité, qui utilise la modélisation d'une machine à bus hiérarchisés, réalisée à partir de la théorie des files d'attente.

Enfin, dans le chapitre V, après avoir exposé les éléments d'un cahier des charges, nous présentons un projet d'une architecture originale, conçue à partir d'un outil de communication utilisant des boites à lettres. Des simulations nous permettent d'évaluer les performances que l'on peut en attendre.

CHAPITRE I

INTERET DES ALGORITHMES ASYNCHRONES EN CALCUL PARALLELE

I - LA TRANSFORMATION DES ALGORITHMES SÉQUENTIELS

1.1. INTRODUCTION

La limitation de la puissance des monoprocesseurs, due à des contraintes physiques multiples (dissipation d'énergie, existence d'un bruit de fond, vitesse de la lumière), faisant apparaître l'intérêt d'une multiplication des ressources de calcul allouées à un processus, a fait naître un certain nombre de préoccupations. Celles-ci se situent à 3 niveaux bien différents, selon la nature des objectifs visés :

- La détection d'un maximum de parallèlisme dans le déroulement d'un programme.
- La conception d'algorithmes parallèles.
- L'implémentation du parallélisme au niveau des programmes comme à celui de leur exécution.

C'est au début des années 60, alors que les projets d'architectures de calculateurs parallèles étaient en train de naître dans les cartons des concepteurs les plus avancés que s'est posé la question de savoir s'il était possible, et à quel prix, de récupérer l'immense investissement intellectuel et financier constitué par l'ensemble des programmes déjà écrits pour les monoprocesseurs.

1.2. TRANSFORMATION AUTOMATIQUE

Un certain nombre de techniques ont alors vu le jour, basées sur la reconnaissance d'un parallélisme intrinsèque dans les algorithmes séquentiels courants, et, plus généralement, dans un programme donné. Des applications sont apparues, qui ont récemment conduit, par exemple, à implémenter sur Cm* [FUL 77] une version d'un compilateur Algol 68, destiné à générer des programmes objets parallèles à partir d'un programme source séquentiel. On trouvera dans [HOB 68], [EVA 80], ainsi que dans la bibliographie correspondante, des références sur ce sujet. Ces techniques sont de natures diverses. Les plus anciennes, développées au sein d'un projet de UCLA, réalisent la décomposition du programme séquentiel en une série de sous-tâches élémentaires. Les liaisons logiques existant entre ces sous-tâches à plusieurs niveaux, ainsi qu'avec les différentes variables, qu'elles utilisent, sont exprimées par des matrices qui caractérisent ainsi le graphe de ce programme. Par des opérations logiques

sur ces tableaux, l'algorithme génère à son tour des matrices décrivant le graphe d'un programme parallèle qui réalise le même processus.

Plus récemment, un certain nombre de travaux ont été publiés, proposant une solution à ce problème par l'utilisation de réseaux de Pétri. On pourra consulter en particulier [YOE 79], [CON 79] à ce sujet.

Les programmes ainsi obtenus, outre le fait que leur exécution réclame un taux de communication et de synchronisation considérable, présentent un défaut majeur : le degré de parallélisme qu'ils permettent d'obtenir est dû en bonne partie au hasard de l'écriture du programme initial. Il ne peut apparaître structurellement que dans la transformation de certaines boucles "Tant que". Illustrons ce propos par un exemple simple :

Soit un programme séquentiel pour monoprocesseur réalisant le calcul de la somme des N premiers entiers :

```
Début
    lire N
    S ← 0
    I ← 0
    Tant que (I ≤ N) faire  S ← S+I
                            I ← I+1
                       fait
Fin
```

Bien que les données soient par nature indépendantes, cette indépendance n'apparaît pas explicitement dans l'écriture du programme. Celui-ci ne pourra être transposé - automatiquement s'entend - en une version parallèle. Or, il est clair qu'une telle version existe et s'exprime simplement.

1.3. PLANIFICATION DU PROCESSUS PARALLELE

On voit donc que, même lorsqu'elle n'échoue pas complètement, la démarche précédente peut très bien ne pas toujours présenter de réel intérêt. Le fort taux de séquentialité introduit dans les programmes transformés a pour corollaire une occupation très faible des processeurs chargés de les exécuter. Aussi est-on le plus souvent conduit, au prix d'une complexité supplémentaire, à introduire la notion de planification du séquencement des tâches indépendantes.

Précisons : chaque processeur, intervenant dans la décomposition du programme primitif, demande un temps d'exécution donné, qui peut être connu d'après la nature et le nombre des instructions qui le composent mais qui peut aussi bien être une variable aléatoire : en effet, en terme de calcul parallèle, l'architecture d'une machine peut influer dans une très large mesure sur le déroulement d'un algorithme. On peut cependant attribuer à un noeud du graphe un temps qui soit, par exemple un temps moyen, ou bien au contraire le maximum des temps correspondant à plusieurs configurations susceptibles de se produire.

Le mode d'affectation des tâches aux processeurs est alors un facteur important de l'optimisation du programme parallèle, et peut modifier dans une très large mesure, le déroulement et la puissance d'un algorithme.

Pour illustrer ce point de vue, imaginons trois tâches indépendantes, réclamant chacune une unité de temps pour être exécutées. Si, disposant de deux processeurs, on affecte une tâche complète à un processeur donné, on a le schéma de la figure 1 sur lequel le temps de calcul est de deux unités de temps. Si, par contre, on admet qu'une affectation puisse être modifiée en cours de calcul, on a le schéma de la figure 2, auquel correspond un temps de calcul qui n'est plus que de 1,5 unités de temps.

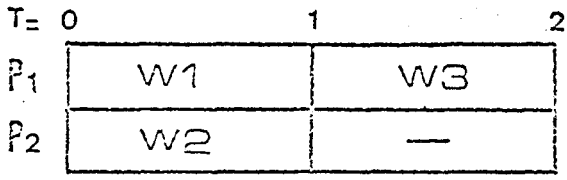
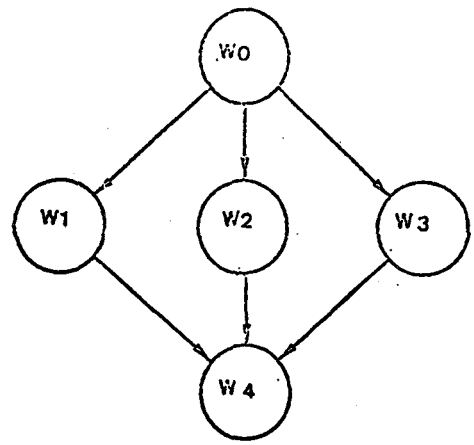


figure 1



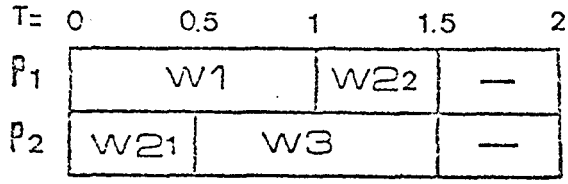


figure 2

Ce genre de problème a été particulièrement bien étudié lorsqu'on ne dispose que de 2 processeurs [COF 69]. Pour un nombre plus élevé, les difficultés d'optimisation sont considérables.

II - ADAPTATION DES ALGORITHMES SÉQUENTIELS

2.1. ADAPTATION

Une autre approche dans la conception des algorithmes parallèles est apparue d'emblée beaucoup plus fructeuse. C'est celle qui consiste évidemment à adapter les algorithmes séquentiels connus, voire à en imaginer d'autres, tout à fait originaux. Dans le premier cas, la démarche s'apparente à réaliser, quoique d'une façon beaucoup plus souple, ce qu'aurait effectué, avec une efficacité médiocre, un traitement automatique du genre de ceux qui viennent d'être décrits. La plupart du temps, elle conduit à restructurer, en partie ou totalement, le schéma du programme initial. L'intérêt apparait immédiatement, quand on considère par exemple une méthode parfaitement séquentielle par essence, comme la méthode de Gauss-Seidel, utilisée dans la résolution de systèmes matriciels. Conrad et Wallachs [WAL 77] décrivent quelques algorithmes qui rompent cette séquentialité et permettent son adaptation au calcul parallèle. De fait, s'il existe plusieurs versions plus ou moins équivalentes d'un même algorithme, c'est qu'il existe aussi différentes façons d'adapter un processus au traitement parallèle, selon que l'on cherche à utiliser tel ou tel aspect de ce mode de calcul.

2.2. PROCEDES DE DECOMPOSITION

D'une façon générale, les procédés employés sont des procédés de décomposition qui tendent à réduire la complexité de l'algorithme. Le choix du modèle de décomposition adopté est lié aux caractéristiques de dépendance entre les sous-ensembles que l'on fait apparaître.

Lorsqu'il s'agit d'une dépendance temporelle, cette décomposition peut s'entendre dans le sens du travail à la chaîne, c'est à dire une subdivision des sous-tâches composant un processus. Ce traitement reste séquentiel, mais comporte un recouvrement des temps d'exécution. On utilise

plus particulièrement ce type de méthode dans le cas de traitement d'image ou de signal ("macro pipelining").

Lorsque la simultanéité d'exécution est possible, deux procédés sont utilisés :

- Dans certains cas, on peut envisager de résoudre des sous-problèmes de même dimension que le problème initial, mais plus simples dans leur formulation [ELT 76]. C'est ce qu'on réalise lorsqu'on cherche à résoudre, par itérations par exemple, des équations non linéaires portant sur plusieurs variables [BUC 77]. D'autres exemples apparaissent dans des algorithmes décrits par MIELLOU sous l'appellation de méthodes de relaxation chaotique à retards [MIE 74].

- Le plus souvent, la décomposition s'opère au niveau des données, de façon à réduire la taille d'un problème ou le champ d'application de l'algorithme au niveau d'un processeur. On est amené à résoudre alors des problèmes de même nature que le problème initial, mais comportant un plus petit nombre de variables. C'est certainement ce point de vue qui permet le plus aisément d'adapter pour des machines parallèles, avec une bonne efficacité, les algorithmes classiques. De ce fait, les références sont nombreuses dans ce domaine et portent principalement sur le calcul matriciel les problèmes d'évaluation de fonctions, de traitement d'image [MIR 71], [WAL 77], [KUC 77].

2.3. ALGORITHMES ET CLASSES D'ARCHITECTURES

Pour les machines SIMD et pipe-line, les algorithmes sont souvent construits autour d'une vectorisation des variables. Pool et Vooght [POO 74] présentent un ensemble complet de telles méthodes de calcul. Ce type de conception nous semble présenter de grandes similitudes avec le style propre à la programmation en APL, et est soumis à des règles identiques en ce qui concerne l'organisation du programme qui en découle.

En ce qui concerne les réseaux de processeurs, de nombreux algorithmes consistent à opérer une décomposition au niveau de certains paramètres, de façon à réaliser une sorte de multiplexage. On trouve dans [MIR 75] et [MIR 77] des études portant sur des méthodes d'interpolation,

de recherche de racines, de résolutions d'équations à point fixe. De même, ces organisations matérielles sont mises à profit pour envisager des méthodes par anticipation consistant à lancer dans différentes directions à la fois des tentatives, et à ne retenir, pour progresser, que celle qui réussit le mieux (parallel shooting). Cette technique est utilisée en simulation de processus, dans des algorithmes d'exploration de graphe ou dans des méthodes numériques de prédiction-correction [MIR 71].

Les machines de type MIMD présentent a priori une adaptabilité plus grande à toutes les méthodes que nous venons d'évoquer. Moyennant des modifications mineures, celles-ci s'appliquent sans problèmes. De nombreux auteurs font référence à des méthodes spécifiques à ce type d'architecture : ce sont les méthodes asynchrones sur lesquelles nous reviendront dans la section 4.4.

III - LES MÉTHODES À PARALLÉLISME INTRINSÈQUE

3.1. CLASSES DE PROBLEMES CONCERNES

Il existe une importante classe de problèmes pour laquelle un parallèlisme apparaît, indépendemment des algorithmes de traitement, dans leur principe même. Importants, ils le sont à plusieurs titres ; par leur taille comme par l'intérêt qu'ils suscitent. Leur étude, ces dernières années, a généré un gros besoin en machines parallèles pour des applications scientifiques, et a impulsé dans une large mesure des recherches sur ces sujets.

De très nombreux processus physiques sont décrits par des équations différentielles aux dérivées partielles, que l'on ne saurait pratiquement résoudre autrement que par des méthodes de discrètisation.

Parmi toutes les applications, citons :

- L'étude de très nombreux types de champ, en régime transitoire ou permanent (méthodes stationnaires ou instationnaires).
- Les problèmes de la prévision météorologique et de l'élaboration d'un modèle théorique de l'atmosphère.

- Le problème d'élasticité en résistance des matériaux, le calcul des modes propres de vibrations des structures.

- En mécanique des fluides, les problèmes d'écoulement, de répartition des vitesses, de pressions.

- En thermodynamique, la diffusion de la chaleur.

- En électricité et en magnétisme, la répartition des grandeurs électriques dans différents milieux (débouchant en particulier, pour la CAO sur la modélisation de tous les types de semi-conducteurs).

- D'une façon générale, toutes les applications se rapportant à la simulation ou à l'étude des milieux continus.

- Un certain nombre de processus ayant pour siège des milieux discrets (chimie, traitement d'image) ou qui ont une formulation mixte (astronomie, interaction de particules).

Parmi ces méthodes, la plus facile à mettre en oeuvre est celle des différences finies.

3.2. LA METHODE DES DIFFERENCES FINIES

Rappelons-en brièvement le principe ; le domaine de variation continue des arguments d'une équation différentielle aux dérivées partielles est remplacé par un ensemble fini Ω constituant un réseau de points. Ce faisant, on dit que l'on a opéré un maillage du domaine initial, correspondant à une portion du plan ou de l'espace. Les noeuds du maillage sont formés par l'ensemble des points du réseau.
Les dérivées, les conditions aux limites éventuelles, sont remplacées par des combinaisons linéaires de valeurs prises en certains noeuds par des variables qui représentent les valeurs discrètes de la fonction étudiée.
La résolution de l'équation différentielle se ramène ainsi à la résolution d'équations algébriques, souvent linéaires, ou que l'on rend linéaires en découplant les arguments.

3.3. DISCRETISATION DES OPERATEURS DIFFERENTIELS

Il est toujours possible de proposer pour un même maillage, plusieurs modèles de discrétisation relatifs à un même opérateur. Le schéma le plus courant est probablement le schéma centré réalisé sur un maillage carré (figure 3). Désignons par ϕ_i les valeurs prises par une fonction

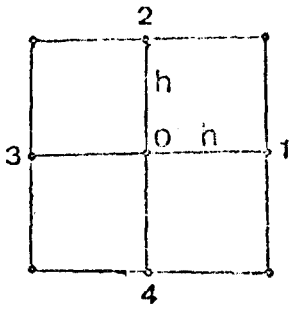


Fig. 3

inconnue ϕ en chacun des noeuds i ($0 \leq i \leq 4$). A l'aide de la formule de Taylor, l'expression discrétisée du laplacien $\Delta\phi = \frac{\partial^2\phi}{\partial n^2} + \frac{\partial^2\phi}{\partial y^2}$ s'écrit :

$$\mathcal{D}(\phi_o) = \frac{\phi_1 + \phi_2 + \phi_3 + \phi_4 - 4\phi_o}{h^2}$$

avec un ordre d'approximation en h^4.

Le domaine de validité d'un tel opérateur approché est donc, en premier lieu, celui de l'approximation de la formule de Taylor : le pas h doit être suffisamment petit, sauf là où la variation de la fonction ϕ est faible.

Cette condition réalisée, la résolution de $\Delta\phi = b$ est équivalente à celle du problème matriciel $A\phi = b$, ou A est une matrice carrée nxn, ϕ et b sont des vecteurs de dimension n, n étant le nombre de points du domaine discrétisé.

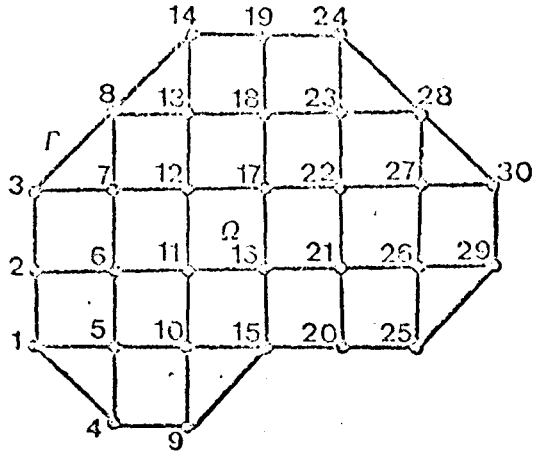


Fig. 4

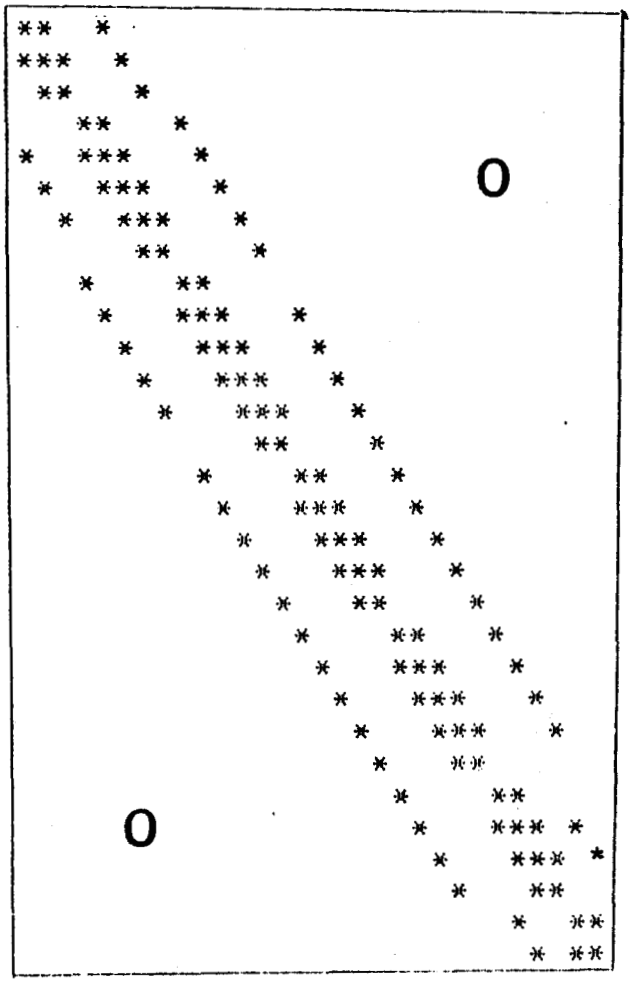


Fig. 5

Cet opérateur matriciel n'est pas quelconque. Supposons que Ω soit un ensemble de points du plan, représentés à la figure 4. On en déduit l'aspect de la matrice A, représentée à la figure 5, qui est celui d'une matrice bande, particulièrement creuse. Symétrique dans le cas de l'exemple étudié ($\Delta\phi$), elle ne l'est pas dans le cas général.

Un tel schéma de discrétisation est source de difficultés (figure 4) lorsque les frontières du domaine ne sont pas constituées d'éléments rectangulaires. On trouvera dans [SAM 78] les expressions de tels opérateurs dans des réseaux constitués d'éléments triangulaires, hexagonaux ou trapézoïdaux, réguliers ou non.

Dans ces conditions, les schémas aux différences finies constituent une méthode de calcul souple, susceptible de s'appliquer à la quasi-totalité des problèmes évoqués précédemment. Conduisant à de très gros systèmes linéaires, leur efficacité réelle dépend principalement de la puissance des algorithmes de résolution de ces dernières.

3.4. LES METHODES DE RELAXATION

3.4.1. *Relaxation*

Le procédé le plus courant utilisé pour résoudre les gros problèmes linéaires, caractérisés par le fait que les matrices des opérateurs correspondants sont très creuses, est un procédé itératif.

Etant donné l'équation matricielle $Ax = b$, on peut toujours trouver une matrice B et un vecteur v tel que x vérifie aussi $x = Bx + v$. On sait que, étant donné un choix initial de x, noté x^0, la suite $x^{k+1} = Bx^k + v$ converge, lorsque k tend vers l'infini, vers la solution du système si l'une des 2 propositions suivantes est satisfaite :

- $\rho(B) < 1$ où $\rho(B)$ désigne le rayon spectral de la matrice B, c'est à dire le module de sa plus grande valeur propre,
- Il existe une norme matricielle (notée $||.||$) telle que $||B|| < 1$,

Le rayon spectral $\rho(B)$ représente une mesure de la vitesse de convergence de la suite itérative. Plus précisément, la méthode converge d'autant plus vite que le rayon spectral est petit.

Dans la pratique, de telles méthodes sont particulièrement lentes, et le nombre d'itérations nécessaires pour obtenir la convergence peut atteindre, voire dépasser le millier, à moins que la valeur initiale x^0 de x ne soit très proche de la valeur x^* retenue en fin de calcul. Cette lenteur dans la convergence est certainement l'inconvénient majeur de ces méthodes et a introduit la forte demande que l'on connait en possibilité d'exécution parallèle pour les très gros problèmes. Cet aspect mis à part, les méthodes de relaxation sont fort intéressantes :

- Elles nécessitent tout d'abord des bases de données simples et de faible volume. Pratiquement, on utilise l'image matricielle du domaine de calcul, ce qui conduit à stocker généralement un réel par variable

et par noeud, deux lorsqu'on a besoin d'avoir simultanément les anciennes et les nouvelles valeurs. Le fait, en calcul parallèle, de découper le domaine de calcul en éléments (sous-domaines) que l'on attribue à différents processeurs tend à réduire considérablement le volume d'informations nécessaires par processeur.

- Leur formulation est d'une très grande simplicité et conduit à des programmes extrêmement courts.

- Elles autorisent la propagation numérique régulière des influences les plus minimes et permettent une résolution particulièrement élégante de phénomènes dans lesquels apparaissent des frontières, des discontinuités variables de part et d'autre desquelles différents opérateurs s'appliquent. Des problèmes mixtes (elliptiques-hyperboliques) ou non linéaires (par découplage) trouvent là un outil de résolution d'un emploi aisé.

3.4.2. *Sur-relaxation*

La méthode de sur-relation s'efforce d'accélérer la vitesse de convergence par l'introduction d'un paramètre w appelé "facteur de sur-relaxation". Par le biais de ce paramètre, on réalise en fait une pondération de l'influence des variables situées en un noeud du maillage et aux noeuds immédiatement voisins. La formulation est classique en séquentiel : soient D, U et L les matrices issues de la décomposition de B, de telle sorte que

$$B = D - U - L$$

où D est la matrice diagonale composée des éléments diagonaux de B, U et L étant consitués des éléments situés respectivement au dessus et au dessous de la diagonale de B.

à l'équation $Dx = (L + U)x + y$ on fait correspondre

$$[D-wL] = (1-w)Dx + wUx + wv$$ qui conduit à la suite itérative

$$x^{k+1} = (D-wL)^{-1}((1-w)D + wU)x^k + w(D-wL)^{-1} y$$

dont la convergence est assurée pour $0 < w < 2$.

L'opérateur $\mathcal{L}_w [D-wL]^{-1} [(1-w)D+wU]$ est appelé opérateur de relaxation. Le rayon spectral $\rho(\mathcal{L}_w)$ est une fonction de w, et on montre qu'il existe une valeur w_o de w pour laquelle il est minimum. Ceci a lieu en fait dans l'intervalle [1, 2]. Ce minimum est très accusé (figure 6), ce qui laisse entendre que le choix de la valeur de ce paramètre réclame une

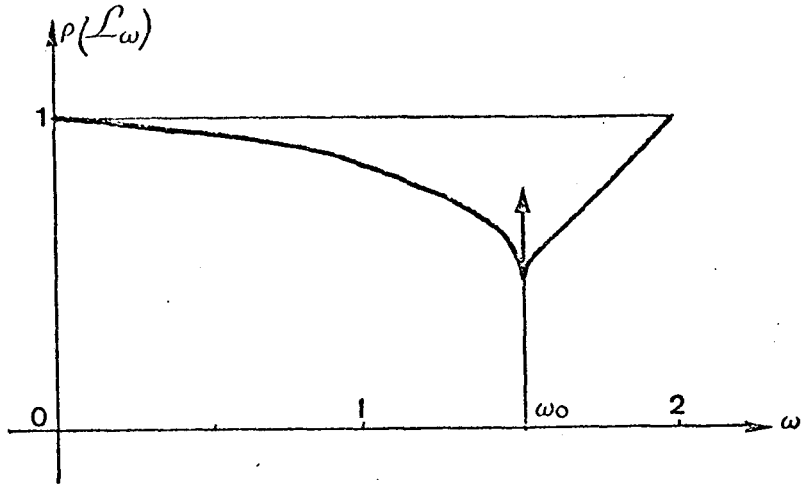


Fig. 6

grande précision. Dans la pratique, une variation de 5% autour de la valeur optimale peut conduire à doubler le temps de calcul.

w_o dépend de la forme du maillage, bien plus que de celle de l'équation et de ses coefficients. Rappelons que, en calcul séquentiel, la théorie indique, dans le cas du maillage régulier d'un domaine rectangulaire, de dimension K1 et K2, que la valeur de w_o est donnée pour l'équation de Poissons, par :

$$w_o = 2 - \sqrt{2}\,\pi \sqrt{\frac{1}{N_1^2} + \frac{1}{N_2^2}}$$

où N_1 et N_2 sont les nombres de mailles dans les deux dimensions du rectangle (à condition que leur valeur soit assez élevée. Le rayon spectral de l'opérateur est donné par :

$$\rho(\mathcal{L}_w) = \frac{1}{2}\left(\cos\frac{\pi}{N_1} + \cos\frac{\pi}{N_2}\right) < \rho(B)$$

L'utilisation de cette théorie, classique en analyse numérique séquentielle, est susceptible de poser un certain nombre de problèmes dans le cas d'un emploi pour un multiprocesseur. Rien n'indique que les résultats qui viennent d'être exposés restent valables en calcul parallèle, indépendemment du type d'architecture sur laquelle serait implémentée un tel algorithme.

IV - LES MÉTHODES ASYNCHRONES

4.1. INTRODUCTION

Les recherches effectuées pour développer des algorithmes parallèles pour multiprocesseurs asynchrones s'inscrivent dans le cadre d'une démarche originale, car, bien sûr, les problèmes rencontrés n'ont aucun équivalent en algorithmique séquentielle. L'avenir prometteur de ces nouvelles méthodes s'est plus particulièrement précisée à partir de la publication des travaux de Chazan et Miranker [CHA 69].

Ces derniers ont insisté sur le coût de la synchronisation entre processus parallèles. Dans l'exemple illustré à la figure 7, on constate l'existence de points obligés de synchronisation, entrainant une réduction élevée de l'activité réelle des processeurs. A cela s'ajoute le fait que la fonction de contrôle peut constituer une composante importante du coût du calcul, et contribuer à faire chuter les performances globales de l'architecture.

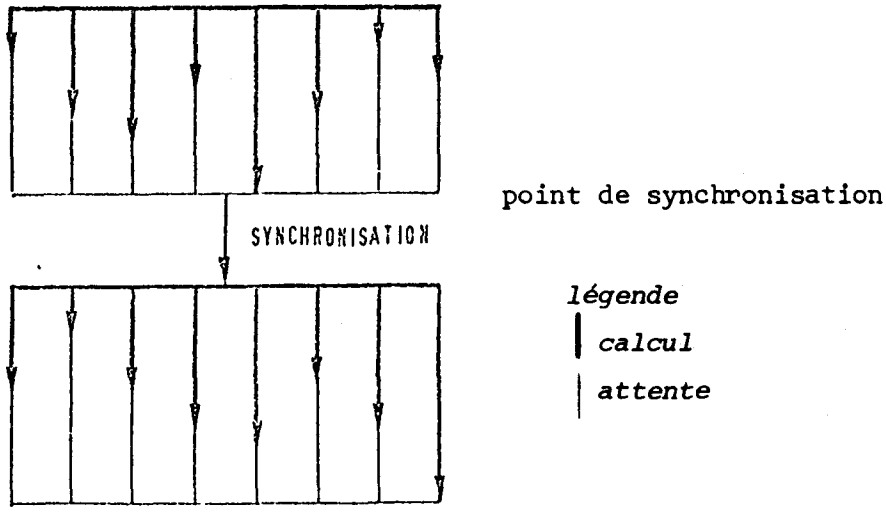


Fig. 7

Ces problèmes sont évoqués dans de nombreux articles sous le nom de "Book keeping problem). Chazan et Miranker indiquent que le meilleur moyen de le résoudre est encore de l'ignorer. Ceci a des conséquences importantes au niveau de la conception d'une machine MIMD :

- La fonction de contrôle de l'exécution est simplifiée à l'extrême.
- Les communications entre les processeurs sont mieux réparties dans le temps que dans une machine SIMD et les possibilités de blocage sont plus réduites.

- Dans la mesure où le système chargé de gérer les ressources de la machine peut décider d'opérer un transfert de tâches d'un processeur à un autre, il est tout à fait impossible d'évaluer le temps d'exécution de celles-ci par rapport au temps d'exécution de chacune des instructions les composant. Ce paramètre doit donc, d'une façon générale, être considéré comme une variable aléatoire.

4.2. EXEMPLE D'UNE METHODE ASYNCHRONE

G. BAUDET [BAU 78] a développé pour des multiprocesseurs capables d'un fonctionnement asynchrone (Cmmp et Cm*) un algorithme qui tient compte des fluctuations du temps d'exécution.

Considérant une séquence de N tâches w_1, ..., w_n, indécomposables et linéairement ordonnées, il définit la règle suivante, en vue de leur exécution par le processeur :

- initialement, tous les processeurs commencent par exécuter la tâche w_1
- Tant que la tâche w_n n'a pas commencé à être exécutée, un processeur libre se lance dans l'élaboration de la première tâche non achevée de la liste.

La figure 8 illustre le déroulement d'un tel algorithme, pour lequel les temps t_i indiqués correspondent à l'achèvement d'une tâche w_i. Les portions de traits renforcés correspondent aux parties du traitement réellement utile pour le déroulement du programme. Cet algorithme, d'une remarquable simplicité de formulation, doit sa puissance non à un partage des tâches parmi les processeurs, mais à l'utilisation rationnelle de leur asynchronisme. Miranker [MIR 77] donne également un exemple d'algorithme de recherche de racine avec 2 processeurs, qui met en évidence les avantages non négligeables des méthodes asynchrones.

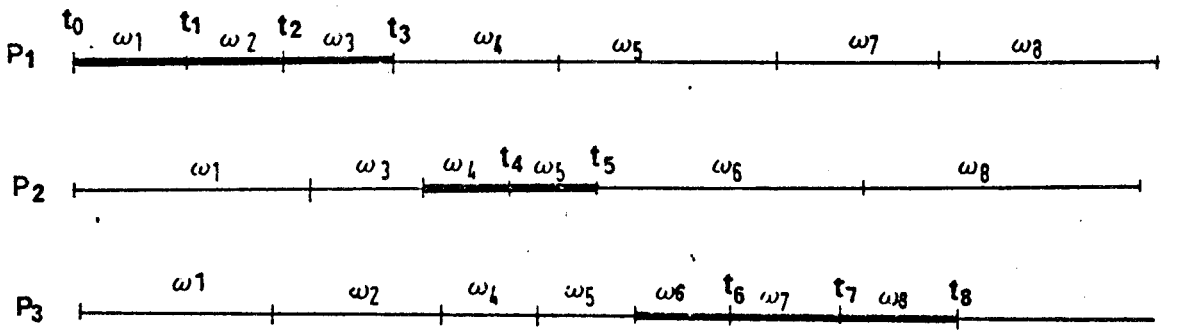


Fig 8. Déroulement d'un algorithme chaotique avec 3 processeurs P1,P2,P3

4.3. QUELQUES DEFINITIONS

Nous donnons les définitions suivantes relatives aux mesures du parallélisme et aux évaluations de performances [BAU 78], [MAS 78]

Parallélisme efficace (Speed-up ratio)

Soit $T_k(n)$ le nombre moyen d'exécution de n tâches $w_1,\ldots, w_n$ sur k processeurs. Nous définissons le parallèlisme efficace par l'expression :

$$S_k(n) = \frac{T_1(n)}{T_k(n)}$$

où $T_1(n)$ est le temps d'exécution en séquentiel. Dans la pratique, $S_k(n)$ est toujours inférieur à k et mesure le nombre de processeurs équivalents, pleinement efficaces dans la résolution du même problème.

Occupation moyenne ou efficacité

C'est le nombre : $E_k(n) = \frac{S_k(n)}{k}$ $\quad E_k(n) \leq 1$

La variation de ce paramètre en fonction du nombre k de processeurs peut permettre de définir s'il existe, le nombre optimal de ces derniers, nécessaire au traitement des n processus considérés.

Efficience [KUC 77]

KUCK introduit cette grandeur qu'il définit par :

$$F_k(n) = \frac{S_k(n)}{C_k(n)} \leq 1.$$

où $C_k(n) = k * T_k(n)$ est une mesure d'espace-temps du coût de l'algorithme.

4.4. LES ALGORITHMES CHAOTIQUES

4.4.1. *Introduction*

Les algorithmes chaotiques de relaxation ont été introduits par A. Ostrowski [OST 55] et S. Schechter [SCH 59] sous l'appellation d'itérations à conduite libre (Free-steering). C'est sous cet aspect qu'ils ont été étudiés dans le contexte de l'optimisation par Fiorot et Huard [FIO 74]. Ils ont été repris à travers une simulation, dans une formulation plus générale, par J. Rosenfeld [ROS 69] et développés par Chazan et Miranker [CHA 69]. Donnelly [DON 71] a examiné leur comportement dans le cas de schémas périodiques. Certains aspects de l'étude de Chazan et Miranker ont été étendus par Miellou [MIE 74], [MIE1 75] à l'approximation d'un point fixe pour des applications non linéaires contractantes. Cet auteur expose dans [MIE275],[MIE 77] des résultats concernant des propriétés de comportement monotone et d'encadrement de solutions, en relation avec des notions d'ordre partiel, ainsi que des propriétés de stationnarité et de convergence vers la solution. Cette notion d'ordre partiel, associée aux algorithmes chaotiques, a été reprise par Couzot [COU 77] et El Tarasi [ELT 76]. Enfin, G. Baudet [BAU 78] a étendu la notion d'itération chaotique à celle d'itération asynchrone ce qui lui permet de regrouper l'ensemble de ces travaux sous un même formalisme.

Considérons le système linéaire $Ax = b$, dans lequel A est une matrice carrée nxn, x et b 2 vecteurs de dimension n. La solution de ce système, lorsqu'elle existe, vérifie une équation à point fixe qui s'écrit :

$$x = Bx + Cb$$

Sans rien perdre de la généralité de l'étude, cette équation peut être réduite à sa forme homogène en posant $b = 0$.

Désignons par b_i la ligne i de la matrice B, et par x_i la $i^{ème}$ composante du vecteur x. L'équation à point fixe peut être résolue à partir d'une valeur initiale x^0 du vecteur x par des itérations successives de terme général $x^{j+1} = Bx^j$. Supposons que nous disposions de n processeurs pour effectuer ce calcul. A chaque processeur, nous affectons une composante de x^j, soit, au processeur $P_i (1 \leq i \leq n)$, la composante x_i^{j+1}.

Ces processeurs ne travaillant pas en phase, seront rapidement amenés à effectuer l'évaluation de la composante dont ils sont chargés à partir d'un état du vecteur x inconnu, imprévisible ; la convergence de la suite prend ainsi une apparence chaotique.

4.4.2. *Définition*

Chazan et Miranker définissent un schéma d'itération chaotique comme une classe de séquences de vecteurs x^j, où j peut prendre toutes les valeurs entières comprises entre 1 et l'infini.

Chaque séquence est définie récursivement dans sa classe par :

$$x_i^{j+1} = \begin{cases} x_i^j & \text{si } i \neq k_{n+1}(j) \\ \sum_{\alpha=1}^{n} b_{i\alpha}\, x_\alpha^{j-k_n(j)} & \text{si } i = k_{n+1}(j) \end{cases} \qquad \forall\, j \in \mathbb{N}^+$$

Soit S le vecteur de n+1 composantes défini par :

$$S = \{k_1(j), \ldots, k_n(j), k_{n+1}(j)\}$$

Les n premières composantes de S représentent un "retard" pris dans l'élaboration des n composantes de x, par rapport au niveau j de relaxation atteint à un instant donné. Ce retard **Chazan et Miranker** le supposent fini : il existe alors un entier, $s > 0$, tel que :

$$0 \leq k_i(j) \leq s \qquad \forall\, i \in [1, n],\ \forall\, j \in \mathbb{N}^+$$

Baudet étend cette définition à tout type d'opérateur de $\mathbb{R}^n$ dans $\mathbb{R}^n$, linéaire et non linéaire, en supprimant la restriction portant sur la limite des $k_i(j)$.

Enfin, le terme $k_{n+1}(j)$ correspond au choix de la composante sur laquelle s'effectue la relaxation. Il prend donc la même valeur i un nombre indéfini de fois, autrement dit :

$$1 \leq k_{n+1}(j) \leq n \qquad \forall\, j \in \mathbb{N}^+$$

On identifie alors un algorithme d'itération chaotique au triplet $\{B, x^0, S\}$.

Précisons que cette définition implique qu'au cours du calcul le processeur P_i effectue la mise à jour de la composante x_i dont il a la charge, à partir de l'état des autres composantes, quel que soit le niveau de relaxation que celles-ci ont atteint. Il est clair qu'un tel algorithme ne nécessite aucune synchronisation au niveau des communications entre les processeurs.

4.4.3. *Exemples d'algorithmes chaotiques*

Nous serons par la suite amenés à évoquer les exemples suivants :

- La méthode de Jacobi (J-méthode) consiste à évaluer la composante x_i^{j+1} à partir de x^j. Programmée en asynchrone, elle revient à opérer par cycles de calculs, en début desquels les valeurs de x sont lues et à la fin desquels les nouvelles valeurs mises à jour sont enregistrées.

L'algorithme correspondant (que Baudet appelle Jacobi-asynchrone, ou J-A-méthode) est défini par le triplet :

$$(B, x^0, S)$$

avec $k_1(j) = k_2(j) = \ldots = k_n(j) = n\ L\ \frac{(j-1)}{n}$ $\forall\ j \in \mathbb{N}^+$, $\forall\ i \in [1, n]$

où L désigne la fonction plancher.

$$k_{n+1}(j) = 1 + (j-1)\ [\text{mod } n],\ \forall\ j \in \mathbb{N}^+$$

- La méthode de Gauss-Seidel (G-S méthode) est de nature essentiellement séquentielle. Elle est donc difficile à décrire en terme de calcul parallèle. Nous pouvons lui faire correspondre la méthode suivante qui sera notée AGS-méthode : Elle consiste à faire évaluer par un processeur un sous-ensemble de variables selon un algorithme de Gauss classique, et à en transmettre les nouvelles valeurs à la fin de chaque balayage.

- Le schéma chaotique pur, ou asynchrone pur (A-P-méthode) ; on ne fait porter aucune condition sur les composantes de S. Un processeur évalue une composante de x à partir des valeurs immédiatement accessibles, et en transmet la nouvelle valeur aux processeurs concurrents aussitôt après son calcul.

Il existe un cas particulier qui peut être amené à jouer un rôle important dans la pratique, et qui a été appelé schéma périodique par Chazan et Miranker [CHA 69], [DON 71].

Supposons que nous disposions de R processeurs identiques ($1 \leq R \leq n$), auxquels est affecté la même charge de travail et négligeons tous les aléas d'exécution pouvant provenir de l'architecture. Dans ces conditions, $k_{n+1}(j)$ est une fonction périodique de j, et le schéma de calcul étant parfaitement régulier, les retards à la relaxation sont identiques.

On a donc : $k_1(j) = \ldots = k_n(j) = \begin{cases} j-1 \text{ si } j < R-1 \\ R-1 \text{ si } j \geq R-1 \end{cases}$

et $k_{n+1}(j) = j-1 \ [\text{mod } n]$

Il est commode de représenter ce schéma par un graphe (figure 9), en fonction de la variable j (à laquelle correspond un temps t_j d'exécution). Chaque courbe de numéro donné correspond à l'activité du processeur de même numéro.

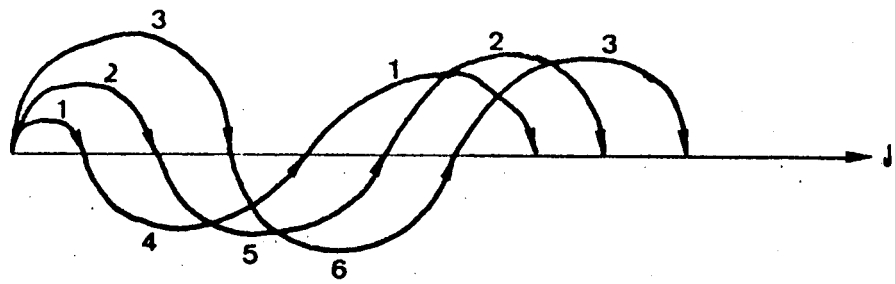


Fig. 9 Itérations chaotiques périodiques
6 variables, 3 processeurs.

De fait, il est extrêmement rare, dans l'implémentation d'un tel algorithme pour la résolution d'un problème donné, d'avoir des charges de calcul identiques pour tous les processeurs, dans le cadre du type de fonctionnement décrit (une ou plusieurs variables, toujours les mêmes attribuées au même processeur). Aussi dans la pratique, pour des charges données, le schéma réel obtenu est pseudo-périodique, la valeur de la pseudo-période étant donnée par celle du PPCM de toutes les charges. A partir du graphe de la figure 10, il est facile de vérifier que le travail de tous les processeurs n'a pas la même efficacité pour le déroulement global de l'algorithme. Il est par conséquent logique de considérer que cette efficacité sera d'autant plus grande que l'on se rapprochera plus du schéma périodique.

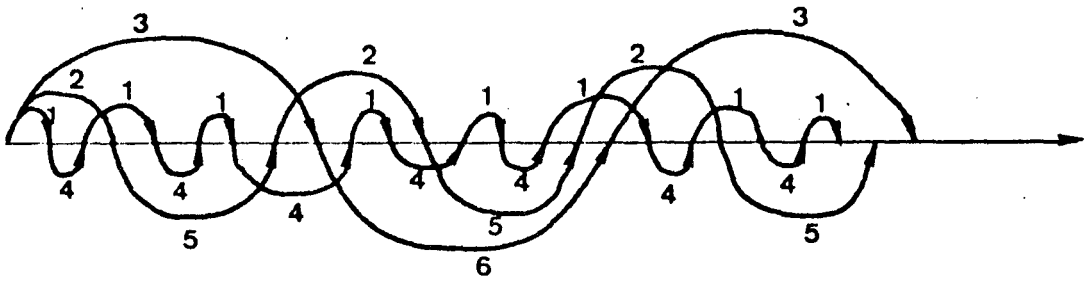


Fig. 10 Itérations chaotiques pseudo-périodiques

4.4.4. Convergence des schémas chaotique

Chazan et Miranker établissent un théorème qui fait dépendre la convergence de la valeur du rayon spectral de la matrice B.

