Simulation du fonctionnement logique de FELIN : algorithmes de calcul simultané de racines de polynômes

Hassan Ouaouicha

THESE

présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE TITRE DE
DOCTEUR DE 3EME CYCLE
MATHEMATIQUES APPLIQUEES
(Analyse Numérique)

par

Hassan OUAOUICHA

SIMULATION DU FONCTIONNEMENT LOGIQUE DE FELIN
ALGORITHMES DE CALCUL SIMULTANE DE RACINES DE POLYNOMES

Thèse soutenue le 16 juin 1987 devant la Commission d'Examen

JURY

PRESIDENT: Monsieur Jean DELLA DORA

EXAMINATEURS:

Messieurs Michel COSNARD
Bernard COURTOIS
Christophe MASSE
François ROBERT

Thèse préparée au sein du laboratoire TIM3.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année 1987

Vice - Présidents : René CARRE
Jean-Marie PIERRARD

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BESSON Jean	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
BUYLE-BODIN Maurice	ENSERG
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHERUY Arlette	ENSIEG
CHIAVERINA Jean	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Mar	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Pierre	ENSIEG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSEEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PAUTHENET René	ENSIEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SERMET PIERRE	ENSERG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
PLA Fernand
GHIBAUDO Gérard
HAMAR Sylvaine
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S

Directeurs de recherche 1ère Classe

CAILLET Marcel
CARRE René
FRUCHART Robert
JORRAND Philippe
LANDAU Ioan
MARTIN

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BINDER Gilbert
BONNET Roland
BORNARD Guy
CALMET Jacques
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
JOUD Jean-Charles
KAMARINOS Georges
KLEITZ Michel
KOFMAN Walter
LEJEUNE Gérard
MERMET Jean
MUNIER Jacques
SENATEUR Jean-Pierre
SUERY Michel
TEDOSIU
WACK Bernard

Personnalités agrées à titre permanent à diriger des travaux de recherche (décision du conseil scientifique)

E.N.S.E.E.G

BERNARD Claude
CHATILLON Catherine
CHATILLON Christian
COULON Michel
DIARD Jean-Paul
FOSTER Panayotis
HAMMOU Abdelkader
MALMEJAC Yves
MARTIN GARIN Régina
SAINTFORT Paul
SARRAZIN Pierre
SIMON Jean-Paul
TOUZAIN Philippe
URBAIN Georges

E.N.S.E.R.G

BOREL Joseph
CHOVET Alain
DOLMAZON Jean-Marc
HERAULT Jeanny

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

BOIS Daniel
DARVE Félix
MICHEL Jean-Marie
ROWE Alain
VAUCLIN Michel

E.N.S.I.M.A.G

BERT Didier
COURTIN Jacques
COURTOIS Bernard
DELLA DORA Jean
FONLUPT Jean
SIFAKIS Joseph

E.F.P.G

CHARUEL Robert

C.E.N.G

CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIB Maurice
VINCENDON Marc

Laboratoires extérieurs

C.N.E.T

DEMOULIN Eric
DEVINE
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M.MERMET
Directeur des Etudes et de la formation: Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

PROFESSEURS DE 1ère CATEGORIE

COINDE Alexandre	Gestion
GOUX Claude	Métallurgie
LEVY Jacques	Métallurgie
LOWYS Jean-Pierre	Physique
MATHON Albert	Gestion
RIEU Jean	Mécanique-Résistance des matériaux
SOUSTELLE Michel	Chimie
FORMERY Philippe	Mathématiques Appliquées

PROFESSEURS DE 2ème CATEGORIE

HABIB Michel	Informatique
PERRIN Michel	Géologie
VERCHERY Georges	Matériaux
TOUCHARD Bernard	Physique Industrielle

DIRECTEUR DE RECHERCHE

LESBATS Pierre	Métallurgie

MAITRE DE RECHERCHE

BISCONDI Michel	Métallurgie
DAVOINE Philippe	Géologie
FOURDEUX Angeline	Métallurgie
KOBYLANSKI André	Métallurgie
LALAUZE René	Chimie
LANCELOT Francis	Chimie
LE COZE Jean	Métallurgie
THEVENOT François	Chimie
TRAN MINH Canh	Chimie

Personalités habilitées à diriger des travaux de recherche

DRIVER Julian	Métallurgie
GUILHOT Bernard	Chimie
THOMAS Gérard	Chimie

Professeurs à l'UER de Sciences de Saint-Etienne

VERGNAUD Jean-Maurice	Chimie des Matériaux et Chimie Industrielle

REMERCIEMENTS

J'adresse mes sincères remerciements à Monsieur le Professeur Jean DELLA DORA de m'avoir fait l'honneur de s'interesser à mon travail en acceptant la présidence de jury.

Monsieur le professeur Michel COSNARD a assuré la direction scientifique de mes recherches avec beaucoup d'intérêt et de disponibilité. Qu'il trouve ici l'expression de ma profonde gratitude pour son aide efficace et pour sa compétence scientifique dont il m'a fait bénéficier tout au long de ce travail au cours duquel j'ai pu aussi apprécier ses qualités humaines.

J'adresse mes vifs remerciements à Monsieur le Professeur François ROBERT de m'avoir fait l'honneur de participer au jury et de l'intérêt qu'il porte à ce travail. Je rends hommage aussi à ses grandes qualités humaines.

Je suis très sensible à l'honneur que me font Monsieur Bernard COURTOIS, Directeur de recherche au CNRS, et Monsieur Christophe MASSE, Ingénieur à PECHINEY, en acceptant de faire partie de ce jury.

Je tiens à exprimer mes remerciements:

A mon ami Jean Michel Muller, j'ai été très heureux de travailler et d'avoir eu de nombreuses discussions scientifiques avec lui; je n'oublierai pas sa gentillesse et son amitié.

Aux membres du "groupe FELIN" notamment Monsieur Alain Guyot, B.Hochet et E. Zysman de l'équipe Architecture des Ordinateurs: j'ai eu beaucoup de plaisir à travailler avec eux.

A mes camarades: N. Akroune, M. Diamoutani, T. Delespierre,C. Lausberg et en particulier P. Ozello et I. Sakho avec lesquels j'ai eu des échanges fructueux.

Aux membres de l'équipe "Algorithmique Parallèle et Calcul Formel" dont j'ai pu apprécier la cordialité.

A C. Dicrescenzo et A. Eberhard, toujours prêts à secourir un utilisateur perdu devant sa console.

A tous les membres du service de reprographie de l'I.M.A.G. pour la réalisation matérielle de ce travail.

Il y'a deux sortes d'esprits, l'un géométrique, et l'autre que l'on peut appeler de finesse. Le premier a des vues lentes, dures et inflexibles; mais le dernier a une souplesse de pensée qu'il applique en même temps aux diverses parties aimables de ce qu'il aime.

B. PASCAL

A

mon grand-père,
mon père et ma mère,
Houssein
et Souad.

SIMULATION DU FONCTIONNEMENT LOGIQUE DE FELIN

ALGORITHMES DE CALCUL SIMULTANE DE RACINES DE POLYNOMES

Table des matières

Deuxième partie
ALGORITHMES DE CALCUL
SIMULTANE DE RACINES DE POLYNOMES

INTRODUCTION

Le travail présenté dans cette thèse est divisé en deux parties.La première est consacrée à la simulation du fonctionnement logique d'un coprocesseur arithmétique de calcul de fonctions élémentaires. La seconde est consacrée à l'étude de quelques méthodes de recherche simultanée des racines d'un polynôme de degré quelconque et à coefficients complexes.

Avant la fabrication d'un prototype, il faut que son fonctionnement logique soit vérifié au préalable. L'outil le plus utilisé par les constructeurs est la simulation.

La simulation sur ordinateur du fonctionnement logique d'un circuit est une méthode numérique qui consiste à conduire des expériences sur ordinateur à l'aide d'un modèle du circuit afin d'analyser et de vérifier son fonctionnement.

La simulation sur ordinateur est un outil indispensable pour étudier le fonctionnement logique des circuits. La simulation séquentielle peut cacher des problèmes de parallélisme, qui sont fréquents dans la réalité. Le parallélisme étant présent dans tout circuit, toute approche de simulation doit répondre aux besoins du parallélisme.

Pour simuler les parties parallèles d'un circuit sur un ordinateur séquentiel leur fonction doit être modélisée sur des bases séquentielles; cette modélisation doit se rapprocher de la réalité. Le simulateur de l'architecture du circuit issu de sa modélisation doit préciser explicitement le parallélisme.

Notre travail est consacré à la simulation du fonctionnement logique de FELIN.

FELIN (Fonctions ELémentaires INtégrées) est un coprocesseur arithmétique de calcul rapide de fonctions mathématiques de base telles que: Sinus, Exponentielle, Arcsinus, Argth, ... Il est composé de deux machines travaillant en parallèle : machine de communication et machine à calcul. Il est destiné à traiter des

expressions arithmétiques complètes sans intervention du processeur hôte.

Pour simuler le fonctionnement logique du coprocesseur arithmétique FELIN, nous avons élaboré une méthodologie de simulation dans laquelle nous avons séquentialisé les parties parallèles par l'adjonction des registres logiques et par la duplication des variables.

Cette méthodologie de simulation consiste en premier lieu à définir un modèle représentant les différents éléments de base du circuit et leurs fonctionnement. Le modèle est ensuite traduit par un logiciel écrit dans un langage de haut niveau, celui-ci constitue le simulateur de l'architecture du circuit dont l'exécution sur des données permet de vérifier le fonctionnement du circuit et de retirer les améliorations à y apporter.

Le premier chapitre décrit le principe des différentes parties qui constituent FELIN, expose les algorithmes de calcul de fonctions élémentaires qui ont été étudiés en détail par J.M. Muller et définit le but, le principe et les outils de la simulation de FELIN.

Dans le deuxième chapitre nous montrons comment nous procédons pour simuler le fonctionnement de la partie opérative. La méthodologie de simulation élaborée est appliquée à une autre architecture.

Puis nous traitons, dans le troisième chapitre, la simulation de la partie contrôle par la méthodologie de simulation du chapitre précédent.