Théorème :

Etant donné le triplet (B, x°, S) définissant un schéma chaotique, la séquence d'itérations définie à partir de x° converge, s'il existe un vecteur positif v et un nombre w tels que :

$|B|\ v \leq w.v$ où $|B|$ représente la valeur absolue de B

Ceci a lieu si le rayon spectral de $|B|$, noté $\rho(|B|)$ est inférieur à 1. Lorsque cette condition n'est pas remplie, on ne peut rien dire de la convergence.

Ce théorème est étendu [MIE 75], [BAU 78] au cas d'opérateurs contractants définis à partir d'une matrice de Lipschitz A. Un cas particulier important d'un tel opérateur est constitué par celui de sur-relaxation, que nous noterons :

$$B_w = I - w\, D^{-1} A = I(1-w) + w\, D^{-1} E$$

où D est la matrice diagonale associée à A et E la matrice complémentaire.

On démontre que le schéma chaotique défini par (B_w, x_o, S) converge pour : $0 < w < \frac{2}{1+\rho(A)}$, A étant une matrice définie positive.

4.4.5. Les expérimentations

4.4.5.1. Les travaux de Rosenfeld

Rappelons que c'est Rosenfeld [ROS 69] qui a, par ses travaux attiré l'intérêt sur les méthodes précédemment exposées. Ceux-ci se situent dans le cadre de simulations,menées sur un modèle général de machine multiprocesseur, d'un algorithme chaotique correspondant à la définition qu'en ont donné par la suite CHAZAN et MIRANKER.

Le schéma d'organisation de la machine considérée, donné à la figure 11, consiste en N_p processeurs connectés à N_{SM} modules de mémoire indépendants à travers un sous-système de communication.

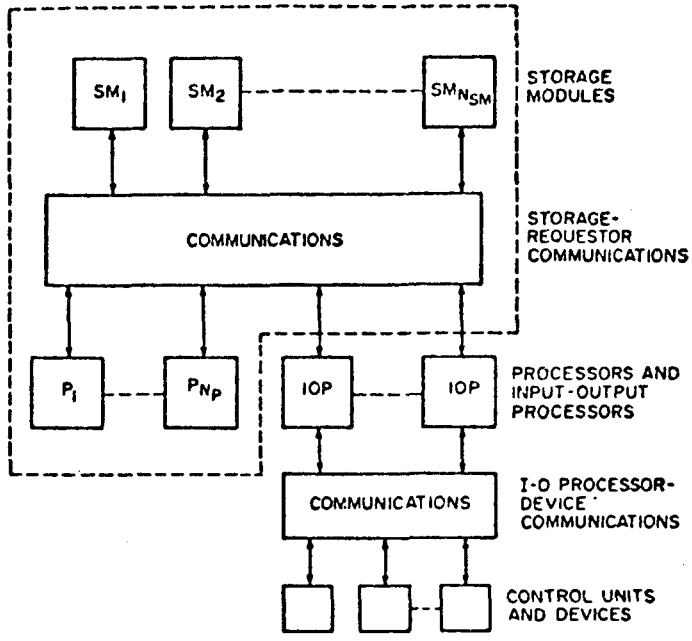


Fig. 11

Le problème sur lequel a porté l'algorithme correspond à la détermination des intensités des courants traversant un réseau de résistances électriques constitué de 26 mailles. Les équations aux noeuds conduisent à un système matriciel AxI = b où A est une matrice carrée 26x26.

Un des paramètres importants du programme est le nombre N_V de variables affectées à un processeur. Chaque processeur opère en relaxant N_V variables, soient I_j à I_{n+N_V-1}, et il est possible, selon la valeur de N_V, que deux processeurs travaillent sur la même variable. Pour éviter

les conflits d'accès à un même banc de mémoire, la simulation utilise un mécanisme d'exclusion mutuelle au niveau de l'attribution d'un indice de variable. Ceci introduit bien sûr un délai, tout processeur essayant de pénétrer dans la zone de programme verrouillée devant rester oisif.

Le principal résultat obtenu correspond à la figure 12 et indique que le temps d'exécution n'est pas une fonction décroissant régulièrement lorsque croît le nombre de processeurs. Le fait que les différentes courbes soient situées au dessus de la fonction $\frac{1}{N_p}$ traduit le fait que l'efficacité E_p est strictement inférieure à 1. L'existence de conflits d'accès aux mêmes bancs de mémoire, d'attente devant un bloc verrouillé, de redondance de calculs, de phases séquentielles en début et fin de programme, expliquent ce phénomène, ainsi que le fait que, pour un nombre donné de bancs de mémoire, il existe un nombre de processeurs qui optimise le temps de calcul.

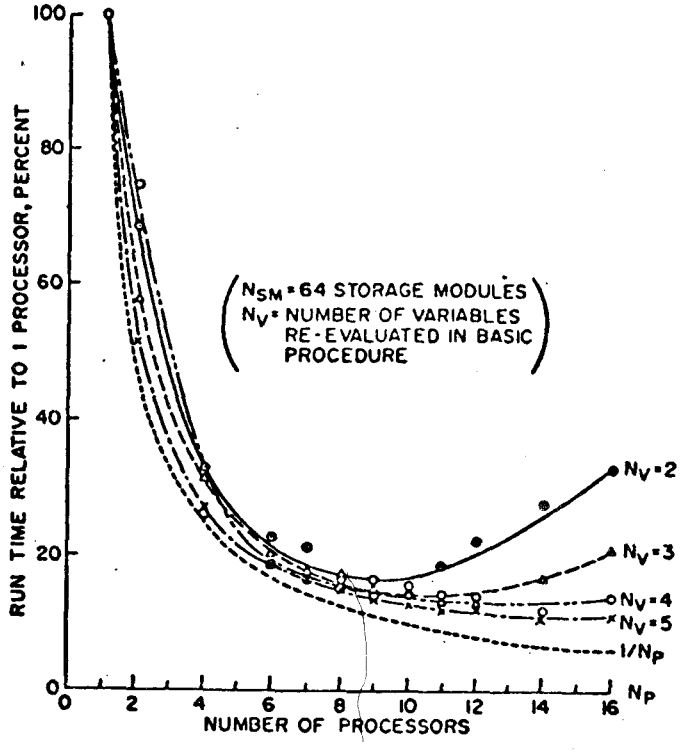


Fig. 12

L'étude du temps de calcul total (somme des temps de tous les processeurs) conduit aux mêmes remarques (figure 13).

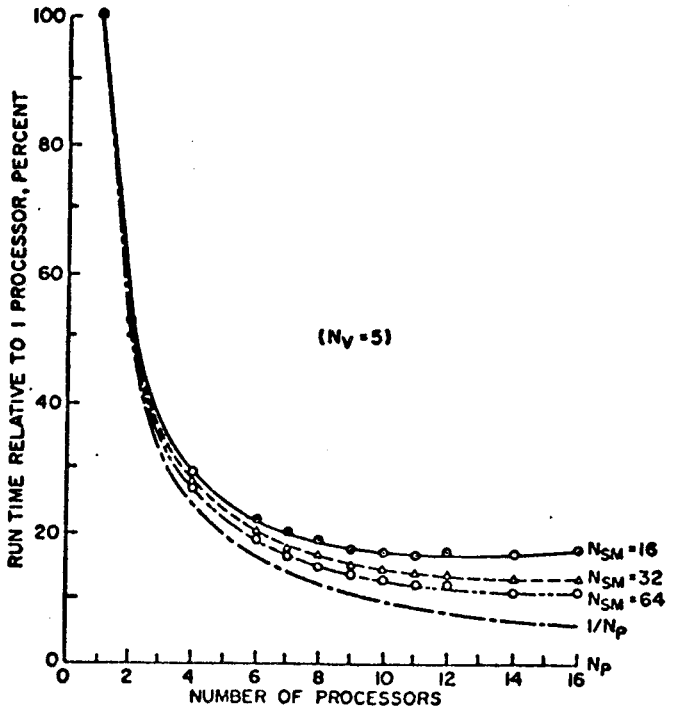


Fig. 13

Les figures 14 et 15 précisent l'influence des problèmes liés aux conflits d'accès aux bancs de mémoire. La seconde est particulièrement intéressante, car elle montre que la mesure de ces interférences est, en première approximation, une fonction linéaire de N_p, de pente relativement faible (au moins pour des valeurs raisonnables du nombre de processeurs.

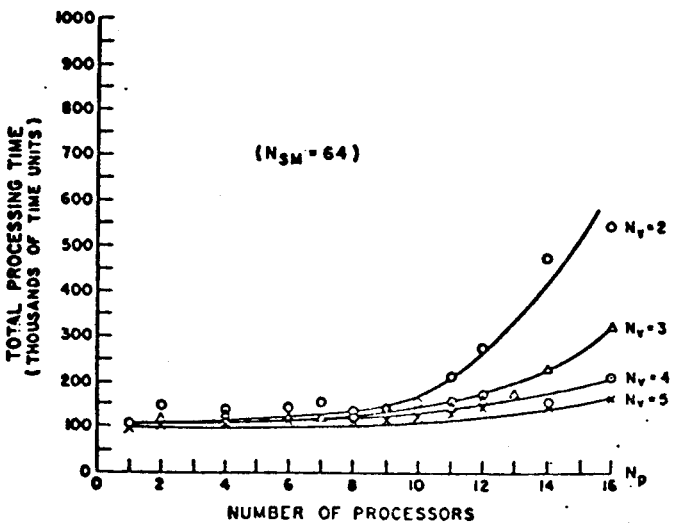


Fig. 14

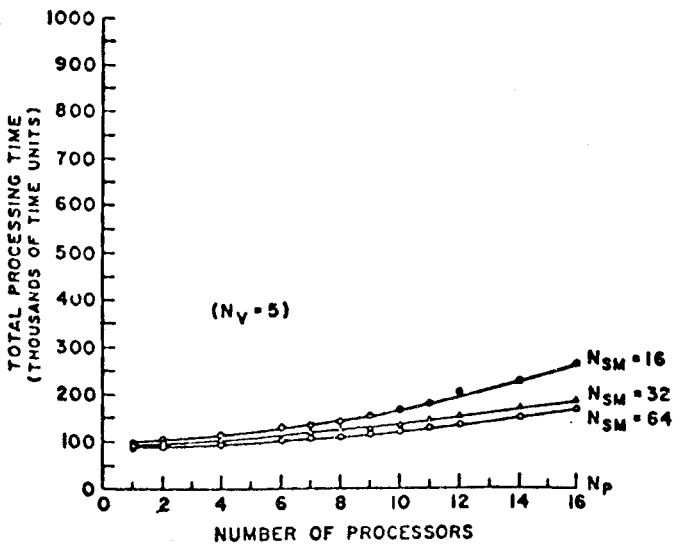


Fig. 15

Une mesure des propriétés des algorithmes chaotiques sur ce modèle de machine est donnée par une grandeur appelée facteur de qualité, et définie par :

$$Q = \frac{K}{T_{RUN} \times T_{PROC}}$$

où T_{RUN} est le temps de calcul de la machine et $T_{PROC} = \sum_{p=1}^{N_p} T_p$ est la somme des temps de calcul de chaque processeur. K est un facteur arbitraire dépendant du problème. Une première approximation de la mesure des performances est en effet donnée par $\frac{1}{T_{RUN}}$, tandis que T_{PROC} correspond à une évaluation grossière du coût.

La figure 16 met particulièrement bien en relief l'existence d'un nombre optimum de processeurs, lié bien sûr au problème, ainsi que, par l'intermédiaire du paramètre N_V, à son implémentation sur la machine.

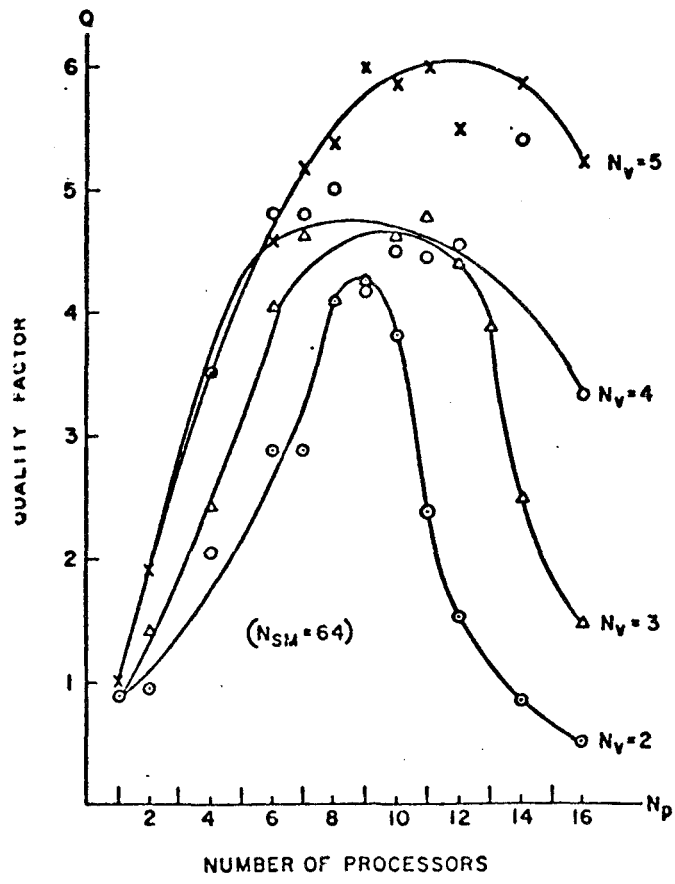


Fig. 16

Le dernier aspect important concernant ces travaux correspond à l'étude de la sur-relatation. On sait que la convergence en est garantie pour $0 < w \leq 1$.

Rosenfeld a obtenu un résultat convorme, en première apparence, aux résultats classiques de l'analyse numérique séquentielle. La figure 17 présente un minimum correspondant à une valeur de w comprise entre 1,4 et 1,5. La divergence est obtenue à partir de $w = 1,6$, pour un nombre de processeurs donné. L'ensemble des paramètres évoqués, à savoir N_p, N_v et N_{SM} influe fortement sur ces résultats.

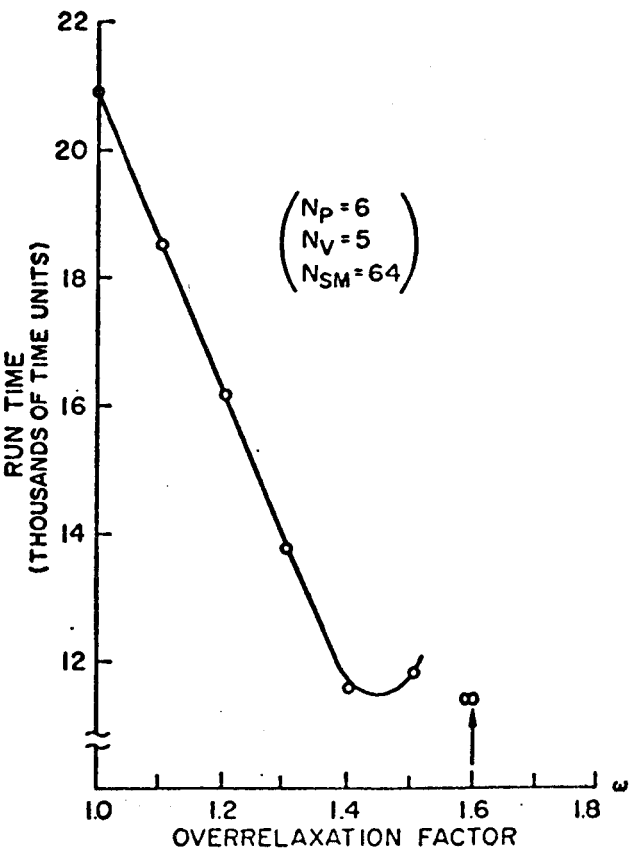


Fig. 17

4.4.5.2. Les travaux de BAUDET

G. Baudet a réalisé une série de mesures portant sur différents algorithmes, sur une machine multiprocesseur à possibilités asynchrones (Cmmp sous système Hydra) pour résoudre le problème de Dirichlet.

On trouvera dans [BEL 72] une description de Cmmp. Au moment des essais, la machine disposait de 6 processeurs PDP-11 et de 13 blocs de

mémoire de 1 M mot, reliés par un réseau d'interconnection. La matrice du laplacien, discrétisée sur un domaine de 21x24 constitué de 504 points, est une matrice bande dont le rayon spectral vaut :

$$\rho(A) = \frac{1}{2}(\cos\frac{\pi}{n+2} + \cos\frac{\pi}{m+1}) = 0.991$$

avec n = 21 et m = 24.

A partir de ce problème, Baudet a comparé différentes méthodes de résolution : Jacobi, Jacobi-asynchrone, Gauss-Seidel-asynchrone, et chaotique pur. Ce dernier algorithme a en fait été exécuté sur Cm* par Raskin [RAS 77], et les résultats normalisés de façon à correspondre à ceux obtenus avec GAUSS-SEIDEL pour un processeur, dans la mesure où ils sont alors équivalents.

Les figures 18 et 19 présentent un état de performances relatives à chacun des algorithmes. Les extrémums qui apparaissent sont dus essentiellement aux différents modes de synchronisation adoptés pour les réaliser (cf. 4,4,3,). Ils font apparaître, pour une configuration donnée de la machine et une dimension du problème, un nombre optimum de processeurs, dans tous les cas, sauf en ce qui concerne l'algorithme purement chaotique.

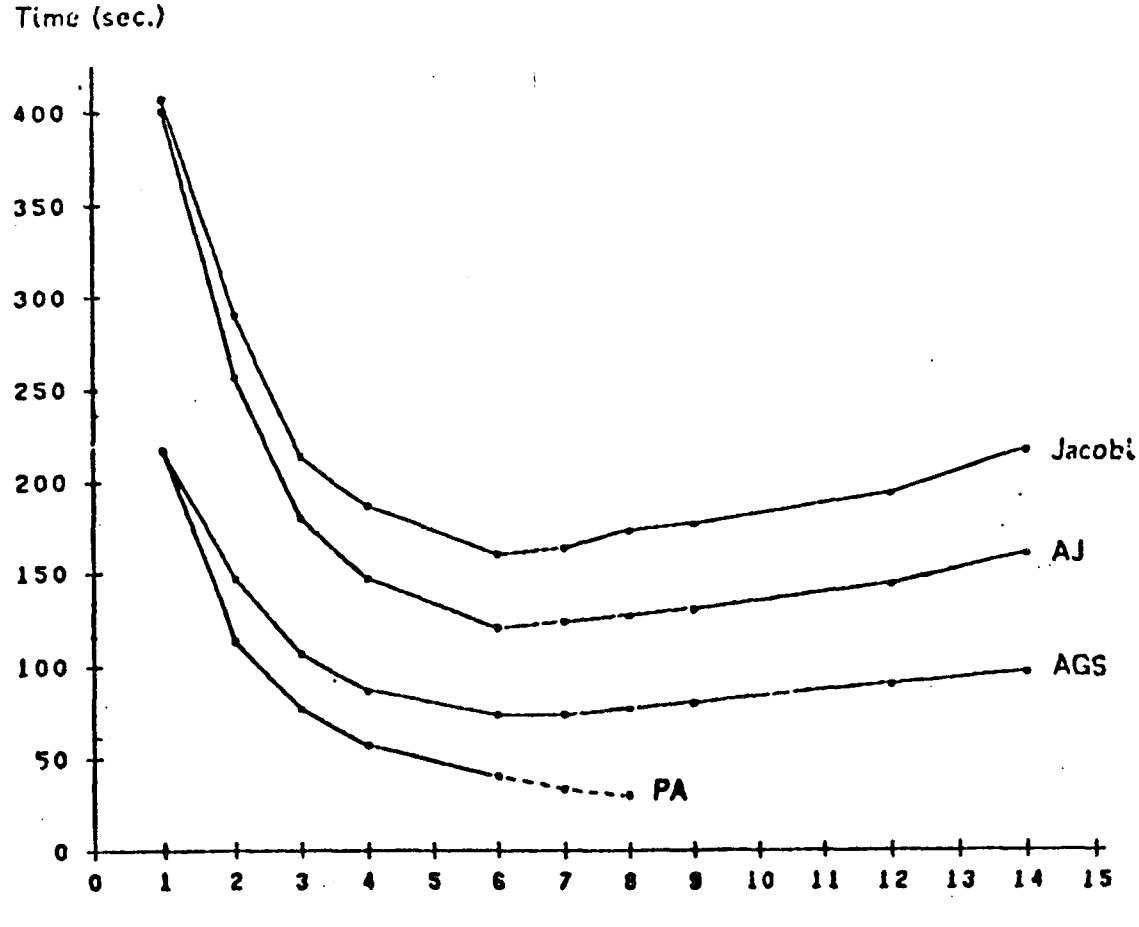


Fig. 18

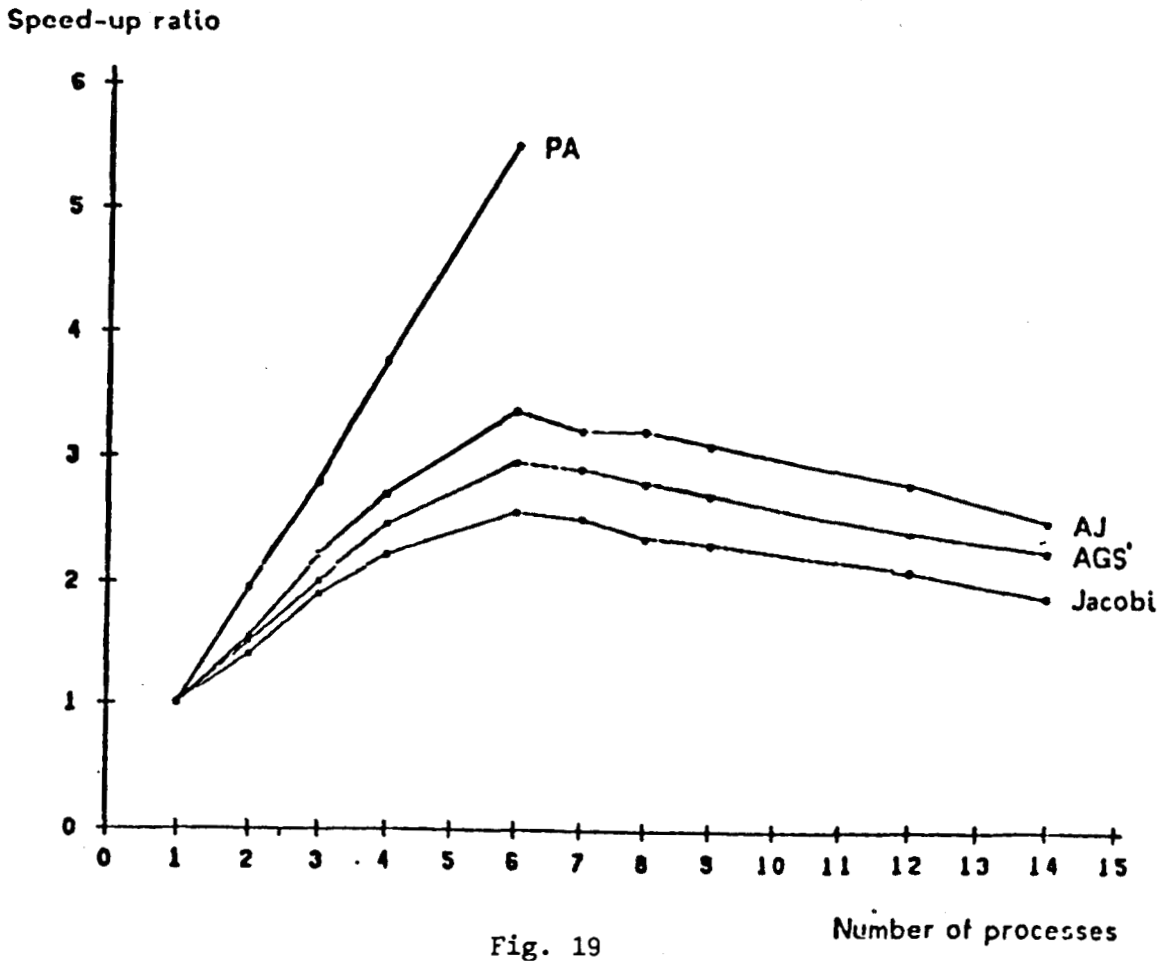


Fig. 19

Le nombre d'itérations nécessaires à la résolution du système est aussi une variable qu'il est intéressant de représenter. La figure 20 en fait apparaitre une variation linéaire.

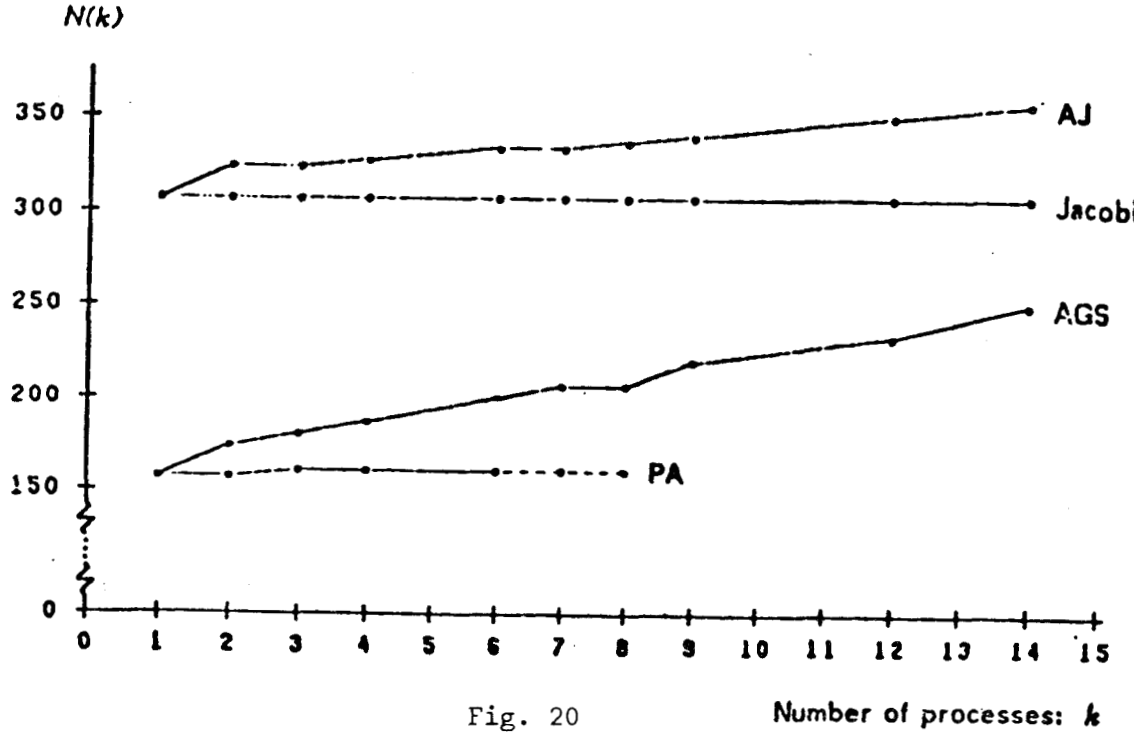


Fig. 20

4.4.5.3. Autres expérimentations

D'autres expérimentations ont été récemment menées à l'Université de Besançon par **Jullian, Perrin** et **Spiteri** [JPS 80], [JPS 81]. Elles correspondent à des simulations évaluant les performances de méthodes asynchrones par bloc, pour des problèmes de Laplacien discrétisés sur des domaines complexes.

L'ensemble de ces travaux, à la fois théorique et expérimentaux mettent en évidence le très gros avantage que l'on peut attendre des méthodes de calcul asynchrone, ainsi que leur remarquable simplicité de formulation pour le traitement parallèle.

CHAPITRE II

COMPLEXITE DES APPLICATIONS ET ARCHITECTURES

Parmi les applications les plus importantes qui réclament des besoins de calcul parallèle figurent les problèmes discrétisés évoqués au paragraphe I.3 du chapitre précédent. Leur résolution rapide constitue l'un des enjeux scientifiques et économiques de cette décennie et fait l'objet actuellement d'efforts très importants. C'est pourquoi il nous semble particulièrement intéressant d'y porter notre attention, afin d'en préciser les difficultés et d'estimer les contraintes qu'ils font peser sur les projets d'architecture.

Dans ce but, nous avons retenu le cas de la prévision météorologique, d'une part parce qu'il est familier à beaucoup et que sa compréhension ne nécessite pas de ce fait, de trop grands développements, et d'autre part parce qu'il nous semble assez représentatif de la complexité à laquelle on se heurte lorsque l'on cherche à résoudre de tels problèmes.

Puis nous évoquerons différentes architectures de machines parallèles, en fonction de leur capacité à traiter cette classe d'application par des méthodes asynchrones.

I - LE PROBLÈME DE LA PRÉVISION MÉTÉOROLOGIQUE

1.1. FORMULATION

Il existe de nombreux modèles théoriques de l'atmosphère. La plupart ont été récemment développés lorsque la puissance des machines a rendu possible leur mise en oeuvre. Ils utilisent tous, à notre connaissance la méthode des différences finies comme méthode de résolution. Ils diffèrent les uns des autres par le type, la forme et les dimensions du maillage employé, ainsi que par l'éventuelle prise en compte de phénomènes dits secondaires, mais qui peuvent jouer un rôle important, et qui sont susceptibles de compliquer de façon notable les équations de base. Le problème de la prévision météorologique est double à ce niveau :

- Une évaluation à court terme de l'état de l'atmosphère dépend essentiellement de l'état initial au début des calculs, et ne nécessite pas d'autres données que les grandeurs physiques qui définissent celui-ci.

- Une évaluation à moyen ou à longterme fait intervenir des paramètres supplémentaires. Ceux-ci sont en partie fonction de l'état instantané, mais parfois aussi, difficilement prévisibles, parce qu'ils sont mal mesurables (vents solaires, évolution de la couche d'ozone) ou extérieurs au système lui-même (pollutions).

Les phénomènes principaux qui interviennent dans un modèle sont évidemment ceux qui caractérisent un fluide en mouvement (l'air humide) soumis à des variations de température.

Les paramètres utilisés sont donc :

la vitesse du vent $\vec{V}(V_x, V_y, V_z)$, la pression p, la température T. Par commodité, on introduit aussi une variable h qui représente l'altitude d'une surface isobare particulière (cf. figure 1). L'influence du mouvement de rotation de la terre se fait sentir à travers f, accélération de Coriolis, qui dépend de la latitude d'un point.

L'équation fondamentale est celle de la dynamique des fluides, dans laquelle on ne fait pas intervenir, dans un premier temps, de force de friction. Elle s'écrit :

$$\frac{\partial \vec{V}}{\partial t} + \vec{V}.\nabla\vec{V} + \vec{V}_z \frac{\partial \vec{V}}{\partial p} + \vec{j} \times \vec{h} \wedge \vec{V} + g\,\nabla h = 0$$

Si l'on suppose que l'atmosphère est à chaque instant en équilibre hydrostatique, on introduit l'équation :

$$\frac{\partial h}{\partial p} + \frac{RT}{gp} = 0 \quad \text{où R est la constante des gaz}$$

Les échanges de chaleur intervenant entre un élément de volume d'atmosphère et le milieu environnant sont traduits par une équation d'équilibre thermodynamique :

$$Cp\ dt - RT\ \frac{dp}{p} - dQ = 0$$

où Cp est la chaleur spécifique à pression constante, et dQ la quantité de chaleur absorbée.

Enfin, la loi de conservation de masse introduit une équation de continuité :

$$\vec{\nabla V} + \frac{\partial v_z}{\partial p} = 0$$

1.2. FACTEURS SUPPLEMENTAIRES DE COMPLEXITE

Parmi les effets supplémentaires dont il faille tenir compte, on peut énoncer, sans que cette liste soit limitative :

- Des effets de friction particuliers et, éventuellement localisés : effets de topographie, déflections dues aux chaînes montagneuses existence de couloirs particuliers (vallée du Rhône) jets-streams dans la haute atmosphère,

- Des problèmes thermiques particuliers ; alternance du jour et de la nuit, variations d'ensoleillement dues aux saisons, variation de l'irradiation. A ce sujet, il faut encore distinguer les effets causés par les ondes courtes (U.V.) absorbées par le sol et l'air ou longues (infra-rouges), en partie émises par le sol, en partie absorbées par les hautes couches de l'atmosphère. A ces éléments complexes s'ajoutent les effets dus à l'existence des mers et des grands lacs, à la présence de courants chauds ou froids (certains étant saisonniers), des déserts, des forêts. L'équilibre air/vapeur d'eau fait intervenir des réseaux de courbes qu'il est nécessaire de stocker dans des fichiers de bonne taille.

- On pourrait inclure aussi les effets causés par les mouvements de convection d'échelles variables, qui vont de la simple ascendance, bien connue des pilotes de Vol à Voile, jusqu'aux puissants et très larges mouvements associés aux fronts orageux et qui ont pour siège les cumulo-nimbus.

En évoquant ce dernier point, on ne peut passer sous silence la difficulté qui apparait lorsque l'on considère les systèmes dépressionnaires, de formes et de dimensions variables. Ils introduisent en effet des discontinuités supplémentaires, de frontières variables, dans le schéma. D'un côté ou de l'autre de cette frontière, l'opérateur (non linéaire) est de type élliptique ou hyperbolique, et la résolution ne peut s'effectuer de la même façon selon que l'on considère tel ou tel type. Aussi, dans la pratique conviendra-t-il d'effectuer un test, localement, en chaque point du maillage, afin de spécifier les frontières de part et d'autre desquelles peut s'appliquer une méthode ou une autre.

1.3. PRINCIPE DU CALCUL

L'ensemble des équations présentées établit les relations entre les paramètres qui régissent globalement le problème. Elles s'appliquent à toute tranche de l'atmosphère comprise entre 2 niveaux voisins d'altitude. Il est donc commode de découper celle-ci en niveaux auxquels on fait correspondre une isobare donnée. La figure 1 montre qu'un tel découpage, sur 3 niveaux, apparait très grossier, et qu'il conviendrait d'en retenir une dizaine, si ce n'est plus, pour attendre une bonne précision des calculs.

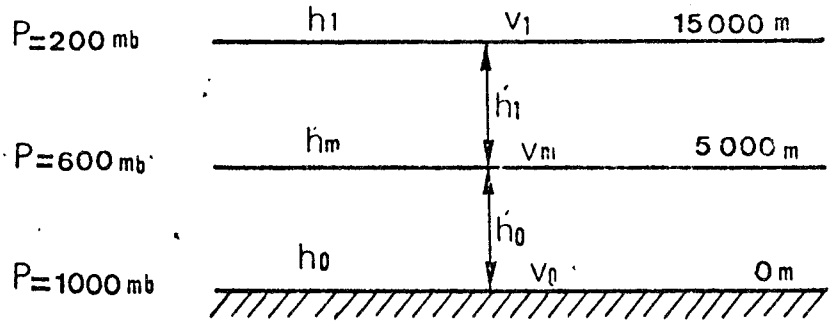


Fig. 1

Le temps est évidemment un paramètre fondamental. Le traitement consiste à évaluer, par itérations successives, un état d'équilibre de l'atmosphère, pour des pas temporels donnés Δt.

Partant d'un état initial au temps 0, on évalue les valeurs des $\frac{\partial h}{\partial t}$ au temps $t + \Delta t$. On détermine ainsi les nouvelles hauteurs, en chaque noeud du maillage, des équipotentielles. On en déduit les composantes de $\vec{V}$ et la valeur de T. On répète ensuite ce cycle de calcul, autant de fois qu'il le faut pour réaliser la précision demandée.

1.4. DISCRETION DU PROBLEME

1.4.1. *Le maillage*

On vient de voir que ce dernier est nécessairement tri-dimensionnel et qu'il peut comporter 3 niveaux et plus. Chaque grille doit assurer une bonne représentation de la surface terrestre à représenter.

A titre d'exemple, un maillage couvrant la totalité du globe, et correspondant à un découpage en méridiens et parallèles, de 1° en 1°, réclamerait 360x360 points, soit environ 130.000 noeuds. A raison de 3 grilles superposées, le problème atteindrait une taille correspondant aux 390.000 noeuds à prendre en compte, ce qui est à coup sur considérable. Or à cette échelle, la France ne serait représentée que par une centaine de points (figure 2).

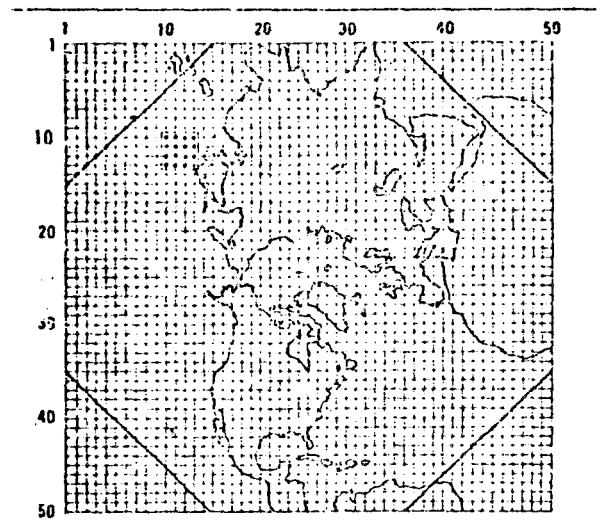


Fig. 2 Horizontal structure of the pronosis model

En réalité, du fait de la barrière représentée par l'équateur un modèle hémisphérique apparait comme suffisant. Le modèle le plus avancé prévoit un découpage en 120 latitudes avec une résolution maximale à l'équateur de 256 points, soit une grille de 20.000 Points environ. Le nombre de grilles envisagées est de l'ordre de 6 à 9, soit en tout un ensemble de 120 à 180.000 points avec, rappelons-le, 5 variables (V_x, V_y, V_z, h et T) par point, et un nombre beaucoup plus grand de paramètres associés, stockés dans des fichiers.

Une pratique courante consiste, partant de l'expression du problème dans un système de coordonnées sphériques, à réaliser une série de transformations géométriques de façon à travailler sur un maillage plan.

1.4.2. *Discrétisation des équations*

Le problème traité est du type instationnaire, ce qui signifie que le temps est le principal paramètre du calcul et que l'évolution du processus doit être considérée comme étant représentée à chaque instant donné, par un état d'équilibre dépendant de l'état précédent. Le calcul peut s'effectuer à l'aide d'un schéma de Mc Cormack, utilisant une méthode à pas fractionnaires, et qui revient à effectuer un maillage virtuel de l'espace selon un axe temporel. Le pas de calcul Δt doit être assez petit pour assurer une bonne précision. Il est couramment de l'ordre de 2 à 5 mn, et est lui-même décomposé en pas $\frac{\Delta t}{2}$ (figure 3).

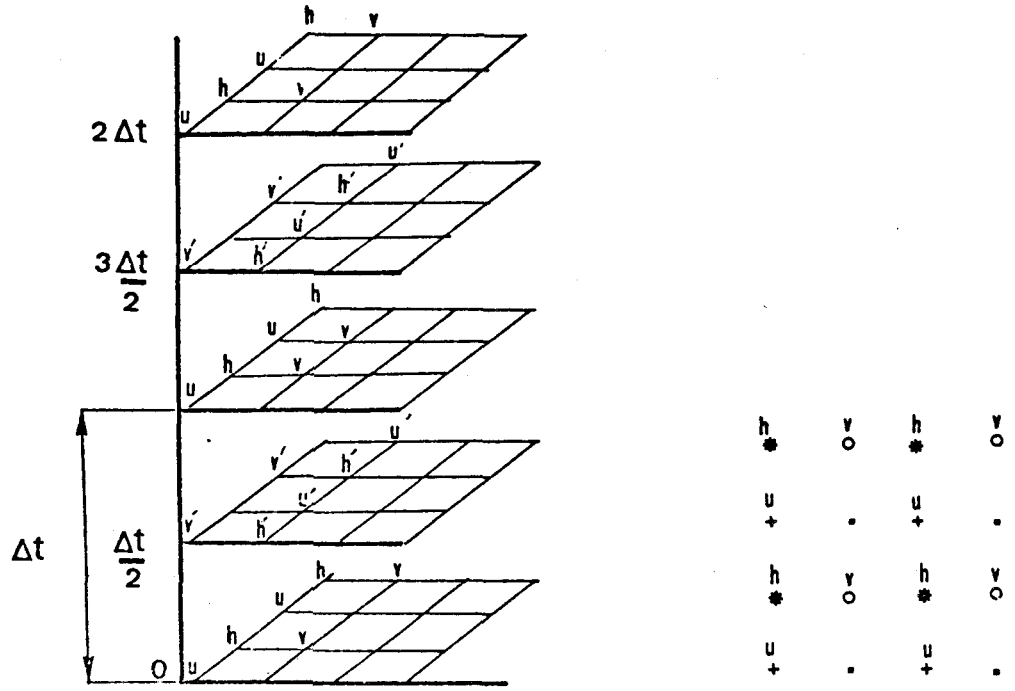


Fig. 3 Répartition des variables sur les grilles.

Pour chaque pas, et pour une grille donnée, les valeurs de h et de la fonction de courant Ψ de l'écoulement sont calculées sur des noeuds disposés en quinconce, à la fois spatialement et temporellement, selon le schéma de la figure 3.

Chaque évaluation réclame un bon nombre de calculs élémentaires, qui peuvent comprendre des interpolations de valeurs à rechercher dans des fichiers divers.

Pour fixer les idées, il convient de préciser que, lorsqu'on veut faire une prévision d'une portée de 14 jours, avec un pas de calcul de 2 mn (ce qui réclame 10.000 pas), on désire que la durée du calcul ne dépasse pas 4 ou 5 heures. La puissance réclamée dans ce cas est au minimum de quelques centaines de milliards d'opérations en virgule flottante par seconde.

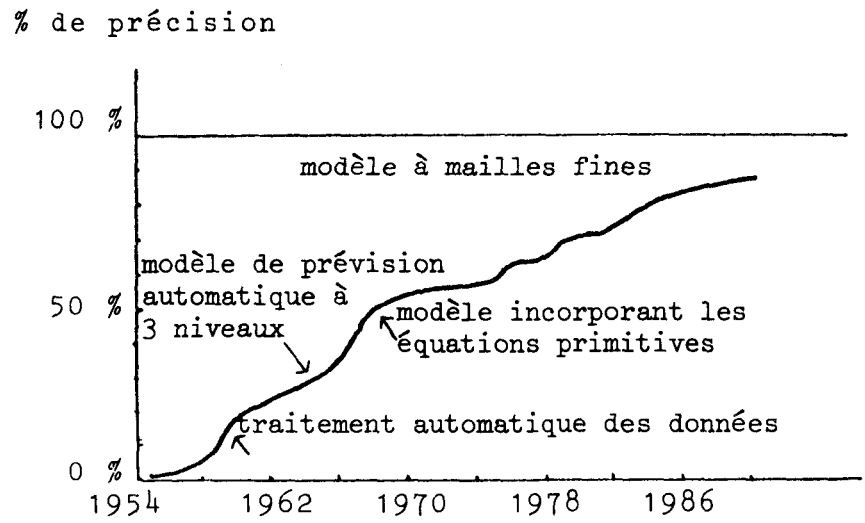


Fig. 4

II - AUTRES PROBLÈMES

2.1. AERODYNAMIQUE

Les contraintes énergétiques et économiques actuelles, qui tendent à amplifier le coût des mesures directes (souffleries, essais en vol),

induisent des recherches de simulation tendant à aboutir à la modélisation complète d'un avion entier. Le problème revient à résoudre les équations de Navier-stockes dans leur formulation la plus générale, en faisant intervenir dans le modèle, les variations de densité et de température de l'air (fonctions de l'altitude et de la vitesse). Le maillage tridimensionnel minimum réclamé pour obtenir une précision acceptable (c'est à dire susceptible de permettre de se passer d'une soufflerie supersonique de grande capacité) est de l'ordre du million de points. Le volume des données annexes est lui aussi énorme et peut atteindre plusieurs millions de mots. En attendant de pouvoir disposer de machines capables de réaliser en mode interactif, une telle simulation, la Nasa a développé d'autres modèles tridimensionnels d'environ 150.000 points, destinés à simuler des effets aérodynamiques particuliers. L'un d'eux ne comporte pas moins de 23 variables par noeud. Un point important à noter, c'est qu'outre la taille du domaine et la complexité des équations, le calcul exige une utilisation continuelle d'énormes fichiers de données. Le nombre d'itérations réclamées pour obtenir la convergence des équations est de l'ordre de 1500 à 3000, ce qui réclame l'équivalent de 25 heures du temps de calcul d'ILLIAC IV.

Etant donné le très grand nombre de calculs différents à effectuer, pour une même géométrie, et les délais de conception réduits qui sont malgré tout imposés, la simulation n'est actuellement compétitive avec les essais en soufflerie qu'au niveau de modèles élémentaires (portions d'ailes).

2.2. RESISTANCE DES STRUCTURES

Là encore, les phénomènes sont décrits par des équations aux dérivées partielles qui ont cependant le mérite d'être assez souvent linéaires. La complexité provient de la taille du problème, Elle se traduit notamment par le fait qu'il faut plusieurs mois pour préparer "manuellement", le modèle discrétisé de la fusée Ariane, ou d'une centrale nucléaire par exemple. La forme même des structures introduit des difficultés ; certaines sont complexes, comportant des vides, ou des angles aigus, , sièges de phénomènes à forts gradiants, nécessitant des maillages localement larges ou serrés, qui influent sur la précision. Quoi qu'il en soit, on retrouve les mêmes caractéristiques que dans l'exemple précédent ; discrétisation tridimensionnelle de 10^5 à 3.10^5 noeuds, avec plusieurs variables par noeud et utilisation de gros fichiers de données,

III - LE POINT SUR LES ARCHITECTURES EXISTANTES

3.1 INTRODUCTION

On mesure, sur la base des exemples qui précèdent, toutes les difficultés qui se posent lorsque l'on envisage de traiter de telles applications. Dans ce cas, la non-linéarité de très nombreux problèmes fait que, bien souvent, des méthodes itératives s'imposent par leur simplicité de formulation et de mise en oeuvre. Mais alors, on est confronté à de sévères contraintes de temps et d'occupation de l'espace mémoire. Pour ces raisons, on est conduit à rechercher des machines de grande puissance de calcul, disposant d'un volume de mémoire élevé. Deux grandes classes de calculateurs répondent à ces critères : les machines pipe-line et les multiprocesseurs.

3.2 ARCHITECTURE PIPE-LINE

Le concept de pipe line n'a rien à voir avec celui du calcul parallèle. Cette technique consiste à accélérer le déroulement séquentiel de l'exécution d'une instruction par un chevauchement de l'enchaînement des cycles. Les performances auxquelles elle conduit la rendent très attrayante à première vue ; aussi n'est-ce pas un hasard si, parmi les principales machines de grande puissance et de grosse capacité récemment construites pour les besoins du calcul scientifique, on trouve une notable proportion de machines pipe-line (CRAY 1 - STAR 100).

Il faut cependant remarquer qu'on ne peut mettre en oeuvre des algorithmes asynchrones que sur des machines disposant de plusieurs unités de pipe-line. L'ASC (Advanced Scientific Computer) est un exemple de machine à la fois pipe-line et multiprocesseur. Il est bien évident que la conception et la mise au point de tels calculateurs posent des problèmes encore plus ardus que ceux auxquels on se heurte dans le cas d'un seul pipe. Leur technologie est en effet complexe, et suppose une bonne maîtrise dans des domaines variés : l'emploi de très hautes fréquences d'horloge nécessite l'utilisation de guides d'onde ; il implique aussi la résolution de problèmes de dissipation de l'énergie (cf. circuits de refroidissement de CRAY 1) et impose l'emploi de mémoires de très grand débit, chargées d'alimenter les divers étages de chaque unité.

3.3 ARCHITECTURES MULTIPROCESSEURS

3.3.1. Définition

Nous définissons un système multiprocesseur comme étant un ensemble d'unités fonctionnelles capables d'exécuter simultanément des traitements différents. Elles constituent des Modules de Calcul MC (appelés parfois PE pour "Processing Element") réunissant un processeur, c'est à dire une unité arithmétique et logique et une unité de contrôle, un peu de mémoire locale, et des dispositifs d'entrée/sortie, qui en font un ensemble capable, éventuellement d'opérer seul, hors de tout environnement pendant un bref laps de temps.

L'organisation de la machine lui confert les propriétés générales suivantes :

- elle possède 2 MC au moins, mais souvent beaucoup plus, généralement identiques, ou de caractéristiques semblables.
- chaque MC peut avoir accès à 2 niveaux de mémoires ; une mémoire locale, contrôlée localement, et une mémoire commune. Il peut accéder aussi à différents dispositifs extérieurs, relisés à des organes d'entrée/sortie par l'intermédiaire d'un système de communication.
- l'ensemble est contrôlé par un système unique (mais qui peut être réparti), permettant de lancer une application, de coordonner les tâches qui en ont besoin, de règler les conflits et de récupérer les résultats.

Les différents modes d'organisation qu'il est possible d'imaginer se différencient les uns des autres par le principe des relations susceptibles de s'établir entre les processeurs. Leur diversité provient des problèmes importants qui sont posés par les communications, dès lorsque le nombre de ces derniers devient assez grand. Des conflits, des engorgements se produisent, que l'on peut dans un premier temps chercher à règler par des solutions architecturales d'ordre général.

Un certain nombre d'études ont été réalisées qui proposent différents modes de classement des types d'architectures qu'il est possible d'envisager. Certaines mettent l'accent avant tout sur l'aspect matériel de la réalisation [ENS 77], [FUL 78], d'autres, au contraire, développent

des modèles logiques et leurs implications [SUL 77], [SPE 77], [BEL 71].

Parmi les schémas qu'il est possible de dégager, nous examinerons
- les structures en anneau
- les structures bus/multibus
- les structures de type réseau.

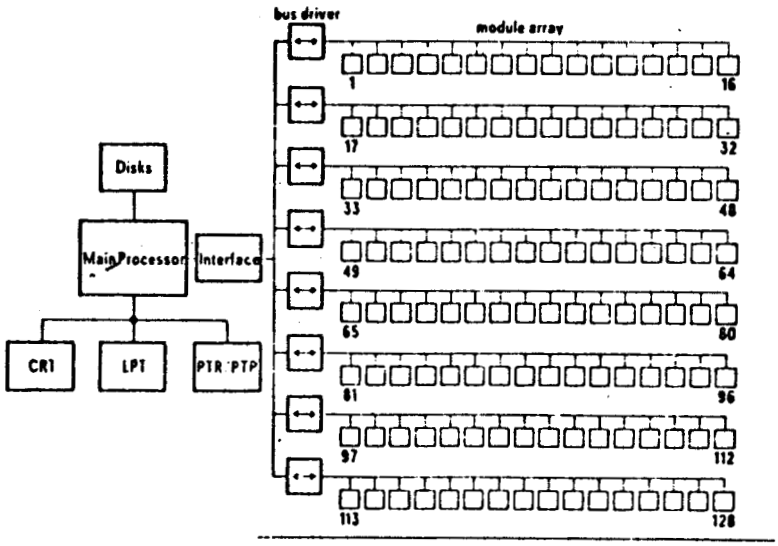
3.3.2. *Processeurs en anneau*

Les processeurs sont ordonnés le long d'un outil de communication unidirectionnel auquel ils sont reliés par un interface approprié. Des informations circulent le long de la voie et peuvent être partagées entre tous les processeurs, ou adressées à l'un d'entre eux plus particulièrement. Certaines d'entre-elles peuvent, cycliquement, effectuer plusieurs tours avant d'être utilisées. Des interfaces gèrent les conflits résultants d'une demande d'insertion d'un message dans la voie et sont également chargés de reconnaître l'adresse de destination d'un message, d'extraire ce dernier et de le transmettre au MC concerné. Un exemple d'une telle réalisation est donné par MAUD [COR 79], où une mémoire circulante joue à la fois le rôle de mémoire de stockage et d'outil de communication. Le schéma de communication monodirectionnel, le cadencement imposé par l'anneau font qu'il nous semble difficile d'envisager l'emploi d'une telle machine pour y implémenter des algorithmes asynchrones, du moins dans l'optique avec laquelle nous en concevons l'utilisation.

3.3.3. *Structures bus/multibus*

Une machine conçue autour d'un bus est naturellement souple et bon marché, mais est rapidement saturée lorsque croît le nombre de processeurs. C'est pourquoi envisage-t-on plutôt des structures hiérarchisées, qui répondent à la méthodologie traditionnelle d'interconnection de plusieurs machines.

* SMS 201-203 [KOH 79] (figure 5) est l'exemple d'un tel calculateur, constitué de 128 MC reliés par groupes de 16 à un bus. L'ensemble de ceux-ci est connecté à son tour au bus d'un processeur principal par l'intermédiaire d'interfaces appropriés. Les processeurs (des 8080 dotés d'une unité de calcul en arithmétique flottante) n'ont accès qu'à leur mémoire locale et à leur mémoire de communication par l'intermédiaire de laquelle ils peuvent dialoguer, les uns avec les autres sous le contrôle du processeur principal.



Block diagram of SMS 201.

Fig. 5

* Les possibilités d'adressage direct sont cependant plus élevées dans une telle architecture que dans le cas de structures plus fortement hiérarchisées, comme EGPA par exemple [HAND 75]. Cette machine récemment mise au point à l'université d'Erlangen présente l'aspect d'une pyramide constituée de trois types de processeurs (figure 6).

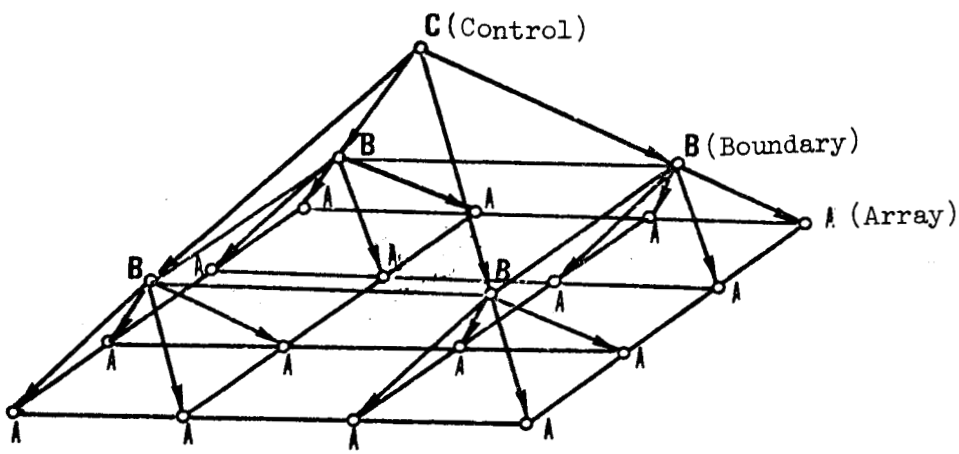


Fig. 6 Architecture d'EGPA

Les A-processeurs (A pour array) forment une matrice carrée aux extrémités connectées entre-elles de façon à générer un tore. Ils sont composés chacun d'un microprocesseur microprogrammable, doté de mémoire

locale multiaccès qui assure l'interconnection avec les 4 modules voisins. Groupés 4 par 4, ils constituent un module de base relié à un B-processeur (B pour boundary) chargé de contrôler des périphériques et d'assurer le chargement des données et l'extraction des résultats. Quatre B-processeurs sont à leur tour connectés à un C-processeur (C pour control), qui joue le rôle de maître et qui assure les fonctions de superviseur.

Les communications entre processeurs voisins sont soumises au seul contrôle de ces derniers. Les communications à plus grande distance, ou entre des processeurs de niveaux différents se font sous le contrôle des B et C processeurs. Il est ainsi possible de modifier la topologie primitive de la machine en créant des modes privilégiés de communication. De ce fait, plusieurs modes de fonctionnement sont prévus et permettent de réaliser des partitions en éléments capables d'opérer séparément des tâches parallèles. La possibilité, pour traiter des algorithmes asynchrones, d'avoir l'ensemble des processeurs travaillant selon un de ces modes est cependant liée à la capacité du C-processeur de distribuer les codes et certains opérandes à un débit suffisant aux processeurs qui en font la demande.

* CM* [FUL 78], [SWA 77] introduit plus de souplesse dans le schéma hiérarchisé. Ceci est du au fait qu'elle a été conçue pour mener des investigations autour de différents modèles d'architecture, et non dans le but de réaliser la meilleure de toute. On lui doit le concept de "cluster" ensemble de 3 à 4 modules de calcul, reliés chacun par leur commutateur slocal à un Map-bus (fig. 7).

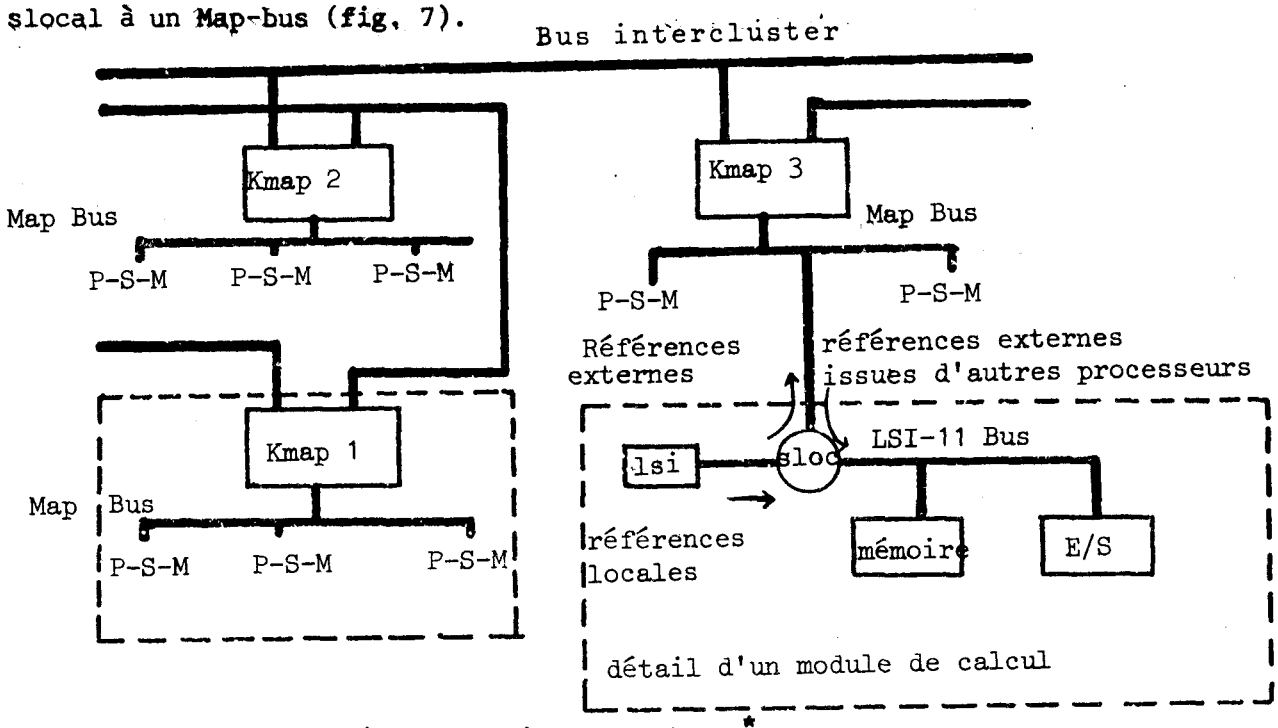


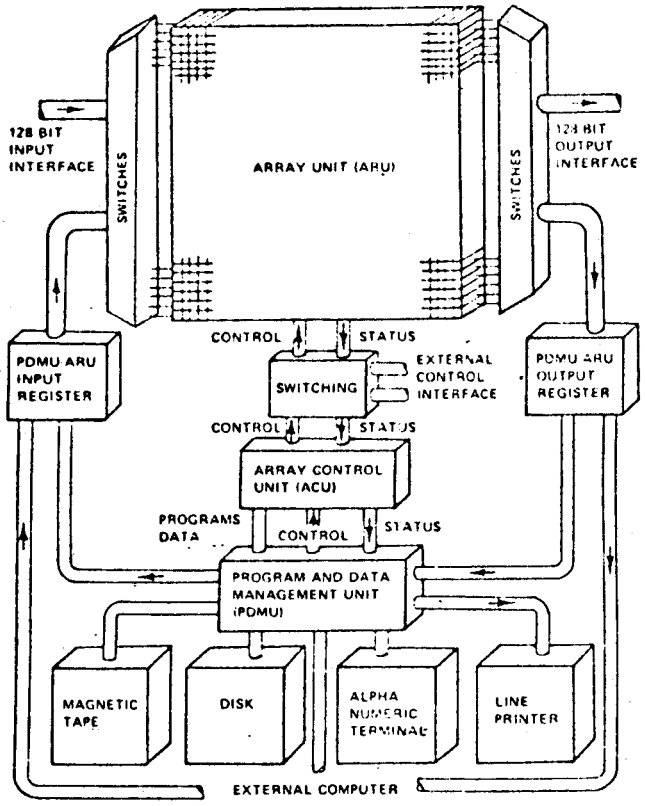
Fig. 7 Architecture de Cm*

Aucune de ces 3 machines ne possède, à l'exception des mémoires de masse, de mémoires indépendantes de celles de ses modules de calcul. La fonction de stockage y est donc entièrement répartie, ce qui implique que les données doivent être autant que possible locales aux processeurs qui les utilisent.

3.3.4. *Structures en réseau*

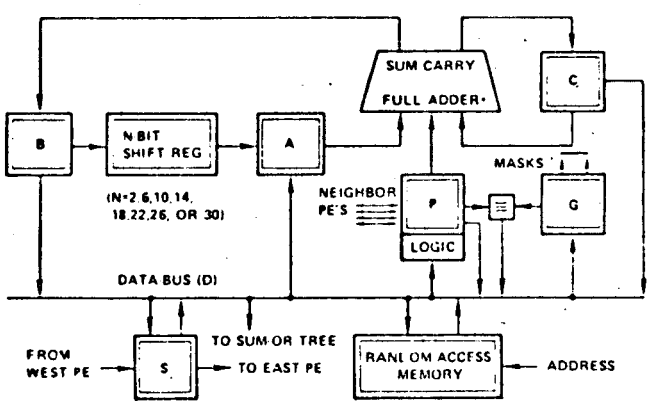
C'est ILLIAC IV [BAR 68], [THU 76] qui, après la machine de Solomon, a fait apparaître le concept de cellularité en matière d'architecture. Ce concept nous semble d'autant plus intéressant qu'il réalise une parfaite analogie entre le schéma structurel et la topologie du domaine de calcul utilisé pour les problèmes discrétisés. La mise au point de cette machine a cependant permis de mettre à jour de très graves défauts, qui ont abouti à l'arrêt de son développement (le quart seulement environ, de la machine prévue initialement a été construit). Le couplage extrêment fort réalisé au niveau de l'interconnection des MC, limitant de ce fait le schéma des communications, se traduit par un manque quasi-total de reconfigurabilité ; ILLIAC IV ne peut traiter à pleine puissance qu'un nombre limité de problèmes.

Tel quel, ce modèle de réseau matriciel ne peut être envisagé que pour des applications très spécialisées ; c'est en particulier le cas pour MPP [BAT 80] (a Massively Parallel Processor), construit par Goodyear Aerospace Corporation dans le but d'analyser les images retransmises par satellite pour le compte de la NASA. Cette machine ne comporte pas moins de 16.384 modules de calcul, connectés de façon à former une matrice de 128x128, à laquelle il faut ajouter 128x4 processeurs supplémentaires destinés à remplacer les unités défaillantes (figure 8). Les processeurs très élémentaires sont destinés à traiter les informations bit par bit, en série, ce qui leur permet de traiter les opérandes de longueur très variable correspondant à ce genre d'application (figure 9).



Block Diagram of Massively Parallel Processor (MPP)

Fig. 8



One Processing Element

Fig. 9

On a surtout développé des multiprocesseurs dont les sous-systèmes de communication soient constitués de réseaux d'interconnection, autorisant un couplage plus lâche, et dont CMMP [BEL 72] est un exemple type. Cette machine a permi de réaliser des progrés notables dans la conception de ces réseaux, complexes et coûteux, ce qui a autorisé le développement de très grosses machines matricielles :

* Le projet S1 [KOZ 80] de Laurence Livermore Laboratory, est celui d'une machine de 16 processeurs interconnectés par un "crossbar" à des bancs de mémoire de capacité non limitée, s'étageant de 128 Méga à 16 Giga octets. La puissance de calcul attendue doit être équivalente à celle de 10 Cray 1.

* La Nasa, en association avec Goodyear, étudie un projet de machine baptisée NASF (Numerical Aerodynamic Simulation Facility) prévue pour 1985, et destinée avant tout à résoudre les équations de Navier-Stockes. Sa puissance prévue est de l'ordre de 6 GFlops.

* Le projet Phoenix [FEI 78] abandonné aujourd'hui en raison des difficultés rencontrées, consistait en une machine matricielle reconfigurable dotée, pour son système de connection, d'un réseau de Benès modifié. Il prévoyait 1024 processeurs spécialement conçus pour effectuer rapidement du traitement vectoriel en virgule flottante. Plusieurs niveaux de mémoire étaient prévus :

- un niveau local de 4 M mots (soit une capacité totale de 4 M mots)
- un niveau étendu de 4 M mots.

La puissance de calcul prévue d'environ 10 G Flop, devait correspondre à celle de 16 ILLIAC IV.

3.3.5. *Conclusion*

Afin de mieux saisir l'évolution de la puissance des calculateurs parallèles qui sont apparus ces dernières années, et de pouvoir les situer les uns par rapport aux autres en fonction de leurs performances, il semble commode d'utiliser le graphe de la figure 10. Les performances moyennes de chaque machine, mesurées en MFLOPS (million d'opérations en virgule flottante par seconde) y sont reportées en fonction de leur année de mise en service (réelle ou projetée).

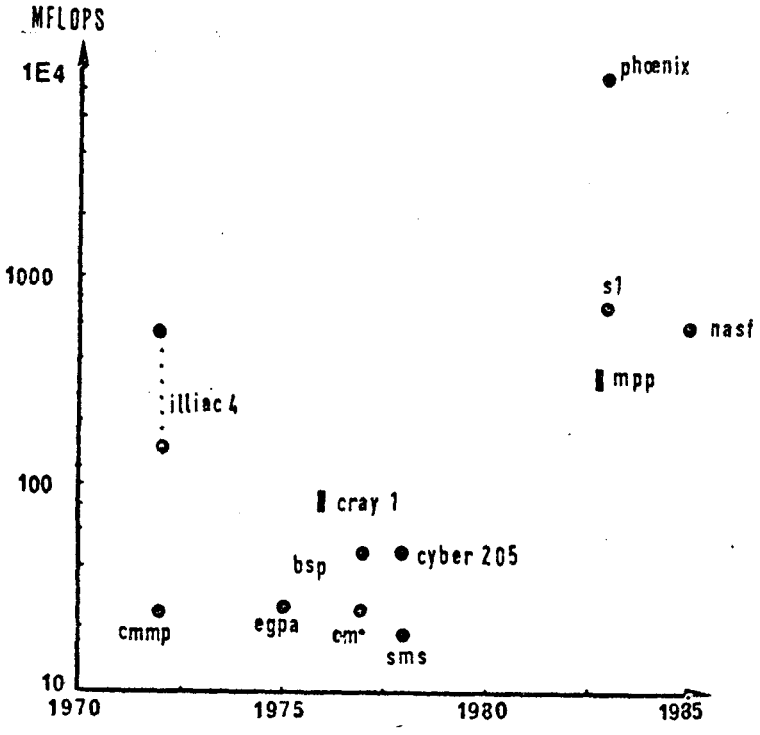


Fig. 10

Si l'on fait exception d'Illiac et de Phoenix, on constate qu'elles sont réparties selon deux grandes directions : une, qui semble horizontale, se situe aux environs des 20 MFlops, et une autre qui exprime une croissance régulière.

CHAPITRE III

ETUDE ET ANALYSE D'UN PROGRAMME TYPE

I - STRUCTURE GÉNÉRALE D'UN PROGRAMME TYPE

1.1. OBJET

Il existe, on l'a vu, un nombre considérable de problèmes dont la résolution peut être menée par une suite d'itérations asynchrones en calcul parallèle. Ils diffèrent les uns des autres par la taille et la complexité, et ce, dans un éventail très large. On peut cependant considérer que tous peuvent se ramener, principalement, au calcul de la répartition d'une grandeur physique par une, ou un système d'équations aux dérivées partielles. Chaque application comportant un certain nombre de spécifités, il est impossible d'en choisir une en espérant en dégager des caractères communs à toutes les autres. Aussi, plutôt que de s'attacher à analyser un problème particulier, essaierons-nous d'étudier, en toute généralité, un type assez courant de problèmes de cette nature. Nous ferons, bien entendu, l'hypothèse que le programme, dont l'examen permettra de préciser progressivement certaines caractéristiques du système, est destiné à être exécuté sur un ensemble de processeurs travaillant simultanément. L'exemple retenu consiste en un calcul de la répartition d'une variable sur un domaine, suivi de calculs de variables annexes dépendant de la première, et qui peut être éventuellement repris en boucle pour de nouvelles valeurs de certains paramètres.

1.2. DECOUPAGE EN ELEMENTS FONCTIONNELS

Soit donc à calculer, dans un domaine plan Ω, limité par une frontière fermée Γ, la répartition de la pression d'un fluide sur un dispositif placé dans une position (calage) donné. De ce premier calcul, on peut déduire, par exemple, la valeur de la force résultante exercée par le fluide sur le dispositif (par $F = \iint_\Omega P dS$), ainsi qu'un débit de fluide , puis de reprendre l'ensemble du traitement pour d'autres valeurs des paramètres de calage. Le programme se découpe alors en un certain nombre de phases :

a) Distribution, pouvant se faire en parallèle, de paramètres globaux, destinés à tous les processeurs, par l'un d'eux ayant un statut de maître. Ces paramètres peuvent être des constantes mathématiques (π, e), des grandeurs physiques ($g = 9.81\ m/s^2$), dont tous les modules de calcul

ont besoin, et qu'il serait coûteux et maladroit, vu leur faible nombre, de vouloir mettre en partage.

b) Distribution, nécessairement séquentielle, des paramètres locaux, du maître aux processeurs esclaves. Ce sont par exemple, les paramètres relatifs aux sous-domaines de calcul Ω_k associés à chaque processeur P_k(= nombre de points du maillage, dimension, etc ...)

c) Calcul des coefficients, intervenant dans les matrices des opérateurs traduisant les équations de définition du problème, ainsi que dans leur second membre.

d) Génération des matrices et vecteurs à partir des valeurs précédentes. Ces deux dernières phases sont autonomes, et chaque module de calcul (MC)les accomplit en toute indépendance, à partir des données dont il dispose.

e) Phase itérative de résolution, utilisant un algorithme asynchrone,

suivit de :

f) Test local de convergence. Selon la valeur de ce test, l'itération est poursuivie ou non. Dans ce dernier cas :

g) Calcul des valeurs locales, pour chaque processeur, des expressions :

$$F_k = \iint_{\Omega k} P dS$$

h) Collecte, séquentielle, des résultats ainsi trouvés, au fur et à mesure qu'ils le sont, par un processeur maître.

i) Sommation par ce dernier, et génération de la variable cherchée, F.

j) Dans de nombreux cas, le résultat trouvé sert de paramètre de fermeture, dans la mesure où on l'utilise pour vérifier la validité du calcul itératif. Si donc, à partir de la valeur de F, on retrouve les valeurs des paramètres de calage du problème, on passe à la phase suivante, sinon, le calcul est repris au niveau de la phse c)

k) La répartition trouvée étant compatible avec les données du problème, on génère les paramètres de calage correspondant à cet état, considéré comme stable à un instant donné.

l) Calcul des valeurs des débits locaux par D_k

m) Collecte séquentielle de ces résultats par un processeur maître.

n) Sommation des résultats partiels et génération du débit D, qui peut être utilisé dans le calcul d'un certain nombre d'autres variables par le maître.

o) Désire-t-on reprendre ces calculs pour de nouvelles valeurs de paramètres, globaux ou locaux ? Dans ce cas, on reprend le traitement au niveau des phases a) ou b). Dans le cas contraire, c'est la fin du traitement.

1.3. EXISTENCE D'UN CYCLE

Sur la base de cet exemple simple, on observe l'existence d'un cycle de calcul, que l'on peut retrouver tout au long de la plupart des programmes, et qui se décompose de la façon suivante :

- distribution en parallèle par un processeur maître, de valeurs communes à tous les processeurs de traitement esclaves
- phase de calcul parallèle, autonome, asynchrone
- collecte séquentielle, mais pas nécessairement ordonnée dans le temps, des résultats, par le maître,
- assemblage de ceux-ci et utilisation par le maître.

C'est sur la base de ce cycle qu'il convient nous semble-t-il, d'analyser les caractéristiques les plus générales de ces applications.

1.4. INFLUENCE DU MODE DE DECOUPAGE EN SOUS DOMAINES

La façon de réaliser le découpage des sous domaines de calcul et leur affectation aux processeurs, joue un rôle important sur l'organisation des programmes de traitement et les échanges de variables entre les processeurs. On peut ainsi proposer ;

1.4.1. Des sous domaines de formes non convexes, définis par leur borne. I étant l'indice d'une ligne et J celui d'une colonne d'un maillage quadrilatère, on a :

$$\forall\ j \in [J\ \text{mini},\ J\ \text{maxi}],\ I \in v_k(J)$$

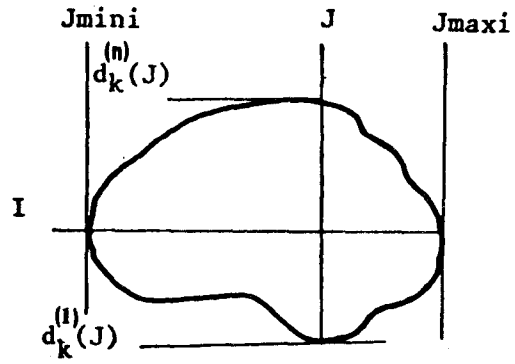


Fig 1

On observe la nécessité de stocker tous les vecteurs v_k au niveau de chaque processeur. Le coût correspondant, joint au fait que le nombre de processeurs voisins, avec qui des dialogues sont nécessaires, nous fait rejeter ce genre de découpage.

1.4.2. *Des sous-domaines réguliers*

α) découpés par bandes verticales.

Si toutes les bandes sont de dimensions égales, le numéro d'un point détermine sans ambiguité son implantation au niveau d'un processeur, par un calcul de modulo ; De plus, dans le cas d'un problème plan, chaque processeur n'a que 2 interlocuteurs au maximum.

β) découpés par pavés.

C'est le découpage que nous retiendrons ici, car c'est le plus souple. Il existe au moins deux façons de procéder en ce qui concerne l'adressage des variables aux noeuds.

*i) On peut faire intervenir, pour un point donné, la direction des points voisins de façon explicite. L'adresse d'un noeud est alors une adresse physique, en ce sens que l'on précise à la fois le numéro de la variable en ce noeud, et celui du processeur chargé d'en gérer la valeur. Par exemple, dans le cas d'un problème plan, on utilisera les notations N, S, E, O pour indiquer les directions Nord, Sud, Est, Ouest, des variables attribuées à un autre processeur, et L pour signifier qu'il s'agit d'une variable locale, gèrée par le processeur pris en considération.

Sur le domaine situé en bas, à droite sur la figure 2 ;

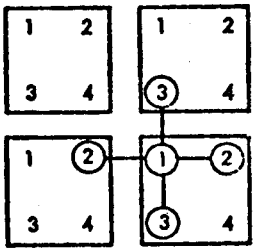


Fig. 2

Point 1 : (L, V2), (L, V3), (0, V2), (N, V3)
Point 2 : (E, V1), (L, V4), (L, V1), (N, V4)
Point 3 : (L, V4), (S, V1), (0, V4), (L, V1)
point 4 : (E, V3), (S, V2), (L, V3), (L, V2)

*ii) Sans faire intervenir une position explicite, on peut générer des adresses logiques déterminant sans ambiguité l'appartenance d'un point à tel ou tel sous domaine. C'est ainsi que le numéro du noeud I peut être défini (figure 3) par les valeurs des indices courants i et j, et des paramètres $\ell 1$, L1 et DEP.

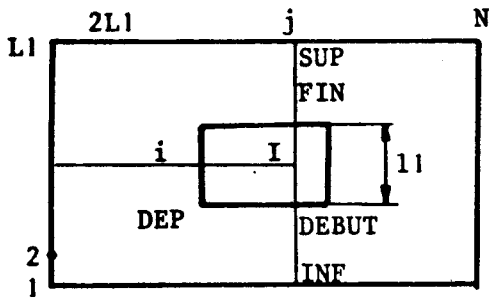


Fig. 3

$\forall\ i \in [1, \ell 1]$
DEBUT = DEP + (j-1) L1
FIN = DEBUT + $\ell 1$ - 1
I = DEBUT + i - 1

Dans le calcul des noeuds voisins, on vérifie que le numéro d'un tel noeud, J, généré par indexation est cohérent avec la géométrie du domaine, à l'aide des relations :

$$1 \leq J \leq N$$

$$J > INF \text{ (noeud sud)}$$

$$J < SUP \text{ (noeud nord)}$$

$$DEP - L1 \leq J \leq DEP - L1 + \ell 1 - 1 \text{ (noeud ouest)}$$

$$FIN + L1 + \ell 1 + 1 \leq J \leq FIN + \ell 1 \text{ (noeud est)}$$

Ce procédé qui réclame un certain nombre de tests et de calculs arithmétiques nécessite peu de mémoire et s'avère assez pratique à l'usage. On remarquera qu'il n'est pas utile que les pavés soient tous égaux.

II - ETUDE DU CYCLE DE BASE

2.1. LE DEBUT DU TRAITEMENT

Au début du traitement se fait sentir, au niveau de chaque processeur, une double besoin de programme et de données. Selon la stratégie adoptée, l'initialisation d'un processus peut nécessiter le chargement de la totalité d'un programme si l'on dispose d'un espace mémoire suffisant, ou plus simplement d'un bloc d'instructions, voire d'une seule (la première) ou de son adresse.

Les données nécessaires sont des paramètres généraux (constantes de calcul) ou locaux (relatifs à la topologie, indices des noeuds concernés, ou encore des variables nécessaires au traitement). Les premières sont communes à tous les processeurs et peuvent, on vient de le voir, être distribuées en parallèle dans des zones mémoires locales à chaque processeur. Les secondes sont chargées séquentiellement, processeur après processeur, par un calculateur externe fonctionnant sur un programme spécifique.

Cette phase d'initialisation se caractérise par l'état passif des processeurs de traitement. Lorsqu'elle est achevée, un signal doit les placer dans un état actif dans lequel ils exécutent le programme qui les concerne.

2.2. PROTOCOLE DE FIN DE CALCUL

Le caractère asynchrone des traitements à effectuer pose un sérieux problème en ce qui concerne la fin de l'exécution d'un processus itératif. En effet, si l'on prend par exemple le cas d'une série de relaxations, un processeur peut, pour toutes sortes de raisons, prendre du retard par rapport à ses voisins. Supposons qu'il mette beaucoup plus de temps qu'eux pour effectuer chaque relaxation ; il se peut alors qu'un processeur ne détectant plus d'évolution de la valeur des variables qu'il est chargé de gérer, soit considéré comme ayant achevé son traitement. En réalité, dès que l'élément le plus lent fournit de nouvelles valeurs, celles-ci sont à même de provoquer une relance du processus itératif.

a) Pour éliminer cette difficulté, on peut laisser la décision à une machine externe opérant par scrutation. Il n'y a alors aucune intervention locale. Cette solution centralisée, coûteuse dès que le nombre de processeurs devient important, nous semble dangereuse ; une panne du scrutateur, non détectée, peut demander un délai important avant d'être prise en compte comme telle.

b) Il semble bien plus intéressant de laisser à chaque MC une part d'initiative. Dès qu'une fin de traitement éventuelle est localement détectée, le processeur génère des paramètres de fin de traitement (par exemple, une norme locale, un booléen). Le calcul itératif se poursuit jusqu'à la réception d'un signal de contrôle commandant l'arrêt de cette phase de traitement. Pour préciser ce point, considérons le processeur P_k effectuant sur un sous domaine Ω_k un calcul itératif portant sur une variable U en un noeud de numéro I. Il calcule :

$$Z_k = U_i^{(n+1)} - U_i^{(n)}$$

où n est le nombre de relaxations effectuées. Le critère à partir duquel on détermine si la convergence est localement atteinte s'élabore à partir du calcul de la valeur absolue de Z_k et correspond à :

$$|Z_k| < \varepsilon_k \quad \text{avec } \varepsilon_k \text{ positif donné}$$

Si Z est l'ensemble des valeurs de $|Z_k|$ ($Z = I|Z_k|\}$), on dira que la convergence globale est atteinte si $||Z|| < \varepsilon$, avec ε donné.

Dans ce cas, le paramètre de fin de traitement sera, localement pour P_k le booléen ($|Z_k| < \varepsilon_k$).

En ce qui concerne le mode de décision, plusieurs scénarios sont possibles :

- Ces paramètres sont transmis à une machine extérieure à qui il revient de prendre la décision finale après réception de l'ensemble de ceux-ci.

- Le MC concerné se contente d'envoyer un message lui signalant qu'il est dans une situation de fin de processus, à charge pour le processeur extérieur de lire les paramètres correspondant quand il estime utile de le faire.

- Le MC transmet un signal de fin de traitement à l'un de ses voisins, selon un schéma pré-établi. Ce signal progresse au fur et à mesure que les autres MC estiment avoir eux-aussi achevé leur calcul, pour aboutir à un MC bien défini qui a donc tout le pouvoir de décision.

Tel quel, ce mécanisme est dangereux et manque de souplesse car il ne permet pas le redémarrage d'un processus qui serait resté quelque temps stationnaire. Il faut non seulement que la décision locale soit cooptée par les voisins en contact avec le MC concerné, mais encore qu'en soit prévue une possibilité d'annulation.

2.3. ANALYSE DE LA FONCTION DE STOCKAGE

Le volume des informations à stocker pour la résolution d'un problème même moyen est considérable, ou doit être en tout cas considéré comme tel. On conçoit que la fonction de stockage joue un rôle prépondérant dans l'organisation d'une machine parallèle, et qu'il convient d'y apporter une grande attention.

2.3.1. Les 3 niveaux

Un des problèmes majeurs du calcul parallèle concernant le partage des informations entre les processeurs, il semble rationnel d'essayer d'opérer une classification des zones de mémoire en fonction du degré plus ou moins grand de mise en commun des données qui y sont stockées. On peut ainsi distinguer 3 zones bien distinctes :

- une zone de données permanentes, au moins pendant une phase de calcul correspondant à une résolution particulière du problème considéré.

Cette zone contient les constantes et les paramètres communs à tous les processeurs pendant cette phase, et qui n'ont pas à être modifiés.

- A l'autre extrémité de la cartographie mémoire, on trouve un ensemble de zones associées chacune de façon exclusive, à un processeur particulier. Chaque zone contient :

  * des paramètres locaux à ce processeur

  * les coefficients des opérateurs relatifs aux lignes des matrices associées aux points du sous-domaine géré par ce processeur

  * les variables de relaxation internes au sous-domaine, à l'exception de celles qui correspondent à ses frontières communes aux sous-domaines voisins.

- Enfin, une série d'espaces partagés deux par deux entre les processeurs. Dans chaque espace, on trouve :

  * les variables de relaxation relatives aux frontières de deux sous-domaines

  * des paramètres particuliers : variables d'états liées à une fonction de contrôle et de surveillance du déroulement des calculs.

2.3.2. *Le mode de partage*

Si l'on pousse plus loin l'analyse d'un processus de relaxation on s'aperçoit qu'il n'y a aucun intérêt à ce que 2 processeurs différents aient à remettre à jour la même variable. Il en est de même pour les variables partagées relatives aux frontières d'un sous-domaine donné : un processeur est chargé d'assurer leur relaxation, et il est maître de ce processus. Le partage avec le ou les processeurs concernés par la valeur prise par une variable, se réduit donc à une "mise au courant" de ces derniers. Dans le déroulement d'un algorithme synchrone, cet échange devant se faire à un instant précis, nécessiterait la mise en oeuvre d'opérations de lecture-écriture. Pour un algorithme asynchrone, la notion de rapidité de la transmission de l'information perd beaucoup de son intérêt : on sait qu'un "retard" plus ou moins important peut exister entre la remise à jour de la valeur d'une variable et la prise en compte de celle-ci dans le processus général de relaxation, sans pour autant que la convergence de celui-ci en soit affectée. Dès lors, des opérations de lecture sont seules nécessaires, de la part des processeurs voisins ; le processeur désirant acquérir une information au moment précis où celle-ci est utile se contentera

de la lire à un emplacement-mémoire devant être connu sans ambiguité.