Le quatrième et dernier chapitre est consacré à la simulation de la machine de communication.

Plusieurs méthodes de recherche des racines d'un polynôme, telles que la méthode de Müller, la méthode de Laguerre, la méthode de Newton ou sa variante la méthode de Bairstow, convergent rapidement si de bonnes approximations initiales des racines sont connues. Si de telles approximations ne sont pas connues, la convergence peut être plus lente ou, dans certains cas, peut ne pas avoir lieu. En plus les méthodes mentionnées donnent une seule racine à chaque fois; lorsqu'une racine est trouvée, la méthode est à nouveau utilisée avec un point de départ choisi aléatoirement.

D'autres méthodes, telles que celles de Bernouilli et Graeffe ne nécessitent aucune approximation initiale, mais la convergence est souvent lente et aussi seulement une (ou deux) racine sont trouvées à chaque fois.

D'autres méthodes comme la méthode Q-D (Quotient-Différence) introduite par Rutishauser, la méthode de Durand-Kerner et la méthode d'Ehrlich ont l'avantage de donner simultanément toutes les racines d'un polynôme donné, en utilisant seulement ses coefficients.

Théoriquement la méthode Q-D peut être utilisée pour déterminer les racines d'un polynôme avec une précision arbitraire; cependant, puisque la méthode converge au moins aussi lentement que celle de Bernouilli, et puisqu'elle propage les erreurs d'arrondi, elle est utilisée uniquement pour fournir les approximations initiales des racines pour la méthode de Newton pour une convergence plus rapide.

Dans la deuxième partie de cette thèse, notre étude porte sur la méthode de Durand-Kerner et la méthode d'Ehrlich. Celles-ci sont:

. méthode de Durand-Kerner:

$$z_r^{k+1} = z_r^k - P(z_r^k) / Q'(z_r^k) \qquad r=1, \dots, n$$

où $Q(z) = \prod_{r=1}^{n} (z - z_r^k)$ et P un polynôme de degré n à coefficients complexes.

. méthode d'Ehrlich:

$$z_r^{k+1} = z_r^k + \delta_r .(1 + \delta_r . \beta_r)^{-1} \qquad r=1, \dots, n$$

où $\delta_r = -P(z_r^k) / P'(z_r^k)$ est le terme de correction de Newton en z_r^k et $\beta_r = \sum_{i \neq r} (z_r^k - z_i^k)^{-1}$.

Dans le premier chapitre nous étudions les propriétés théoriques et pratiques des deux méthodes de base de Durand-Kerner et d' Ehrlich pour lesquelles nous introduisons cinq variantes: méthodes de Durand-kerner modifiées et méthodes d'Ehrlich modifiées. L'étude comparative de ces méthodes montre que celles de Durand-Kerner et d'Ehrlich en mode parallèle (approche de Jacobi) et en mode série (approche de Gauss-Seidel) sont les plus performantes car elles font la synthèse d'un ordre de convergence assez élevé et d'une itération peu coûteuse.

Le deuxième chapitre traite la convergence de la méthode de Durand-Kerner. Un des résultats obtenus est l'équivalence de celle-ci et de la méthode de Newton.

Le troisième chapitre est consacré à l'adaptation d'une architecture parallèle au calcul des itérations de Durand-Kerner.

Dans le quatrième et dernier chapitre nous menons une expérimentation sur une architecture vectorielle (FPS264: Floating Point Systems) des méthodes étudiées dans le premier chapitre. La vectorisation des algorithmes de la méthode de Durand-Kerner en mode parallèle et en mode série permet d'obtenir une vitesse de traitement trois fois meilleure que celle des versions non vectorisées.

Première partie

SIMULATION

DU FONCTIONNEMENT LOGIQUE DE FELIN

Chapitre1

INTRODUCTION A FELIN

1.1 Coprocesseur: Historique et Intérêt:

Dans les dernières années, la technologie des semi-conducteurs, est arrivée à un degré de maturité telle qu'il est devenu possible de fabriquer des circuits comprenant plusieurs milliers de transistors, réalisant des fonctions de plus en plus complexes.

Les fabriquants de composants éléctroniques ont été conduits à imaginer des circuits s'adaptant à un nombre d'applications aussi grand que possible, mais restant utilisables en petite série par de nombreux utilisateurs.

L'aboutissement de cette tendance fut le circuit logique programmable par l'utilisateur, circuit appelé "microprocesseur".

Dans le cas où les microprocesseurs ont un emploi trés général, un circuit spécialisé appelé "coprocesseur" lui est intégré pour réaliser le traitement d'une fonction particulière (calcul scientifique, gestion des entrées et des sorties, ...).

Actuellement Les coprocesseurs 8087 d'Intel et MC68881 de Motorola associés respectivement aux microprocesseurs 8086/8088 et MC68020 sont les plus couramment implantés dans la plus part des micro-ordinateurs et dans de nombreuses cartes systèmes.

1.2 Principe général de FELIN:

FELIN (Fonctions ELémentaires INtégrées) est un coprocesseur arithmétique de calcul rapide de fonctions mathématiques de base telles que: Sinus, Exponentielle, Arcsinus, Argth, ... Il est composé de deux machines travaillant en parallèle : machine de communication et machine à calcul [HAM85] (figure 1.a.). Il est destiné à traiter des expressions arithmétiques complètes sans intervention du processeur hôte.

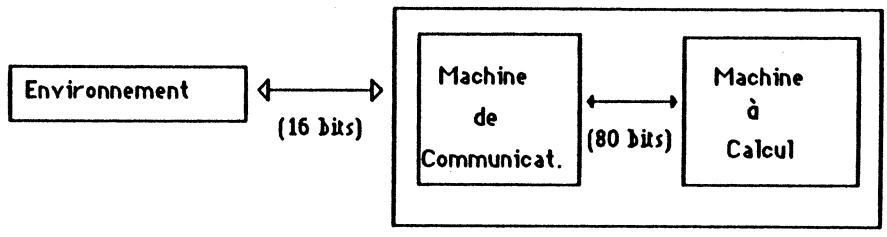


Figure 1.a.

Le coprocesseur arithmétique FELIN est le fruit de la collaboration entre les équipes d'Algorithmique Parallèle et d'Architecture des Ordinateurs du laboratoire TIM3.

Les algorithmes de calcul implémentés dans FELIN sont étudiés en détail par J.M.Muller. La partie opérative et la partie contrôle de FELIN sont conçues et élaborées respectivement par B.Hochet et E.Zysman. Les algorithmes et l'architecture de sa machine de communication sont étudiés par J.Duprat. Dans le présent travail notre étude est consacrée à la simulation du fonctionnement logique du coprocesseur FELIN.

1.2.1 Machine de Communication:

Elle gère les entrées et les sorties réalisant ainsi l'interface entre l'environnement extérieur et la machine à calcul (figure1.b.). Le processeur lui envoie les opérandes et les opérateurs à traiter en utilisant une notation polonaise inversée. Elle génère alors les séquences d'exécution pour la machine à calcul.

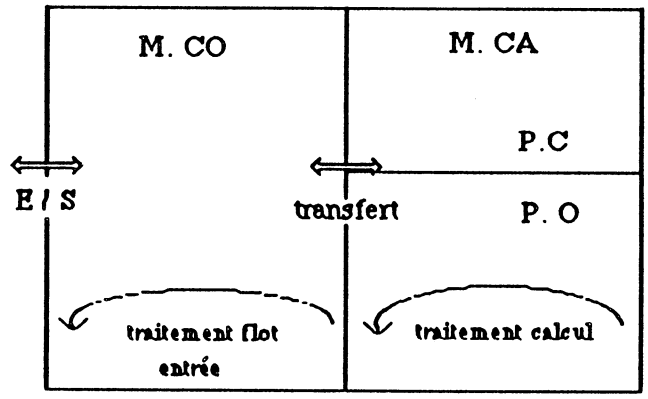


Figure 1.b.

1.2.2 Machine à Calcul:

Elle effectue les calculs. Elle est constituée de deux parties: partie contrôle et partie opérative. La partie contrôle génère des commandes qui par leurs déroulements font fonctionner la partie opérative. Celle-ci se charge du calcul proprement dit (figure 1.b.).

1.3. Algorithmes de calcul de fonctions élémentaires:

L'importance du calcul des fonctions mathématiques de base dans le domaine scientifique est telle que la recherche d'une implémentation satisfaisante de ces fonctions a fait l'objet d'un grand nombre d'études.

Au début, l'implémentation de ces fonctions a été conçue de manière logicielle. Cette conception est basée sur des approximations par des polynômes ou des fractions rationnelles [COD80].

Les possibilités croissantes de l'intégration sur silicium ont ouvert le champ à une réalisation matérielle [BRE76].

En 1959, Volder [VOL59] a proposé l'algorithme CORDIC (Cordinate Rotation on a Digital Computer) pour le calcul des fonctions: sinus, cosinus, et arctangente. Cet algorithme fait intervenir uniquement des additions et des multiplications par des puissances entières de 2.

Plus tard, Walther [WAL71] a pu étendre l'algorithme CORDIC au calcul des fonctions hyperboliques ainsi qu'à l'exponentielle, au logarithme et à la racine-carrée.

L'algorithme CORDIC, modifié par Walther est basé sur l'itération suivante:

$$X_{n+1} = X_n - m\,\zeta_{Zn}\,Y_n.2^{-n}$$

$$Y_{n+1} = Y_n + \zeta_{Zn}\,X_n.2^{-n}$$

$$Z_{n+1} = Z_n - \zeta_{Zn}\,\varepsilon_n$$

où le paramètre m, les variables ζ_{Zn} et les constantes ε_n sont présentées en figure1.c.

J.M. Muller [MUL85], à partir de ces algorithmes, a défini une méthodologie de calcul des fonctions élémentaires. L'outil mathématique de base utilisé est la notion de base discrète qui est une généralisation de la notion de base de numération. Tous ces algorithmes sont construits à partir d'une approximation de l'argument de la fonction que l'on désire calculer par une expression de la forme:

$d_0 e_0 + d_1 e_1 + ... + d_n e_n$ où $d_i \in \{0,1\}$ et $\{e_i\}$ une base discrète [MUL85].