En ce qui concerne les autres variables mises en partage entre quelques processeurs, c'est à dire des paramètres définissant leur état et le degré d'avancement de leur activité, aucune opération d'écriture particulière n'est à envisager. **Le bon déroulement de l'algorithme ne nécessite donc qu"un seul rédacteur pour une variable donnée.**

Les conséquences que l'on peut en attendre sont importantes, et permettent de prévoir une plus grande simplicité de l'organisation de la mémoire de la machine, et de ses outils de communication.

Il existe cependant des cas où des opérations d'écriture sont nécessaires, qui mettent en jeu éventuellement plusieurs rédacteurs : lors de l'initialisation des processeurs et du chargement, dans des zones bien précises, mais qui restent encore à définir, de toutes les données nécessaires au lancement du traitement. On conçoit que de telles opérations qui peuvent avoir lieu périodiquement, aient à se faire sous un contrôle renforcé, et réclament une gestion particulière. On est ainsi amené à définir 2 modes d'échanges d'information :

* Un mode de chargement, sous lequel des opérations d'écriture sont autorisées pour plusieurs rédacteurs, et un mode de protection de l'espace mémoire est assuré par une fonction de contrôle.

* Un mode de traitement sous lequel les opérations de lecture de l'espace mémoire alloué à un processeur sont seules autorisées pour les autres processeurs.

2.3.3. *Volumes des zones de mémoire*

De l'analyse d'un certain nombre de programmes, on peut retenir quelques chiffres concernant les volumes relatifs à réserver pour les différentes zones de mémoire.

La zone de données permanentes ne contient que quelques dizaines d'objets et, en tout état de cause, n'occupe qu'un pourcentage infime de la mémoire totale nécessaire.

La zone comportant l'ensemble des données privées est la plus importante, et sa taille est fonction principalement du nombre de noeuds et de variables que doit gérer chaque processeur. Si les paramètres locaux (le plus souvent entiers) ne sont qu'une poignée il faut compter une dizaine de variables réelles associées à chaque noeud et à chaque variable de relaxation. Les différences de volumes entre l'espace privé à réserver à un processeur et l'espace que ce dernier doit partager avec d'autres tiennent essentiellement au rapport existant entre le nombre de noeuds internes et celui des noeuds aux frontières d'un sous-domaine. La figure 4 en donne le pourcentage pour des sous-domaines constitués de pavés carrés de densité de noeud constante.

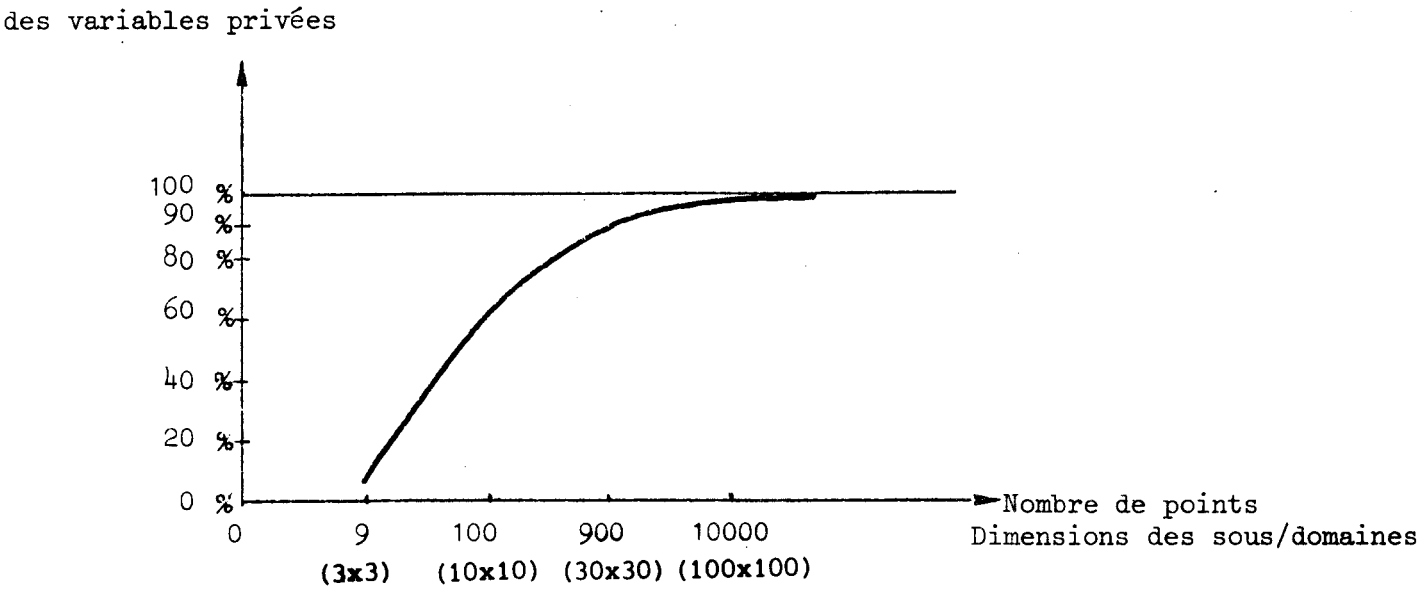


Fig. 4

Il apparait comme parfaitement admissible de retenir une proportion moyenne de 90 - 10 % entre les variables privées et les variables partagées, au niveau de l'espace alloué à chaque processeur, ce qui permet de prévoir des volumes d'échanges relativement faibles.

2.4. ETUDE FONCTIONNELLE DE LA MEMOIRE DE DONNEES

L'étude fonctionnelle de la mémoire de données s'inscrit dans un contexte producteur-consommateur pour lequel on peut envisager deux stratégies :

- le producteur sait qui consomme et dépose son produit dans une mémoire du consomateur qui n'a qu'à l'y prendre : c'est un pilotage par disponibilité

- le consomateur sait où est le produit et dirige lui-même la recherche : c'est un pilotage par nécessité.

Nou pouvons donc distinguer 2 modes de fonctionnement : un mode boite-aux-lettres et un mode panneau d'affichage.

2.4.1. *Mémoire partagée "boite aux lettres"*

Chaque processeur dispose d'un élément de mémoire privée, et d'éléments partagés qui lui permettent de communiquer avec d'autres processeurs (figure 5). Le processeur P_i, pour communiquer avec P_j dépose dans l'élément M_{ij} un message à l'usage exclusif de P_j

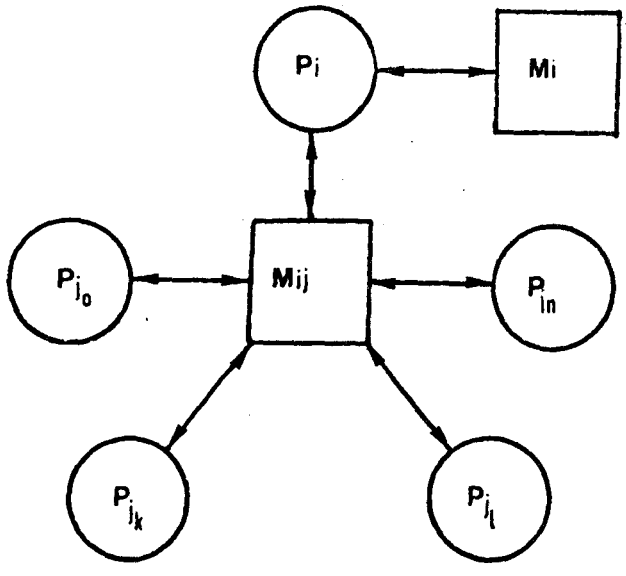


Fig 5

- l'avangage principal que l'on peut attendre d'une telle organisation réside dans la très grande simplicité de la gestion des accès à la mémoire, qui représente un des problèmes fondamentaux posés par la

conception d'une machine parallèle. La difficulté est en effet réduite, au niveau de l'implémentation à celle qui se pose au niveau d'une machine bi-processeurs.

D'autre part, la protection des informations est assurée par le caractère explicite de l'adressage vers tel ou tel élément, ainsi que par le fait qu'un objet donné soit partagé entre 2 processeurs, et deux seulement.

Il convient de remarquer que la mémoire ne joue pas seulement un rôle de dispositif de stockage, mais assure aussi celui d'outil de communication, ce qui permet du même coup de réduire la complexité de ce dernier.

- Inconvénients :

L'obligation de diriger une information donnée vers un élément de mémoire parfaitement défini, si elle assure une bonne sécurité quant à son utilisation, représente malgré tout une certaine lourdeur dans le principe des communications. Elle se traduit par des contraintes au niveau de la programmation qui doivent influer sur le délai de mise au point des programmes.

D'autre part, on privilégie ainsi fortement les communications entre un nombre limité de processeurs, au détriment de toutes les autres qui doivent se faire au travers d'une "chaîne" de transmissions. Ces dernières réclament de ce fait un temps au moins égal à la somme des temps de lecture/écriture nécessaires à la circulation de l'information le long d'un maillon de la chaîne.

Enfin, il risque d'apparaître une mauvaise utilisation de l'espace mémoire qui peut se traduire par des besoins accrus. Le fait de fractionner la mémoire en zones indépendantes de taille fixée fait que, dans chaque zone, existent quelques mots qui ne sont pas utilisés, et sont de ce fait perdus. Le grand nombre de zones nécessaires (5 à 7 par processeur) génère globalement un volume important.

Précisons que ce mode de fonctionnement conviendrait particulièrement bien à une machine de type réseau cellulaire.

2.4.2. Mémoire partagée "panneau d'affichage"

Dans ce mode d'organisation, un élément de la mémoire est partagé entre un nombre variable de processeurs. Pour acquérir une information générée par un producteur, un consomateur doit aller la chercher à une adresse de rangement où il sait devoir la trouver.

- avantages :

On notera tout d'abord la grande simplicité qui en découle au niveau de la programmation : on n'a pas à tenir compte du nombre de processeurs, pas plus que de l'attribution des sous-domaines et les problèmes de communication deviennent transparents au programmeur.

Les communications entre 2 processeurs gérant 2 sous-domaines éloignés l'un de l'autre ne posent plus de difficultés particulières et sont traitées de la même façon que toutes les autres.

La mémoire n'est plus ici atomisée en une multitude de zones bien distinctes, mais forme un ensemble continu. Il en résulte, par rapport au schéma précédent, une meilleure utilisation de l'espace disponible et une réduction notable du coût de stockage.

- Inconvénients :

Ce sont ceux qui sont communs aux machines multiprocesseurs classiques, à savoir :

Une gestion délicate et généralement assez coûteuse des conflits d'accès. On sait que la complexité croît, grossièrement, comme le carré du nombre de processeurs se partageant la même mémoire.

De sérieux problèmes sont posés par la sécurité des informations, principalement en cas de panne ou de mauvais fonctionnement d'un processeur au cours d'un accès à la mémoire. On est conduit ainsi à protéger certaines zones contenant des informations particulièrement critiques. Ce dernier point, cependant, n'est pas aussi délicat qu'il devrait l'être. En effet, lorsque les processeurs fonctionnent en mode de traitement, le partage d'une information stockée ne se fait que par des opérations de lecture. Les opérations d'écriture, pour chaque processeur, ne peuvent se faire que dans une zone bien précise et ne peuvent, normalement altérer des éléments indispensables au bon déroulement de l'algorithme.

2.5. LE PROBLEME DES ENTREES/SORTIES

Les échanges avec des périphériques posent de sérieuses difficultés dans le traitement des gros problèmes scientifiques évoqués. On peut envisager encore une fois deux niveaux de communication, se différentiant par la fréquence et le volume des informations échangées ainsi que par l'utilisation qui en est faite.

2.5.1. *Un niveau de contrôle*

Les échanges se font par l'intermédiaire d'une machine hôte lors de la mise au point d'un programme, d'une interruption du traitement d'une phase d'initialisation ou de fin de calcul. Dans ce cas, l'échange entre un processeur et un périphérique a la priorité sur toutes les autres communications entre processeurs.

Si la communication se fait non par une voie privilégiée mais par l'intermédiaire du sous-système de communication (SSC), le volume est tel que, du fait de la priorité accordée, toutes les autres communications sont bloquées.

2.5.2. *Un niveau de traitement*

Les échanges se font dans ce cas avec des mémoires de masse (disques, lecteurs de bandes magnétiques). Il semble difficile d'en imaginer avec des dispositifs de visualisation (écrans, tables traçantes) qui, normalement n'utilisant que les résultats issus d'un assemblage des résultats partiels qui sont fournis par un monoprocesseur controlant la machine parallèle.

Les volumes concernés sont a priori extrêmement importants. Les besoins s'étendent à plusieurs dizaines d'articles pouvant atteindre 1 K octet, par processeur de traitement. La distribution de ces communications se fait donc par paquets de centaines, voire de milliers de mots, et nécessite un très grand débit de la part du SSC. Ici réside une très grosse contrainte pour l'architecture : les risques d'engorgement des voies de communications sont élevées même avec très peu de processeurs.

Une difficulté supplémentaire vient du fait qu'a priori, tout processeur peut avoir besoin d'accéder à un fichier stocké sur un disque

donné. La mémoire de masse doit donc être considérée comme totalement partagée. Compte tenu des échanges que cela représente, il y aurait probablement avantage à dupliquer un certain nombre de fois des fichiers particulièrement utilisés, sur des disques dont l'accès serait limité à un nombre déterminé de processeurs. Le coût à envisager est nécessairement considérable, mais cette façon de concevoir les choses est probablement la seule qui permette d'assurer l'utilisation rationnelle des gros fichiers de données, et donc le fonctionnement de la machine. On pourra évaluer quantitativement les différents paramètres qui interviennent à ce niveau en réalisant une extrapolation des travaux de Fung et Torng [FUN 79] d'une part, de COVO [COV 74] d'autre part.

2.6. LA FONCTION DE COMMUNICATION

Il faut distinguer, en fonction du rôle qu'ils ont à jour, 3 types d'objets qui doivent transiter dans le SSC afin d'assurer le travail de chaque processeur. Ce sont :

- des mots d'instruction
- des variables ou opérandes
- des articles de fichier

A chaque type d'objet correspond un volume, une fréquence et des directions de transfert spécifiques. Il y a donc lieu d'opérer une distinction précise se traduisant par une séparation fonctionnelle des communications au niveau du SSC.

On peut ici faire la remarque qu'une transmission du type commutation de ligne est en opposition avec le concept de l'asynchronisme : autoriser un processeur à se placer dans un état passif d'attente après l'émission d'une requête revient à instaurer un cadencement implicite de l'exécution. On peut tirer un bien meilleur parti de cette notion en utilisant le principe de la commutation de message : un processeur après l'émission d'une requête, peut gérer d'autres tâches, sans que l'on aie à se préoccuper des retards introduits dans le processus de mise à jour.

III - ETUDE DÉTAILLÉE D'UNE SÉQUENCE PARALLÈLE

Nous prenons comme exemple le cas d'un problème plan, nécessitant une résolution itérative. Chaque processeur éffectue une série de relaxations sur une variable U_o selon un schéma d'interpolation qui fait intervenir 4 valeurs voisines, soit, respectivement :

U_1 à l'ouest, U_2 au sud, U_3 à l'est, U_4 au nord.

On effectue alors une série de calculs de la forme :

$$U_o = C_1 U_1 + C_2 U_2 + C_3 U_3 + C_4 U_4 + B_o$$

Cet exemple n'est exposé que pour permettre une estimation des coûts de calcul. Il représente un cas d'espèce relativement simple, pour lequel en particulier, mis à part quelques coefficients numériques, le code est commun à tous les processeurs.

L'ensemble du calcul peut être écrit en langage évolué de façon à en donner une représenation compacte. L'APL nous semble intéressant, autant pour sa concision que pour sa capacité à rendre compte simplement des opérations vectorielles.

```
      ∇ESSAI[□]∇
    ∇ ESSAI
[1]  ET1:MAX←0
[2]  ET:S←0
[3]   SUP←(1+⌊IND÷L1)×L1
[4]   INF←SUP+1-L1
[5]   J←((IND-L1)×IND>L1),((IND-1)×IND>INF),((IND+1)×IND<SUP),(IND+L1)×IND≤N-L1
[6]   J←(J=0)/⍳⍴J]←N+1
[7]   W←(+/COEF[IND;]×U[J])+B[IND]
[8]   Z←|W-U[IND]
[9]   U[IND]←W
[10]  MAX←(MAX×MAX>Z)+Z×MAX≤Z
[11]  IND←((IND+1)×(IND+1)≤SUP)+(IND-D1)×IND+1)>SUP
[12]  →(IND≤N1)/ET
[13]  →(MAX>EPS)/ET1
    ∇
```

Dans ce programme, la variable IND désigne l'indice d'un noeud courant. J est celui d'un noeud voisin intervenant dans le processus de relaxation qui porte sur IND-MAX désigne le terme de valeur $|U_o^{n+1} - U_o^n|$, où n est le niveau de relaxation atteint par la variable U_o

Dans la mesure où il est constitué d'opérations très fréquentes il serait possible de donner à ce programme de 13 instructions, qui réclame 568 octets dans le système APL SV/360, une représentation codée sous un faible volume. Jones [JON 80] propose de représenter chacune de ces opérations complexes par un octet particulier (supercode). Dans ces conditions le volume de stockage du programme complet serait, au plus, de l'ordre de 80 octets.

En s'inspirant des jeux d'instructions de divers microprocesseurs courants disponibles sur le marché, on peut écrire un programme équivalent en langage machine, avec environ 150 instructions. En négligeant les cycles d'attente liés à l'architecture, et sur la base des catalogues correspondants le nombre de cycles d'exécution nécessaires est alors de l'ordre de 2500.

Indépendemment du langage, le nombre de requêtes d'opérandes est ici égal à 11 (problème plan), et se partage en 7 requêtes de variables privées et de 4, susceptibles d'être partagées avec une probabilité de 0.1. Ceci nous conduit aux chiffres moyens de 10.6 requêtes privées pour 0.4 requêtes externes, concernant toutes des variables réelles exprimées en virgule flottante sur des formats allant de 32 à 64 bits. Ces chiffres doivent, en toute généralité, être considérés comme exprimant une caractéristique commune pour les applications de ce type.

Si l'on admet une bande passante de 16 bits (format des adresses) pour chaque voie du SSC, on a donc :

- un taux de requêtes en opérandes avoisinant $\frac{44}{2500}$ (avec des réels sur 64 bits) auquel s'ajoute un taux de réponses égal soit au total un taux de messages de $\frac{88}{2500}$. Sur ce chiffre, la fréquence du traffic privé est de $\frac{79}{2500}$, celui concernant des données partagées avec un autre processeur n'étant que de $\frac{9}{2500}$.

- Si l'on examine les messages concernant les instructions en langage machine, en admettant qu'elles soient toutes codées sur 16 bits, le taux correspondant a une valeur proche de 0.12 avec les chiffres proposés. Le taux de messages correspondant à un programme équivalent, écrit en code "évolué" serait très probablement inférieur à 0.1, sans qu'il soit possible ici de le chiffrer avec précision.

IV - PROBLÈMES POSÉS PAR LE CODE

On vient de voir qu'une part notable du volume des communications au sein d'une machine parallèle provient des besoins en code, et que ceux-ci sont variables selon le type de langage utilisé. Un certain nombre de problèmes se posent, impliquant chacun des choix éventuellement contradictoires. Nous nous proposons donc de les étudier, ainsi que les implications des différentes options qui en découlent.

4.1. GENERALITES

4.1.1. *Découpage d'un programme en blocs fonctionnels*

Nous nous plaçons dans la perspective dans laquelle différents éléments d'un même processus sont exécutés, en parallèle, par un ensemble donné de processeurs. Le mode de calcul asynchrone retenu fait qu'à un instant donné, les instructions qui leur sont nécessaires se trouvent globalement réparties dans l'ensemble du programme dont ils se partagent l'exécution. Il semble intéressant dans ces conditions, d'envisager le découpage de ce dernier en blocs fonctionnels cohérents qui sont :

- des routines
- des boucles "Tant que"
- des fonctions particulières du traitement

Selon le niveau auquel on se place, on aura à considérer :

- une instruction élémentaire en langage machine
- un bloc élémentaire de quelques instructions constituant une boucle ou une macro-instruction
- un ensemble de blocs élémentaires fermé (c'est à dire ne comportant pas de branchements permettant d'en sortir autrement que par l'intermédiaire de la dernière instruction). Un tel ensemble peut éventuellement constituer une routine.

On aura donc à raisonner différemment sleon que l'on part :

- de l'instruction en assembleur, voire de la micro-instruction
- de l'instruction en langage de haut niveau (LHN) qui peut elle-même se décomposer en un bloc élémentaire d'instructions en langage machine.

4.1.2. Enchaînement des séquences de traitement

En fonction de ce qui précède, l'enchaînement des séquences de traitement peut s'envisager à deux niveaux différents.

- Sur la base d'un enchaînement d'instructions.
- Ou bien sur celle d'un enchaînement de blocs de programmes.

4.1.3. Intérêt d'un langage de haut niveau [JON 80] [KAL 79]

L'utilisation d'un langage de haut niveau présente des avantages importants

- elle réclame un moindre nombre d'instructions
- elle permet de regrouper des séquences d'instructions élémentaires intervenant avec une certaine fréquence
- elle permet de réaliser économiquement des opérations relativement complètes.

Un certain nombre de fonctions semblent nécessaires, ou simplement utiles, dans l'optique des calculs scientifiques envisagés au chapitre I.3.1. Ce sont :

- des opérations de calcul matriciel et, en premier lieu, le produit scalaire. La multiplication et l'addition matricielle interviennent aussi fréquemment. De même la résolution de systèmes linéaires et l'inversion de matrices de faibles dimensions
- des calculs de différentiation
  de gradient
  de divergence
  de rotationnel
- des opérations de sommation et de multiplication (Σ, π)
- des opérations de génération d'ensembles d'indices, extrêmement utiles pour le maniement pratique des variables discrétisées sur des espaces complexes.

On trouve ici un ensemble de fonctions constituant un échantillonage du langage APL. Celui-ci, extrêmement concis par ailleurs, semblerait parfaitement bien adapté, sur ce point, à ce genre d'utilisation.

4.2. PRESENTATION DES PROBLEMES

On distingue 3 problèmes relatifs à l'implantation du programme, son niveau de complexité et son mode d'acquisition par les processeurs.

4.2.1. *Implantation du programme*

Soit C l'ensemble du code relatif à un programme parallèle. Désignons par C_i le code nécessaire à un processeur P_i pour l'exécution d'un processus donné. On a :

$$C = \{C_1, C_2, \ldots C_i, \ldots C_n\}$$

on remarque que $C_i \cap C_j$ est en général non vide.
Soit $V(C_i)$ le volume de mémoire requis pour le stockage de C_i.

L'estimation du volume total occupé et du temps de chargement nous conduit à envisager plusieurs types d'implantation.

- Locale

C'est une solution entièrement éclatée pour laquelle le volume total occupé est :

$$V1(C) = V(C_1) + \ldots + V(C_i) + \ldots + V(C_n)$$

Le code nécessaire à chaque processeur est implanté une fois pour toutes avant l'exécution du processus. Soit T le temps d'acquisition d'une instruction dans une mémoire de masse. Ce chargement initial réclame un temps égal à V1(C) x T, qui constitue la mesure de la perte de temps relative au code dans l'exécution de l'algorithme.

- Centrale

Le volume nécessaire au stockage de la totalité du code est alors le volume V2 défini par :

$$V2(C) = \bigcup_{i=1}^{n} V(C_i) - \bigcap_{i=1}^{n} V(C_i)$$

on constate que : $V2(C) < V1(C)$ car $V1(C) = \bigcup_i V(C_i)$.

Cependant dans ce cas, les conflits d'accès sont nombreux et entrainent un engorgement important des voies d'accès au sein du SSC. Le gain en volume est donc contrebalancé par l'existence d'un taux de conflit θ.

Soit f la fréquence moyenne d'utilisation d'une instruction. Le temps total moyen nécessaire pour le chargement du code est :

$$T = V1(C) \times f \times \theta$$

On ne peut le réduire qu'en minimisant les facteurs f et θ. On peut jouer sur le second de deux façons possibles.

* Par une implantation mixte du programme. Les portions de programme utilisées durant un intervalle de temps donné sont stockées dans des mémoires liées à un certain nombre de processeurs, de façon à réaliser un moyen terme entre les deux types d'implantation exposés. Les processeurs s'adressent, pour acquérir les instructions dont ils ont besoin, au site chargé de leur assurer ce service. Le risque auquel on s'expose alors est celui d'un engorgement des sites distributeurs qui tendent à se spécialiser au détriment des calculs qu'ils ont à effectuer.

* Par l'utilisation d'un langage de haut niveau qui réduit également la valeur du paramètre f. On peut, dans ces conditions, se poser le problème du niveau de complexité du langage utilisé, relativement à son implémentation.

4.2.2. *Niveau de complexité du langage*

Le LHN suppose, pour son exécution, l'existence d'un interpréteur. Chaque instruction du LHN est, par son intermédiaire, associé à une routine en langage machine, stockée éventuellement en RAM, mais plus probablement en ROM (exemple du P code du Pascal). Suivant les implantations respectives des instructions LHN et des routines correspondantes, on peut envisager 4 schémas possibles, d'un intérêt très variable :

implantation locale pour le LHN, locale pour le langage machine
implantation locale pour le LHN, centrale pour le langage machine
implantation centrale pour le LHN, locale pour le langage machine
implantation centrale pour le LHN, centrale pour le langage machine.

4.2.3. *Mode de chargement du programme*

Il peut se faire instruction par instruction, ou par blocs fonctionnels.

- Par instructions

Dans cette perspective, l'implantation du programme n'est pas réalisée sur un site. Le coût de stockage est nul au niveau local, ce qui n'est pas le cas du coût de calcul qui, du fait des temps d'attente associés à l'acquisition de chaque instruction, devient éventuellement important.

- Par blocs

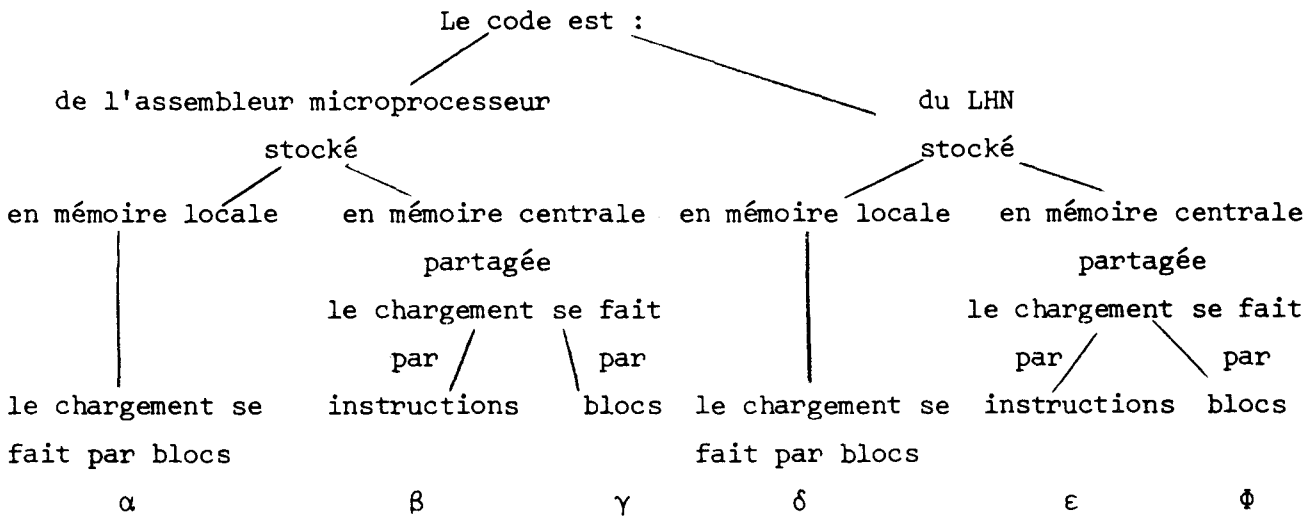
Lorsque le chargement se fait par blocs, il y a, relativement au code, une totale indépendance au cours du traitement de chaque bloc. Lorsque celui-ci contient une boucle majeure correspondant à un traitement de longue durée, les problèmes de conflits sont réduits au minimum. Cependant, on est obligé d'assurer la réplication du même bloc en un certain nombre d'exemplaires, ce qui accroît notablement les coûts de stockage.

4.3. DISCUSSION

Les problèmes que nous venons d'évoquer ne sont pas indépendants les uns des autres. Les différentes façons de les aborder font naître un ensemble d'implications dont il faut essayer d'évaluer la protée, afin de dégager la ou les options intéressantes au niveau de la conception d'une machine du type envisagé.

4.3.1. *Graphe des options*

Pour schématiser d'une façon commode ce qui vient d'être dit au paragraphe précédent, il est pratique d'utiliser le graphe suivant :



4.3.2. *Etude des options*

- Parmi toutes ces options, δ, la plus coûteuse, ne présente aucun intérêt.

- Dans le cas de l'option α, la machine constitue un réseau de micro-ordinateurs. Le coût matériel en est important du fait de la

duplication des blocs, et est proportionnel à leur taille et au nombre de processeurs. On notera qu'il y a intérêt à charger des blocs de grande taille de façon à bénéficier de boucles de haut degré de répétition. Ce dernier point joue un rôle encore plus important dans le cas de l'option γ, pour laquelle le coût des communications est élevé. Dans ces conditions, la solution consistant à charger des blocs de taille fixe est à rejeter. Celle qui consiste à charger des blocs de taille variable pour s'adapter aux structures répétitives du programme pose toute sorte de difficultés. Peut-on, en effet, toujours prévoir la taille optimum d'un tel bloc et décider de l'opportunité de tel ou tel découpage du programme ? Il n'est pas du tout sûr que tout partitionnement conduise à des performances acceptables.

- Le coût principal de l'option β est un coût de calcul, qui dépend étroitement des communications. Evaluons le grossièrement.

Désignons par f la fréquence moyenne des requêtes d'instructions émises par un processeur en fonctionnement. Soient T1 et T2 les temps de service respectifs du sous-système de communication SSC et de la mémoire partagée, c'est à dire les nombres moyens de cycles pendant lesquels ils sont mobilisés pour assurer le transfert, ou la réception d'une requête et la délibrance d'une instructions. Pour une configuration donnée de la machine, il existe un temps moyen E(T1, T2) qui caractérise le délai, en nombre de cycles, entre l'émission d'une requête et la réception de l'instruction demandée. La valeur de ce paramètre nous permet d'évaluer le nombre maximum de sites processeurs qu'il est possible de connecter au SSC sans que l'ensemble ne s'engorge par :

$$NSC = \frac{1}{f \times E(T1,T2)}$$

Graphiquement, on peut, pour différentes valeurs de E, représenter les courbes correspondant à diverses valeurs de f.

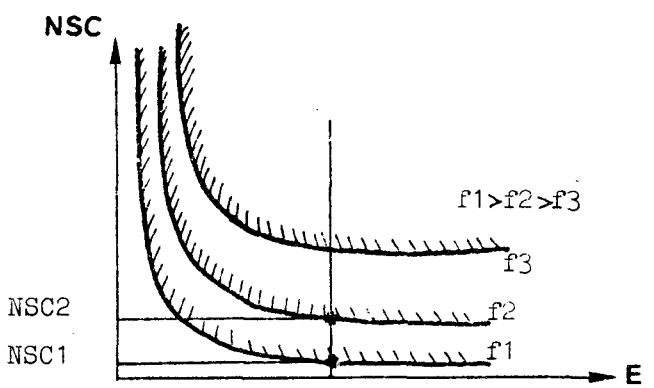


Fig. 6

L'existence de conflits d'accès à la mémoire dans un premier temps, ne bloquerait pas le fonctionnement de la machine, mais aurait pour conséquence de diminuer la fréquence des requêtes au niveau de tous les processeurs, cette baisse étant égale au nombre de cycles d'attente nécessaires pour obtenir l'accès à la mémoire et au SSC.

Désignons par NSC1 le nombre maximum de sites connectés, de telle sorte qu'il n'y ait aucun conflits avec une fréquence f1. Soit NSC2 un nombre de sites légèrement supérieur. On peut considérer, un nouvel équilibre dans les communications étant atteint, avec une fréquence f2, que la différence $\frac{1}{f2} - \frac{1}{f1}$ soit égale au nombre de cycles supplémentaires d'attente.

La fréquence maximum de requêtes dépendant du programme, il existe 2 moyens possibles de réduire le coût :

* Le premier consiste à diminuer la valeur de E, c'est à dire à travailler avec une technologie très rapide, mais aussi très chère.

* Le second consiste à utiliser un procédé d'anticipation qui permet en outre d'accroître le nombre de processeurs connectable au SSC.

- Les options ε et Φ ne présentent de l'intérêt que lorsque l'on envisage la technologie VLSI, qui permet d'implanter en ROM, dans le chip processeur lui-même, le code correspondant au LHN choisi. En ce qui concerne la seconde option, étant donné la concision des programmes écrits en LHN, on peut envisager de les stocker en totalité dans des mémoires locales à chaque processeur.

CHAPITRE IV

ETUDE DU COUT DE CALCUL

Afin d'étudier le coût du calcul parallèle, il nous a semblé nécessaire de faire la part entre les performances autorisées par les algorithmes sur lesquels s'est porté notre choix, et celles des différents modèles d'architecture qu'il est possible de retenir pour les y implémenter.

Pour cela, nous avons décidé d'étudier, en simulation, le comportement des algorithmes chaotiques, en dehors de tout cadre architectural. Puis nous nous proposons d'utiliser les résultats obtenus pour développer un modèle analytique dans lequel il soit possible d'inclure un facteur de dégradation, lié au conflits et aux contraintes imposées par un type donné de machine.

I - SIMULATION

1.1. INTRODUCTION

Partant des expérimentations de Rosenfeld [ROS 69] et Baudet [BAU 78] nous avons tenté de développer leur méthode pour obtenir les renseignements les plus précis possibles sur le comportement de l'algorithme.

Le principal problème à partir duquel nous avons simulé le déroulement chaotique d'un algorithme de relaxation est exposé en annexe. Son choix est dû au fait qu'il apporte une légère complexité supplémentaire par rapport aux calculs de Laplacien (Baudet) ou aux équations à matrices positives monotones (Rosenfeld).

Une des conditions aux limites est en effet une relation de symétrie, ce qui, compte tenu des conditions sur les autres frontières du domaine de calcul, permet d'introduire une plus grande disparité dans les temps d'évaluation des différentes inconnues.

D'autre part, les coefficients de la matrice de l'opérateur sont variables et dépendent de l'indice de la colonne du maillage régulier choisi. Leurs valeurs, exprimées à partir de relations sinusoïdales, sont alternativement positives et négatives. Il est possible que cela soit la cause de résultats observés en sur-relaxation.

Il était intéressant de voir aussi en quoi la discontinuité du domaine d'application de l'opérateur au sein du domaine de calcul était susceptible d'interférer sur les résultats. On trouvera dans [SER 67] un exemple de résolution par un algorithme de sur-relaxation classique en séquentiel du même problème.

1.2. PRINCIPES DE LA SIMULATION

Dans un premier temps, il a semblé nécessaire d'étudier le comportement d'un algorithme chaotique indépendemment de toute architecture c'est à dire d'un point de vue purement théorique. Nous avons donc admis les points suivants :

- On considère une famille de processeurs tous identiques, disposant de leur propre mémoire de programme et de données de taille suffisante pour assurer toutes les duplications nécessaires.

- Les temps de communication de ces processeurs entre eux sont négligeables.

- Aucun blocage du sous-système de communication n'est susceptible de se produire.

- On attribue à chaque processeur P_k un sous domaine Ω_k de façon exclusive, c'est à dire qu'aucun point du maillage n'est confié à plus d'un processeur. Autrement dit, l'algorithme fonctionne selon le principe du partage des tâches et non selon celui de la concurrence entre tâches. Les sous-domaines sont réguliers et généralement convexes ; ils réalisent un pavage du domaine Ω.

- Chaque processeur a accès, dès qu'elles sont disponibles, aux variables calculées par les autres processeurs. Celles dont il a besoin sont situées sur les frontières les plus proches des sous-domaines voisins de celui dont il a la charge.

- Lorsqu'il n'y a qu'un processeur, le déroulement de l'algorithme est identique à celui de Gauss-Seidel, classique, pour monoprocesseur. Quel que soit le nombre de processeurs, le fonctionnement est identique : chaque processeur "balaye" cycliquement le domaine dont il est chargé, effectuant une relaxation de façon séquentielle sur les noeuds de numéros croissants.

On peut objecter le fait que le principe du balayage séquentiel ne conduit pas nécessairement aux meilleurs résultats. Effectivement, sur un monoprocesseur, un balayage de type noir-blanc en damier conduit à un gain de temps de l'ordre de 20%. Un autre type de balayage a été testé, qui consiste à prendre les points en quinconce, sur des lignes non consécutives du domaine. Le gain réalisé est alors de 30%. Ce phénomène est du au fait que dans le temps d'une itération, on balaye en fait plusieurs fois le domaine, et on favorise une meilleure propagation des valeurs recalculées. Tout se passe, numériquement "comme si les variables se faisaient la courte-échelle" pour progresser vers la convergence. Pour un algorithme asynchrone, les choses sont plus délicates dès lors qu'il s'agit de mesurer des performances dans des conditions aussi semblables que possible.

Non seulement les effets précédents s'amenuisent, mais quand change la répartition des domaines, ou, ce qui peut revenir au même, dans un sens, le nombre de processeurs, le principe même de ces types de balayage est bouleversé. Aussi avons-nous adopté le balayage séquentiel, plus facile à comparer avec des résultats connus sur monoprocesseurs.

On notera que le système de découpage du domaine Ω en sous domaines Ω_k indépendants implique qu'il y a toujours plus de variables que de processeurs. Ce point de vue nous semble justifié par la taille des problèmes qui en pratique, créent un besoin actuel en calculateurs parallèles. D'autre part, dans le cas contraire, il aurait fallu introduire des contraintes supplémentaires incompatibles avec la virtualité de l'architecture intervenant dans cette simulation.

Celle-ci s'effectue selon le principe suivant.

Le temps est découpé en une succession d'instants de durée constante δt (paramètre DEL) au cours desquels on observe ce qui se passe pour chacun des R processeurs (pris dans l'ordre séquentiel). La valeur à donner à δt est choisie de telle sorte qu'elle soit un sous multiple de toutes les durées des évènements susceptibles de se produire au cours de la simulation. Le vecteur logique B, de R éléments, est un booléen permettant, au niveau de chaque processeur, de décider ou non de la validation d'un calcul. Lorsque l'on incrémente l'indice I de la variable U à relaxer sur le processeur K, B_k est égal à 1. La valeur est remise à jour et stockée dans l'élément K du vecteur W. Puis B_k reprend la valeur 0. Le temps local $T1_k$ du processeur, initialisé à 0 au début de ce processus, est alors progressivement incrémenté de la quantité δt, jusqu'à ce qu'il égale le temps estimé pour l'exécution de la tâche (donné par TAU). La variable U_i prend alors la valeur de W_k, et les paramètres B_k et $T1_k$ sont réinitialisés pour le calcul sur le noeud suivant.

Le tableau donne la liste des symboles utilisés dans les programmes et leur signification.

N : Nombre de noeuds du domaine
U : Vecteur des inconnues au noeud, dimension : N+1
Z : Vecteur des écarts absolus entre 2 relaxation, dimension N+1
W : Variable de sauvegarde de la dernière valeur calculée
A : Matrice des coefficients de l'opérateur discrétisé, dimension : (N+1)x5
SM : Vecteur second nombre de l'équation linéraire à résoudre
CT : Compteur de relaxation pour la méthode de GAUSS
EPS : Valeur du test de précision
R : Nombre de processeurs simulés
DEL : Intervalle de temps élémentaire d'observation des évènements de la simulation
L : (paramètre effectif) numéro d'un processeur courant
K : (paramètre formel) numéro du processeur courant
B : Vecteur booléen permettant de prendre en compte ou non un évènement
IND : Matrice des points attribués à chaque processeur, dimension (R, N/R)
I : Indice d'unpoint dans Gauss, indice du point pris en charge par le processeur K dans PROC. C'est alors un vecteur de R éléments qui mémorise les tâches entamées
I1 : Paramètre local, indice du point choisi.
J : Vecteur de 5 éléments indices des points voisins du point de numéro I1 les éléments correspondant à des points inexistants ou mal placés ont pour indice N+1
T1 : Temps local d'un processeur. Vecteur de R éléments
T : Temps de la machine simulée.
CMT : Compteur de relaxation de chaque point. Vecteur de N éléments.
TAU : Vecteur des temps d'exécution de chaque point du maillage, dimension N
OMG : Scalaire ou vecteur représente le paramètre de relaxation ; si c'est un vecteur sa dimension est R

Table des symboles

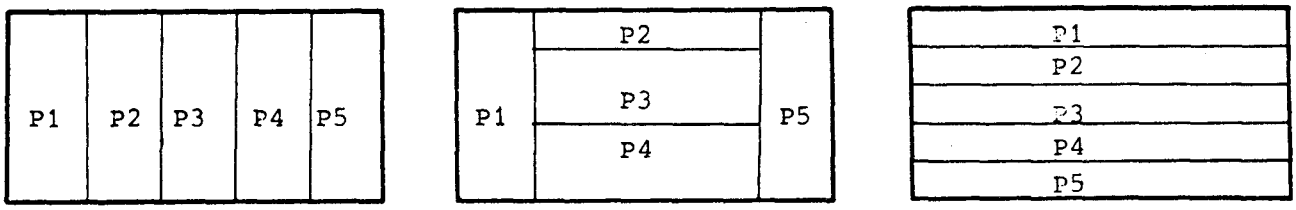


Figure 1 Différentes distributions a) verticale
b) mixte
c) horizontale

1.3. LES EQUATIONS DU PROBLEME

Bien que les simulations aient porté sur des dimensions variables du problème, les points suivants se trouvent toujours vérifiés par le principe même de la méthode.

Le domaine Ω est toujours un rectangle de N1xN2 noeuds avec N2 > N1 (figure 1). Il comporte une frontière Γ, de telle sorte que

$$\Omega = \Gamma \cup \Omega'$$
$$\Gamma \cap \Omega' = \emptyset$$

- pour tous les points de Ω', l'équation discrétisée :

$$U_{ij} = C\, U_{i-1,j} + d\, U_{i+1,j} + \ell(U_{i,j-1} + U_{i,j+1}) + b_i$$

fait intervenir les 4 variables aux noeuds voisins, plus le terme indépendant b_i.

- sur les frontières sud, est et ouest, exception faite des quatre sommets du rectangle, on a les équations :

à l'est : $U_{ij} = C\, U_{i-1,j} + \ell(U_{i,j-1} + U_{i,j+1}) + b_i$

à l'ouest : $U_{ij} = d\, U_{i+1,j} + \ell(U_{i,j-1} + U_{i,j+1}) + b_i$

au sud : $U_{ij} = c\, U_{i-1,j} + d\, U_{i+1,j} + \ell\, U_{i,j+1} + b_i$

qui font intervenir 3 variables voisines

- aux deux coins inférieurs, on a :

à gauche : $U_{ij} = \ell\, U_{i+1,j} + d\, U_{i+1,j} + b_i$

à droite : $U_{ij} = \ell\, U_{i+1,j} + c\, U_{i-1,j} + b_i$

- Enfin, sur la frontière supérieure en raison de la relation de symétrie, on a :

$$U_{ij} = U_{i-2,j}$$

Nous avons adopté, pour les temps d'exécution, les données suivantes :

addition : 10 µs

multiplication : 20 µs

temps moyen de transfert d'une variable : 5µs

Ce qui nous a conduit à envisager les temps d'exécution de chacun des groupes d'équations évoqués, considérés comme fixes :

160 µs, 130 µs, 90 µs, 5 µs respectivement.

Ces valeurs sont stockées dans le tableau TAU.

Dans les schémas des programmes réalisés, les variables indicées U et Z sont des vecteurs obtenus à partir d'une numérotation continue des noeuds du domaine, colonne par colonne, partant du coin inférieur gauche pour aboutir au coin supérieur droit (voir annexe A).

Algorithme de GAUSS-SEIDEL pour monoprocesseur :

```
Début
  lecture des données : A, SM, TAU, N
  initialisation : CT = 0  U = {0} Z = {1}
  TANT QUE (Max (Z_i) > ε) Faire
                 i
      I ← 1
      TANT QUE (I ≤ N) faire
          W ← expression de U(I)
          Z(I) ← |W-U(I)|
          U(I) ← W
          CT ← CT + 1
          I ← I + 1
      fait
      calcul de Max (Z_i)
                  i
  fait
  imprimer U
  imprimer T = CT x  Σ   TAU(I)
                   i=1
Fin
```

Le programme de simulation proprement dit se compose de 2 procédures. La première, nommée MIMD, décrit le déroulement général de l'algorithme, tandis que la seconde, appelée PROC, simule le fonctionnement d'un des processeurs de la machine virtuelle.

MIMD

```
DEBUT
  lecture des données : A, SM, TAU, IND, R
  initialisations : U = {0}, B = {1}, T1 = {0}, Z = {1}, CMT = {0}, T = {0}
  Tant que (Max (Z_i) > ε) faire
                 i
      L ← 1
      Tant que (L ≤ R) faire
          PROC (L)
          L ← L + 1
      fait
      calcul de Max (Z_i)
                    i
  fait
  imprimer U, T, CMT, Σ CMT(I)
                      I
FIN
```

PROC (K)

```
DEBUT
  Si (B(K) = 1) alors
      . calculer l'indice I1 d'un nouveau point à partir
        des indices de sous domaine Ω_k contenus dans IND
      . calculer les indices J des points voisins
      B(K) ← 0
      W(K) ← nouvelle valeur de U(I)
      Z(I1) ← |W(K) - U(I1)|
  FSI
  T1(K) ← T1(K) + DEL
  Si (T1(K) < TAU (T1)) alors
      B(K) ← 1
      T1(K) ← 0
      U(I1) ← W(K)
      CMT(I1) ← CMT(I1) + 2
  FSI
FIN
```

Les listings correspondants sont donnés à l'annexe A.

1.4. RESULTATS

Un très grand nombre de mesures a été obtenu, desquelles sont issus les résultats suivants. Il s'est tout de suite avéré que, dans ce type de problèmes, la façon de découper le domaine de calcul en sous domaines présente une grande importance. Nous avons donc étudié l'influence des charges de travail d'un cycle de calcul des processeurs sur le résultat global. Puis nous avons essayé de déterminer l'influence d'un fonctionnement discordant des processeurs en sur-relaxation. Enfin, nous avons examiné l'influence d'une méthode asynchrone dans la programmation d'un problème instationnaire (équation de la chaleur).

1.4.1. Influence du facteur de charge-effets de bord

Nous appelons facteur de charge la somme des temps d'exécution des équations liées aux noeuds du sous-domaine d'un processeur k. Autrement dit, compte tenu des notations du programme, ce paramètre est égal à :

$$F_k = \sum_i \text{TAU (IND(K, J))}$$

Nous avons été amené, parmi toutes les configurations possibles à en relever 3 principales, que nous avons appelées respectivement "verticales", "mixtes" et "horizontales". Ces 3 types de répartition sont illustrées à la figure 1.

Sur un domaine de 75 points (5 x 15), ces répartitions conduisent aux facteurs de charge suivants, avec 5 processeurs (l'unité étant égale à 10^{-6}s). La dernière colonne du tableau précise le nombre total de relaxations effectuées.

distribution	P1	P2	P3	P4	P5	Moyenne	T5	total des relaxations
verticale	1565	1815	1815	1815	1565		19665	860
mixte	1565	2250	1250	945	1565	1715	22480	879
horizontale	2050	2150	2150	2150	75		20005	5705

Dans le cas d'un monoprocesseur, on a touvé un temps d'exécution de 93325 µs en 825 relaxaitons. Il convient d'ajouter que le facteur de précision adopté a été pris égal à $\varepsilon = 0.05$ seulement.

Les conclusions qu'inspirent ce tableau apparaissent plus clairement sur la figure 2 ou l'on a représenté la variation du temps de résolution T5 en fonction de la variance du facteur de charge. Ce temps est d'autant plus faible que l'écart est petit, ce qui s'interprète par le fait que certains processeurs ont été amenés à effectuer des calculs redondants du fait des différences trop marquées des charges de travail.

Le tableau ci-dessous donne, pour les mêmes configurations, les valeurs des différentes mesures définies au chapitre I.

découpage	Parallélisme S5	Efficacité E5	Efficience F5
vertical	4.74	0.948	2410^{-5}
mixte	4.15	0.83	1810^{-5}
horizontal	3.73	0.74	1510^{-5}

Sur un aussi petit domaine, on peut objecter l'influence des effets de bord et penser que le phénomène décrit disparait dès que les deux dimensions du maillage rectangulaire s'accroissent. Une nouvelle série de mesures sur des domaines de 410 points (10x41) et 854 points (14x61) montre qu'il n'en est rien et que les variations de temps restent sensibles (de l'ordre de 15-20%) pour des choix du nombre de processeurs en rapport. La durée prohibitive d'exécution du programme de simulation nous a empêché de poursuivre ces mesures sur des domaines de plus de 1000 points, mais il est probable, et surtout pour des problèmes décrits par des équations complexes, que cet effet de charge se ferait encore sentir.

Nous en déduisons que le fonctionnement optimal des algorithmes chaotiques se produit lorsque tous les processeurs ont des charges de calcul identique, ce qui survient dans le cas du fonctionnement périodique. Dans la pratique, il conviendra d'approcher le mieux possible une telle configuration. La figure 3 précise la variation des temps de résolution pour

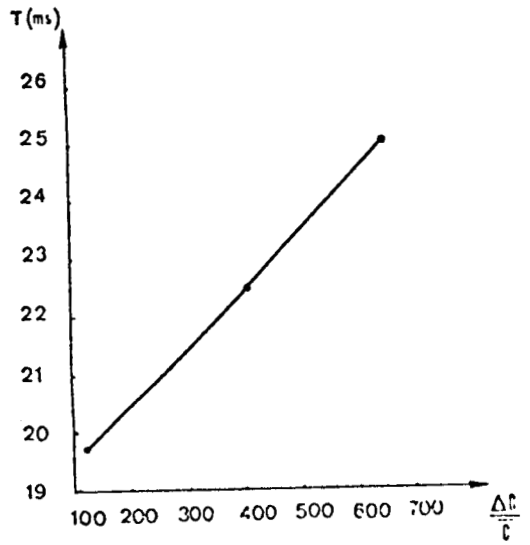


Fig. 2

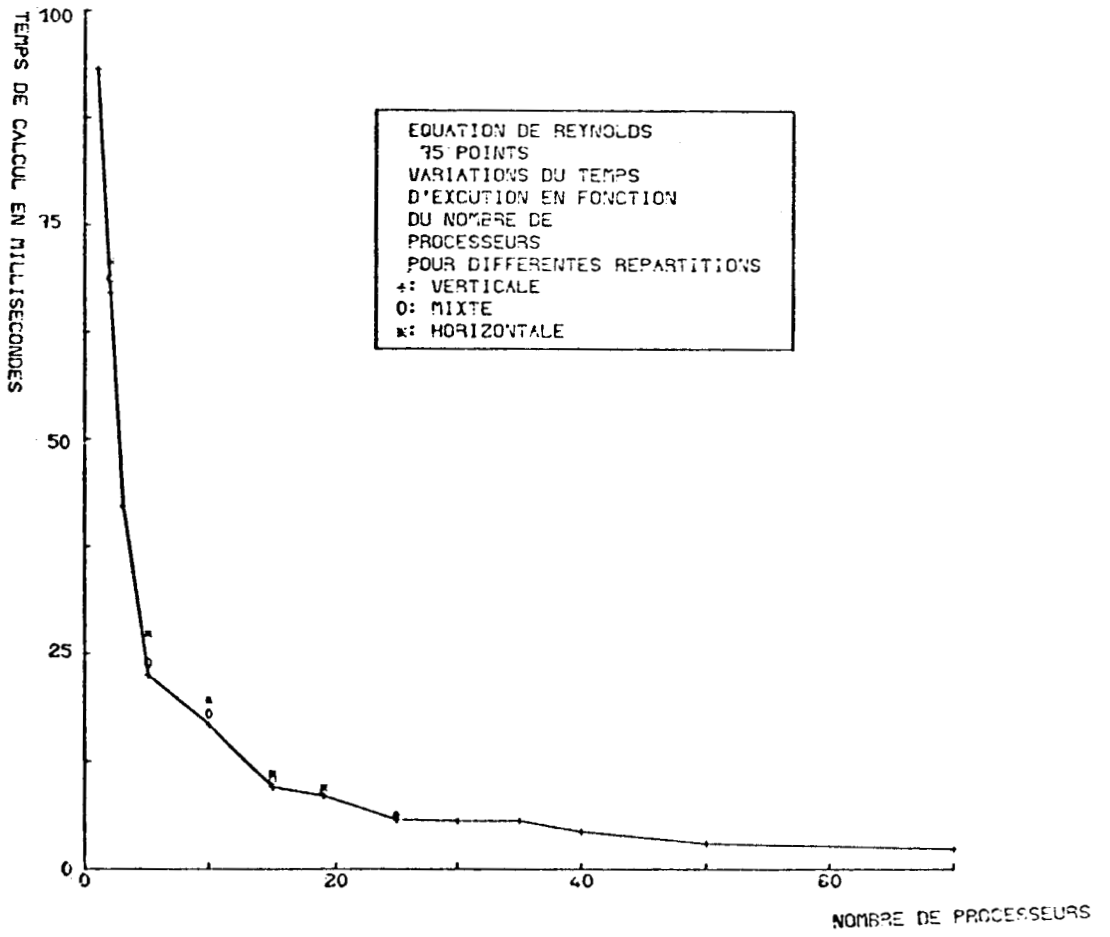


Fig. 3

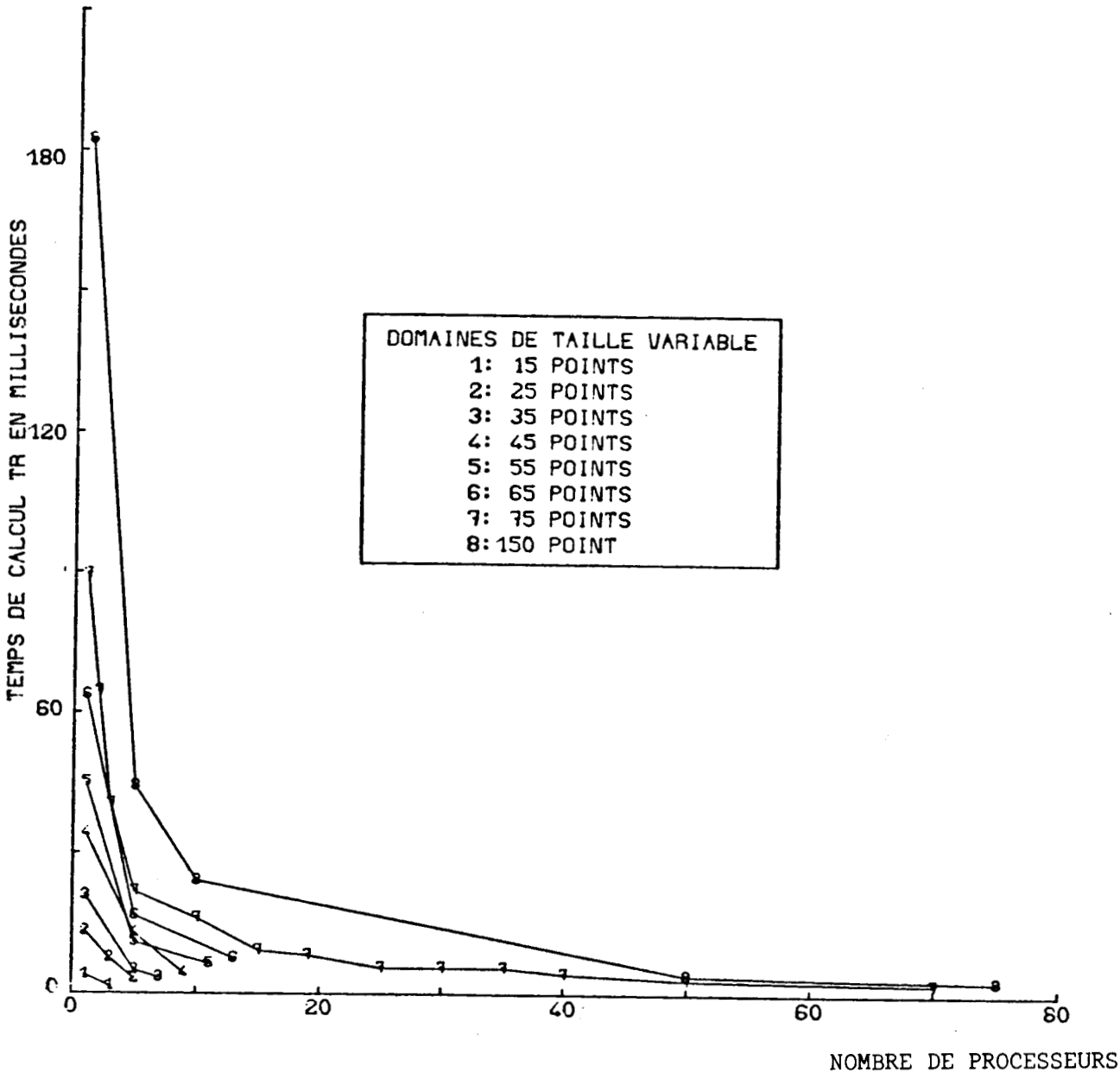


Fig. 4

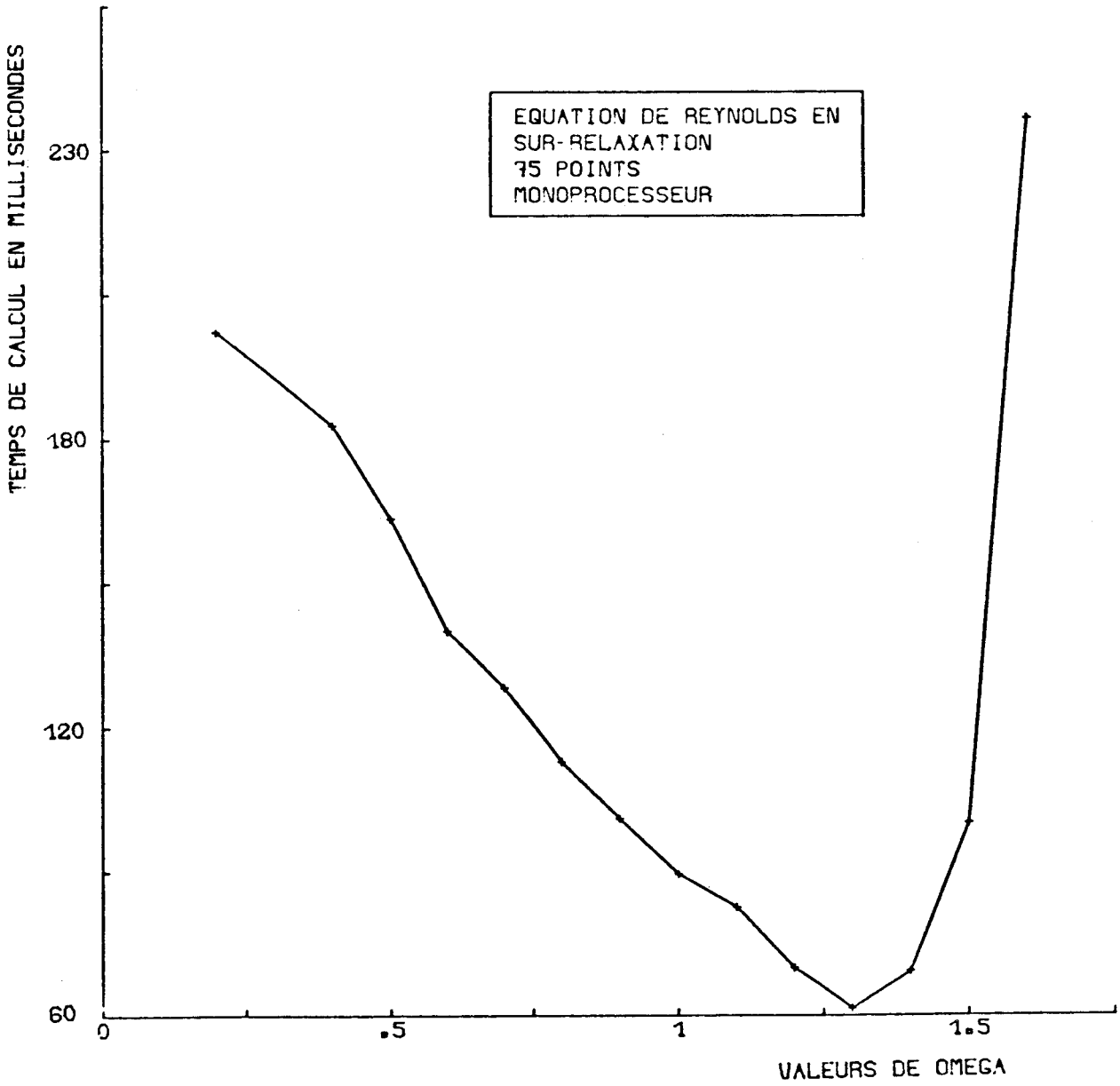


Fig. 5

un nombre croissant de processeurs, principalement pour la configuration dite "verticale", ainsi que quelques valeurs de temps correspondant aux autres configurations.

Dans tout ce qui suit, nous nous situerons toujours dans la première, pour laquelle, avec le domaine de 75 points, les écarts de charge avec la valeur moyenne ne dépassent pas 150 µs quand on utilise 5 processeurs.

Dans le même ordre d'idées, nous avons voulu étudier l'influence d'une augmentation régulière (linéaire) de la taille du même problème. Ceci a été réalisé avec des domaines de longueur croissante, de 2 en 2 colonnes, soit 10 points supplémentaires à chaque fois, de façon à travailler sur des ensembles allant de 15 à 150 points. L'influence des effets de bord se traduit par le fait que, pour une charge donnée, (exprimée ici en terme de nombre de points par processeur), les temps de résolution se situent sur une courbe inclinée le long de l'axe horizontal (figure 4).

1.4.2. *Sur-relaxation*

Une série de simulations en sur-relaxation a été réalisée sur le domaine de 75 points. Pour un monoprocesseur, il y a bien convergence pour $0 < w < 2$, avec un minimum prononcé du temps de résolution, obtenu pour la valeur : $w_o = 1.31$ (figure 5).

Avec 5 processeurs, on retrouve un résultat semblable à celui présenté par Rosenfeld : la variation du temps de résolution en fonction de w subit un minimum, avec toutefois une plage comprise entre 1.28 et 1.35. On remarque cependant que, dans cette étude, la valeur du paramètre optimum w_o est identique, en mono et en multiprocesseur, ce qui n'est pas le cas pour le problème du réseau électrique (cf chapitre 1).

L'étude a porté sur les effets d'une variation de w parmi les processeurs. Il est difficile de dégager un résultat cohérent de l'ensemble des mesures effectuées. La figure 6 présente les variations du temps de résolution en fonction du paramètre w du processeur P3 (tous les autres fonctionnant avec le même), dont le sous-domaine occupe une position centrale

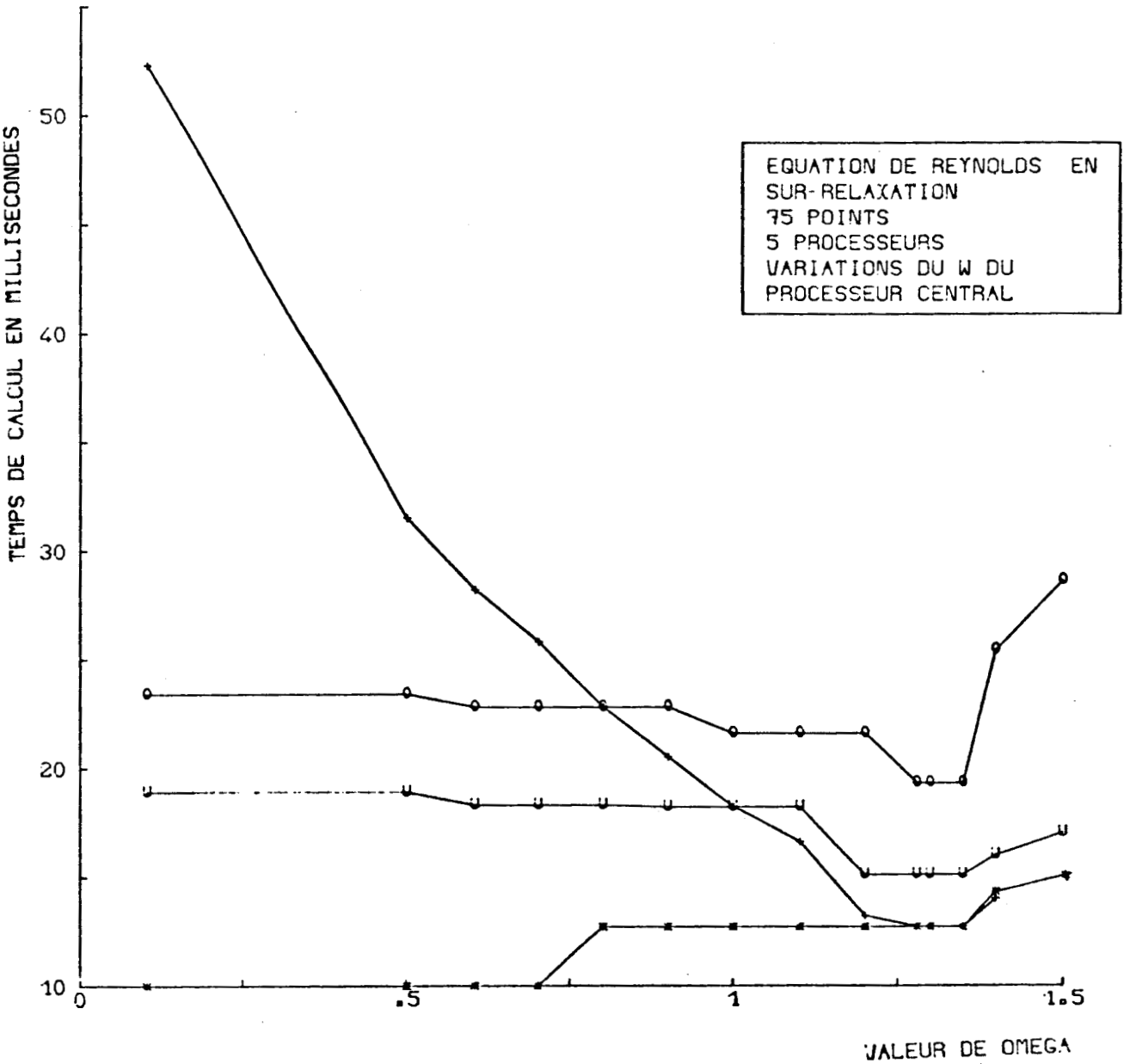


Fig. 6

Tous les autres processeurs travaillent avec la même valeur de w

0 : 8

U : 1

* : 1.3

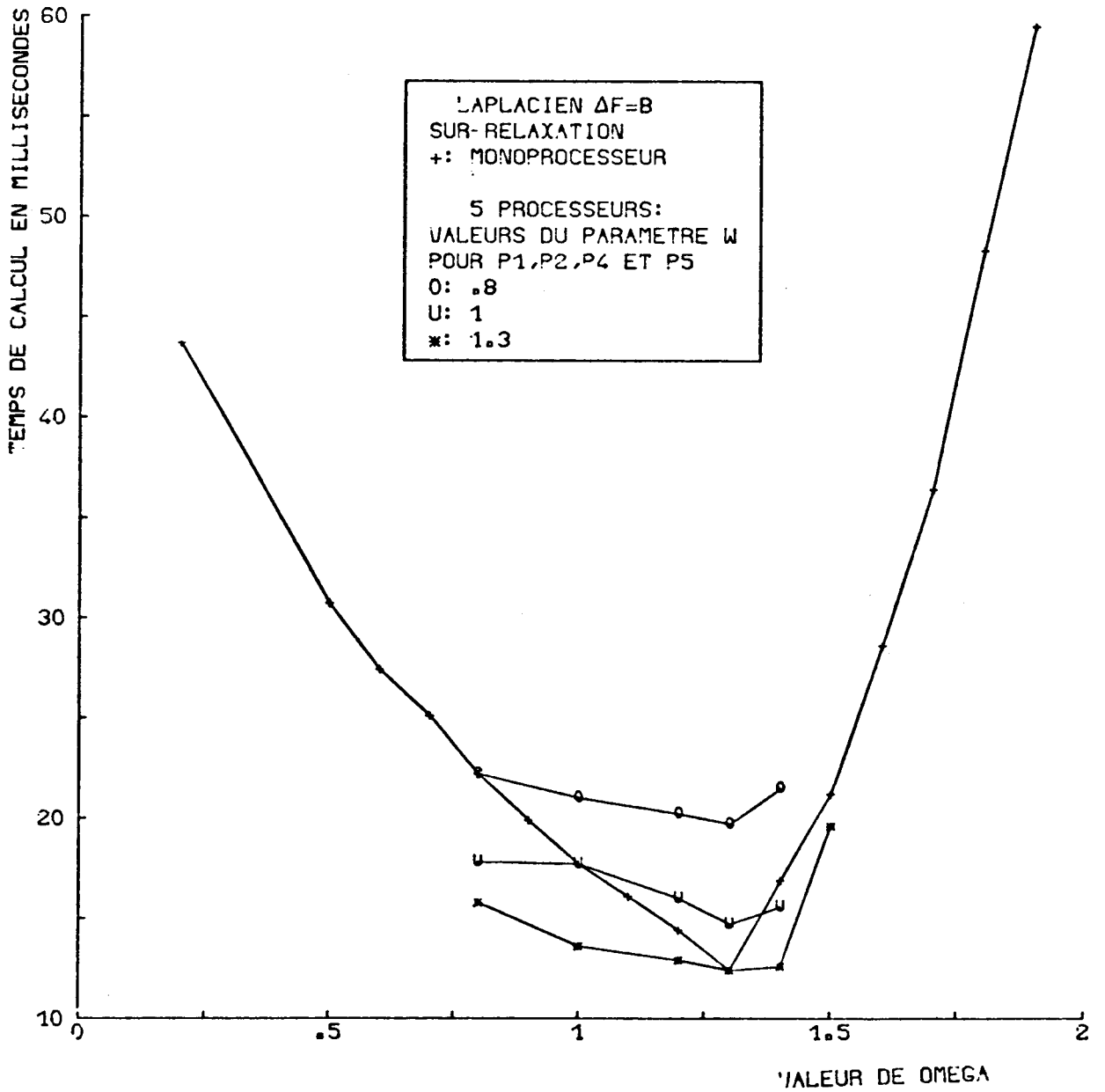


Fig. 7

R	w=1	1.1	1.2	1.3	1.4	1.5
1	825	753	646	574	682	789
2	801	746	634	568	616	734
3	818	746	640	569	676	
4	843	771	661	588	698	
5	860	784	673	598	711	
7	861	786	674	599	711	
10	888	811	695	618		
12	885	808	693	616		
15	901	822	706	627		

Fig. 8.1

R	w=1	1.1	1.2	1.3	1.4	1.5
1	825	753	646	574	682	789
2	1165	1063	908	800	979	
3	1473	1339	1150	1014		
4	3025	2757	2347	2090		
5	5705	5186	4467	3955		
6	5912	5396	4579	4034		
7	6708	6098	5252			
8	6833	6236	6218			
9	7012	6527	6484			

Fig. 8.2.

avec la distribution choisie. Il apparait que le fonctionnement en sous-relaxation de ce processeur contribue à accélérer la convergence. De plus, on peut faire reculer la valeur limite de w provoquant la divergence en donnant à w_3 une valeur faible. Cet effet est d'autant plus prononcé que la zône d'influence des points du sous-domaine correspondant est étendu.

Cependant, la même étude réalisée pour vérification, avec un opérateur Laplacien pur, conduit à un résultat tout à fait différent et plus conforme à ce que nous en attendions intuitivement (figure 7). Nous estimons que le résultat précédent est principalement dû au fait que l'opérateur a pour représentation une matrice dont les éléments sont de signes variables, sans pour autant pouvoir en apporter une justification théorique.

Enfin, nous avons étudié la variation des zones de convergence et de divergence en fonction du nombre de processeurs. Les résultats que nous présentons font apparaître une instabilité moindre que celle intervenant dans l'étude de Rosenfeld, du fait des contraintes du programme qu'il utilise. Nous pensons que les variations observées de la frontière entre les 2 domaines est encore une fois une conséquence de disparités de charge de travail pour les processeurs. Quand on accroit encore cette disparité en modifiant la répartition, les variations observées sont encore plus fortes (figure 8.2.).

1.4.3. Autres expérimentations

Nous avons repris l'ensemble de ces mesures à partir d'une méthode instationnaire, en utilisant un schéma explicite centré qui s'apparente à celui de Richtmayer. Le problème traité a été celui de la diffusion de la chaleur dans un corps rectangulaire, de hauteur infinie dont 2 faces opposées de la largeur sont supposées être à des températures constantes θ_2 et θ_1. L'équation de diffusion ($K\Delta\theta = \frac{\partial\theta}{\partial t}$, où K est le coefficient de conductabilité thermique du corps), outre l'intervention de la variable temps, est représenté par un opérateur beaucoup plus régulier que celui de l'équation de Reynolds. Les calculs ont été menés avec des pas de 0.1 secondes, sur un domaine de 80 points (8x10).

Mis à part l'introduction d'une synchronisation périodique, nécessaire entre chaque pas Δt de calcul, les résultats ont été globalement identiques à ceux qui viennent d'être présentés. Les gains de temps réalisés à chaque pas de calcul par rapport à une méthode synchrone, s'additionnent pour présenter au bout de quelques pas, un écart notable qui justifie pleinement le choix de l'algorithme.

1.4.4. *Conclusion*

Les résultats que nous venons de présenter développent ceux qui ont été fournis auparavant par BAUDET et ROSENFELD, et précisent l'intérêt des méthodes de calcul asynchrone. Ils mettent en évidence certains problèmes susceptibles de se poser lors de la mise en oeuvre d'un problème sur une machine réelle.

- Quand bien même la simulation s'est réalisée dans le cadre d'une architecture virtuelle, non contraignante, nous pouvons, dans l'optique d'une mise en oeuvre effective, affirmer que, pour un problème de taille donnée, il existe un nombre optimum de processeurs qui conviennent à sa résolution. L'étude de la variation du temps de calcul en fonction du nombre de processeurs (figure 3) montre qu'au delà d'un certain point, le gain de temps réalisé devient trop faible par rapport au coût représenté par l'adjonction de processeurs supplémentaires au processus de calcul. Ce coût, outre sa composante matérielle (en prix et en complexité) est aussi un coût en performance, puisque le volume de communications à prévoir, et les contraintes du système, augmentent nécessairement.

- Nous pensons avoir montré d'une façon générale que l'algorithme chaotique optimum à mettre en oeuvre dans des problèmes de relaxation est un algorithme périodique. Plus on s'en écarte, et plus les performances chûtent dans des proportions parfois considérables, ce qui se mesure bien par l'observation du paramètre d'efficacité. Cette remarque implique que, pour optimiser les résolutions, il faille faire précéder l'algorithme par une procédure, probablement dynamique, de découpage et de répartition de sous-domaines à un nombre optimum de processeurs, calculé selon la taille du domaine et la complexité du problème. Une telle procédure, coûteuse en temps, peut fort bien ne pas pouvoir tenir compte de toutes les contraintes de programmation et nécessiter une intervention humaine (préparation manuelle, analogue à la préparation d'un maillage pour un algorithme de discrétisation automatique). Il nous semble que c'est là un phénomène peu

connu dans la mise en oeuvre de méthodes chaotiques.

- Nous avons mis en évidence la moins grande sensibilité de ces algorithmes au facteur de sur-relaxation, ce qui autorise une recherche moins précise, donc plus rapide, du paramètre w_o. Un autre point intéressant mais qu'il n'est pas possible de généraliser et de préciser, est la possibilité dans certains cas, d'accélérer encore le calcul en introduisant une relative hétérogénéité au niveau des valeurs de ce paramètre, pour tous les processeurs.

II - MODÉLISATION DE L'ALGORITHME

2.1. INTRODUCTION

Tout monoprocesseur, pourvu d'une mémoire suffisante, est capable de résoudre des problèmes semblables à ceux qui viennent d'être présentés. L'appel à des architectures multiprocesseurs ne se justifie que dans la mesure où elles apportent des réponses satisfaisantes en temps de calcul. C'est le cas, par exemple, de la prévision météorologique. On n'acceptera de se doter d'un outil capable de faire une prévision à 24 heures que s'il peut la faire en moins de 24 heures. A la limite, on consentira à payer un prix plus élevé pour une machine plus puissante encore, si elle permet de faire cette prévision en seulement quelques heures, de façon à disposer du temps nécessaire pour prendre en compte tous les paramètres qui découlent du résultat.

Malgré le volume extrêmement élevé des données à traiter, il existe probablement, en terme de rentabilité de calcul, un nombre optimum de processeurs à mettre en oeuvre lors de l'exécution d'un calcul parallèle. Même en faisant abstraction des problèmes posés par leur coordination, il ne sert probablement à rien de faire travailler 10000 processeurs sur un problème comportant 10000 variables. Un résultat peut être obtenu dans un délai raisonnable avec seulement 10 à 100 fois moins d'éléments de calcul.

Précisons :

Soit $P_1(A)$ la puissance de calcul maximum, mesurée en MFLOPS (millions d'opérations en virgule flottante par seconde), d'un monoprocesseur

classique, pour une année A. $P_1(A)$ évolue au rythme des progrès technologiques. En 1980 sa valeur est de l'ordre de 10 MFLOPS. Pour obtenir des performances supérieures, il faut grouper R processeurs de performances moyennes ou même modestes, définies par $P_2(A)$.

Dans un contexte multiprocesseur, chaque processeur est affecté d'une perte de performance θ due aux conflits, aux communications etc ...

Le gain en performances est, dans ces conditions :

$$G = R \frac{(P2 - \theta)}{P1}$$

On désire naturellement que G soit très supérieur à 1. Or θ est une fonction de R, globalement croissante avec R, ce qui tend à diminuer la valeur de G. Même, lorsque l'on néglige conflits et communications, ce qui était le cas lors de la simulation précédente, le gain observé tend à devenir stationnaire (figure 3), ce qui montre que, pour $R \neq 1$, θ ne peut être nul.

Il nous parait intéressant d'analyser ce phénomène et d'établir un modèle qui permette d'évaluer les performances que l'on peut espérer atteindre.

2.2. MODELE SANS CONTENTION DE MEMOIRE

Ce modèle a pour fonction de formaliser les résultats obtenus lors de la simulation précédente. Il en utilise donc toutes les hypothèses, à savoir :

- L'architecture n'influe pas sur le déroulement de l'algorithme Ceci veut dire qu'on se place dans des conditions telles qu'aucun conflit n'est à envisager à quelque niveau que ce soit. Ce serait le cas pour un réseau d'ordinateurs connectés, à l'image du maillage, possédant chacun en propre le programme et les données du problème, et n'échangeant avec leurs quelques voisins directs que les variables qu'ils se partagent, sans que le délai de transmission ne se fasse sentir.

- Les phases d'initialisation et de fin de calcul réclament le même temps pour chaque processeur. Le temps T_P, égal au cas monoprocesseur,

sera négligé ici ; on ne se préoccupe donc que de la phase itérative elle-même.

- En chaque noeud, le temps que demande l'exécution d'une relaxation dépend :

* de la nature de l'équation, c'est à dire du problème initial et de sa discrétisation.

* du nombre de noeuds considérés comme voisins dans ce schéma (4 ici).

La méthode de relaxation chaotique est équivalente, quand R = 1, à la méthode de Gauss-Seidel pour monoprocesseur.

Soit I le nombre d'itérations effectuées jusqu'à l'obtention d'une précision minimum donnée, et T_e le temps nécessaire pour effectuer une itération pour une technologie et une architecture donnée. Le temps de la résolution, T_1, du monoprocesseur est :

$$T_1 = I \quad T_e$$

quand on tient compte de la seconde hypothèse.
Le domaine de calcul Ω est divisé en R domaines $\Omega_k (k \in [1, R])$. Chaque domaine comporte n_1 points, et on suppose que $n_1 = \frac{N}{R}$ est un nombre entier
Le temps nécessaire pour effectuer une itération sur les n_1 points de Ω_k est ici égal à t_k. Quant au nombre d'itérations nécessaires pour atteindre la même précision que précédemment, il prend une valuer fractionnaire égale à C_k.

Le temps de résolution est :

$$T_R = C_k \; t_k, \; \forall \; k \in [1, R]$$

Compte tenu des hypothèses exprimées plus haut, on peut dire que :

$$t_k = d_k \; \frac{T_e}{R}$$

où dk est un coefficient variable, mais tel que $\sum_{k=1}^{R} dk = 1$.

Il dépend de paramètres difficiles à cerner tel que : forme des découpages des sous-domaines Ω_k, spécificité du maillage et de l'opérateur. Si $\bar{d}_k$ indique la valeur moyenne de d_k, on peut écrire :

$$I = \bar{C}_k . \bar{d}_k$$

En réalité, on observe l'apparition, dans le cas de relaxations chaotiques non périodiques, d'une redondance de calculs qui provient du fait qu'en certains noeuds, la même variable puisse être relaxée deux ou plusieurs fois, sans que sa valeur ne soit modifiée. C'est le cas lorsqu'une grande disparité de charge de travail existe entre deux processeurs se partagant une frontière commune. L'un peut faire plusieurs itérations dans le temps ou l'autre n'en fait qu'une. La perte d'efficacité qui en résulte se traduit par une variation locale du paramètre c_k. Si g(R) est une fonction de redondance de calcul, nulle pour R = 1, ou, dans le cas périodique, pour toute valeur de R, on peut écrire :

$$N_R = \sum_{k=1} n_1 C_k$$

où N_R est le nombre total de relaxations effectuées (paramètre égal à $\sum$ CMT dans les simulations).
en posant : $N_R = N.I.\ (1 + g(R))$, on en déduit:

$$\sum_{k=1}^{R} C_k = \frac{R}{N}.N.I\ (1 + g(R))$$

avec $R\ \bar{d}_k\ \bar{C}_k = \sum_{k=1}^{R} C_k$, on obtient $\bar{C}_k = \frac{I}{\bar{d}_k}\ (1 + g(R))$

Les résultats obtenus lors de la simulation précédente, montrent que g(R) est une fonction linéaire, de pente très faible, que l'on peut mettre sous la forme :

$$g(R) = \alpha(R-1)$$

d'où :

$$\bar{C}_k = \frac{I}{\bar{d}_k}\ (1 + \alpha(R-1)) \text{ avec } \alpha \text{ petit}$$

Dans ces conditions, le temps de résolution peut se mettre sous la forme :

$$T_R = \frac{IT_e}{R}\ (1 + \alpha(R-1)) = \frac{T1}{R}\ (1 + \alpha(R-1))$$

C'est, en fait, le terme en $\frac{\alpha(R-1)}{R}$ qui est responsable de l'écart mesuré avec la fonction en $\frac{1}{R}$ qui représente la limite idéale de l'évolution.

Le temps T_R, le parallélisme $P_R = \frac{T1}{TR} = \frac{R}{1+\alpha(R-1)}$ et l'efficacité.