Fonction	Choix de δ^x et δ^y	Valeurs initiales	Domain	Résultats	Error
Exponentielle complexe	If $x_k \geq \varepsilon_k^x$ then $\delta^x = 1$ else 0 If $y_k \geq 0$ then $\delta^y = 1$ else -1	$a_0 = k_0$ $b_0 = 0$ $x_0\, y_0$ given	$x_0 + iy \in D$	$a_N + b_N = e^{x_0 + iy_0}$	erreur relative
Sinus et Cosinus	$\delta^x = 0$ If $y_k \geq 0$ then $\delta^y = 1$ else -1	$a_0 = k_0$ $b_0 = 0$ $x_0 = 0$ y_0 given	$-E_y \leq y_0 \leq E_y$	$a_N = \cos y_0$ $b_N = \sin y_0$	erreur absolue
Exponetielle	$\delta^y = 0$ If $x_k \geq \varepsilon_k^x$ then $\delta^x = 1$ else 0	$a_0 = 1$ $b_0 = 0$ x_0 given	$0 \leq x \leq E_x$	$a_N = e^{x_0}$	erreur relative
Logarithme	If $a_k + 2^{-k} a_k \leq a$ then $\delta^x = 1$ else 0	$a_0 = 1$ a given $b_0 = 0$ $x_0 = 0$	$1 \leq a \leq e^{E_x}$	$x_N = \ln a$	erreur absolue
Arctangente	If $b_k > b$ then $\delta^y = 1$ else -1	$a_0 = k_0$ $b_0 = 0$ b given $y_0 = 0$	$b \in R$	$y_N = \arctan b$	erreur absolue

Itération :

$$a_{n+1} = (1+2^{-n})^{\delta_n^x} . (a_n - b_n \delta_n^y 2^{-n})$$

$$b_{n+1} = (1+2^{-n})^{\delta_n^x} . (b_n + a_n \delta_n^y 2^{-n})$$

$$x_{n+1} = x_n - \delta_n^x \varepsilon_n^x \qquad y_{n+1} = y_n - \delta_n^y \varepsilon_n^y$$

De nombreux algorithmes de calcul de fonctions élémentaires sont obtenus(figure 1.c.). Ces algorithmes englobent ceux de CORDIC et se prêtent bien à une réalisation matérielle car ils ne sont basés que sur les deux primitives suivantes: addition et décalage. L'exemple suivant de calcul de la fonction élémentaire exponentielle illustre cette étude.

Dans [MUL85], il est montré qu'un réel x peut être décomposé dans la base discrète $\{\ln(1+2^{-n})\}$. On obtient:

$$x = d_0 \ln(1+2^{-0}) + d_1 \ln(1+2^{-1}) + \ldots + d_n \ln(1+2^{-n}) + \ldots .$$

alors l'exponentielle de x s'écrit:

$$e^x = (1+2^{-0})^{d_0} . (1+2^{-1})^{d_1} \ldots (1+2^{-n})^{d_n} . \ldots \text{ où } d_i \in \{0,1\}.$$

Cette expression pourra être aisément évaluée car son calcul se ramène au calcul des additions et des décalages.

A l'aide de cet outil puissant que sont les bases discrètes, les autres fonctions élémentaires s'obtiennent facilement.

Pour calculer une fonction élémentaire sur son domaine de définition I nous procédons en trois étapes:

- On réduit l'argument au domaine de convergence J de l 'algorithme,
- On calcule la fonction sur l'argument réduit,
- On restitue le résultat pour obtenir la valeur finale de f.

Pour rammener l'argument au domaine de convergence de l'algorithme, deux méthodes peuvent être utilisées suivant les propriétés de la fonction:

.La réduction multiplicative : soit un réel x élément de I, alors on cherche un entier n tel que $x * 4^{-n}$ appartient à J. Cet algorithme s'applique pour les fonctions racine-carrée et logarithme.

.La réduction additive: soit un réel x élément de I, alors on cherche un entier n tel que x-n*C appartient à J. Beaucoup d'algorithmes ont été étudiés. Nous avons retenu l'algorithme suivant qui est implémenté dans le coprocesseur FELIN et qui a été proposé par J.M. Muller [MUL85].

Début

y : = x ;

k : = $\lfloor \log_2 (x) \rfloor$;

pour i : = k jusqu'à 0 faire y : = y - signe (y) C. 2^i;

Fin.

où C est une constante positive dépendant de la fonction calculée (par exemple $\pi/2$ pour les fonctions trigonométriques sinus et cosinus).

Cet algorithme traite la réduction d'argument des fonctions élémentaires telles que: exponentielle, cosinus, sinus .... Il est important de noter qu'il est du même type que ceux de calcul des fonctions élémentaires. Le temps nécessaire à son exécution est proportionnel à $\log_2 x$. Il est de l'ordre de $(2/\ln 2) * \ln(x)$.

1.4 Machine de communication:

Les algorithmes de la machine de communication [DUP86] [PAU86] consistent à traiter l'entrée des données qui lui sont envoyées par le processeur hôte en notation polonaise inversée et la sortie des résultats, le transfert des registres d'états (registres d'entrée) dans les 2 RAMs des opérandes et des instructions, le calcul et enfin le transfert des résultats dans les registres de sortie.

La gestion des données transférées dans les 2 RAMs nécessite 2 files : file attente opérande et file attente instruction. Ces données doivent être ensuite traitées et envoyées à la machine à calcul; ceci est réalisé par une pile de calcul. La gestion des résultats de calcul et des comptes rendus nécessite 2 autres files: file attente compte rendu dans la RAM des instructions et file attente resultat dans la RAM des opérandes. Ces 2 RAMs constituent la partie opérative de la machine de communication.

La gestion de l'ensemble des files et pile se ramène à la gestion de 7 registres que l'on fait correspondre à la tête et à la queue de chacune des files et pile.
L'ensemble de toutes ces parties est géré par un contrôleur global qui constitue la partie contrôle de la machine de communication.

1.5 Partie contrôle de la machine à calcul:

Les fonctions à traiter sont de deux sortes: les fonctions élémentaires (exemple: cosinus, sinus,...) et les fonctions composées (exemple: fonction puissance, tangente,...). Le traitement de celles-ci se fait par une décomposition préalable en fonctions simple. Par exemple: y^x sera décomposée en trois fonctions simples:

- calcul de ln (y),
- calcul de x * (ln (y)),
- et calcul de exponentielle de (x* ln (y)).

Pour les fonctions simples, le calcul fait intervenir un certain nombre d'opérations telles que la traduction de formats, la réduction d'arguments, le calcul proprement dit, ... etc.

Les algorithmes de calcul sont décrits sur trois niveaux d'interprétation:

a) Le premier niveau est divisé en 2 étapes:

. La première extraît des macroétapes traduisant la décomposition d'une fonction;

. La seconde associe à chaque macroétape un certain nombre de paramètres utilisés par le niveau inférieur.

b) Le second niveau génère une suite de procédures correspondant à l'interprétation des paramètres du niveau supérieur.

c) Le troisième niveau correspond au déroulement des procédures précédentes.

Le premier et le second niveau d'interprétation sont intégrés dans un même bloc fonctionnel, conçu et élaboré par Zysman [ZYS87] avec dix registres attribués aux variables algorithmiques décrites plus haut (fonction, macroétapes, paramètres,..), deux incrémenteurs et deux PLAs: un PLA de séléction qui séléctionne les différents registres en lecture et en écriture et un PLA de valeurs qui calcule les paramètres et contrôle le séquencement de la machine elle-même.

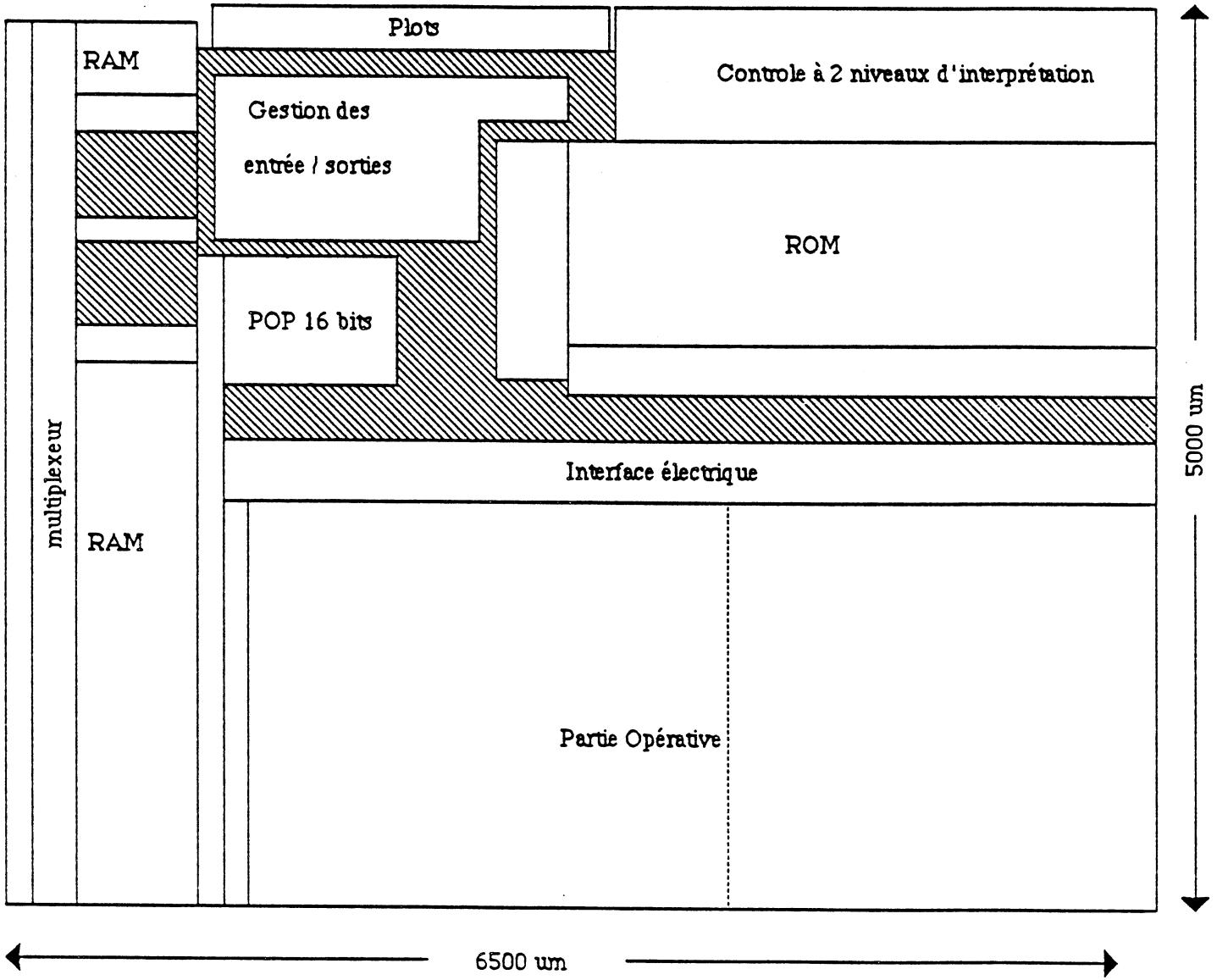
Le troisième niveau est intégré dans une ROM où sont placés tous les algorithmes micro-programmés [ZYS87] .