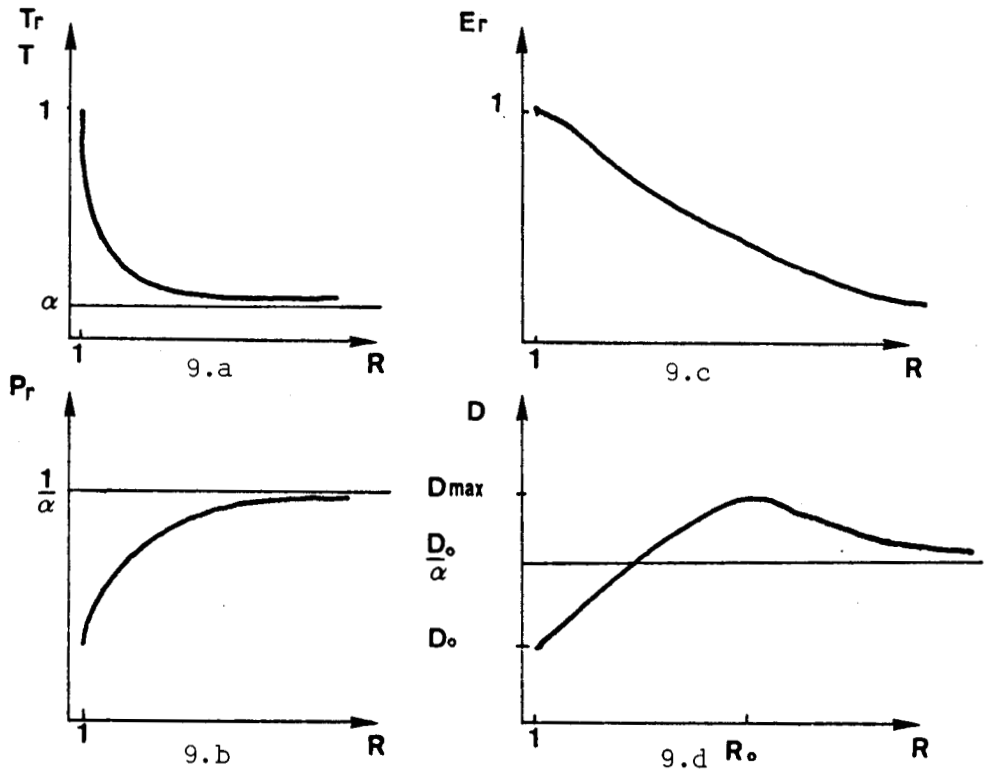


Fig. 9

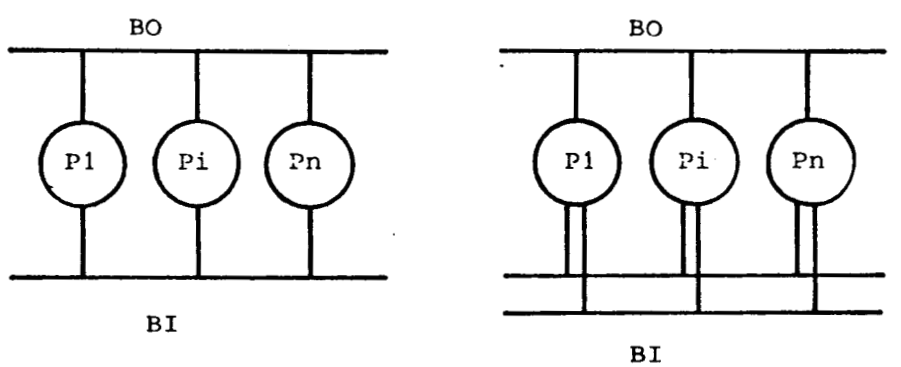


Fig. 10 et 11 : Architecture modellisée
et son extension

$ER = \frac{T1}{RTR} = \frac{1}{1+\alpha(R-1)}$ sont représentés graphiquement par les courbes de la figure 9 a-b-c.

- Evaluation de la puissance de calcul.

Soit D_1 le nombre moyen d'opérations par secondes que peut effectuer un processeur seul. Dans le temps T_1, il effectue $m = D_1 T_1$ opérations qui sont toutes utiles à la progression de la convergence.

Pour le multiprocesseur, m opérations utiles réclament $m(1+\alpha(R-1))$ opérations effectives qui sont faites dans le temps :

$$T_R = \frac{T1}{R} (1+\alpha(R-1))$$

Le débit utile de la machine est danc ce cas : $D_R = \frac{RD1}{1+\alpha(R-1)}$ au lieu du maximum théorique RD1 qui n'est normalement jamais atteint.

Pour de faibles valeurs de R, la variation est pratiquement linéaire, mais à partir d'une certaine valeur R_o, le débit utile décroît après être passé par un maximum (figure 9 d).

La valeur de α, selon nos résultats, est de l'ordre de 10^{-2} et montre que les algorithmes étudiés se prêtent bien à un emploi massif de processeurs, puisque la valeur limite $\frac{1}{\alpha}$ du parallèlisme n'est atteinte, à quelques pour-cent près, que pour une valeur élevée de R, ce qui est conforme aux résultats présentés par Baudet.

2.3. MODELE AVEC CONTENTION

Rosenfeld avait bâti ses simulations autour d'un modèle simplifié d'architecture, où des éléments de calcul étaient connectés à des bancs de mémoire indépendants par une matrice de commutation du type "Crossbar", dépourvue de délais de transmission. De ce fait, il n'avait qu'à introduire dans ses programmes un mécanisme de verrouillage interdisant à un utilisateur d'accéder à un banc de mémoire en même temps qu'un autre. Les candidats n'avaient plus alors qu'à se mettre en attente pour, au cycle suivant tester à nouveau le mécanisme. Autrement dit,le modèle

implicite était celui de la commutation de lignes, avec autant de lignes que de processeurs et de mémoires (modèle d'interconnection totale).

Le principal résultat obtenu par Rosenfeld est donné par la variation du temps total des calculs effectués par l'ensemble des processeurs. Pour un nombre donné de bancs de mémoire, cette variation, tout d'abord linéaire pour un faible nombre de processeurs, tend à s'incurver par la suite par rapport à la droite initiale (cf figure 14 du chapitre I). Grossièrement parlant, les interférences dues aux accès à la mémoire croissent aussi linéairement avec R (figure 15). Il est clair que ces résultats font intervenir globalement les deux phénomènes distincts qui nous intéressent : la redondance de calculs, inhérente à l'aspect chaotique du déroulement de l'algorithme, et les conflits qui constituent le problème majeur de toute architecture parallèle.

Un autre aspect important de ce travail est que la décision a été prise, compte tenu du faible nombre d'instructions du programme de résolution simulé (calcul d'intensités aux noeuds d'un réseau électrique), d'en effectuer des répliques stockées dans chaque module de mémoire, ce qui limite de façon importante le taux des conflits.

Quoi qu'il en soit, ces résultats conduisent dans l'analyse précédente, à introduire une fonction de contention f(R), égale à 0 pour R = 1. Alors, t_k devient $t_k(1 + f(R))$, ce qui permet d'écrire :

$$T_R = \frac{T1}{R} [1+\alpha(R-1)][1+f(R)]$$

où f(R) est de la forme :

$f(R) = \beta(R-1)$, tant que R n'est pas trop grand.

Firestone [FIR 77], analysant les travaux de Rosenfeld, propose de modéliser les conflits en utilisant la fonction

$$f(R) = \frac{q^2}{2m} (R-1)$$

où m est le nombre de bancs de mémoire, et q la probabilité qu'une référence à un banc soit faite par un processeur pour un cycle particulier. Dans ce cas, on trouve que :

$$\frac{TR}{T1} \simeq \frac{\alpha BR^2+(\alpha+\beta)R+1-(\alpha+\beta)}{R} \quad \text{avec } \alpha\beta \text{ petit}$$

De ce fait, ce résultat est tout à fait conforme à celui qui correspond à la figure 12 du chapitre I. La variation de $\frac{RTR}{T1}$, d'abord linéaire pour R faible, prend une allure progressivement parabolique. Il faut s'attendre à voir chûter les performances de façon sensible, ce qu'on observe avec les expressions développées au paragraphe précédent en remplaçant le terme 1+g(R) par le produit [1+g(R)][1+f(R)]. En particulier, le parallèlisme P_R après avoir atteint un maximum, diminue pour tendre vers 0.

Avec les valeurs q = 0.5 et m = 10, on obtient $\beta = 1.2510^{-2}$. Dans ce cas, le facteur $\alpha\beta$ est compris entre 10^{-4} et 10^{-5}, ce qui montre qu'il serait possible dans les conditions précisées, d'envisager une machine de plusieurs centaines de processeurs.

Ces résultats nous paraissent cependant criticables, car ils dépendent étroitement des travaux de Rosenfeld, déjà anciens, et limités à une architecture à la fois trop vague (simplifiée à l'extrême) et liée à des choix trop précis (modèle du "crossbar"). Ils sont inadaptés à l'étude d'une architecture de quelqu'autre type que ce soit.

Aussi nous a-t-il semblé intéressant d'évaluer cette fonction de contention dans un cadre très différent.

III - MODÉLISATION DES CONFLITS : UN EXEMPLE

3.1. INTRODUCTION

L'une des architectures les plus simples à simuler est probablement celle qui consiste à réunir les différents processeurs le long d'un bus ou d'un ensemble de bus. L'étude qui va suivre a donc été menée dans un tel contexte. Etant donné l'importance variable du volume des communications selon la nature des variables, nous avons posé le principe de la séparation des voies d'accès pour le code et pour les opérandes, ce qui va de pair avec celui de la séparation physique des mémoires de programmes et de données. L'étude pouvait être menée selon deux conceptions différentes :

- soit par une simulation fine des divers composants de l'architecture qui décrive leur état pour chaque cycle d'horloge
- soit par l'utilisation d'un modèle analytique.

Nous avons choisi la seconde possibilité. La première suppose en effet, pour fournir un résultat significatif, que le calcul soit mené sur plusieurs milliers de cycles, ce qui, pour un nombre élevé de processeurs, réclame un temps prohibitif.

3.2. PRINCIPES

L'architecture étudiée se compose :

- de R modules de calcul, dotés chacun de leur mémoire de donnée MØ et d'un commutateur SL. Ce dernier joue un rôle d'interface et assure la gestion des messages émis ou reçus. Il permet aux processeurs d'avoir un fonctionnement asynchrone.
- de NM bancs de mémoire d'instructions MI
- de bus BØ et BI, respectivement bus de données et d'instructions

Etant donné le traffic plus élevé sur le second que sur le premier, le schéma de la figure 10 a été étendu, à l'approche de la saturation de BI à celui de la figure 11.

Un modèle analytique relativement simple peut être obtenu à partir de la théorie des files d'attente. Celui que nous avons utilisé est déduit d'un modèle proposé dans Allen [ALL 80]. Nous précisons que ces files ne jouent qu'un rôle fonctionnel, et servent à évaluer les flots d'informations en circulation . Notre modèle n'impose en rien l'existence physique effective de telles files dans l'architecture que nous simulons. La régulation des accès à ses divers éléments est représentée par des files du type FIFO (first in, first out).

Afin de préciser le comportement d'un processeur, nous l'avons divisé en deux éléments fonctionnels : l'un uniquement affecté à la recherche et au traitement des opérandes, l'autre à la recherche et au décodage des instructions. En ce qui concerne ces dernières, on suppose

que le format des adresses (16 bits) est le même que celui des mots de code. Il y a alors autant de requêtes que de réponses. Nous avons admis qu'il en était de même pour les variables privées ou partagées.

Chaque processeur émettant une requête d'instruction, celle-ci est placée par le commutateur dans la file d'attente du bus BI, puis dans celle de la mémoire d'instruction. Dès que l'accès en est obtenu et le mot de code délivré, ce dernier se retrouve de nouveau dans la file d'attente de BI, et renvoyé au commutateur qui le transmet au processeur.

A tout instant, en moyenne, il y a R messages en circulation dans les divers constituants de la machine, concernant aussi bien des instructions que des opérandes. Le flux de messages peut être évalué à partir de la relation de Little qui s'écrit (voir annexe 2) :

$$R = \sum_i L_i \, W_i$$

où L_i est la valeur moyenne du taux de messages en circulation dans un élément donné.

W_i est le temps moyen passé par un message dans le composant correspondant.

Le modèle proposé est schématisé à la figure 12.

3.3. CALCUL DES TEMPS DE SERVICE

Le programme de relaxation utilisé comporte 135 instructions, correspondant, pour un processeur opérant seul, à un nombre de 2450 cycles d'exécution. Ces valeurs ont été déduites du catalogue d'un microprocesseur récent (Z8000). Au cours d'un cycle de relaxation, il y a émission de 44 requêtes d'opérandes.

Soient W5, W10 et W2, respectivement, les temps moyens d'attente d'un message (adresse ou donnée), dans le commutateur, la mémoire d'opérande (locale ou non), et le bus d'opérande (cf figure 12).

Si 90% des lectures se font, en moyenne, en mémoire locale, le temps moyen d'attente, lors de la lecture d'un mot d'opérande est :

$$W\emptyset = 0.9 \times (2W5 + W10) + 0.1 \times (2W5 + W10)$$

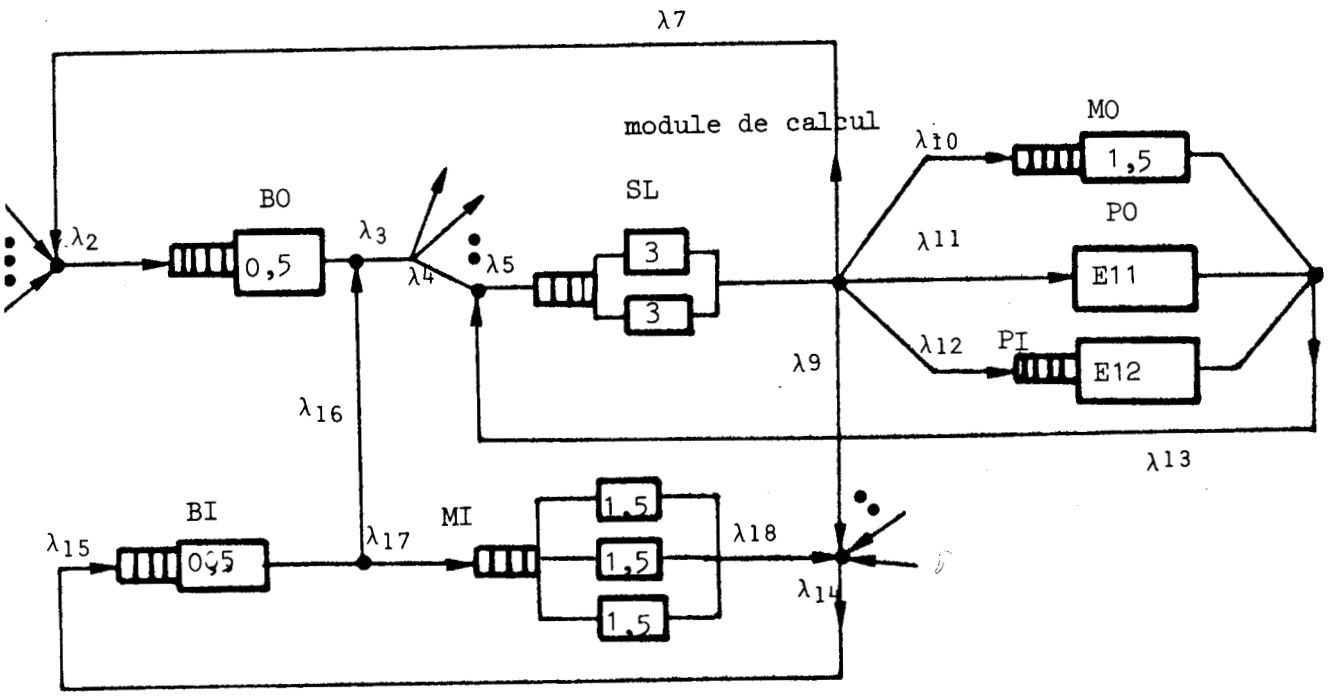


Fig. 12

Et le temps moyen d'exécution d'une instruction (c'est à dire le temps moyen séparant deux recherches d'instructions) s'écrit :

$$E12 = \frac{2450 + 44 \times W\emptyset}{135}$$

Le temps moyen séparant la réception d'un opérande à la suite d'une lecture, de l'émission de l'adresse de l'instruction varie selon le mécanisme mis en place pour la recherche des instructions. Sa valeur minimale est :

$$E11 = \frac{2450}{135}$$

Il est normal de considérer que les instructions soient entrelacées sur plusieurs bancs, afin de diminuer encore les possibilités de conflit à ce niveau. Dans ce cas, la probabilité pour tous les bancs d'être utilisés est la même, et il y a autant de serveurs que de bancs dans le modèle. En se basant sur des catalogues de mémoires, on a admis une valeur du temps de service égale à 1.5 cycles en unité de cycle-processeur. Il va de soi que ce paramètre, de même que le nombre de serveurs de la mémoire d'instruction, peut être modifié sans difficultés.

3.4. CALCUL DES PROBABILITES DE DISTRIBUTION DES MESSAGES AU SORTIR DU COMMUTATEUR

Soient :

x_1 le taux moyen de requêtes d'instructions

x_2 le taux moyen de requêtes à la mémoire locale

x_3 le taux moyen de requêtes aux mémoires extérieures

On admet que le taux de requête est égal à celui des réponses. Si l'on néglige les effets de bord, on admettra que le taux de requêtes d'opérandes émises par un processeur vers les mémoires des modules voisins est égal à celui que l'ensemble de ces modules est susceptible d'emettre vers sa propre mémoire.

Le flux total moyen traversant le commutateur est

$$D = 4x_1 + 4x_2 + 8x_3$$

Soient : P_1 la probabilité d'émission d'un message d'instructions

P_2 celle concernant un opérande destiné à la mémoire locale

P_3 celle concernant un opérande destiné à une mémoire lointaine

on a :

$$P_1 = \frac{x_1}{D} \; ; \; P_2 = \frac{x_2}{D} \; ; \; P_3 = \frac{x_3}{D}$$

Soient PI et PØ les probabilités qu'une requête concerne, respectivement une instruction et un opérande.

On a :

$$PI = \frac{x_1}{x_1+x_2+x_3} = \frac{135}{44+135} \simeq 0.75$$

$$P\text{Ø} = \frac{x_2+x_3}{x_1+x_2+x_3} = \frac{44}{179} \simeq 0.25$$

On en déduit :

$P_1 = 0.183$

$P_2 = 0.069$

$P_3 = 0.006$

La répartition des flux au sortir du commutateur est régie par les probabilités suivantes :

vers le bus opérande : 2P3

vers le bus d'instruction : P1

vers la mémoire locale : P2+P3

vers la partie processeur émettant les requêtes en opérande P2+P3

vers la partie processeur émettant les requêtes en constructions : P1

Le calcul de la valeur des flux et des temps d'attente s'effectue alors sans problèmes.

3.5. RESULTATS

Les résultats obtenus sont illustrés par la figure 13. Ils représentent pour une telle architecture, la fonction de contention f(R) en fonction du nombre R de processeurs.

Plusieurs configurations ont été simulées par l'intermédiaire de ce modèle.

La courbe n° 1 correspond à une machine pourvue d'un bus d'instructions, dont la saturation est atteinte, pour les valeurs adoptées des paramètres, aux environs de 45 processeurs.

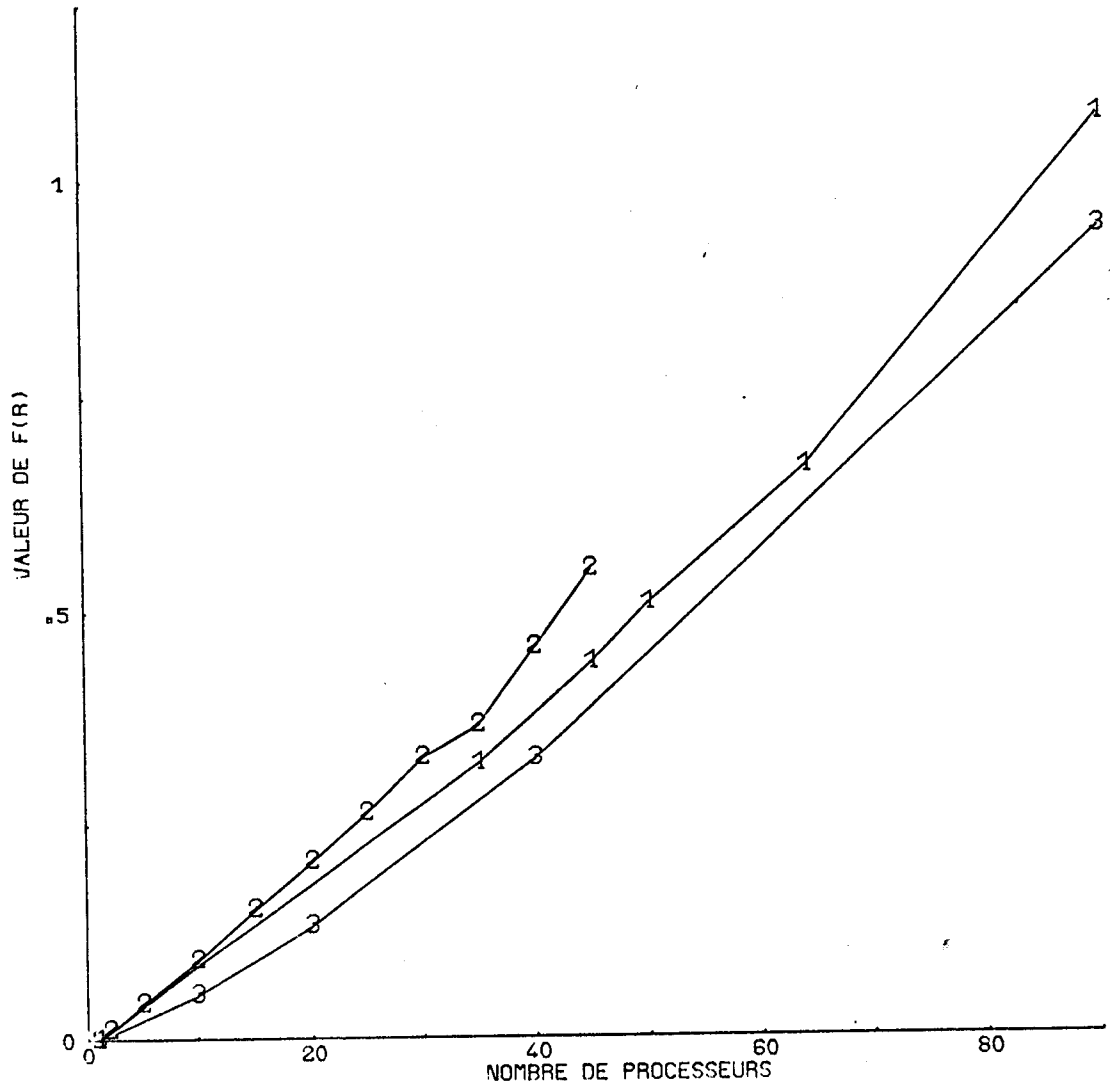


Fig. 13

Une autre configuration, comportant un deuxième bus d'instructions a donné les résultats représentés par la courbe n° 2. La saturation est alors obtenue sur ces bus, pour 90 processeurs, environ.

Enfin nous avons simulé le comportement d'une machine comportant un mécanisme d'anticipation dans la recherche des instructions (en négligeant les problèmes posés par les branchements conditionnels). Il y a alors, en moyenne, 2R messages dans les divers éléments, et la formule de Little doit être modifiée pour en tenir compte. La courbe n° 3 représente l'expression des conflits et attentes qui en découlent.

Dans tous les cas, la fonction f(R), que l'on peut considérer comme linéaire pour de faibles valeurs de R, accuse ensuite une croissance légèrement plus marquée, qui lui confert l'aspect d'une parabole.

CHAPITRE V

PROPOSITION D'ARCHITECTURE

I - INTRODUCTION

Les considérations très diverses qui résultent des chapitres précédents permettent d'aborder les problèmes de l'architecture et de proposer un certain nombre de lignes directrices.

Tout système de traitement de l'information se compose des fonctions suivantes :

- fonction de traitement
- fonction de contrôle
- fonction de stockage
- fonction de communication

Chacune de ces fonctions peut être unique et globale, ou au contraire, multiple et répartie. Dans ce cas, la répartition peut s'inspirer de deux grandes familles de modèles :

- Le modèle horizontal, où la fonction se trouve répartie dans sa totalité entre des sites disposant de pouvoirs comparables. Ceux-ci sont donc amenés à en négocier le partage, à l'aide de protocoles et d'outils de synchronisation appropriés.

- Le modèle hiérarchique, où la fonction est distribuée verticalement selon un schéma d'analyse arborescente, et peut relever, de ce fait, d'une autorité supérieure.

La multiplicité des choix que rendent possible ces différentes combinaisons explique l'extrême variété des architectures parallèles existentes. Il parait difficile de dégager, à partir des conclusions des chapitres précédents, une solution optimale qui puisse tenir compte des divers compromis à adopter. Il est plus logique, dans le contexte précis de cette étude, de tenter de mettre en évidence quelques caractéristiques précises de chacune de ces fonctions.

S'il est clair que l'utilisation d'algorithmes chaotiques simplifie le contrôle de la machine, et que les traitements ne posent pas de difficultés particulières au niveau de leur exécution, on est conduit, par contre à constater que les fonctions de communication, et, dans son aspect dynamique, de stockage, peuvent tirer avantage de solutions spécifiques.

C'est précisément à leur propos que sera développé une étude détaillée.

II - MODÈLE FONCTIONNEL

2.1. INTRODUCTION

On a vu au chapitre III que le modèle fonctionnel du schéma des communications s'inscrit dans le cadre d'un processus liant des producteurs à des consommateurs. Nous ne le décrivons que dans le contexte fort large il est vrai, d'une procédure itérative, au cours de laquelle N variables subissent des remises à jour répétées. Chacune des variables du problème est tour à tour produite, puis consommée, et l'on doit considérer qu'un producteur unique lui est attaché. Par contre, elle peut être utilisée par un nombre indéterminé de consommateurs, sa diffusion s'effectuant à partir d'une liste de routage. Deux cas sont susceptibles de se présenter :

- la liste de routage ne contient qu'une seule référence, et il n'y a donc qu'un seul consommateur qui utilise la variable produite. C'est naturellement le cas le plus fréquent, qui intéresse, rappelons-le de 80 à 90% du total des variables du problème.
- la liste contient plusieurs références ; elles sont dans ce cas en nombre réduit (le plus souvent, de l'ordre de 4 ou 5, rarement plus d'une quinzaine).

Nous devons donc envisager N producteurs et N consommateurs, qui s'échangent les N éléments d'un vecteur U de variables, selon le schéma fonctionnel suivant :

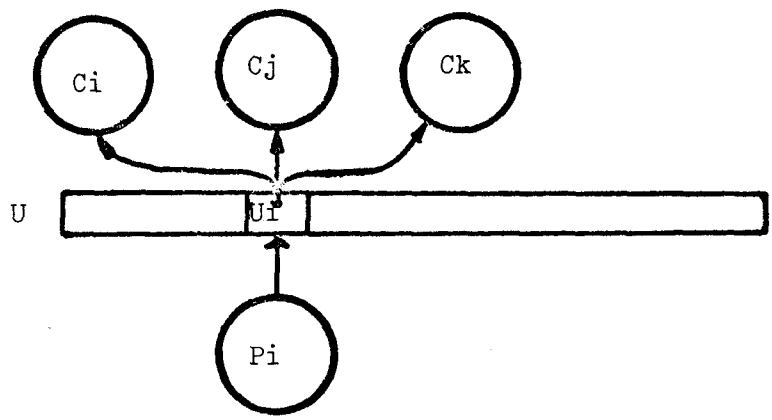


Fig. 1

Le processus d'échange est caractérisé par l'asynchronisme des algorithmes utilisés, ce qui revient à dire qu'il doit être réalisé selon un mode de commutation de messages.

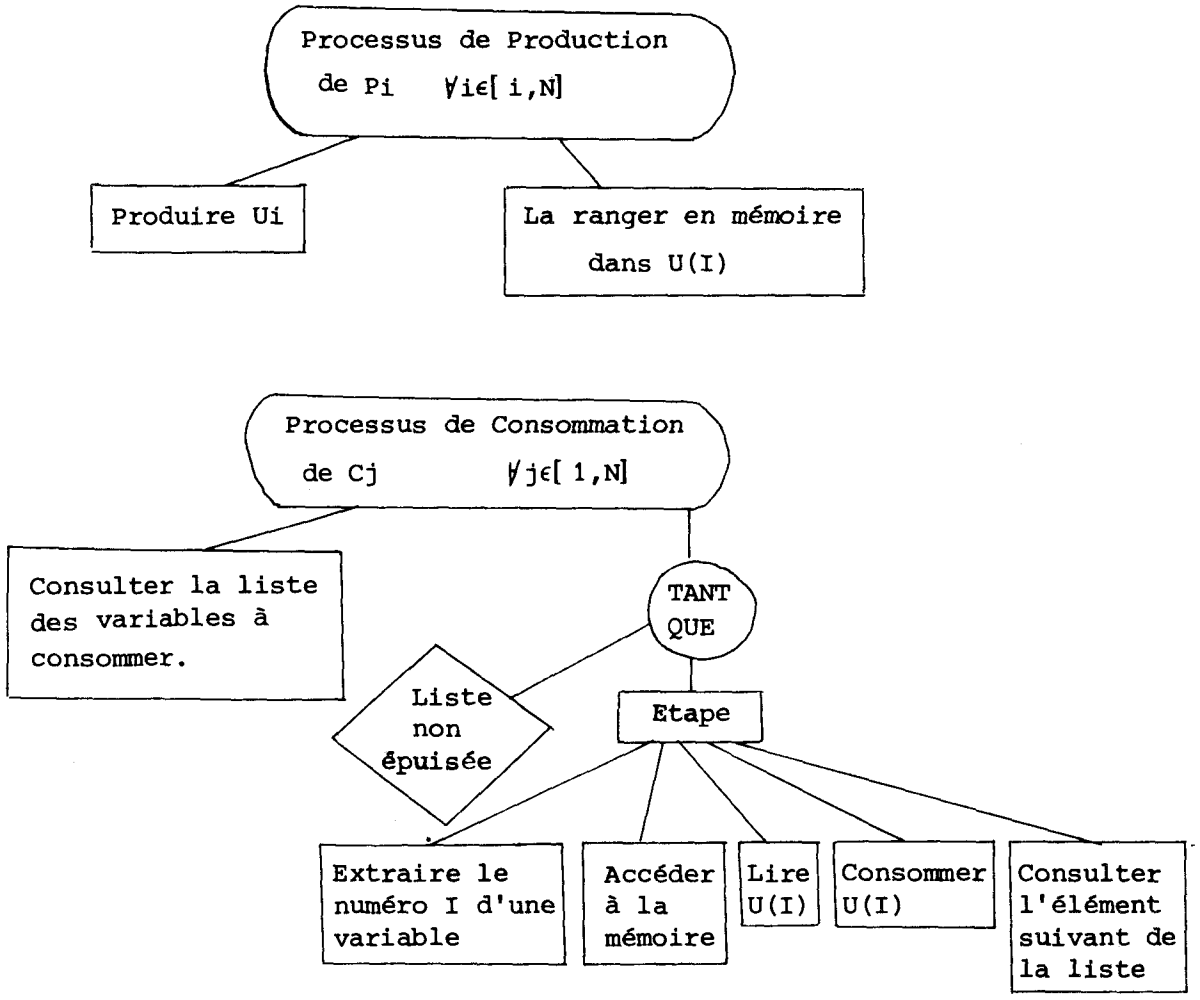
2.2. DESCRIPTION FONCTIONNELLE

Après avoir évalué la nouvelle valeur de la variable dont il est chargé, le producteur Pi se met en devoir de la communiquer au consommateur Cj auquel entre autres, elle est destinée. A cette fin, il dépose l'information qu'il détient dans un dispositif de stockage, temporaire ou permanent, qui est constitué par un registre ou une mémoire.

C'est un processus actif : Pi décide tout à la fois d'expédier le message qu'il a produit et de lui associer une adresse qu'il a élaboré. La consommation, elle, peut être active ou passive.

- elle est active si Cj vient, lorsqu'il en éprouve le besoin chercher lui-même le message là où il a été déposé, c'est à dire dans ce cas dans une mémoire (pilotage par nécessité). Nous avons vu au chapitre III que celle-ci peut être assimilée à un panneau d'affichage. Elle peut être commune à l'ensemble des producteurs et des consommateurs, mais aussi être répartie en éléments associés à quelques uns d'entre-eux, à savoir, pour chacun, un producteur et les consommateurs correspondants.

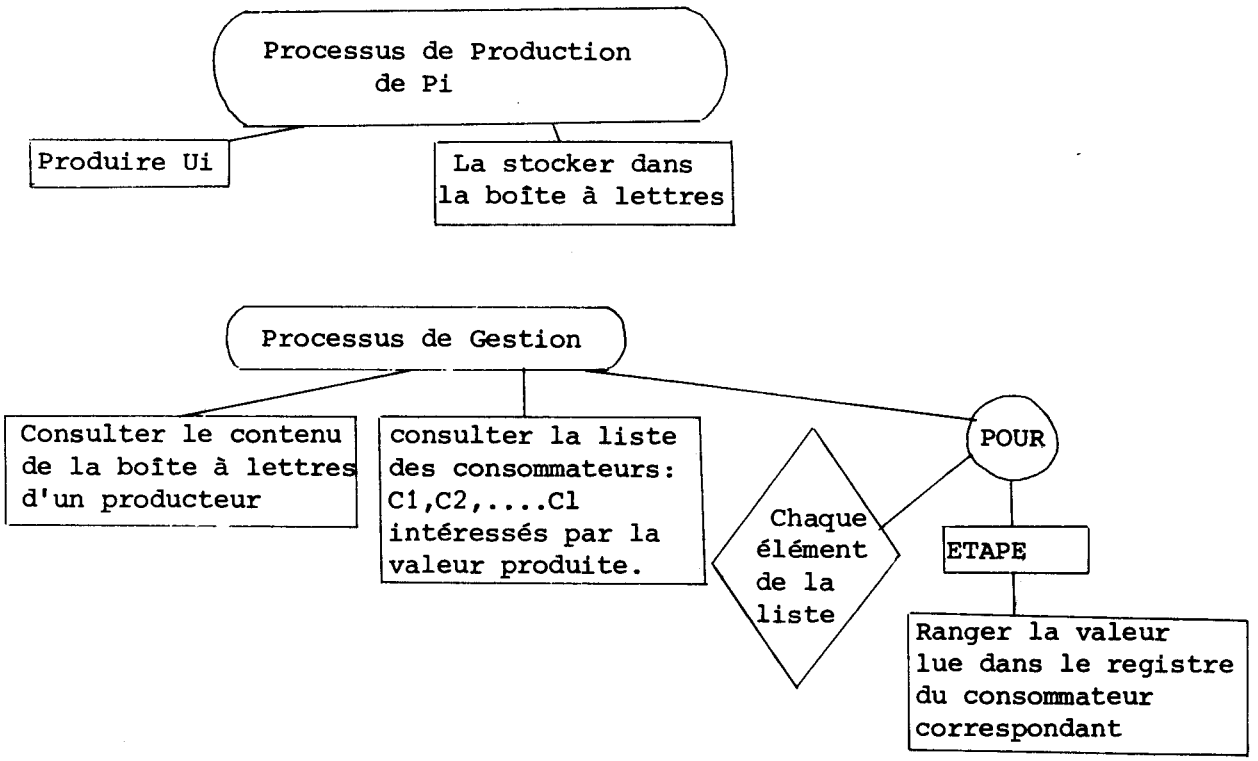
L'ensemble de ce processus peut être illustré par les algorithmes suivants :

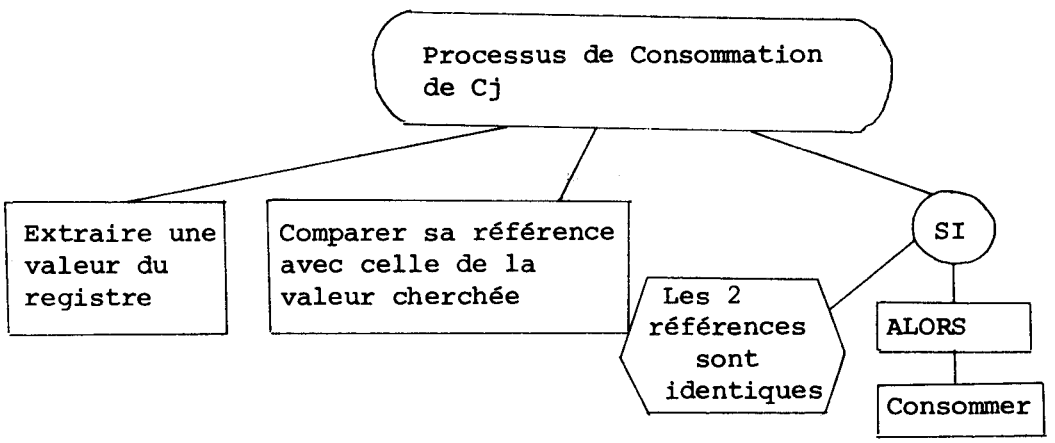


Nous avons déjà évoqué précédemment certains avantages et inconvénients inhérents à ce mode de fonctionnement. Rappelons que l'intérêt présenté par un volume de stockage réduit est contrebalancé par le temps important que doit attendre chaque consommateur désirant accéder à la mémoire.

- elle est passive si un organe de gestion dépose l'information dans une mémoire (un registre) où le consommateur Cj n'aura qu'à venir la prendre (pilotage par disponibilité). Dans ce cas, le mode de fonctionnement est celui d'une boîte aux lettres.

Le processus correspondant est décrit par les schémas de programmes suivants :


Processus de Production de Pi
Produire Ui
La stocker dans la boîte à lettres
Processus de Gestion
Consulter le contenu de la boîte à lettres d'un producteur
consulter la liste des consommateurs: C1,C2,....Cl intéressés par la valeur produite.
Chaque élément de la liste
POUR
ETAPE
Ranger la valeur lue dans le registre du consommateur correspondant


Processus de Consommation de Cj
Extraire une valeur du registre
Comparer sa référence avec celle de la valeur cherchée
Les 2 références sont identiques
SI
ALORS
Consommer

Ce point de vue permet l'exploitation d'une des caractéristiques très importantes des algorithmes chaotiques : la notion de retard à la relaxation d'une variable (cf. chapitre I). Ce concept a le corollaire suivant : La perte d'une information, lorsqu'elle est aléatoire, n'a pas de conséquences dramatiques pour la poursuite du traitement. Il autorise un phénomène d'effacement, dans la boîte à lettres, des variables qui ne seraient pas utilisées immédiatement après leur production ou leur acheminement vers un consommateur.

Ainsi, peut importe que Cj tarde à consommer le message que lui adresse Pi. Ce dernier peut même en émettre plusieurs qui s'effacent les uns les autres. Lorsque Dj est prêt à examiner le contenu de la boîte à lettres, il y trouve soit la valeur initialement produite (lorsque le processus de Pi est stationnaire), soit une valeur réactualisée de la variable.

Plusieurs avantages en découlent :

- Une plus grande simplicité du schéma des communications entre producteurs et consommateurs qui n'a pas alors besoin d'être optimisé.

- L'existence d'un système de gestion permet de lui attribuer des rôles secondaires, mais néanmoins importants. En particulier, sa base de donnée étant constituée de l'ensemble du vecteur des variables, il peut assurer de ce fait la détermination de la fin du traitement itératif, en examinant globalement l'état de progression des éléments. La stationnarité, constatée sur l'ensemble des valeurs est, dans ces conditions un critère suffisant pour arrêter la poursuite de l'algorithme.

III - IMPLÉMENTATION

3.1. HYPOTHESES

Une tentative d'implémentation du schéma précédent doit tenir compte des éléments suivants :

- Le domaine de calcul sur lequel travaille chaque processeur ne se réduit pas, en général, à une seule variable, mais à un ensemble qui

représente les valeurs de fonctions inconnues aux noeuds d'un sous-domaine. Les noeuds sont groupés de façon à constituer un espace de forme quelconque, mais que, pour des raisons de simplicité, nous supposerons le plus souvent régulier (carré ou rectangulaire dans le plan).

- Il faut bien noter que les deux processus de production et de consommation décrits sont interdépendants, et qu'un processeur ne peut produire une variable que dans la mesure où il peut en consommer d'autres. Dans ces conditions, les fonctions de production/consommation sont, deux à deux, réalisées alternativement au sein de chaque processeur.

- La majeure partie de ces processus est interne : Le même processeur est tour à tour producteur et consommateur de ses propres variables, qui sont des variables privées. Tout calcul sur les variables aux frontières du sous-domaine nécessite la connaissance de variables (une seule le plus souvent) correspondant au point en vis à vis sur la frontière la plus proche, et qui est produite par un autre processeur. Les échanges entre processeurs ne concernent alors qu'une faible partie de l'ensemble des variables du problème.

- Afin de fixer des ordres de grandeur pour les paramètres principaux intervenant au sein d'un cahier des charges, nous nous proposons de raisonner sur l'exemple suivant :

Considérons un problème, pouvant éventuellement comporter plus d'une variable par point de maillage, mais que nous limiterons de toute façon à 50000 variables. Supposons qu'il faille, en moyenne 150 relaxations par variable pour obtenir la convergence, et que chaque relaxation nécessite l'équivalent de 26 opérations en virgule flottante. Le nombre d'opérations à réaliser (Flop) est alors, dans le cas d'un mono-processeur, de l'ordre de 200 MFlop (million d'opérations en virgule flottante).

Le volume global de stockage nécessaire, si l'on tient compte des coefficients des matrices associés à chaque variable correspondant en moyenne à environ 10 objets par variable. Il atteint, rien que pour celles-ci, la valeur de 4 M Octets, en adoptant des réels sur 64 bits, et sans tenir compte des données annexes, paramètres et portions de fichiers, pour lesquels on peut chiffrer un ordre de grandeur de l'ordre de 200000 octects.

Toujours en raisonnant globalement sur l'ensemble du problème, le volume des échanges entre les processeurs porte sur quelques 40 à 80000 octets qui représentent en fait moins de 2% du volume total de la mémoire de la machine parallèle.

3.2. ETUDE DES DIFFERENTES IMPLEMENTATIONS POSSIBLES

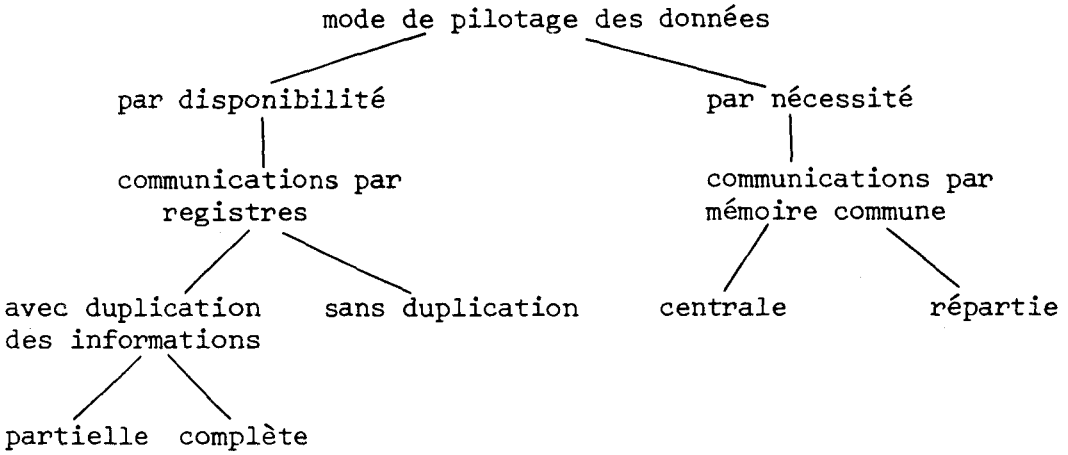
3.2.1. *Présentation*

L'étude du modèle fonctionnel précédent permet d'envisager plusieurs types possibles d'implémentations, qui paraissent plus ou moins bien répondre aux exigences du cahier des charges proposées. Pour évaluer plus précisément l'intérêt que présente chacune d'elle, on peut proposer comme critères de classement :

- le volume de stockage réclamé
- le temps d'accès à l'information
- le flux des messages dans l'outil de communication.

Toutes offrent des qualités opposées, et seul un bilan chiffré peut conduire à l'adoption d'un compromis acceptable. Le critère essentiel que nous retiendront en dernier lieu pour guider notre choix, sera celui de la puissance de calcul maximal, dans la limite où elle ne semblera pas devoir être obtenu au prix d'un coût et d'une complexité prohibitive.

Les solutions proposées peuvent être résumées par le schéma suivant :



3.2.2. *Solution à mémoire commune*

Désignons par D1, D2 et D, respectivement, les débits de la mémoire, de l'outil de communication et d'un processeur. Le nombre de ces derniers est limité par l'existence de conflits qui peuvent conduire au blocage de la machine. Dans le cas d'une mémoire centrale, il doit respecter la relation : $R \leq \frac{\min(D1,D2)}{D}$. Dans le cas d'une solution répartie, cette relation devient : $R \leq \frac{D_{D2}}{D}$, car il n'y a que quelque processeurs à se partager chaque mémoire. Or, D2 est toujours plus élevé que D1, et le taux de conflits en dehors d'un état de saturation, est alors plus faible. Il n'est cependant jamais nul et confert une valeur relativement élevé au temps d'accès à l'information partagée en mémoire.

Dans tous les cas, le flux des communications, qui se décompose en un flux de requêtes et un flux de réponses, est important, d'autant plus que la même variable peut être utilisée par plusieurs processeurs.

Nous avons présenté au chapitre précédent un modèle d'évaluation des performances d'une architecture répartie. Quoique comportant un certain nombre de spécificités, il permet de chiffrer le nombre maximum de processeurs qu'elle peut utiliser, et donne un moyen d'évaluer l'importance et l'influence des conflits sur la puissance de traitement.

Les résultats que nous avons obtenus prévoient une "perte de rendement" assez importante du fonctionnement de l'algorithme chaotique lorsque croît le nombre de processeurs.

3.2.3. *Solution à registre*

Elle correspond à un mécanisme de boite aux lettres dans laquelle chaque processeur dépose et retire ses messages. Son avantage principal est de permettre un temps d'accès à la fois bref et constant, aux informations échangées entre processeurs. Le flux de messages est de l'ordre de 2 fois le nombre de variables partagées produites. Il est très inférieur à la valeur correspondant à la solution précédente. Le taux de conflit semble donc a priori extrêment faible, et autorise de ce fait une très grande extensibilité à une machine conçue selon ce principe. Son inconvénient majeur réside dans le volume de mémoire supplémentaire exigé lorsque la

duplication est totale. Rappelons cependant qu'elle ne concerne que les variables partagées et qu'elle ne réclamme que de 1 à 2% du volume total.

Nous avons suggéré, dans l'arbre présenté plus haut, l'existence de plusieurs variantes, introduisant différents degrés de duplication des informations, avec des avantages variables pour l'efficacité de l'algorithme. Elles correspondent à divers modes d'extraction de l'information du registre où elle a été déposée. On peut en effet proposer plusieurs protocoles d'échange, se traduisant chacun par des contraintes variées, aussi bien au niveau de l'architecture que de la programmation.

3.2.4. *Etude des protocoles d'extraction*

Tel que nous l'avons rapidement évoqué, le fonctionnement du registre de la boite à lettre est le suivant :

a) Un processeur explore son registre lorsqu'il désire acquérir une variable étrangère à la structure de données qu'il gère. S'il ne l'y trouve pas, il a le choix entre deux attitudes

- il saute le calcul dont il a cependant assuré la préparation, ce qui lui a demandé un certain temps.
- il effectue le calcul en se servant d'une ancienne valeur qu'il a stocké. Nous sommes ici dans le cas d'une duplication totale.

La critique principale que l'on peut adresser à ce principe est qu'il est en opposition avec un point que nous avons dégagé dans l'analyse fonctionnelle du chapitre III, à savoir que la double écriture d'une variable partégée n'est nullement nécessaire.

Dans les deux cas, la probabilité pour que la variable attendue se trouve effectivement dans le registre lors de la lecture est extrêmement faible. Les retards à la relaxation, ou qui,du moins, sont ressentis comme tels par le processeur consommateur, peuvent devenir énormes. La puissance de calcul de la machine, même avec un nombre très raisonnable de processeurs, a toutes les chances d'être bien inférieure à celle d'un monoprocesseur équivalent.

On peut alors imaginer d'autres scénarios ; par exemple, lorsqu' après un balayage de ses frontières, un processeur ne constate aucune progression de la convergence, il se met en attente et explore en boucle

le contenu du registre. Après lecture, il peut alors adapter son traitement à la valeur qui lui aura été fournie. Quoi qu'il en soit, il n'est pas certain que l'accélératon de la convergence ainsi obtenue soit très sensible. De plus, on introduit des contraintes au niveau de la programmation, qui sont un peu trop fortement liées à l'architecture et à son système d'exploitation.

b) Une autre stratégie consiste pour un processeur, à lire le registre et à stocker ce qu'il y trouve, qu'il en ait besoin ou pas. Cette opération de lecture peut même se faire à intervalles régulièrement répartis parmi l'ensemble des instructions.

Si l'on retrouve les critiques précédentes en ce qui concerne le volume de mémoire perdu, on est dans ce cas assuré de la progression de la convergence de l'algorithme, car l'échantillonage est alors suffisant pour rentabiliser les autres calculs.

c) Dans le cas d'une duplication partielle, on peut utiliser un certain nombre de NR registres fonctionnant en mémoire associative.NR informations y sont rangées en boucle au fur et à mesure de leur production, chaque nouvelle valeur effaçant la plus ancienne. A partir d'un masque utilisant un nom de la valeur cherchée, on peut déterminer si celle-ci est disponible ou non ; on est alors ramené au premier cas en ce qui concerne la conduite des opérations à effectuer, mais on a multiplié par NR la probabilité de trouver une information donnée.

d) Il est aussi possible de multiplier les registres en les spécialisant

- par rapport aux processeurs
- par rapport à certaines variables.

Dans le premier cas, on disposerait en sortie d'un registre par processeur voisin, ce qui est coûteux, et de nature à géner considérablement toute tentative d'extension ou de reconfiguration.
Dans le second cas, on stockerait dans ces registres des variables particulières, par exemple celles qui sont nécessaires pour calculer les valeurs aux quatre coins d'un domaine plan, ou au milieu d'une frontière. Les autres variables seraient alors extraites d'un registre courant, et

exploitées selon une des stratégies précédentes.

e) une dernière stratégie nous semble cependant plus intéressante. Elle sauvegarde la simplicité initiale de l'architecture, en ne considérant qu'un registre pour la boite à lettres, sans pour autant réclamer un volume de mémoire supplémentaire. Elle devrait conduire à de bonnes performances globales, au prix, il est vrai, d'une légère contrainte au niveau de la programmation.

Elle consiste à faire lire le registre par le processeur qui en examine le contenu et calcule la valeur de la variable qui s'en déduit. Par exemple, supposons que Pi trouve la valeur U(ℓ) dans son registre. Il décide alors de l'utiliser pour calculer celle de U(k) (figure 2).

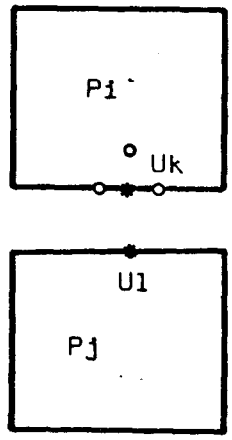


Fig. 2

On réalise ainsi une sorte de pilotage par les données, qui est cependant limité aux variables frontières intervenant seules dans le processus d'échange.

On observe que les variables concernées sont situées en vis à vis sur les frontières correspondantes de 2 domaines de calcul voisins, et qu'il en résulte un mouvement de va et vient qui, une fois amorcé, ne peut plus s'arrêter. Pour l'interrompre, il est possible d'imposer au processeur d'effectuer, alternativement avec le schéma précédent, une relaxation au niveau d'un point quelconque, situé éventuellement sur une autre frontière. Par exemple, il pourrait, par extrapolation, en générer une valeur approchée, permettant ainsi au processeur à qui elle serait transmise, d'opérer dans cette zone de données et d'en affiner la précision.

Pour être efficace, un tel processus doit tenir compte du rapport qui existe entre variables privées et partagées, dans une configuration donnée, de façon à réduire le couplage qui s'introduit dans le fonctionnement des processeurs.

3.2.5. *Choix de l'architecture*

Parvenus à ce stage de notre étude, il nous est désormais possible de faire, du point de vue des modes de stockage et de communication, un choix parmi les modèles d'architectures envisagés. Ce choix ne peut que se porter sur un modèle de machine à registres, en raison de la plus grande puissance de traitement qu'il permet d'atteindre. Il est toujours possible, en effet d'y ramener le taux de conflit au niveau de celui qui peut exister dans une machine bi-processeur. De plus, ces conflits, en raison de la brièveté des temps d'accès aux registres, ne peuvent être que relativement peu fréquentes et de faible durée.

Un tel modèle présente bien d'autres avantages : la simplicité de son principe, tout d'abord, qui permet d'envisager une mise au point relativement aisée, et un coût modéré en ce qui concerne la construction du système de communications.

D'autre part, nous venons de voir, en étudiant les différents protocoles d'extraction des informations, que l'on peut fort bien cumuler, sous certaines conditions qui restent à préciser, les avantages de la rapidité d'accès à l'information avec ceux de l'économie de la mémoire, en se passant de toute recopie de variable partagée. Ce mode de fonctionnement, qui permet d'étendre les possibilités de la machine dans une configuration donnée, peut se révéler utile dans la pratique.

Ce type d'architecture exploite naturellement bien toutes les propriétés de l'asynchronisme des algorithmes,et, du fait de son peu de sensibilité aux contraintes de communications, semble parfaitement bien adaptée à l'aspect chaotique des processus de convergence : elle n'introduit que peu de perturbations par rapport à leur schéma théorique, et doit en permettre une mise en oeuvre aisée.

Enfin, au contraire d'une architecture cellulaire (type ILLIAC IV) ou à bus hiérarchisés (type Cm*), elle n'apporte aucune contrainte

particulière au niveau des échanges de données entre processeurs chargés de gérer des sous-domaines de variables très éloignés l'un de l'autre au sein du domaine général de calcul. Toutes les communications inter-processeurs y sont traitées sur le même plan, et réclament, de ce fait, sensiblement le même temps.

IV - DESCRIPTION DE L'ARCHITECTURE PROPOSÉE

4.1. DESCRIPTION GENERALE

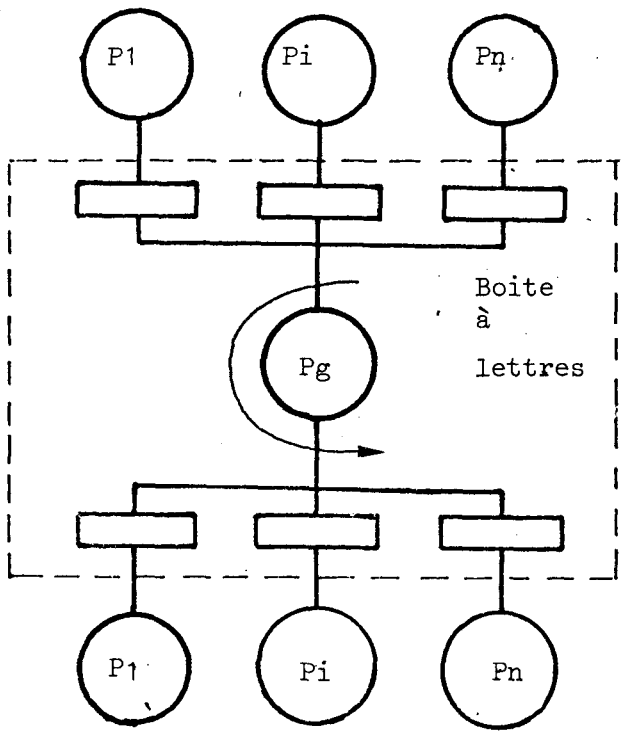


Fig. 3

L'architecture que nous proposons est conçue autour d'un mécanisme de boite aux lettres (B.A.L.) qui utilise des registres spécialisés en entrée et en sortie. Lorsqu'un processeur a évalué une variable sur une frontière de son sous-domaine, il en dépose la valeur dans le registre d'entrée auquel il a accès, ainsi que l'adresse du processeur qui gère la frontière voisine.

Le système de gestion de la B.A.L. est assuré par un processeur spécialisé P_G. Celui-ci explore l'ensemble des registres d'entrée, les uns après les autres, s'empare des informations qui s'y trouvent, et les place dans le registre de la B.A.L. auquel a accès le processeur destinataire. Ce dernier peut éventuellement les stocker dans sa mémoire locale à laquelle, en temps normal, il est seul à avoir accès.

4.2. CHOIX DU TYPE DE PROCESSEUR

Le choix du type de processeur qu'il convient d'utiliser n'a pas à proprement parler, de réelle importance. Tout au plus lui réclamons-nous une grande puissance de traitement, et en particulier, la capacité d'effectuer rapidement des opérations en virgule flottante.

Frailong [FRA 79] a développé l'étude d'un tel processeur, susceptible d'être construit en circuits intégrés LSI, pour une architecture cellulaire. Toutefois, l'emploi de microprocesseurs nous semble beaucoup plus intéressant pour des raisons de simplicité de conception et de sureté lors de l'implémentation.

Or il apparait actuellement sur le marché des microprocesseurs répondant à un concept nouveau : celui de co-processeur ; il s'agit d'unités spécialisés, implantées sur un "chip", et possédant un jeu d'instructions spécifiques adaptées à certains calculs scientifiques. Ils sont destinés à être utilisés comme esclaves d'un microprocesseur de la même famille, qui se charge de leur transmettre les opérations spécialisées.

Nous avons relevé deux références relatives à des produits déjà commercialisés : ce sont le 8087 d'Intel [PAL 80], destiné à être couplé au 8086 ou au 8087, et le NS 16081 de National Semiconducteur [BAL 80]. D'autres sont annoncés chez les concurrents pour le courant de 1981.

Parmi les performances annoncées, nous avons relevé les suivantes :

microprocesseur	multiplication 32 bits	multiplication 64 bits
8087	18 µs	27 µs
NS 16081	10 µs	18 µs

Elles montrent toute la puissance que l'on peut désormais atteindre pour un prix modique et une très faible complexité sans chercher à utiliser des processeurs "en tranche" très difficiles à mettre en oeuvre.

Ces processeurs, l'un comme l'autre, disposent d'un jeu de registres internes et d'instructions spécialisées. En particulier, le NS 16081 est capable de faire de la multiplication matricielle (donc du produit scalaire), ce qui s'avère plus important, dans le cadre de cette étude, que le seul calcul de fonctions transcendantes. Leur présence dans l'architecture est complètement transparente au programmeur qui n'a pas à spécifier si telle ou telle instruction doit être exécutée par l'un ou l'autre des co-processeurs. Quand le microprocesseur détecte une opération flottante, il en transmet l'exécution au module esclave et reste disponible pour effectuer d'autres opérations, jusqu'à ce que ce dernier le prévienne de la fin de son traitement. Le fonctionnement de l'ensemble est schématisé par la figure 4.

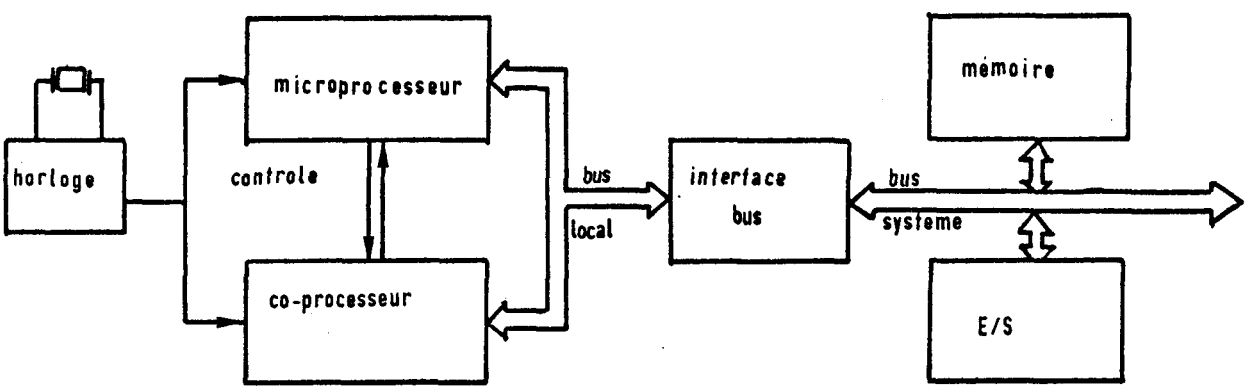


Fig. 4

Les autres critères extrêmement importants pour arrêter le choix d'un processeur sont ceux de l'espace adressable, qui doit être aussi grand que possible, et des facilités de communication. Or, les produits que nous venons d'évoquer sont tous des microprocesseurs 16 bits permettant d'adresser 64 K octets. Couplés à des unités de gestion de mémoire (MMU = Memory Management Unit), ils peuvent adresser plusieurs millions d'octets (48 M octets pour le Z8000) [HU 80]. La largeur de bande du bus local de ces processeurs est également une garantie d'efficacité en ce qui concerne les échanges avec les registres de la B.A.L.

Pour le processeur de gestion PG, rien n'impose à le choisir du même type que celui des processeurs de calcul. Le caractère extrêmement spécialisé de ses fonctions conduit à proposer un microprocesseur rapide, microprogrammable, principalement efficace au niveau du transfert des données, au détriment des propriétés de calcul proprement dites.

On peut citer, parmi les produits répondant à ces spécifications le Signetics 8x300, dont chacune des micro-instructions réclame un temps d'exécution constant, égal à 1 µs.

4.3. FORMAT DES MESSAGES - TAILLE DES REGISTRES

Quatre informations sont nécessaires pour constituer un message déposé dans la B.A.L. Les deux premières sont liées au problème lui-même, les deux autres à l'architecture proposée

- la valeur de la variable qui, la plupart du temps, est un réel représenté en format flottant sur un nombre de bits allant de 32 à 64. Elle occupe donc 4 à 6 octets.

- une adresse, virtuelle ou physique. Dans le premier cas, ce peut être l'adresse, dans une table de l'implantation du vecteur des variables, accompagné d'un déplacement correspondant, par exemple, au numéro du point pour lequel la valeur a été calculé.

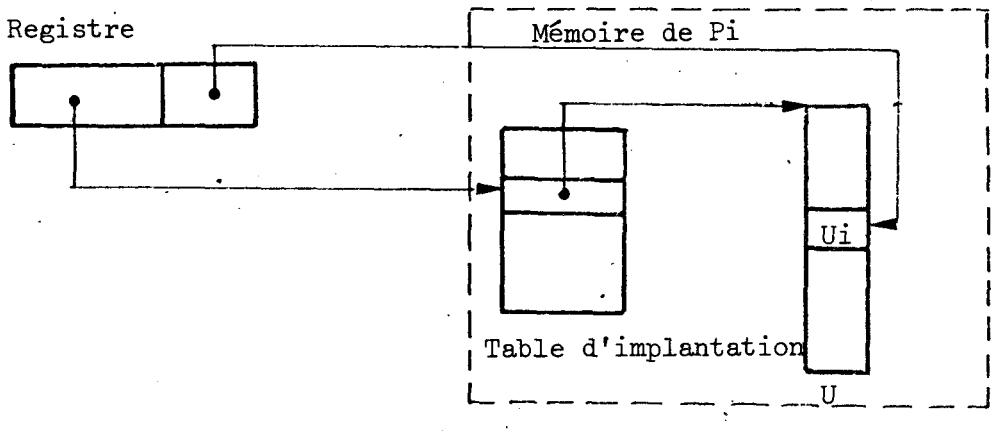


Fig. 5

- Le numéro du processeur destinataire. Un octet permet d'adresser jusqu'à 256 processeurs. Dans le cas où un nombre plus faible serait retenu, rien ne s'oppose à ce que les bits inutilisés servent à accroitre le champ d'adressage de la variable. Ceci est justifié par l'éventuel accroissement de la charge de calcul qui en résulterait pour chaque processeur.

- On peut y adjoindre le numéro du processeur producteur de l'information, qui peut être utile, dans certains cas, pour simplifier et accélérer l'identification complète de l'information, par rapport à la topologie de la base de donnée utilisée.

En tout donc, de 8 à 12 registres à 8 bits sont nécessaires à l'entrée de la BAL. A la sortie, seule la référence du processeur destinataire n'est plus nécessaire.

4.4. ETUDE DU PROCESSEUR DE GESTION

Evaluons le rythme avec lequel P_o doit assurer la gestion des registres. Les fréquences de production et d'acquisition des variables aux frontières, en moyenne, sont très voisins (si l'on néglige le problème des coins du domaine). Si nous supposons que les rapports entre les deux types de variables sont de l'ordre de 10% - 90%, et si T est le temps moyen d'une relaxation, leur valeur est sensiblement égale à :

$$f = \frac{1}{9T}$$

Par exemple, si l'on estime la valeur de T à 1 ms, on voit qu'un processeur de calcul fait, en moyenne, un accès à l'un de ses registres toutes les 9 ms.

Pour le processeur de gestion P_o, la période d'accès correspondante doit, au plus, avoir la moitié de cette valeur. Il est inutile d'envisager pour P_o un rythme d'exploration des registres supérieur à celui de leur chargement ou de leur lecture.

Nous venons cependant de dire qu'il était intéressant d'envisager à ce niveau, un microprocesseur très rapide, éventuellement microprogrammé. Si l'on prend l'exemple d'un microprocesseur effectuant une opération par microseconde, le nombre de registres qu'il doit gérer étant de 12+11, et compte tenu d'une analyse de l'adresse du destinataire, le transfert d'un message lui demande 25 µs. Supposons que l'on dispose de 256 processeurs de calcul, il lui reste donc 2600 µs pour effectuer un certain nombre de tâches supplémentaires entre deux distributions de messages.

- Des tâches de surveillance, portant sur le bon fonctionnement des processeurs (détection des pannes, suivi d'exécution ...)
- Des tâches de gestion de l'algorithme lui-même, parmi lesquelles la principale concerne la détection de la fin du calcul. Dans ce cas, un processeur estimant avoir achevé son traitement peut déposer dans son registre un message spécifique destiné à P_o. Celui-ci l'analyse et, en fonction du contexte, lui donne la suite qu'il convient.

Dans le cas d'une extension de la machine, il existe une configuration limite au delà de laquelle P_o n'est plus assez rapide pour suivre le rythme des productions/consommations. Il est cependant parfaitement possible de le dédoubler, et d'affecter alors chacun des processeurs mis en place à la gestion d'un sous ensemble de registres (figure 6). La transmission de certaines informations nécessitant le passage d'une boite à lettre à une autre, un certain nombre de registres, partagés entre PG1 et PG2, peuvent être utilisés à cet effet.

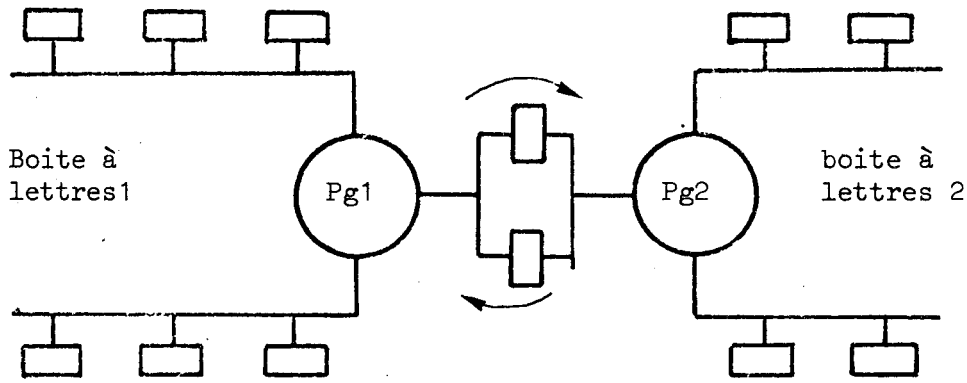


Fig. 6

4.5. LES MECANISMES D'EXCLUSION

Il est bien évident que l'accès à un registre est régi en ce qui concerne P_o et chaque processeur de calcul, selon le principe de l'exclusion mutuelle, de façon à préserver la cohérence des données qui y sont déposées. On peut envisager de deux façons la réalisation d'un tel mécanisme.

- par une solution câblée:un arbitre simple (figure 7) conçu à partir de 2 bascules, élimine les risques d'accès simultanés et interdit à un processeur de lire le contenu du registre, alors qu'un autre est en train de l'effacer pour y stocker une autre information.

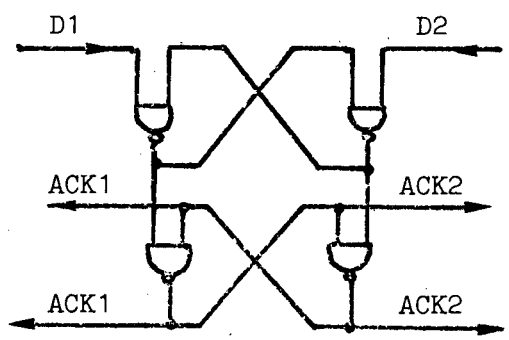


Fig. 7

Précisons qu'il est inutile de prévoir ce mécanisme pour l'ensemble des registres de chaque boite à lettres particulière, mais qu'il suffit de l'appliquer globalement en interdisant deux accès simultanés au premier d'entre-eux.

- par une solution programmée. Elle nécessite l'existence, au niveau des microprocesseurs, d'instructions test & Set, de durées identiques. On peut alors imaginer la séquence suivante, pour chacun des processeurs concernés, où C représente un registre de contrôle supposé contenir un bit d'occupation (figure 8)

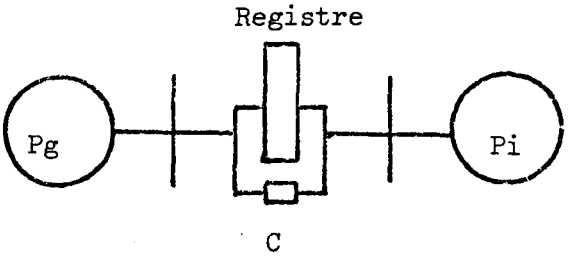


Fig. 8

Cette solution, économique au point de vue du matériel, ne peut cependant fonctionner correctement que si l'on interdit l'accès simultané au registre de contrôle.

4.6. EVALUATION DES PERFORMANCES

4.6.1. *Modes de fonctionnement retenus*

Afin d'évaluer le comportement de cette architecture, nous avons mené une série de simulations, correspondant à divers modes d'utilisation. Ceux-ci ont été décrits au § 3.2.4. Nous en avons retenu trois, qui nous ont semblé, soit présenter des caractéristiques particulières, soit conduire à des résultats convenables.

Le premier correspond à l'utilisation de registres spécialisés par rapport aux variables situées aux 4 coins d'un sous-domaine.

Le second correspond à un mode de lecture et de stockage systématique des informations du registre de sortie.

Le troisième correspond au procédé d'adaptation du calcul à la lecture du contenu du registre.

4.6.2. *Simulation*

Le problème simulé, par souci de simplicité, se réduit à un calcul de Laplacien, sur un domaine plan, carré, de N = 256 points (16x16).

L'architecture se compose de R processeurs. L'environnement de chaque processeur est décrit par l'ensemble des tableaux suivants :

- La topologie du domaine de calcul par :

  * IND(R, $\frac{N}{R}$) qui contient les numéros des points affectés à un processeur ; selon le mode simulé, il s'agit des points intérieurs, ou de l'ensemble des points du domaine.

  * FR(R, 4, P) contient, pour chaque sous-domaine, les numéros des P points aux frontières, respectivement Nord, Sud, Est, Ouest. Selon les cas, il peut contenir ou non les coins du sous-domaine.

  * COIN (R, 4) contient les numéros de ces 4 coins, dans l'ordre correspondant au schéma de la figure 9.

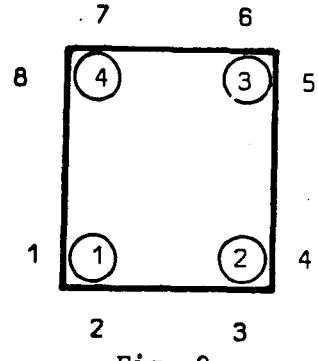


Fig. 9

- Son environnement par :

  * VS(R, 4, 2) contient les numéros des points voisins de chaque coin, dans un ordre correspondant au schéma.

  * PV(R, 4, 2) contient les numéros des processeurs chargés de gérer ces mêmes points.

  * VSN(R, 4) contient les numéros des processeurs voisins situés respectivement au Nord, au Sud, à l'Est et à l'Ouest symboliquement d'un processeur donné.

- Les objets du traitement

  * les variables à calculer sont stockées dans le tableau U(R,N)

  * un compteur CMT (R, N) comptabilise le nombre de relaxations effectués en chaque point d'un sous domaine et de son voisinage immédiat. A ce niveau, il sert à mesurer le taux de réception des objets, transmis à travers la boite aux lettre.

- Les registres sont représentés par les tableaux REGIN (R, 4) et REGOUT (R, 3), contenant pour le premier les références du destinataire de la valeur et de l'emetteur, et, pour le second, les trois dernières.

Le programme principal, au cours d'un intervalle de temps donné, adapté à la plus petite durée de relaxation d'une variable, examine le comportement de chaque processeur. Lorsque ce cycle est achevé, les objets placés dans les registres d'entrée sont, en ordre successifs, rangés dans les registres de sortie de la boite au lettres. Pour des raisons de simplification, et d'économie du temps de calcul, il nous a semblé en effet légitime de considérer que cette façon d'opérer rend assez bien compte du fonctionnement du processus de gestion. Les variations que l'on pourrait observer avec un fonctionnement réel seraient, pensons-nous, tout à fait négligeables.

4.6.3. *Résultats*

Les calculs ont été menés avec un nombre de processeurs égal aux puissances successives de 2, de 1 à 256.

Le premier mode de fonctionnement correspond à l'utilisation de registres spécialisés qui stockent, pour une durée assez longue, des informations relatives aux coins du domaine de calcul. Le contenu du registre courant REGOUT n'est utilisé que lorsqu'il correspond à la valeur cherchée. La simulation correspondante n'a pas été poussée. Nous nous sommes borné à constater que le temps de calcul obtenu avec 2 processeurs est très supérieur à celui que l'on mesure avec un monoprocesseur.

Les deux autres modes de fonctionnement ont conduit à des résultats assez voisins les uns des autres. Les temps de calcul observés pour le premier sont les plus faibles, ce qui était prévisible, étant donné que l'on a réalisé ici la simulation du comportement d'un algorithme chaotique pur, à peine perturbé par de très faibles contraintes architecturales. Ils ont été cependant obtenus, rappelons-le, aux prix d'une occupation plus importante de la mémoire propre à chaque processeur.

Le faible écart mesuré entre ces 2 cas devrait se réduire, dans la réalité, par rapport à celui que nous rapportons ici. En effet, le temps de transfert du contenu du registre à la mémoire n'a pas été pris en

compte, et le temps d'évaluation d'une variable a été le même dans tous les cas.

Les résultats de la simulation sont illustrés sur le graphe de la figure 10, ou l'on a porté, en fonction du nombre de processeurs, le rapport $\frac{TR}{T1}$ du temps de calcul mené avec R processeurs au temps correspondant au monoprocesseur. La courbe inférieure représente la fonction $\frac{1}{R}$.

Si l'on se reporte aux notations du modèle présenté au chapitre précédent, on constate que cette simulation fait apparaître principalement les effets de la fonction de redondance de calculs g(R), alors que ceux de la fonction de contention f(R) sont faibles, et restent pratiquement constants pour toute valeur de R.

Le meilleur des deux résultats montre que l'efficacité du fonctionnement correspondant est de l'ordre de 85 à 90 % de de celle d'un algorithme chaotique théorique. Dans ces conditions, l'influence des divers conflits et calculs inutiles se chiffre par un coefficient dont la valeur maximale est de 1,15, et qui traduit une perte de puissance relative de chaque processeur. Si l'on reprend les divers chiffres que nous avons proposé tout au long de ce chapitre, on est conduit à adopter, pour chaque processeur, une puissance de calcul égale à 25000 Flops. Alors, nos 256 processeurs vont pouvoir résoudre le problème évoqué (correspondant à un total de 200 millions d'opérations en virgule flottante) en un temps égal à

$$T = \frac{200 \times 1.15}{25000 \times 256} = 36 \text{ secondes environ}$$

Pour résoudre ce même problème, dans un temps de 10" qui correspond à la limite psychologique autorisant un traitement interactif, il faudrait mettre en oeuvre 920 processeurs.

Avec des processeurs 4 fois plus puissants (100 M Flops), le temps de résolution ne serait plus que de 9 secondes environ avec 256 processeurs.

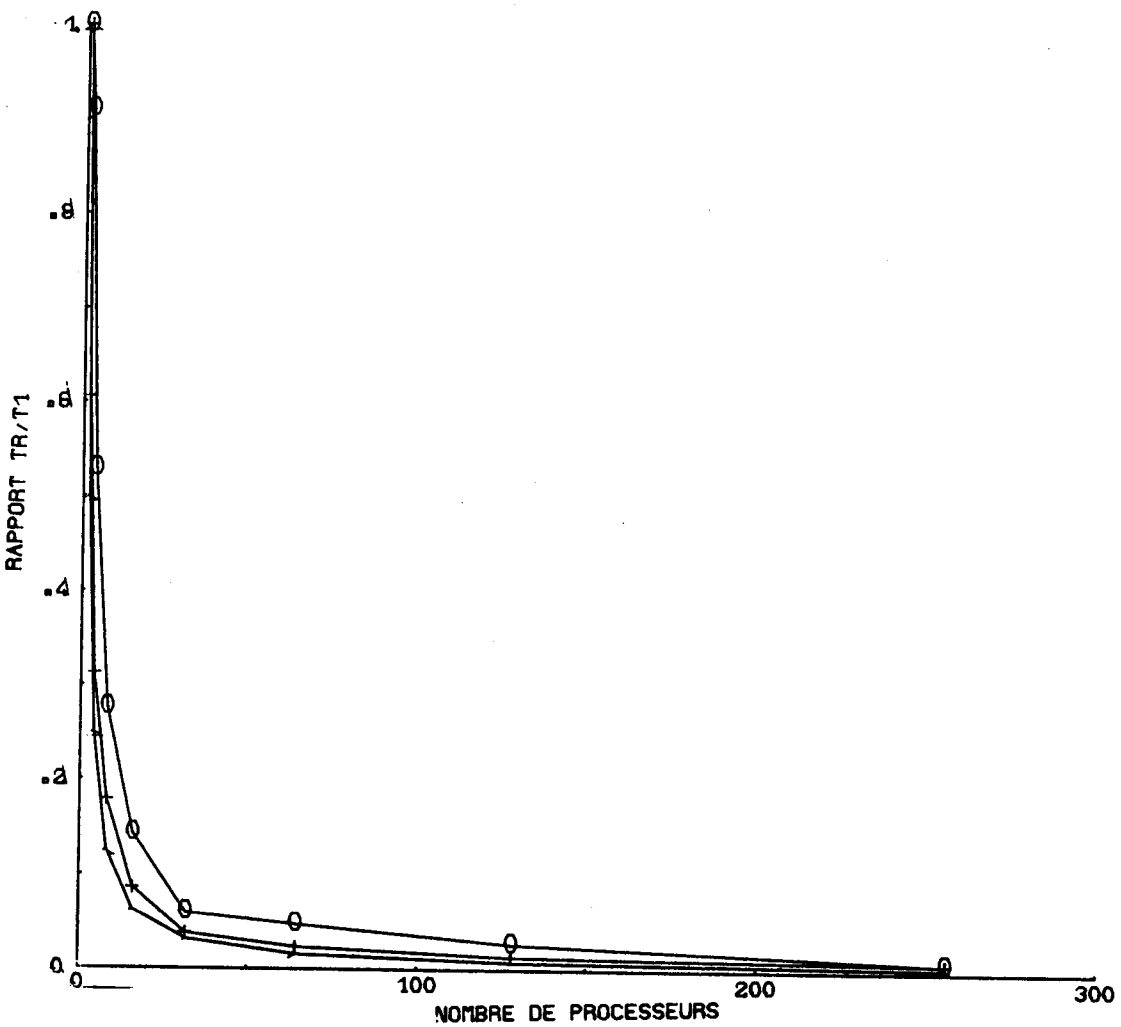


Fig. 10

CONCLUSION

Jusqu'à maintenant, il était normal, en informatique, de considérer que la sécurité d'une information utilisée au cours d'un traitement devait être absolue, et justifier toutes les mesures de sauvegarde que l'on pouvait envisager. Dans ces conditions, les transferts asynchrones d'un processeur à un autre étaient une source de difficultés considérables au niveau de l'organisation du système. S'il est facile, en effet d'implémenter des primitives de type sémaphore, celles-ci ne sont pas à même d'éviter les blocages, et de nombreux auteurs pensaient que le gain de temps réalisé à l'aide de méthodes chaotiques n'était pas suffisant pour justifier des efforts importants de programmation.

Nous avons pu montrer qu'il est possible de lever de telles objections, ce qui confert aux algorithmes chaotiques toute leur efficacité. Nous n'avons pas cherché à la comparer avec celle d'autres classes d'algorithmes. Sans doute en exite-t-il de plus puissants. Nous nous sommes contenté d'essayer de préciser comment l'on pouvait approcher l'efficacité optimale, et les résultats obtenus indiquent que le degré de parallèlisme atteint est très proche du maximum théorique. L'absence quasi complète de conflits dans l'architecture proposée permet d'envisager une extensibilité régulière adaptée aux besoins d'une application quelconque, même de taille très importante.

Nous sommes cependant bien convaincus que l'étude effectuée ne peut se substituer à des expérimentations réelles : la poursuite de simulations mais aussi la réalisation d'une machine asynchrone, devraient être l'un des prolongements de ce travail.

ANNEXE 1

Les équations de Navier-Stockes, par leur complexité et l'importance de leur champ d'application sont à l'origine des récents projets de construction de super-calculateurs. Il nous parait intéressant d'en présenter une résolution dans un cas très particulier et relativement simplifié, portant sur l'étude de la lubrification hydrodynamique d'un palier fini. La première phase de cette étude, qui est celle à laquelle nous nous arrêterons, concerne la répartition des pressions dans un film d'huile qui assure un double rôle de lubrification et de sustentation de l'arbre dans son coussinet (figure 1).

Le palier supposé avoir une longueur finie ℓ, est alimenté à travers une rainure longitudinale faisant un angle β avec la ligne joignant les centres respectifs de l'arbre et du coussinet.

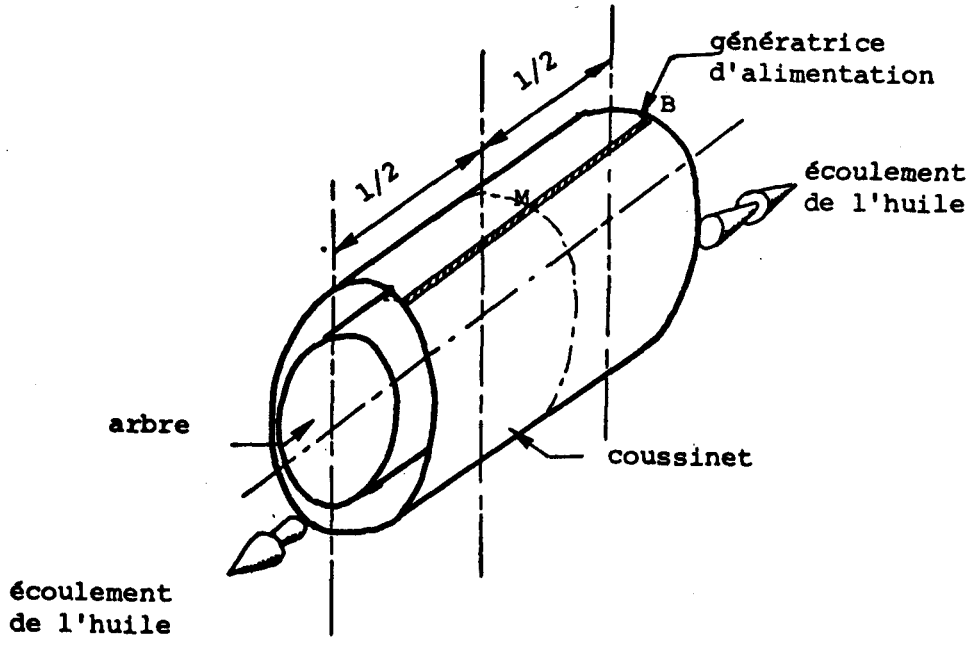


Fig. 1

La différence des deux rayons, appelée jeu radial, noté a, est extrêmement faible. De ce fait (palier lisse), les rayons de courbure de l'arbre et du coussinet sont très grands par rapport à l'épaisseur du film d'huile, et il est légitime de développer la surface du coussinet et d'y définir un repère cartésien d'origine C quelconque sur le coussinet et d'axes x_1, x_2, x_3 (figure 2) tels que x_3 soit parallèle à l'axe de ce dernier.

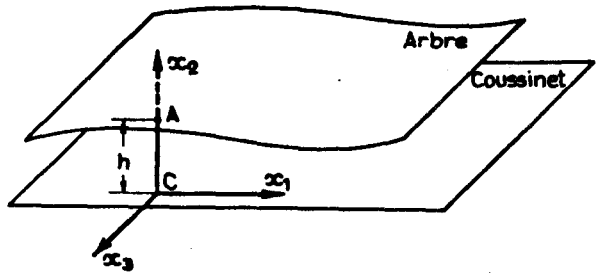


Fig. 2

Soit $\vec{v}(v_1, v_2, v_3)$ le vecteur vitesse d'un élément du fluide dans ce système d'axe, et p sa pression.

On suppose que :

- le fluide est newtonien c'est à dire que son comportement est décrit par son seul coefficient de viscosité μ.
- que le fluide est incompressible (ρ : poids spécifique = c^{ste})
- que le nombre de Reynolds, défini par $R = \rho\frac{v1a}{\mu}$ est faible

l'écoulement est alors laminaire et les forces d'inertie sont négligeables.