1.6 Partie opérative de la machine à calcul:

Elle constitue la deuxième partie de la machine à calcul et représente un moyen permettant l'éxécution des opérations qui interviennent dans le traitement d'une fonction élémentaire: traduction de format d'entrée en format flottant 80 bits, traitement des exceptions, ..., calcul proprement dit, traduction du résultat en format de sortie.

L'implémentation de ces primitives de calcul nécessite deux types d'opérateurs: l'addition et le décalage. Ceux-ci doivent par ailleurs être associés à quelques registres et une R.O.M contenant les contantes nécessaires aux algorithmes (exemple: $\ln(1+2^{-n})$, ...).

L'architecture de la partie opérative, étudiée par Hochet [HOC87] est une structure pipeline où l'additionneur et le décaleur travaillent en parallèle. La partie opérative est subdivisée en deux parties égales de 64 bits disposées l'une à coté de l'autre et communiquent entre elles à travers le décaleur. Seule la partie 'poids forts' communique avec l'extérieur. Par la figure 1.d. nous présentons la plan de masse du



Plan de masse du circuit

circuit.

1.7 Simulation:

Avant la fabrication d'un prototype, il faut que son fonctionnement logique soit vérifié au préalable. L'outil le plus utilisé, par les constructeurs est la simulation [BOR76] [NAY68].

Dans notre contexte, la simulation peut-être définie comme étant une méthode numérique pour conduire des expériences sur un ordinateur à l'aide d'une représentation simplifiée du système à étudier [NAY68] [FEU78].

1.7.1 But:

La technique de notre simulation est utilisée dans un triple but: l'analyse, la construction et la vérification du fonctionnement logique de FELIN. En fait au cours des différentes étapes de notre simulation sont réalisées des améliorations et des ajustements correctifs de l'architecture aussi bien que des algorithmes déja conçus au départ. Ces réalisations sont au fur et à mesure analysées de manière à vérifier son fonctionnement .

1.7.2 Principe:

Notre méthode de simulation est compsée de trois étapes (figure 1.e.):

. Représentation simplifiée:

Le problème est la vérification du fonctionnement logique de FELIN. Pour celà une représentation simplifiée est élaborée. Cette représentation consiste tout d'abord à donner une description architecturale et algorithmique. Les algorithmes sont ensuite transformés de manière qu'ils soient adaptés à l'architecture et puis vérifiées.

.Programmation sur ordinateur:

La programmation débute par la mise sur pieds de l'organigramme qui décrit le déroulement logique des algorithmes adaptés. Cette étape initiale est la plus importante. Le codage en effet n'est qu'une transposition de l'organigramme qui décrit le déroulement du programme, celui-ci s'exécute. Les résultats obtenus constituent la deuxième étape de la vérification du bon déroulement de l'organigramme.

.Analyse des résultats:

Les résultats de programmation obtenus sont présentés dans des tableaux. Ils sont analysés et comparés aux valeurs réelles ($|f(x)-f_n(x)| \leq 2^{-n}$, ...) . Suivant l'éfficacité des résultats, il peut y avoir des retours en arrière pour ajustement et changement nécessaires soit des données, soit des algorithmes ou de l'architecture.

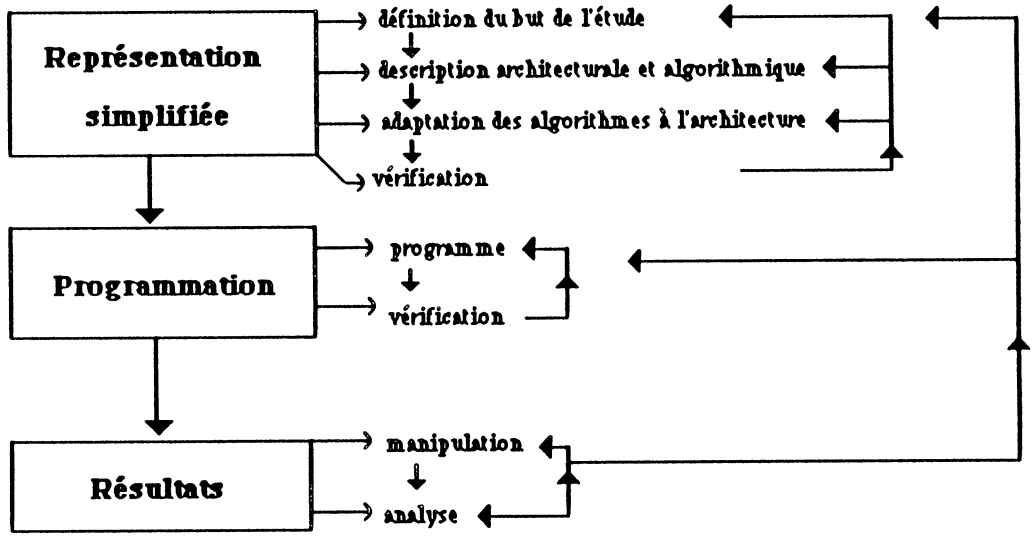


Figure 1.e. étapes de notre méthode de simulation.

1.7.3 Outils:

Notre simulation est menée à l'aide d'un certain nombre d'outils d'ordre:

- Informatique: fonctionnement en mode pipeline.
- Eléctrique: un cycle d'horloge est divisé en trois phases.
- Méthodique: les algorithmes sont transformés pour qu'ils soient adaptés à l'architecture.

Chapitre 2

SIMULATION DE LA PARTIE OPERATIVE

2.1 Introduction:

La machine à calcul est divisée en 2 parties: la partie opérative et la partie contrôle. celle-ci génère des commandes qui par leur déroulement font fonctionner la partie opérative qui se charge du calcul proprement dit en exécutant chaque commande.

2.2 Description algorithmique:

Les algorithmes de calcul de fonctions élémentaires [MUL85] qui sont intégrés dans le coprocesseur arithmétique FELIN sont divisés en 3 classes. La première classe, appelée classe bidirectionnelle [MUL85], contient quatre algorithmes: cosinus, sinus, arctangente et division. La deuxième, classe unidirectionnelle, comporte: exponentielle, logarithme et racine-carrée. la dernière contient l'algorithme de calcul de la multiplication.

Dans ce qui suit nous décrivons un algorithme de chacune des trois classes.

a) Classe bidirectionnelle : calcul de l' arctangente.

La fonction arctangente converge sur l' intervalle $]-\infty, +\infty[$. Son algorithme est le suivant: les variables X, Y et Z sont respectivement initialisées aux valeurs 0, t (argument de la fonction) et 1. Ensuite le calcul est éffectué par la boucle de la représentation de l'algorithme, ci-dessus:

$X := 0, Y := t, Z := 1;$
Pour i:=0 jusqu'à N faire
Début
$X := X + d_Y \mu_i;$
$Z := Z - d_Y Y.2^{-i};$
$Y := Y + d_Y Z.2^{-i};$
d_Y:=signe(Y);
Fin.

où μ_i=Arctan(2^{-i}), sont des constantes précalculées. Le résultat final du calcul se trouve dans X.

Les algorithmes de calcul des autres fonctions sont du même type que celui du calcul du l'arctangente.

b) Classe unidirectionnelle: calcul de l'exponentielle.

Le domaine de convergence de la fonction exponentielle est [0, ln4]. L'algorithme de calcul de cette fonction procède par l'initialisation des variables X et Z aux valeurs 1 et t (argument) respectivement, suivie du calcul effectué par la boucle de la représentation de l'algorithme, ci-dessus:

```
X := 1, Z :=t;

Pour i:=0 jusqu'à N faire
Début
    si Z > μi alors
    Début
        Z := Z - μi;
        X := X + X.2^-i;
    Fin;
Fin.
```

où μ_i=ln($1+2^{-i}$), sont des constantes précalculées. Le résultat final du calcul se trouve dans X.

c) Calcul de la multiplication:

Pour l'algorithme de calcul de la multiplication , une solution plus performante a été retenue. Celle-ci consiste à traiter l'un des opérandes en base 4.

Soient X et Y deux réels, on peut les écrire en représentation virgule flottante sous la forme suivante:

$X=(+/-)m_X.2^I$ et $Y=(+/-)m_Y.2^J$ où m_X et m_Y, éléments de [0.5,1[, sont respectivement les mantisses de X et de Y et I et J sont des entiers relatifs.
Le calcul de $X*Y=(+/-)(m_X*m_Y).2^{(I+J)}$ se ramène à l'évaluation de m_X*m_Y.

$$m_X*m_Y=(\sum_{1}^{32} x_j.2^{-j}).m_Y + 2^{-32}(\sum_{1}^{32} x'_j.2^{-j}).m_Y \text{ où } x'_j=x_{j+32} \text{ et } x_j \in \{0,1\}.$$

Le calcul de $(\Sigma x'_j.2^{-j}).m_Y$ se fait de la même façon que celui de $(\Sigma x_j.2^{-j}).m_Y$; nous traitons ce dernier.

$$(\Sigma x_j.2^{-j}).m_Y = [\,[\ldots[\,(x_{32}+2*x_{31}).m_Y)].2^{-2} + (x_{30}+2*x_{29}).m_Y)].2^{-2} + \ldots + (x_2+2*x_1).m_Y)].2^{-2}$$

$$= [\,[\ldots[\,(\mu_{16}.m_Y)].2^{-2} + (\mu_{12}.m_Y)].2^{-2} + \ldots + (\mu_1.m_Y)].2^{-2}$$

où $\mu_j = x_{2j} + 2.x_{2j-1}$, $1 \leq j \leq 16$.

Ce qui peut s'écrire sous la forme suivante:

$$(\Sigma x_j.2^{-j}).m_Y = [\,[\ldots[\,(\mu_{16}.m_Y)].2^{-4} + (\mu_{14}.m_Y)].2^{-4} + \ldots + (\mu_2.m_Y)].2^{-4}] +$$

$$[\,[\ldots[\,(\mu_{12}.m_Y)].2^{-4} + (\mu_{13}.m_Y)].2^{-4} + \ldots + (\mu_3.m_Y)].2^{-4} + (\mu_1.m_Y)]].2^{-2}.$$

L'algorithme de calcul de la multiplication procède par la représentation de la mantisse de X, m_X, en base 4 puis par l'exécution de $m_X * m_Y$ en multipliant m_Y par la partie de m_X correspondant aux exposants impaires, en multipliant m_Y par la partie de m_X correspondant aux exposants paires et ensuite en aditionnant les deux résultats obtenus.

2.3 Description architecturale

La partie opérative [HOC87] est constituée de deux blocs appelés partie mantisse et partie exposant (figure1).

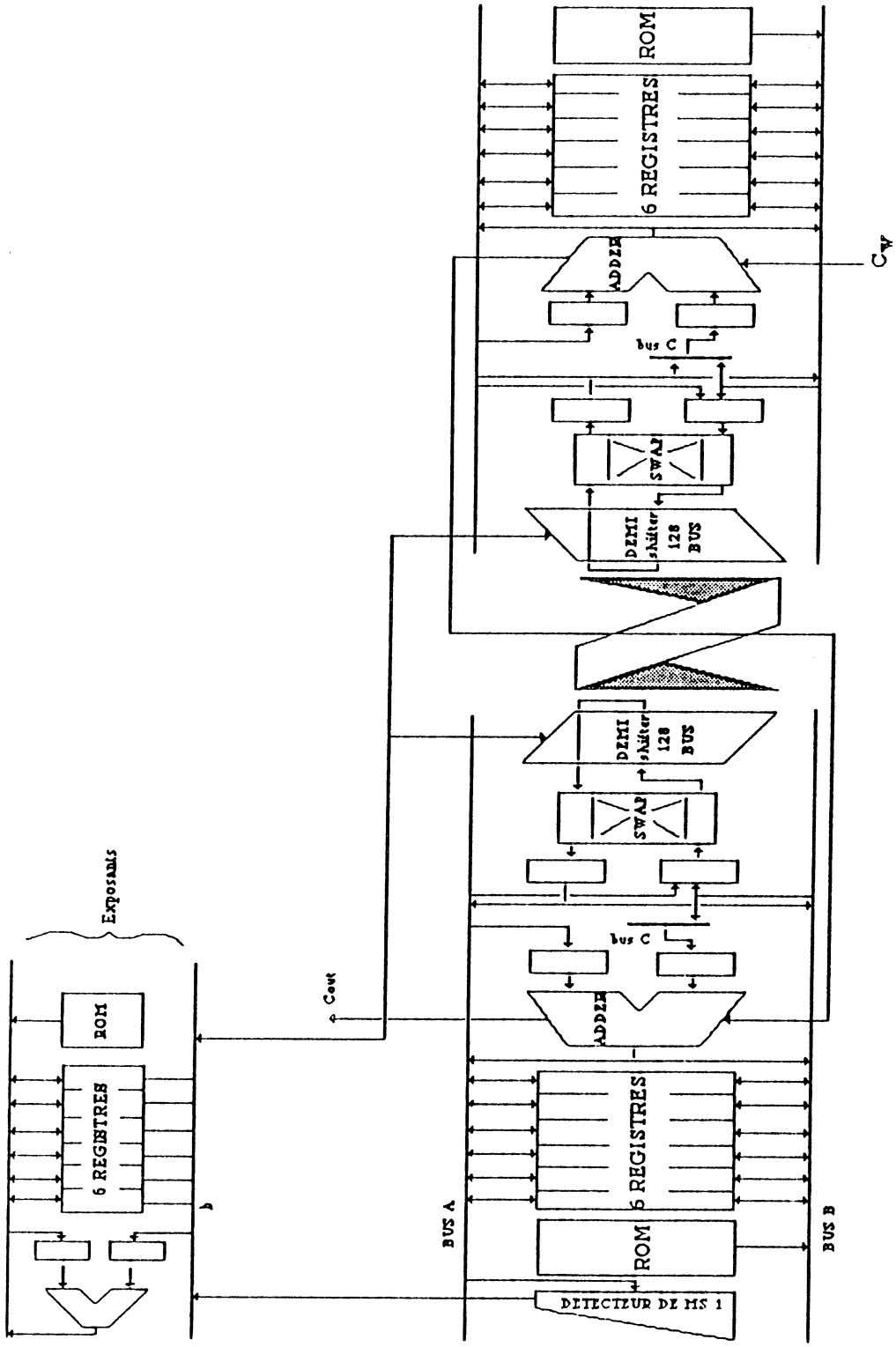
La partie mantisse est constituée des éléments de base suivants:

- Une mémoire morte R.O.M. (Read Only Memory) [ANC82] dans laquelle sont implémentées toutes les constantes nécessaires au calcul des fonctions élémentaires ($(\ln(1+2^{-I})$; $\mathrm{Arctan}(2^{-I})$, I=0,1,...,64; 1fixe; ...).
- Six registres X,Y,Z,Q,I1 et I2 dont les trois derniers servent pour le calcul intermédiaire.

La ROM et les 6 registres communiquent avec les autres éléments par les deux bus A et B.

-Une unité arithmétique et logique : U.A.L., de laquelle sortent deux fils qui écrivent le résultat de l'UAL, l'un dans le bus A et l'autre dans le bus B dans l'un des six registres ci-dessus.
- Un décaleur d'un nombre quelconque de positions: SHIFT.

- Un registre tampon T qui reçoit le résultat sortant du décaleur.



PARTIE OPERATIVE DU CIRCUIT.

- Trois registres intermédiaires Ual1, Ual2, Ual3 dont les deux premiers sont liés à l' entrée de l'additionneur et le troisième à l'entrée du décaleur.
- Le bus C qui lie le registre tampon T et le bus B au registre Ual2.
- Un détecteur de zéros ZLD (Zero Lower Detector) qui détermine la position du premier "1" rencontré dans l'écriture d'une valeur en base deux en commençant par les bits de poids forts.

La partie opérative est subdivisée en deux parties égales de 64 bits disposées l'une à côté de l'autre et communiquant entre elles à travers le décaleur.

La partie exposant est constituée des éléments de base suivants:

- Une R.O.M. où sont implémentées toutes les constantes nécessaires au calculs (19,-14,...).
-Six registres X,Y, Z, Q,I 1, I2 dont les trois derniers servent au calcul intermédiaire.
- Une unité arithmétique et logique: U.A.L.

La ROM et les 6 registres communiquent avec les deux registres Ual1 et Ual2 liés à l' entrée de l'additionneur par les deux bus a et b.

La partie exposant est liée à la partie mantisse par 2 bus : le premier arrive du détecteur de zéros, ZLD, au bus a. De celui-ci part le second vers le décaleur.

2.4 Outils de simulation:

La simulation du fonctionnement de la partie opérative est menée à base d'un certain nombre d'outils qui interviennent au cours des étapes successives suivantes: simulation de l'architecture, exécution des algorithmes sur le simulateur de l'architecture et vérification des résultats.

En effet nous avons introduit le mode pipeline dans le fonctionnement de la partie opérative. Pour ce mode, nous avons défini le cycle de base que nous avons découpé en trois phases: phase de lecture, phase de traitement et phase d'écriture. Par ailleurs, nous avons traduit la fonction des éléments de base du circuit par des fonctions mathématiques et introduit la relation $|f(x) - f_n(x)| \leq 2^{-n}$ pour la vérification des résultats de l'exécution des algorithmes sur le simulateur de l'architecture.

La forme des algorithmes fait qu' on se ramène à chaque fois au calcul de l'expression $a +/- b.2^{-i}$ où a et b sont des réels et i est un entier relatif. Nous avons entre 2 et 4 opérations de ce type à exécuter pour chaque itération des algorithmes. Il est par suite interessant d' effectuer ces opérations en parallèle. Cela revient à réaliser un bon séquencement de la partie opérative. L'exemple de l' algorithme de calcul de sinus et cosinus illustre ceci:

$$X_{n+1} = X_n - d_n Y.2^{-n} \qquad Y_{n+1} = Y_n + d_n X_n.2^{-n}$$

$$Z_{n+1} = Z_n - d_n \text{Arctg}(2^{-n})$$

En fait le calcul de X_{n+1} nécessite X_n et $Y_n.2^{-n}$. Ensuite pour le calcul de Y_{n+1} nous avons besoin de Y_n et $X_n.2^{-n}$. On remarque que le calcul simultané de X_{n+1} et de $X_n.2^{-n}$ fait gagner du temps. Pour cela un fonctionnement en *mode pipeline* [HWA84] est choisi de manière que l'additionneur et le décaleur travaillent en parallèle. Dans ce mode une opération élémentaire du type a +/- b.2^{-i} est exécutée en un *cycle machine* qui comporte trois *phases* [SAU] : phase de lecture, phase de traitement (addition et décalage) et phase d'écriture.