Les équations de Navier-Stockes se simplifient alors beaucoup, pour s'écrire :

$$\vec{grad}\ p = \mu\Delta\vec{u} \qquad (1)$$

soit : $$\frac{\partial p}{\partial x_1} = \mu\left(\frac{\partial^2 v1}{\partial {x_1}^2} + \frac{\partial^2 v1}{\partial {x_2}^2} + \frac{\partial^2 v1}{\partial {x_3}^2}\right)$$

et 2 équations semblables en v2 et v3.

On introduit l'équation de continuité $\bar{V}\ \vec{u} = \vec{0}$, indépendante de ρ du fait de la seconde hypothèse :

soit $$\frac{\partial v1}{\partial x_1} + \frac{\partial v2}{\partial x_2} + \frac{\partial v3}{\partial x_3} = 0 \qquad (2)$$

La figure 3 fait apparaître les grandeurs suivantes :

h : hauteur AB du film d'huile

$\Psi = \Psi(r)$: angle instantané qui fait la ligne des centres avec une charge verticale $\vec{F}$, intervenant comme force extérieure.

e = e(t) : distance instantanée des centres 01 et 02

$\vec{V}$: vitesse périphérique de l'arbre par rapport à son centre 01 ; $\vec{V} = (V_1, V_2, V_3)$.

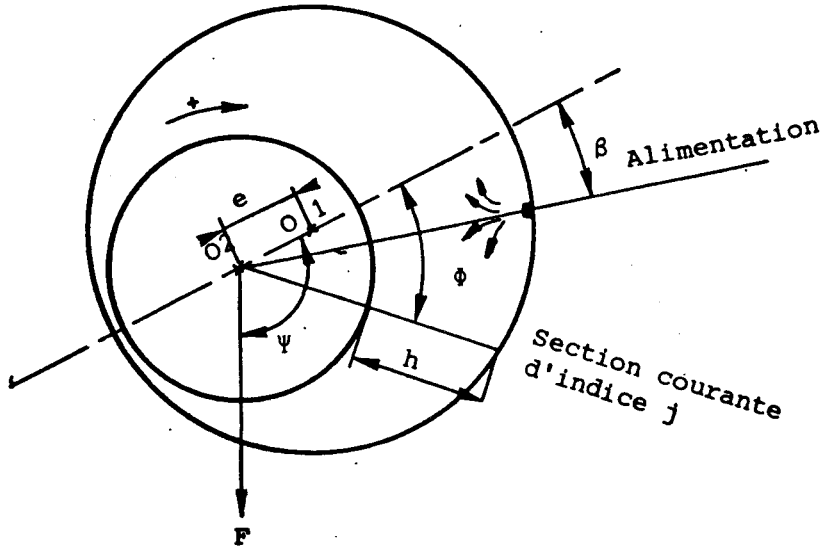


Fig. 3

Le système de 4 équations obtenu à partir de l'équation de Navier-Stockes et de l'équation de continuité conduit, moyennant quelques hypothèse et développements que nous n'expliciterons pas, à l'équation de Reynolds, établie par ce dernier dès 1886 :

$$\frac{\partial}{\partial x_1}\left[\frac{h^3}{\mu}\frac{\partial p}{\partial x_1}\right] + \frac{\partial}{\partial x_3}\left[\frac{h^3}{\mu}\frac{\partial p}{\partial x_3}\right] = 6\left[V1\,\frac{\partial h1}{\partial x_1} + h\,\frac{\partial v1}{\partial x_1} + 2\,\frac{\partial h}{\partial t}\right] \quad (3)$$

qui fait intervenir les inconnues :

$V1 = V1\,(x_1, x_3, t)$

$h = h(x_1, x_3, t)$

$p = p(x_1, x_3)$

et $\mu = \mu(p, x_1, x_3)$

Cette dernière variable dépend beaucoup, de plus, de la température (variable dans tout l'intervalle où circule l'huile). On la supposera indépendante de la pression, et on supposera pour simplifier le film isotherme. On fera les hypothèses supplémentaires suivantes :

- la charge est constante
- le système étudié est stable (absence de vibrations !)

l'équation se simplifie pour devenir :

$$\frac{\partial}{\partial x_1}\left(h^2\,\frac{\partial p}{\partial x_1}\right) + \frac{\partial}{\partial x_3}\left(h^3\,\frac{\partial p}{\partial x_3}\right) = 6\mu\,V1\,\frac{\partial h}{\partial x_1} \quad (4)$$

On le transforme de façon à ne plus avoir qu'une seule variable réduite, non dimentionnelle :

on pose à cet effet :

$x_1 = r\,\emptyset \quad \emptyset \in [0, 2\pi]$

$x_3 = s\,n \quad n \in [0, 2n]$ où n est le rapport de la largeur ℓ du coussinet au diamètre 2r de l'arbre.

$\varepsilon = \frac{e}{a}$: exentricité relative

$V1 = 2\pi\, rN$ introduisant la vitesse angulaire N de l'arbre.

on a : $h = (1 + \varepsilon \cos \emptyset)$

après calculs et changements de variables, on obtient une équation ne dépendant plus d'une variable P, mesure non dimentionnelle de la pression réduite $\frac{p}{pm}$, ou p_m est la pression moyenne sur le coussinet :

$$\frac{\partial^2 p}{\partial \emptyset^2} + \frac{\partial^2 p}{\partial n^2} - \frac{3\,\varepsilon \sin \emptyset}{1 + \varepsilon \cos \emptyset} \frac{\partial p}{\partial \emptyset} + \frac{12\,\pi\,\varepsilon \sin \emptyset}{(1 + \varepsilon \cos \emptyset)^3} = 0 \qquad (5)$$

Choisissons comme domaine d'étude le développement de la surface du coussinet. Si l'on remarque que, pratiquement, les pressions se répartissent symétriquement par rapport à la section droite passent par M (figur 1), on est réduit à n'utiliser que la demi-longueur du coussinet (en négligeant la déformée de l'arbre).

Les conditions aux limites (fuite du fluide selon AA et position de la rainure d'alimentation AB) imposent P = 0 sur les trois frontières AA et AB (figure 4).

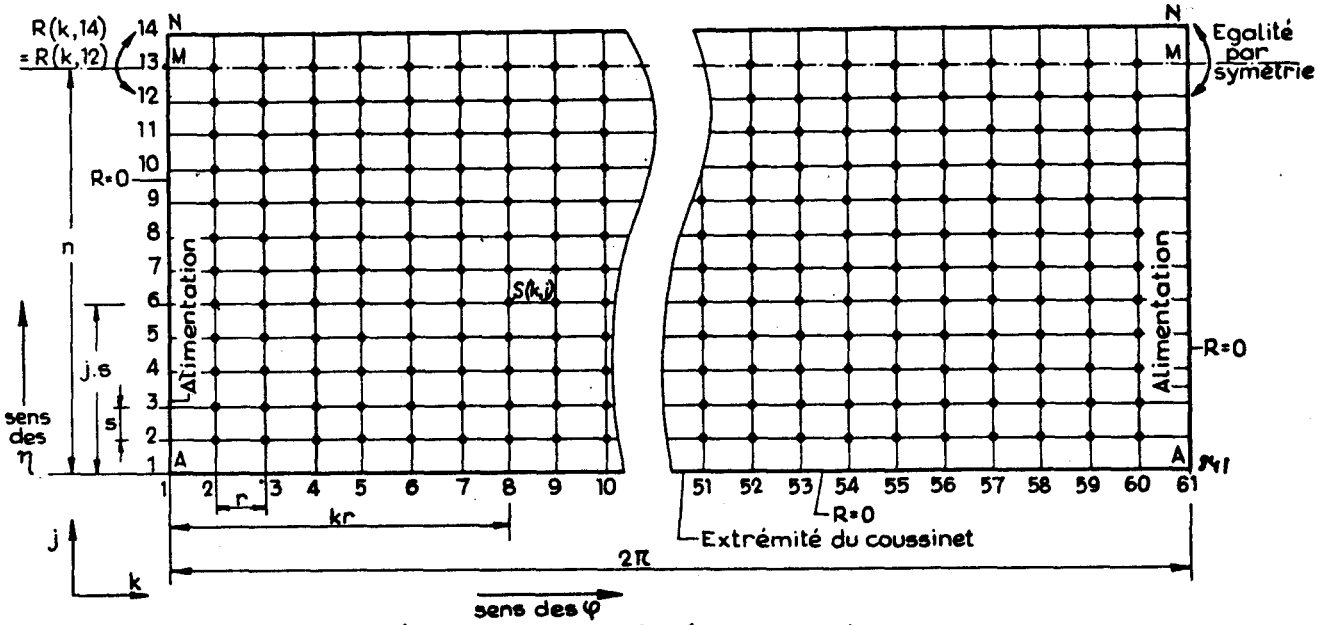


Fig. 4 : développement maillé du coussinet

Le choix d'un schéma centré à 4 points de discrétisation, nous conduit, pour pouvoir exprimer la symétrie précédente, à introduire une ligne supplémentaire au delà de l'axe de symétrie MM.

Le maillage adopté est un maillage régulier, de dimensions ($\Delta \varnothing = r$) x ($\Delta n = s$). Un noeud de ce maillage, d'indices I et J, est à une distance (I-1) s de l'extrémité de l'arbre et (J-1)r de l'alimentation soit à une disante angulaire $\varnothing = (j-1)r+\beta$ de la ligne des centres (figure 5). Pour un angle β donné, on a donc $\varnothing = \varnothing(j)$.

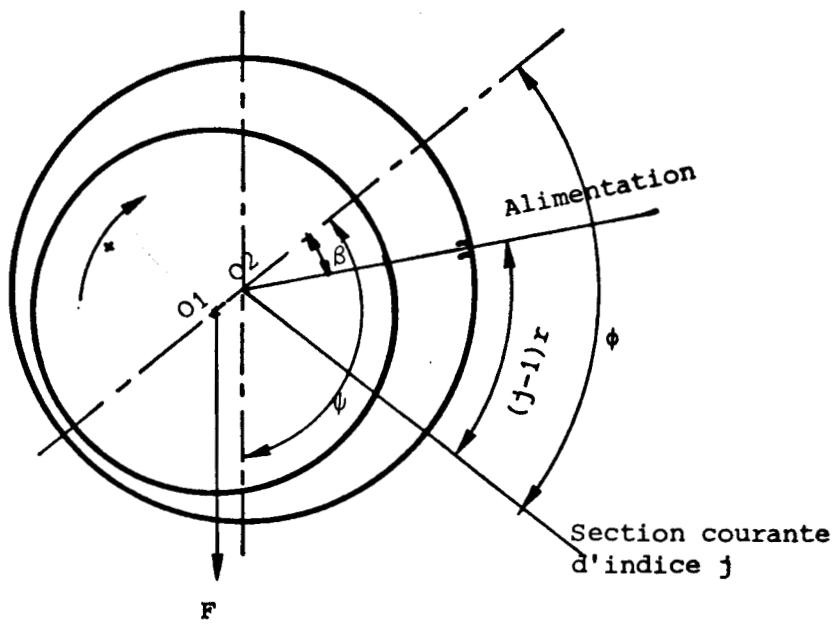


Fig. 5

Introduisons les coefficients $a(j) = \frac{3\,\varepsilon \sin \varnothing}{1+\varepsilon \cos \varnothing}$ et $a'(j) = \frac{12\,\pi\varepsilon \sin \varnothing}{(1+\varepsilon \cos \varnothing)^3}$

discrétisons sur le maillage de la figure 4 l'équation (5)

$$\left(\frac{\partial^2 p}{\partial n^2}\right)_{ij} = \frac{P(I+1,j) + P(I-1,J) - 2P(I,J)}{s^2}$$

$$\left(\frac{\partial^2 p}{\partial \varnothing^2}\right)_{ij} = \frac{P(I,J+1) + P(I,J-1) - 2P(I,J)}{r^2}$$

et $$\left(\frac{\partial p}{\partial \varnothing}\right)_{ij} = \frac{P(I,J+1) - P(I,J-1)}{2r}$$

d'où $$\frac{P(I,J+1)+P(I,j-1)-2P(I,J)}{r^2} + \frac{P(I+1,3)+P(I-1,3)-2P(I,J)}{s^2} - a\,\frac{P(I,J+1)-P(I,J-1)}{2r} + a' = 0$$

posons : $g = 2(\frac{1}{r^2} + \frac{1}{s^2})$

$$L = \frac{1}{gs^2}$$

$$c(j) = \frac{1}{g}(\frac{1}{r^2} + \frac{a(d)}{2r})$$

$$d(j) = \frac{1}{g}(\frac{1}{r^2} + \frac{a(j)}{2r})$$

$$b(j) = \frac{a^1(j)}{g}$$

il vient :

$$P(I,J)=(C(j)xP(I,J-1)) + \ell x(P(I-1,J)+P(I+1,J)) + (d(j)xP(I,J+1))+ b(j) \quad (6)$$

Nous avons modifié le caractère matriciel de cette équation en effectuant une renumérotation des noeuds du maillage. Nous avons pris une suite continue d'indice en partant du coin inférieur gauche pour aller jusqu'au coin supérieur droit, en effectuant un "balayage" en remontant colonne par colonne. La figure 6 précise ce procédé qui, d'une matrice de valeurs inconnues P(I,J) conduit à un vecteur inconnu P(I)

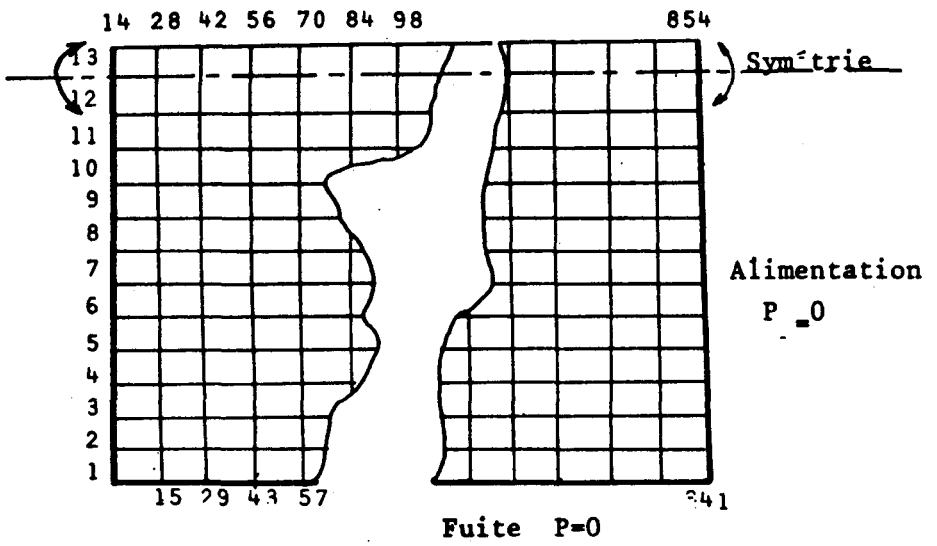


Fig. 6 : renumérotation des noeuds

Dans ces conditions, si N1 est le nombre de points contenus dans la largeur du domaine, l'équation (6) devient :

$$P(I) + C_I(P(I-L1) + \ell \times (P(I-1) + P(I+1)) + d_I P(I+L1) + b_I$$

Pour le maillage de la figure 6, par exemple, on a :

$$P_{16} = C_{16}\, P_2 + \ell(P_{15} + P_{17}) + d_{16}\, P_{30} + b_{16}$$

Les conditions aux frontières s'écrivent :

A l'ouest : $P(I) = 0 \quad I \in [1, 2, \ldots 14]$

Au sud : $P(I) = 0 \quad I = 1 + 14 \times J \quad J \in [1, 2, \ldots 60]$

A l'est : $P(I) = 0 \quad I \in [841, \ldots 854]$

Au nord : $P(I) = P(I-2) \quad I = 14xJ \quad J \in [1, 2, \ldots 61]$

L'aspect de la matrice de l'opérateur est évoqué figure 7. On observe que la bande de largeur L = 1+2x14, est particulièrement vide, et que les valeurs significatives sont toutes disposées sur les mêmes diagonales secondaires. Ceci suggère naturellement de simuler la matrice 854 x 854 par une matrice 854 x 5, compte tenu que la sixième information (le 1 sur la diagonale principale) n'a pas à y être introduite.

L'équation de sur-relaxation s'écrit :

$$P_I = P_I + w\,[C_I\,P_{I-14} + \ell(P_{I-1} + P_{I+1}) + d_I\,P_{I+14} + b_I - P_I]$$

Signalons que, pour ce maillage de dimensions moyenne, Servaës [SER 67] a trouvé $w_o = 1.83$, et qu'il précise que le temps de calcul double pour w = 1.7 ou 1.95.

Signalons enfin qu'il existe dans le développement du coussinet une zone non portante de grande dimension variable en fonction des différents paramètres de calcul (β et ε). Ceci se traduit par une pression nulle, alors que, numériquement, on en vient à calculer des pressions négatives. En relaxation, les fonctions p et $\frac{\partial p}{\partial \theta}$ étant continues, il suffit d'imposer p = 0 en tout point où elle devient négative. En fait, il y a là une discontinuité d'opérateur de part et d'autre d'une frontière à priori indéfinie. L'équation de Reynolds ne s'applique plus dans la zone de pression nulle, qui se caractérise par l'existence d'un phénomène de cavitation (présence de bulles d'air dans le fluide.

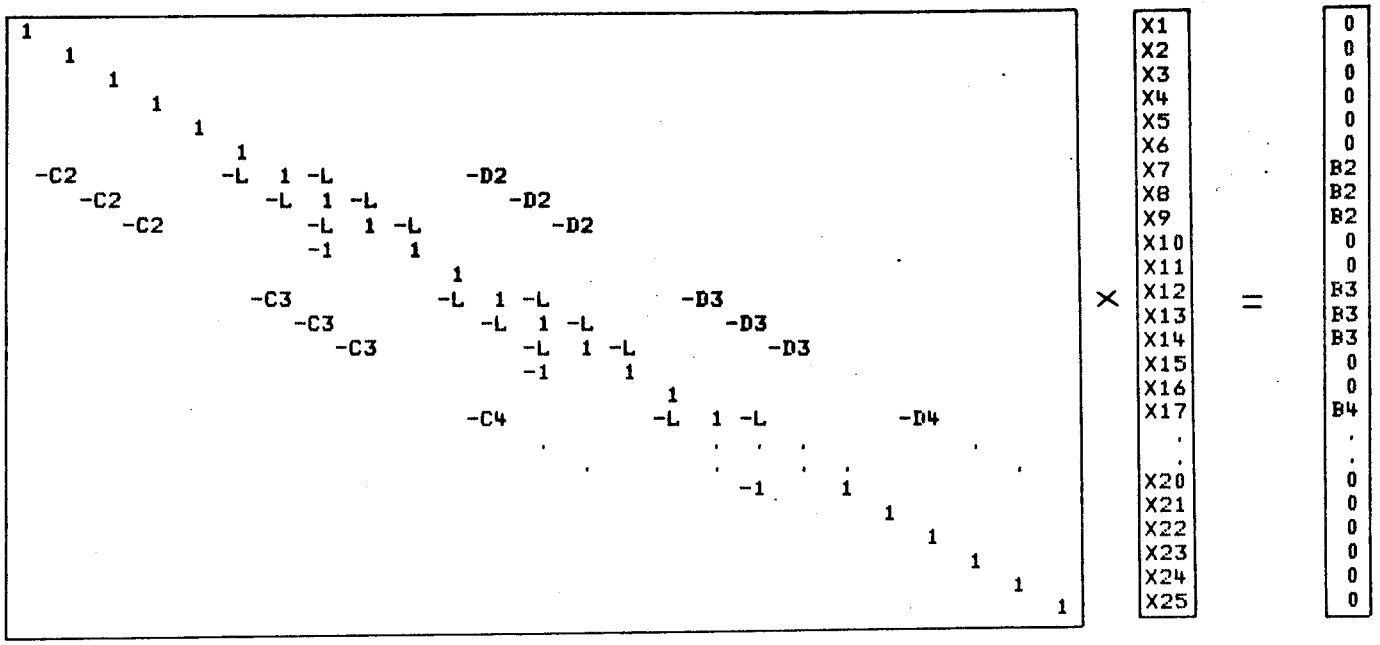


Fig. 7

ANNEXE 2

ÉTABLISSEMENT D'UN MODÈLE ANALYTIQUE D'ÉVALUATION DES PERFORMANCES D'UNE ARCHITECTURE

Ce modèle est établi commodément à partir de la théorie des files d'attente.

I - RAPPELS SUR LA THEORIE DES FILES D'ATTENTES

La figure 1 montre les éléments d'un système ouvert de file d'attente. On considère une population qui forme une source d'utilisateurs potentiels. Chaque utilisateur est un individu désirant recevoir un certain service d'un dispositif donné. Etant donné le nombre d'utilisateurs et le temps mis pour assurer ce service, une queue se forme devant des guichets. Au niveau de chaque guichet se trouve un serveur, qui travaille tant qu'il existe encore une personne dans la queue, et se repose sinon.

Un certain nombre de variables aléatoires permettent de modéliser une file d'attente, et sont représentées sur la figure 2. Ce sont :

- le taux moyen λ d'arrivée des usagers provenant de la population initiale
- le temps τ séparant 2 arrivées successives d'usagers, dont la valeur moyenne est $E(\tau) = \frac{1}{\lambda}$
- la valeur moyenne Lq du nombre Nq d'usagers dans la file : $Lq = E[Nq]$
- la valeur moyenne L du nombre N d'usagers dans l'ensemble du dispositif $L = E[N]$
- le temps moyen Wq passé par chaque usager dans la file
- le temps moyen W passé par chaque usager dans le dispositif complet
- le temps moyen E[s] mis par un serveur à accomplir le service rendu. On a donc la relation $W = Wq + E[s]$
- le facteur d'utilisation ρ d'un serveur, probabilité pour que l'un quelconque d'entre-eux soit occupé.

Les calculs statistiques que l'on peut effectuer sont beaucoup plus simples lorsque l'on considère que la source de population est de taille infinie. Il suffit dans la pratique qu'elle soit suffisamment grande pour pouvoir se placer dans le cadre de cette hypothèse. Le fait d'utiliser une description probabiliste revient à remplacer un phénomène essentiellement discret par un phénomène continu équivalent, décrivant l'état stationnaire d'un système.

Les arrivées des utilisateurs dans la file se font à des instants successifs $t_1 < t_2 < t_3 \ldots$ Leur distribution dans le temps se fait normalement de façon quelconque. Lorsqu'elle est exponentielle, la probabilité P pour que τ soit inférieur ou égal à un temps t donné correspond à la loi de Poisson : $P(\tau \leq t) = 1 - e^{-\lambda t}$.

Il existe naturellement d'autres types de distribution ; on classe les files d'attente en distinguant les lois de probabilité attribuées aux variables aléatoires τ et s, et leur mode de fonctionnement. Dans notre cas, nous considérons des modes de fonctionnement FIFO.

Pour distinguer les différentes files, Kendall a introduit la notation :

A / B / c / K / m / Z

- A spécifie la distribution du temps d'arrivée des utilisateurs, B celle du temps de service, c le nombre de serveurs.

On donne à A et B les valeurs symboliques suivantes :

* GI lorsque les temps d'arrivée sont indépendants
* G lorsque les temps de service sont indépendants
* D lorsque ces deux temps sont constants
* E_k lorsque la distribution est Erlangienne à k étages
* M lorsqu'elle est exponentielle
* H_k lorsqu'elle est hyper-exponentielle à k étages

- K dénote la capacité du système, m le nombre de sources, Z le fonctionnement de la file. Pour une FIFO et une source infinie, on utilise la notation A / B / c.

Le modèle que nous utilisons est décrit en terme d'un réseau de M / M / c. Dans ce cas, en effet, des résultats analytiques ont été énoncés.

En fait, le fonctionnement réel correspond soit à une distribution constante du temps de service, soit à une distribution par paquets (Erlangienne). Il est cependant possible, moyennant quelques approximations de se ramener au premier cas. Ces simplifications sont justifiées par les faibles écarts numériques que l'on constate au niveau des résultats. On peut mesurer la validité des hypothèses faites en examinant la valeur du paramètre :

$$C_x^2 = \left(\frac{Var[X]}{E[X]} \right)^2$$

elle est égale à 0 si X est cst

$0 < \frac{1}{k} < 1$ si la distribution de X est Erlangienne

1 si elle est exponentielle

> 1 si elle est hyper-exponentielle.

Les formules définissant une file de type M / M / c dans son état stable font état du paramètre u mesurant l'intensité du traffic dans la file, c'est à dire le rapport des temps moyens entre 2 arrivées E[Z] et du service E[s]

$$u = \frac{E[s]}{E[\tau]} = \lambda \, E[s]$$

Dans ce cas, le facteur d'utilisation ρ est égal à $\frac{u}{c}$, et la probabilité C(c, u) pour que tous les c serveurs soient occupés à tout instant est donnée par la formule d'Erlang :

$$C(c, u) = \frac{\frac{u^c}{c!}}{\frac{u^c}{c!} + (1-\rho) \sum_{n=0}^{c-1} \frac{u^n}{n!}}$$

dans le cas d'une M / M / c, elle se simplifie pour valoir $C(c, u) = \frac{f}{c} = \frac{\lambda}{c} E[s]$. Le temps moyen passé dans la file est :

$$Wq = \frac{C(c, u) \, E[s]}{c(1-\rho)}$$

le temps moyen total dans le système est : $W = Wq + E[s]$.

Dans le cas des files de type GI / G / c, M / M / c, M / G / c on admet les approximations suivantes :

$$Wq = \frac{C(c, u) \, E[s]}{c(1-\rho)} \times \left(\frac{C_\tau^2 + C_s^2}{2} \right) \text{ où } \frac{C(c, u)}{c(1-\rho)} \simeq \frac{\rho^c}{1-\rho^c}$$

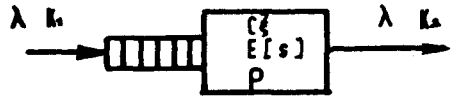
Dans ce cas, la formule de Little nous permet d'évaluer les valeurs des différentes variables du système en écrivant :

$$L = \lambda.W \text{ et } Lq = \lambda.Wq$$

Lorsque l'on dispose de plusieurs files, si P est le nombre moyen d'usagers dans la totalité du système, elle devient : $P = \sum_{i=1}^{m} \lambda_i W_i$ où m est le nombre de files.

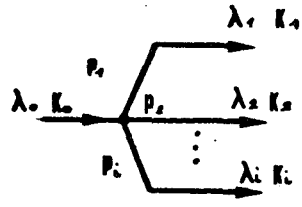
La machine modélisée peut être considérée comme un réseau de files d'attentes fonctionnelles. En tenant compte des relations précédentes, on peut obtenir un certain nombre de relations permettant d'évaluer la répartition des flots de messages dans les diverses branches du réseau, en fonction des 3 cas suivants :

- pour une file



la valeur de λ reste inchangée
K mesure la valeur de C_τ^2.
on a :
$K_2 \simeq 1+\rho^2[C_S^2-1] + (1-\rho^2)(K_1-1)$
avec $\rho = \frac{\lambda E[1]}{C}$ s'il y a c serveurs

- pour une bifurcation

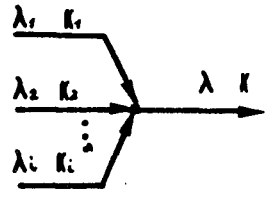


$\lambda = p_i \lambda 0$

$K_i = 1 + p_i(K_o - 1)$

p_i : probabilité de choisir la première branche.

- pour un regroupement



$$\lambda = \sum_{i=1}^{n} \lambda_i$$

$$K = 1 + \sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda}\right)^2 (K_i-1)$$

II - LE MODELE

Il est schématisé à la figure 13 du chapitre IV. Chaque élément est défini par les valeurs des paramètres λ, K et E qui lui sont associés, E représentant le temps de service. Lorsqu'un élément existe à plusieurs exemplaires (exemple des bancs de mémoire), il est représenté par plusieurs serveurs.

L'algorithme de calcul des flots consiste à définir à priori la valeur du taux moyen λ_6 des messages sortant du commutateur pour se diviser en 3 flots avec des priorités P1, P2 et P3. On calcule alors les valeurs des autres paramètres λ_i, et on itère jusqu'à ce que le λ_6 final soit égal au λ_6 initial.

On apelle WØ le temps séparant l'émission d'une requête pour opérande de sa réception, et WI le temps correspondant à l'attente d'une instruction. Sans dispositif d'anticipation, les temps de service des processeurs sont :

$$E_{12} = \frac{2450 + 44W\emptyset}{135} \quad \text{et} \quad E_{11} = \frac{2450}{135} + WI$$

avec WØ = 2W5 + W10 + 0.2W2 et WI = 2W5 + 2W15 + W17

Lorsqu'il y a anticipation, on a

$$E_{12} = \frac{2450 + 44W\emptyset}{125} \quad \text{et} \quad E_{11} = \frac{2450}{135} \quad \text{de plus, on doit poser } Wq12 = 0$$

La formule de Little nous permet par rebouclage, de calculer la valeur d'équilibre de λ_6. A tout instant, en moyenne, s'il y a R messages en circulation dans le système, on doit avoir :

$$R = \lambda_2 W_2 + \lambda_{15} W_{15} + L_{17} W_{17} + R(\lambda_5 W_5 + \lambda_{10} W_{10} + \lambda_{11} W_{11} + \lambda_{12} W_{12})$$

Connaissant les λ_i, on peut calculer les valeurs des K_i correspondant et mesurer la validité des hypothèses faites. Là encore, un calcul itératif peut être nécessaire pour obtenir les valeurs d'équilibre.

L'unité utilisée pour exprimer les valeurs des paramètres utilisés est le cycle d'horloge, et nous nous sommes référés au catalogue décrivant la famille du Z800 (cycle de base de 250 ns) pour l'initialisation du programme de calcul.

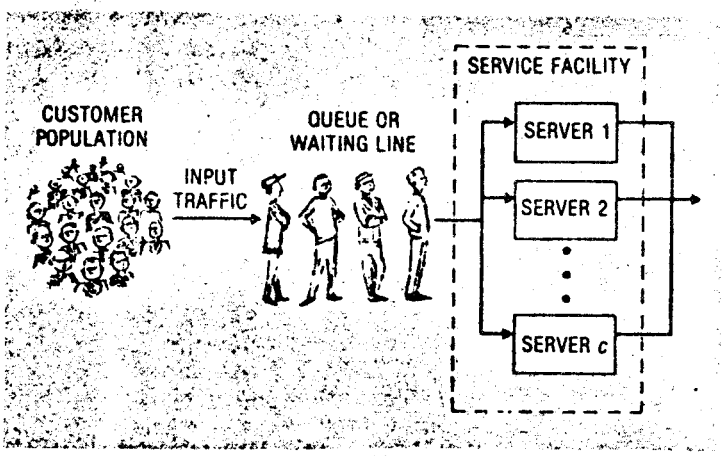


Fig. 1

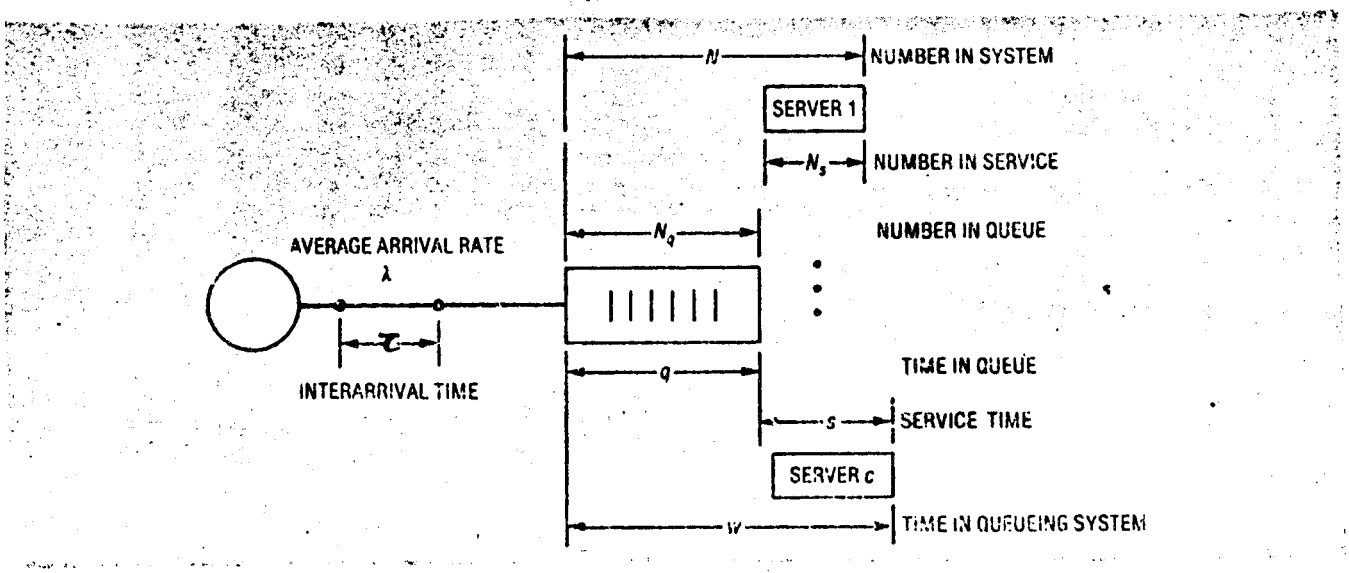


Fig. 2

BIBLIOGRAPHIE

[ALL 80] A.O. ALLEN : *"Queing models of computer systems"*
Computer n° 12 Avril 80

[BAL 80] S. BAL, E. BURDICK, R. BARTH, D. BODINE : *"Systems capabilities get a boost from a high-powered dedicated slave"*
Electronic design n°5 Mars 80

[BAR 68] G.H. BARNES : *"The ILLIAC IV computer"*
IEEE Trans. on comp. Vol C-17 N°8 aout 68

[BAT 68] K.E. BATCHER : *"Sorting network and their application"*
SJCC 1968

[BAT 80] K.E. BATCHER : *"Architecture of a massively parallel processor"*
La Baule symposium ACM-IEEE Mai 1980

[BAU 78] B. BAUDET : *"The design and analysis of algorithms for asynchronous multiprocessors"*
Thèse Carnegie Mellon University avril 1978

[BEL 72] C.G. BELL, W.A. WULF : *"Cmmp : a multi-mini-computer"*
AFIPS FJCC 1972

[BUC 77] R.S. BUCY, K.D. SENNE : *"Non linear filtering algorithms for parallel and pipe-line machines"*
Parallel computer, Parallel Mathematics Ed. Feilmeir 1977

[CAR 67] A.B. CAROLL, R.T. WETHERALD : *"Application of parallel processing to numerical weather prediction"*
JACM Vol 14 N° 3 juillet 1967 p 591-614

[CHA 69] D. CHAZAN, W. MIRANKER : *"Chaotic relaxation"*
Linear algebra and its application 2 (1969) 199-222

[COF 69] E.G. COFFMAN, R.R. MUNTZ : *"Optimal preemptive scheduling on a 2-processor system"*
IEEE Trans. on comp. vol C-18 n° 11 novembre 1969 p 1014.

[CON 79] S.E. CONRY, R.M. MATHEYSES : *"On construction of control networks for parallel systems"*
1st European conference on Parallel and Distributing processing Toulouse fév. 1979 p 170.

[COR 79] V. CORDONNIER : *"La mémoire circulante de Maud"*
Publication du laboratoire de calcul de Lille I 1979

[COV 74] A. COVO : *"Analysis of multiprocessor control organizations with program memory partial replication"*
IEEE Trans. on Comp. vol C-23 n° 2 Fév. 74 p 113.

[DON 71] J.D.P. DONNELLY : *"Periodic chaotic relaxation"*
Linear algebra and its applications 4 (1971) p 117-128

[ELT 76] EL TARAZI : *"Sur des algorithmes mixtes par blocs de type Newton-relaxation"*
Thèse 3e cycle Besançon 1976.

[ENS 77] P.H. ENSLOW J.R. *"Multiprocessors organizations : a survey"*
Computer Survey vol 9 N° 1 mars 77

[EVA 80] D.J. EVANS, S.A. WILLIAMS : *"Analysis and detection of parallel processable code"*
The computer journal vol 23 N° 1 fév 80.

[FEI 78] G. FEIERBACH, D.K. STEVENSON : *"The phoenix array processing system"*
Rapport.

[FIO 74] J.Ch. FIOROT, P. HUARD : *"Relaxation chaotique en optimisation"*
Publication du Laboratoire de Calcul de l'Université de Lille I 1974

[FIR 77] R.M. FIRESTONE : *"An analytic model for parallel computation"*
IMACS 1977

[FRA 79] J.M. FRAILONG : *"Definition of an integrated processing element for a parallel numerical architecture"*
Euromicro 1979 p 297-304

[FUL 77] S.H. FULLER, R.J. SWAN, D.P. SIEWIOREK : *"Cm* : a modular multiprocessor"*
AFIPS NCC Vol 46 1977 p 637-643 et 645-655.

[FUL 78] S.H. FULLER and al. :*"Multi-microprocessors : an overview and working examples"*
Proceeding on the IEEE vol 66 n°2 fév 78

[FUN 79] K.T. FUNG, H.C. TORNG : *"On the analysis of memory conflicts and bus contention in a multiple microprocessor system"*
IEEE Trans. on Comp. Vol C-27 N°1 janv. 79 p 28.

[HAN 75] W. HANDLER, F. HOFMANN, H.J. SCHNEIDER : *"A general purpose array with a broad spectrum of applications"*
Workshop on computer architecture erlangen mai 1975

[HOB 70] HOBSS : *"Parallel processor technologies and applications"*
Mc Millan Spartan books N.Y. 1970

[HU 80] J. HU, H. YONEZAWA, B. PUETO : *"Memory management units help 16-bit MPS to handle large memory systems"*
Electronics designs 9 avril 1980

[JON 80] J. JONES : *"Toward a hill orientated microprocessor instruction set"*
Euromicro journal n° 6 (1980) p 158-160.

[JPS 80] D. JULLIAND, G.R. PERRIN, P. SPITERI : *"Simulations d'exécutions parallèles d'algorithmes de relaxation asynchrones"*
Publication du labo. d'informatique de l'université de Besançon oct. 1980

[JPS 81] D. JULLIAND, G.R. PERRIN, P. SPITERI : *"Simulation de types de communications appliqués des algorithmes numériques"*
Besançon janv. 1981.

[KAL 79] A.V. KALYAYEV : *"MP homogeneous calculating structure with distributed memory and universal commutation"*
Euromicro Journal 5 (1979) p 73-81.

[KOB 79] R. KOBER, C. KUZNIA : *"SMS 201 : A powerful parallel processor with 128 microprocessors"*
Euromicro Journal n° 5 (1979)

[KOZ 79] E.W. KOZDROWICKI, D.J. THEIS : *"2D generation of vector super computers"*
Computer nov 1980

[KUC 77] D.J. KUC, A.H. SAMEH : *"Parallel direct linear system solver : a survey"*
Parallel Computer, parallel mathematics 1977 p 25

[MAR 79] M.A. MARSAN, G. CONTE, D. DEL CORSO, F. GREGORETTI : *"Analysis of the communication network in a multiprocessor system"*
Euromicro Journal 1979 p 381

[MAZ 78] MAZARE : *"Structures multiprocesseurs, problèmes de parallélisme définition et évaluation d'un système particulier"*
Thèse d'état Besançon 1978

[MIE 74] J.C. MIELLOU : *"Itérations chaotiques à retards"*
CRAS T 278 (1974).

[MIE1 75] J.C. MIELLOU : *"Algorithmes de relaxation chaotique à retards"*
RAIRO R1 avril 1975

[MIE2 75] J.C. MIELLOU : *"Itérations chaotiques à retards. Etude de la convergence dans le cas d'espaces partiellement ordonnés"*
CRAS T 280 1975

[MIE 77] J.C. MIELLOU : *"Relaxations chaotiques de type monotone"*
Séminaire d'analyse numérique Grenoble 1977

[MIR 71] W.L. MIRANKER : *"A survey on parallelism in numerical analysis"*
SIAM Review vol 13 N° 4 oct 71 P 524

[MIR 77] W.L. MIRANKER : *"Parallel method for solving equations"*
Parallel computer, Parallel mathematics 1977

[OST 55] A. OSTROWSKI : *"Determination mit uberwgegenden hauptdiagonale und die absolute konvergenz von linear iterations"*
Comm. Math. Helv. 30 (1955) p 175-210

[PAL 80] J. PALMER, R. NAVE, C. WYMORE, R. KOEHLER, C. MC-MINN :*"Maiking mainframe Mathematics accessible to microcomputers"*
Electronics 8 mai 1980

[POO 74] W. POOL, R. VOIGHT : *"Numerical algorithms for parallel and vector computers : an annotated bibliography"*
Computing Reviews vol 15 1974 p 379-388

[RAS 77] L. RASKIN : *"Performance of a stand alone Cm* system"*
Cm* Review. CMU Computer science département report juin 1977 p 26-56.

[ROS 69] J.L. ROSENFELD : *"A case study in programming for parallel processors"*
Communication of ACM vol 12 n° 12 Déc. 69 p 648

[SAM 78] A. SAMARSKI, V. ANDREEV : *"Methode aux différences pour équations elliptiques"*
Editions de Moscou.

[SAU 77] H. BELLM, A. SAUER : *"Method of data exchange between micro-computers"*
Euromicro Journal 1977

[SCH 59] S. SCHECHTER : *"Relaxations method for linear equations"*
Comm. Pure and Appl. Math. 12 (1959) p 313-355.

[SER 68] H. SERVAES : *"Contribution à l'étude de la lubrification du coussinet complet de largeur finie"*
Publication n° 4 de l'Université de Liege

[STR 70] W.D. STRECKER : *"Analysis of the instruction Rate in certain computer structures"*
These CMU 1970

[THU 76] K.J. THURBER : *"Large scale computer architecture"*
1976

[WAL 77] Y. WALLACH, V. CONRAD : *"Iterative solution of linear equations on a parallel processor system"*
IEEE Trans. on comp. vol C-26 N° 9 Sept 1977 p 838.

[YOE 79] M. YOELI : *"A structured approach to parallel programming and control"*
1st European conference on parallel and distributed processing
Toulouse fév 1979.

[ZEY 74] R.K. ZEYTOUNIAN : *"Notes sur les écoulements rotationnels de fluides parfaits"*
Springer Verlag Berlin Heidelberg New York 1974.

[ZIL 79] The Z8000 Family