Pour traduire le fonctionnement de la partie opérative au cours des trois phases nous utilisons les *fonctions mathématiques* suivantes:

SHIFT: $\mathbb{R}X\mathbb{Z} \to \mathbb{R}$: $(X, i) \mapsto X.2^i$ pour le décaleur, ADDSUB : $\mathbb{R} X \mathbb{R} \to \mathbb{R}$: $(X,Y) \mapsto Z$ telle que $Z = X+Y$ pour l'additionneur et AFFECTE2:$(X,BUS,Y) \mapsto (Y := X)$ pour le transfert entre 2 registres par un bus.

L'exécution des algorithmes sur le simulateur de l'architecture se fait en tenant compte de leur découpage en phases décrit dans des tableaux, appelés *tableaux de cycles.*

A la fin de chaque exécution, les résultats de la simulation sont vérifiés au moyen de la relation $/ f(x)-f_n(x)/ \leq 2^{-n}$ [MUL85] où $f_n(x)$ est la valeur de la fonction au point x à la $n^{ième}$ itération et f(x) la valeur exacte.

2.5 Tableaux de cycles:

Les algorithmes de calcul des fonctions élémentaires sont transformés et adaptés à l'architecture de la partie opérative. Le déroulement d'une itération de ces algorithmes est traduit par des tableaux, appelés tableaux de cycles et dont le contenu est illustré pour le cas des algorithmes: arctangente et exponentielle (tableaux 1 et 2). Nous expliquons ces tableaux de cycles en traitant l'exemple de l'algorithme arctangente. Précisons quelques notations: la phase de lecture, la phase de traitement (addition et décalage) et celle d'écriture sont respectivement notées par phase1, phase2 et phase3 et l'ensemble des trois phases constituent un cycle. Une itération de l' algorithme de calcul de la fonction arctangente nécessite trois cycles. Au cours de la première phase du premier cycle, on envoie la valeur du registre X ($=X_n$) par le bus A dans le registre Ual1 et la valeur du registre Y ($=Y_n$) par le bus B dans Ual3. Puis on transfère la valeur $\mu_n.2^{-n}$ (μ_n est décalée vers la droite de n

$t \in]-\infty,+\infty[$ **Algorithme (ARCTAN)**

t Affecte → X
X → Y
0 → X
1 → Z

$$\begin{cases} X = X + d_Y \mu_I * 2^{-I} \\ Z = Z - d_Y Y * 2^{-I} \\ Y = Y + d_Y Z * 2^{-I} \end{cases}$$

$$\begin{cases} d_Y = \text{signe}(Y). \\ \mu_I = 2^I * \text{ARCTAN}(2^{-I}). \end{cases}$$

$X \longleftarrow \text{ARCTAN}(X).$

ARCTAN	phase 1	phase 2	phase 3
cycle 1	.X $\xrightarrow{(A)}$ ual1 .Y $\xrightarrow{(B)}$ ual3 .T $\xrightarrow{(C)}$ ual2	.ual1 $\oplus_Z$ ual2 .shift(ual3,1)	.ual1 $\oplus_Z$ ual2 $\xrightarrow[(B)]{(A)}$ X
cycle 2	.Z → ual1 .Z → ual3 .T → ual2	.ual1 $\oplus_Z$ ual2 .shift(ual3,1)	.ual1 $\oplus_Z$ ual2 → Z
cycle 3	.Y → ual1 .μ → ual3 .T → ual2	.ual1 $\oplus_Z$ ual2 . shift(ual3,I+1)	.ual1 $\oplus_Z$ ual2 → Y

$\oplus$ = (+/-).signe(Y)

Tableau 1.

tableau 1

$t \in [0, LN(4)]$

t Affecte → X
X → Z 1 → X

Algorithme (EXPONENTIELLE)

POUR I:=1 JUSQU'A N FAIRE
DEBUT
  SI Z>μ ALORS
  DEBUT
    Z:=Z-μ;
    X:=X+X*2^-I ;
  FIN
FIN

$\mu_I = LN(1+2^{-I})$

$X \leftarrow Exp(X)$.

EXP	phase 1	phase 2	phase 3
cycle 1	.Z —(A)→ val1 .X —(B)→ val3 .T —(C)→ val2	.val1 - val2 .shift(val3,I)	.val1 - val2 —(A)/(B)→ Q
cycle 2⁺	.X → val1 .μ → val3 .T → val2	.val1 + val2 .shift(val3,I)	.val1 + val2 → X
cycle 3⁺			.Q → Z
cycle 2⁻	. μ —(β)→ val3	. shift(val3,0)	

Tableau 2.

positions au cours du dernier cycle de la n ^{ième} itération) du registre T à Ual2 par le bus C. Dans la seconde phase, on calcule la somme des 2 valeurs de Ual1 et Ual2 (Ual1 +signe(Y) Ual2) et on décale vers la gauche la valeur Ual3 de n positions (le registre T contiendra alors la valeur $Y_n.2^{-n}$ qui servira au prochain cycle pour le calcul de la nouvelle valeur de Z ($=Z_{n+1}$)). Dans la troisième phase, on transfère le résultat de l'additionneur par les deux bus A et B dans le registre X.

Les trois transferts entre les registres se font simultanément ainsi que les deux traitements en phase2.

Au bout de ces trois phases, la nouvelle valeur de X (= X_{n+1}) est calculée. Le calcul de Y_{n+1} au cours du cycle 2 et celui de Z_{n+1} au cours du cycle3 se déroulent de la même manière que celui de X_{n+1}.

2.6 Simulation:

La simulation de la partie opérative pour le calcul de toutes les fonctions élémentaires consiste à décrire la fonction de chaque élément de base par une procédure que nous utilisons pour traduire le fonctionnement de la partie opérative au cours de chaque phase. L'enchainement de celles-ci constitue le simulateur de l'architecture. Sur celui-ci, nous exécutons les algorithmes en tenant compte de leurs découpages en phases. Les résultats obtenus de chaque exécution sont vérifiés au cours de chaque itération de calcul d'une fonction élémentaire.

Pour modéliser le fonctionnement des différentes parties du circuit, nous avons procédé ainsi:

Nous avons représenté les 11 registres X, Y, Z, Q, I1, I2, Ual1, Ual2, Ual3, T et UAL12 par un tableau Δ à 11 éléments, les 3 bus A,B et C par un autre tableau Σ à 3 éléments et enfin le bus qui transporte la valeur entière de décalage par une variable k.

Nous avons fait correspondre à chacune des trois phases une procédure. La procédure PHASE1(Δ,Σ) correspondant à la phase de lecture phase1 décrit trois transferts par trois bus d'un registre à un autre:

procédure PHASE1(Δ,Σ);
Début
AFFECTE2(Δ_9,Σ_1,Δ_1);
AFFECTE2(Δ_7,Σ_2,Δ_2);
AFFECTE2(Δ_8,Σ_{10},Δ_3);
Fin;

où AFFECTE2(Z,Y,X) est la procédure qui traduit l'affectation de X à Y puis cette dernière à Z.

La phase2, phase de traitement, est représentée par la procédure PHASE2(Δ,Σ,k) qui fait intervenir 2 autres procédures qui traduisent respectivement la fonction de l'additionneur et du décaleur:

procédure PHASE2(Δ,Σ,k) ;
Début
ADDSUB(Δ_7,Δ_8,Δ_{11});
SHIFT(Δ_{10},k);
Fin;

La phase3 est un transfert du résultat de l' additionneur par les deux bus A et B dans un registre. La procédure utilisée est :

procédure PHASE3(Δ,Σ);
Début
AFFECTE2(Δ_1,Σ_1,Δ_{11});
AFFECTE2(Δ_2,Σ_2,Δ_{11});
Fin;

Les variables de sortie de la procédure PHASE1 sont des variables d'entrée de la procédure PHASE2 et les variables de sortie de celle-ci sont des variables d'entrée pour PHASE3.

L' enchaînement des 3 procédures traduit le fonctionnement de la partie opérative au cours d' un cycle, et constitue le *simulateur de son architecture.*

La simulation logicielle de la partie opérative est décrite en transposant en langage PASCAL l'ensemble des procédures précédentes. Les procédures du simulateur sont basées essentiellement sur les 3 procédures suivantes:
ADDSUB (X ,Y ,Z), SHIFT(X, k) et AFFECTE2(Y, A, X) qui traduisent respectivement la fonction de l'additionneur, du décaleur et le transfert d'une donnée d' un registre à un autre par un bus.

Les procédures du logiciel de simulation constituent 3 groupes:
Le premier contient celles qui correspondent aux PHASE1, PHASE2 ,PHASE3 et CYCLE.

Le second est composé des 3 procédures principales qui traduisent le décalage, l'addition/soustraction et le transfert.

Le troisième , utilisé par le second groupe, est formé des procédures qui traitent la comparaison des mantisses et leur addition/soustraction.

Dans ce qui suit, nous présentons les types et les déclarations des variables utilisées ainsi que les procédures puis nous traitons l'exemple de calcul de la fonction sinus et cosinus.

```
Type
    mantfixe= record
       signe: byte;
       mantisse: array[1..128] of byte;
    end;

(* les données en partie mantisse sont représentées en virgule fixe sur 128 bits *)

    exposfixe= record
       signe: byte;
       exposant= array[1..12] of byte;
    end;

(* Les données en partie exposant sont représentées sur 12 bits *)

    expmantfixe=record
       part1: exposfixe;
       part2: mantfixe;
     end;

var
    registres :    array[1..10] of expmantfixe;
    busm :         array[1..3] of mantfixe;
    buse :         array[1..2] of exposfixe;
```

procédures et leur explication:

Groupe1:

procédure **ADDFIXE**(X,Y: mantfixe; var Z: mantfixe);
(* fait la somme de la mantisse de X et de celle de Y puis met le résultat dans la mantisse de Z*)

procédure **SUBFIXE**(X,Y:mantfixe; var Z: mantfixe);
(* calcule la différence entre la mantisse de X et celle de Y ensuite place le résultat dans la mantisse de Z*)

fonction **GEF**(X,Y: mantfixe): boolean;
(* GEF="true" si la mantisse de X est supérieure à celle de Y*)

procédure **DCBN**(C: real; var X:expmantfixe);
(* convertie de décimale en binaire*)

procédure **BNDC**(X: expmantfixe;var C: real);
(* convertie de binaire en décimale*)

Groupe2:

procédure **ADD**(X,Y: mantfixe; var Z: mantfixe);
(* calcule la somme de X et de Y et affecte le résultat à Z*)

procédure **SUB**(X,Y: mantfixe; var Z: mantfixe);
(* calcule la différence entre X et Y puis place le résultat dans Z*)

procédure **AFFECTE**(var Y: mantfixe; X: mantfixe);
(* affecte X à Y *)

procédure **SHIFTRIGHT** (X: mantfixe; k: integer; var Y: mantfixe);
(* décale à droite la mantisse de X de k positions et sort le résultat dans la mantisse de Y*)

procédure **SHIFTLEFT**(X: mantisse; k: integer; var Y: mantfixe);
(* décale à gauche la mantisse de X de k positions et sort le résultat dans la mantisse de Y*)

procédure **REDARGLR**(var X: mantfixe; var n: integer);
(* réduit l'argument X, des 2 fonctions ln et sqrt, à l'intervalle [1,4] *)

procédure **REDEXP**(var X: mantfixe; var n: integer);
(* réduit l' argument X, de la fonction exp, à l'intervalle [0,ln4]*)

procédure **REDCSN**(var X: mantfixe; var n: integer);
(* réduit l' argument des deux fonctions sinus et cosinus, à l'intervalle [-π/2,π/2]*)

procédure **LIRE**(var X: expmantfixe);
(*lit en entrée la valeur de X écrite en binaire*)

procédure **ECRIRE**(var X: expmantfixe);
(*écrit en sortie la valeur de x représenté en binaire*)

Groupe 3:

procédure **SUBADD**(X,Y: mantfixe; CRCT: char; var Z: mantfixe);
(* calcule (X+Y) si CRCT='+' sinon (X-Y) et place le résultat dans Z*)

procédure **SUBADDCOND**(X,Y,Z: mantfixe; CRCT: char; var t: mantfixe*)
(*calcule (X+Y) si CRCT='+'('-') et signe de Z='+'('-') sinon (X-Y) et affecte le résultat à Z*)

procédure **PHASE1**(X,Y,T: mantfixe;B:busm;var Ual1,Ual2,Ual3: mantfixe);
(* affecte les valeurs X,Y et T respectivement à Ual1,Ual3 et Ual2 par les 3 bus A,B et C*)

procédure **PHASE2**(Ual1,Ual2,Ual3: mantfixe;K: integer; var t,Ual12: mantfixe);
(*calcule (Ual1+Ual2) et place le résultat dans Ual12 puis décale Ual3 de k positions et sort le résultat dans t*)

procédure **PHASE3**(Ual12,A,B: mantfixe; var X: mantfixe);
(* affecte la valeur de Ual12 à X par les 2 bus A et B*)

procédure **CYCLE**(var X,T: mantfixe; Y: mantfixe;k:integer);
(* calcule (X+Y), décale Y de k positions et place les 2 résultats respectivement dans X et T*)

Exemple: calcul de la fonction sinus et cosinus

On commence par lire, par la procédure LIRE(X), les valeurs 0.607...,0, 3.14..., et t (argument de la fonction), stockées dans le fichier <text> "entree", et on les place respectivement dans X, Y, PI et Z. Le fichier "entree" contient aussi les constantes arctangente(2^{-i}) (i=0,...,64) écrites en base 2 et en représentation virgule fixe sous la forme: (+/-)$\partial_{21}\partial_{20}...\partial_1\mu_0\mu_1...\mu_{106}$, où ∂_i, correspond à 2^i, et μ_i, à 2^{-i}, sont des éléments de {0,1}.Ensuite on réduit l'argument Z à l'intervalle de convergence par la procédure REDCSN(Z, n). Par CYCLE(CORB,T,Y,0) on décale la valeur de Y de 0 position et on écrit le résultat dans T. Le déroulement de cycle1 du tableau de cycles est traduit par la procédure CYCLE(X , T, X , n) qui nous donne , à la fin de son exécution, les valeurs X_{n+1} et $X_n.2^{-n}$.

CYCLE(X, T, X, n), est l'enchaînement des trois procédures PHASE1(X,X,T,Ual1,Ual2,Ual3), PHASE2(Ual1,Ual2,Ual3,n,T,Ual12) et PHASE3(Ual12,A,B,X), fait l' addition des valeurs des 2 premières variables X et T et place le résultat dans la première variable X puis décale la valeur de la troisième variable X de n positions et écrit le résultat dans T.

Ensuite on lit la constante arctangente(2^{-n}) et on la place dans EPS. Puis on exécute successivement CYCLE(Y,T,EPS,n) et CYCLE(Z,T,Y,n+1) qui correspondent respectivement à l'exécution du cycle2 et cycle3. A la fin de ces deux procédures nous obtenons les valeurs de Y_{n+1} et Z_{n+1} .

L'enchaînement de ces trois procédures traduit une itération complète de calcul de la fonction sinus et cosinus. En itérant cette enchaînement pour n variant de 0 à 64 nous obtenons la valeur finale de sinus(t) (= X_{64}) et de cosinus(t) (=Y_{64}) avec une erreur absolue inférieur à 2^{-n}.

L'exécution des algorithmes de calcul de toutes les fonctions élémentaires sur le simulateur de l'architecture a conduit à des résultats qui traduisent l'état de fonctionnement de la partie opérative au bout de chacune des 3 phases, d' un cycle et par suite de chaque itération.

A partir de ces résultats nous vérifions le bon fonctionnement en examinant respectivement si en phase1 les bus A,B et C ont reçus les valeurs des registres X,Y et T et s'ils ont ensuite envoyé celles-ci respectivement dans les registres Ual1, Ual3 et Ual2. Puis nous vérifions en phase2 si la nouvelle valeur de T correspond au décalage de n positions de la valeur de Ual3 et si Ual12 a reçu le résultat de l'addition des deux valeurs de ual1 et Ual2. En phase3 nous examinons si les bus A et B ont tous les deux reçu la valeur de Ual12 et s'ils ont ensuite écrits dans le registre X. Ainsi on arrive à vérifier ces résultats pour un cycle et par suite pour tous les cycle d'une itération complète.

Par la relation $|f(x) - f_n(x)| \leq 2^{-n}$ (c'est à dire qu' on gagne un bit à chaque itération) on juge de l'exactitude des résultats à la fin de chaque cycle.

Moyennant ces vérifications successives, les résultats de la simulation de la partie opérative pour calculer toutes les fonctions élémentaires nous ont permis de rendre compte du bon fonctionnement à chaque itération.

2.7 Conclusion:

Dans cette étude nous avons élaboré une méthodologie de simulation qui consiste d'abord à construire un simulateur de l'architecture du circuit moyennant certains outils de simulation, à exécuter les algorithmes sur celui-ci en tenant compte de leurs découpages en phases puis à vérifier les résultats de ces exécutions.

Cette simulation s'est révélée efficace au cours de ces différentes étapes et notament pour la vérification du fonctionnement logique. Par ailleurs elle nous a permis de minimiser le nombre de cycles d'exécution des algorithmes.

Cette méthodologie de simulation pourrait être appliquée pour simuler d'autres architectures de la partie opérative.

2.8 Architecture virgule flottante:

La forme des algorithmes de calcul des fonctions élémentaires fait qu'on se ramène à chaque fois au calcul de l'expression a +/- $b.2^{-i}$ où a,b sont des réels et i un

entier relatif. Pour exécuter l'opération ci-dessus nous procédons ainsi:

a) Comparaison des mantisses des 2 nombres a et $b.2^{-i}$; si elles sont égales on compare les exposants.

b) Alignement des deux nombres pour avoir les mêmes exposants et l'exécution de l'opération addition.

c) Normalisation du résultat pour que sa mantisse appartienne à l'intervalle [1,2[.

Le circuit virgule flottante conçu pour faire les trois opérations a), b) et c) est composé de 3 grandes parties (figure1 en Annexe 1):

i1) COMPARE.
i2) ALIGN-ADD(SUB).
i3) NORMALISATION.

Chacune des 3 parties comporte 3 autres parties:

j1) Partie signe.
j2) Partie exposant.
j3) Partie mantisse.

Celles-ci fonctionnent simultanément. Les variables de sortie des parties i1),i2) et i3) sont respectivement des variables d'entrée pour i2), i3) et i1).

Après la transformation des algorithmes pour les adapter à cette architecture, nous avons représenté leur déroulement par des tableaux de cycles dont un exemple est donné en tableau A1 en Annexe pour la fonction sinus et cosinus.

Les notations utilisées sont telles que:
PHI1, PHI2, PHI3 et PHI4 pour désigner les phases de traitement des 3 parties COMPARE, ALIGN_ADD, NORMALISATION et le transfert à partir de cette dernière vers la première partie respectivement. PHI représente le déroulement simultané de PHI1 et PHI3 et PHIbarre celui de PHI2 et PHI4.

Le fonctionnement de la partie opérative virgule flottante est réalisé en mode pipeline. Celui-ci consiste en une opération en 2 étapes: dans la première étape seule la phase PHI est active et dans la seconde c'est la phase PHIbarre qui s'active. L'ensemble de ces 2 étapes constituent le cycle de base.

Pour simuler la partie opérative nous avons commencé par décrire la fonction des éléments de base par des procédures que nous avons utilisées pour traduire le fonctionnement de la partie opérative au cours des phases PHI1, PHI2, PHI3, PHI4 et par suite PHI et PHIbarre . L'enchaînement de ces 2 dernières traduit le cycle de base. Ensuite nous avons transposé en langage PASCAL l'ensemble de ces

procédures et obtenu un logiciel de simulation qui est constitué de 2 groupes de procédures: le premier contient celles qui correspondent aux 4 phases de base. Le second est composé des 2 procédures qui traduisent PHI et PHIbarre.

Pour chaque fonction élémentaire, le bon enchaînement des procédures correspondant à son exécution ainsi que les résultats intermédiaires qui traduisent l'état de fonctionnement de la partie opérative sont vérifiés au cours de chaque cycle et par suite de chaque itération.

Chapitre 3

SIMULATION DE LA PARTIE CONTROLE

3.1 Introduction:

Le coprocesseur arithmétique FELIN est divisé en 2 parties: machine de communication et machine à calcul. Celle-ci commence à exécuter les calculs quand elle reçoit les opérateurs et les opérandes de la première machine.

Pour faire le calcul, la partie contrôle génère des commandes qui par leur déroulement font fonctionner la partie opérative qui se charge du calcul proprement dit, en exécutant chaque commande. Le bloc contrôle de la machine à calcul tel qu'il est conçu par Zysman [ZYS87] est divisé en deux parties:

.Premier et second niveau d'interprétation qui reçoivent le code de la fonction à traiter de la machine de communication. Ce code sera placé dans le registre FONCT de 6 bits (figure1).

Dans le premier niveau d'interprétation la fonction est décomposée en une (si elle est simple: sinus par exemple) ou en plusieurs primitives (si elle est composée: fonction puissance). A chaque primitive, on fait correspondre une macroétape. Au cours de celle-ci on extrait 8 paramètres; les quatres premiers correspondent aux deux traductions: traduction de format d'entrée en format interne et inversement du format interne en format de sortie. Le cinquième paramètre est le code de la primitive. Les 3 derniers traduisent les transferts entre registres de 80 bits et entre registres de 128 bits.

Ces 8 paramètres interviennent dans le séquencement du deuxième niveau d'interprétation avec le traitement des exceptions, la réduction d'argument ainsi que le calcul et les restitutions.

Le deuxième niveau se charge de l' interprétation de chaque paramètre et des 3 procédures citées ci-dessus.

.Troisième niveau d'interprétation qui est la R.O.M. des micro-instructions où sont implémentés tous les algorithmes de calcul microprogrammés.

3.2 Description algorithmique:

3.2.1 Second niveau d'interprétation:

Nous allons expliquer la définition de chaque paramètre et leurs enchaînements. Ces paramètre utilisent des procédures élémentaires (exemple: TRADUCTION_FLOTTANT(n, formatopérande(n)) qui seront présentées au paragraphe suivant. La procédure de base utilisée dans ce niveau d'interprétation est: procédure NIVEAU2(k1, k2, k3, k4, R,T1,T2,T3).

Nous allons tout d' abord décrire rapidement le rôle de chacun des 8 paramètres, puis nous expliciterons leur valeur dans le cas des fonctions simples et des fonctions composées.

.k1 est un élément de {0,1,2}. Sa valeur permet de savoir s'il y a 0,1 ou 2 opérandes à traduire du format d' entrée en format virgule flottante. Les formats possibles en entrée sont: entier sur 16, 32 ou 64 bits et réel sur 32, 64 ou 80 bits. Le format virgule flottante sur 64 bits est tel que $(+/-)1.\mu_1\mu_2\ldots\mu_{63}2^{\text{exposant}}$ où $\mu_i \in \{0,1\}$, $1 \leq i \leq 63$ et exposant est un entier relatif écrit en base 2 sur 16 bits. La traduction est exécutée par la procédure TRADUCTION_FLOTTANT(n,formatopérande(n)); formatopérande(n) est le format du $n^{ième}$ opérande.

Après le traitement du paramètre k1 on exécute l'opération de transfert en faisant appel à la procédure TRANSFERT(T1, X,Y). Celle-ci transfère la valeur qui se trouve dans le registre X au registre Y si la valeur du paramètre T1 est différente de 0. La valeur des 2 registres est représentée en virgule flottante sur 80 bits. Si T2=0 alors le transfert n'a pas lieu.

A ces 2 procédures, on enchaîne le traitement des exceptions. La procédure appelée est EXCEPTION_EN_ENTREE(R). R est la primitive pour laquelle on traite l'exception; par exemple si R est égale à la primitive division, on examine la division par 0: comme le deuxième opérande est écrit en représentation flottante, vérifier s'il est nul revient à voir si le premier bit est nul ou pas. Dans le cas où R est égale à l'exponentielle dont l'argument , écrit en représentation flottante, est inférieur à 2^{-31} (c.à.d. que l'exposant est inférieur à -31) ,on remplacera exponentielle (argument) par (1+ argument), ...

.k2 est un élément de {0,1,2}. Sa valeur permet de savoir s'il y a 0,1 ou 2 opérandes à traduire du format en représentation virgule flottante au format en

virgule fixe. Un nombre représenté en virgule fixe s'écrit sous la forme suivante : (+/-) $\mu 0 \mu 1 \ldots \mu 128$ où $\mu i \in \{0,1\}$ correspond à 2^{21-i}. La traduction est exécutée par la procédure TRAD_FLOT_FIXE(n, R) où n est le numéro de l' opérande à traiter et R est la primitive pour laquelle on veut faire la traduction.

Après le traitement de ce paramètre, on passe au transfert du deuxième type. On exécute la procédure de base TRANSFERT(T2, X,Y) qui consiste à transférer la valeur du registre X dans le registre Y si la valeur du paramètre T3 est différente de 0 sinon le transfert n'a pas lieu. Les valeurs des 2 registres sont représentées en virgule fixe.

A ce traitement on enchaine la réduction d' argument. Pour celà on fait appel à la procédure RED_ARG(R), procédure élémentaire qui consiste à réduire l'argument du domaine de définition au domaine de convergence pour la primitive donnée R.

Après cette exécution, on fait le calcul proprement dit pour l'argument réduit et on restitue le résultat grâce à la procédure de base CAL_CORR(R) où R est la primitive pour laquelle on veut effectuer le calcul.

.k3 est un élément de {0,1,2}. Sa valeur permet de connaître l'opérande à traduire du format virgule fixe au format virgule flottante. La traduction est exécutée par la procédure TRAD_FIXE_FLOTTANT(n); n est le numéro de l'opérande à traiter.

Après cette traduction on effectue le transfert du type 3 si la valeur de T3 est différente de 0 ,en faisant appel à la procédure TRANSFERT(T3, X, Y). Les valeurs qui sont dans les deux registres X et Y sont représentées en virgule flottante. Si T3=0 alors le transfert n'a pas lieu.

Enfin pour le paramètre k4 , c' est un élément de {0,1,2}. Sa valeur permet de connaitre l'opérande à traduire du format virgule flottante en format de sortie. Les résultats possibles en sortie sont entier sur 16, 32 ou 4 bits ou réel sur 32, 64 ou 80 bits(format étendu). La traduction est exécutée par la procédure TRADUCTION_FLOTTANT(n, formatsortie) où formatsortie est l'un des six formats précédents.

3.2.2 Premier niveau d'interprétation:

Dans ce qui suit nous expliquons les algorithmes [ZYS87] de décomposition des fonctions et les procédures utilisées.

Le premier niveau reçoit la fonction à traiter de la machine à format. Nous avons deux sortes de fonctions: les fonctions simples: Sin, Cos, Sin&Cos, Arctg,

Exp, Ln, Sqrt, Div, et Mult et les fonctions composées: Tg, 1/X, X^2, X^Y, Log_2 Argsh, Argch, Argth, Arccos, Arcsin, Arctg, SH, CH et TH.

Dès que le premier niveau reçoit le code de la fonction à traiter, celle-ci est placée dans le registre FONCT, de 6 bits. La fonction à calculer est décomposée en sa première primitive. Avant de chercher la primitive suivante on extrait les paramètres d'enchainement pour la primitive trouvée. La procédure de base utilisée est celle du deuxième niveau d'interprétation: procédure NIVEAU2(k1,k2,k3,k4,R,T1,T2,T3). Les fonctions simples sont décomposées en une seule primitive. Les fonctions composées sont exprimées en fonction des primitives simples:

$Arccos(x)=Arctg[(\sqrt{1-x^2})/x]$, $y^x= e^{x*ln(y)}$, ...

Donnons 2 exemples l'un pour les fonctions simples et l'autre pour les fonctions composées.

Si le premier niveau reçoit, dans son registre FONCT le code de la fonction sinus et cosinus alors à la première étape, on aura donc la primitive Sin&Cos: FONCT(Sin&Cos), puis on extrait les paramètres. Dans ce cas:

.k1 est égal à 1. On traduit le premier opérande du format d' entrée en format virgule flottante.

.k2 est égale à 1. On traduit le premier opérande de flottant en représentation virgule fixe.

.k3 est égal à 3 ce qui veut dire que les deux résultats stockés dans les 2 premiers registres X et Y (X:=Sin(X) et Y:= Cos(X) seront traduits de leur représentation en virgule fixe en représentation en virgule flottante.

.k4 est égal à 3. On traduit les deux opérandes de format virgule flottante en format de sortie.

Les résultats seront dans les 2 registres tampon OP1 et OP2 (voir interface entre la machine de communication et la machine à calcul).

Comme il n'y a aucun transfert à faire, les paramètres correspondants aux transferts seront nuls. La procédure qui sera utilisée est NIVEAU2(1,1,3,3,Sin&Cos,0,0,0).

Traitons maintenant l'exemple d' une fonction composée. Prenons le cas de la fonction Argsh. ($Argsh(x)=ln(\sqrt{(x^2+1)}+x)$). Dans le registre FONCT on a le code

de la fonction Argsh:

La fonction FONCT(Argsh) est décomposée en cinq macro-étapes permettant de calculer: x^2, x^2+1, $\sqrt{x^2+1}$, $\sqrt{(x^2+1)}+x$ et enfin $\ln(\sqrt{(x^2+1)}+x)$.

Pour calculer x^2 on traduit le premier opérande de format d'entrée en flottant, puis on le transfère dans le deuxième registre et on fait la multiplication entre les 2 registres(la multiplication est réalisée en flottant). Le résultat est stocké dans le premier registre. La procédure qui sera donc appelée est NIVEAU2(1,0,0,0, MULT, 4, 0, 0).

Pour calculer x^2+1, on transfère Y (Y=x) dans le registre Z pour que la valeur x soit sauvegardée; on transfère 1.fixe dans le registre Y et on fait appel à la procédure : niveau2(0,1,0,0,ADD,1,3,0).

Pour calculer $\sqrt{x^2}+1$, en première étape R prend la valeur SQRT et en seconde étape on extrait les paramètres. La procédure utilisée est Niveau2(0,0,0,0,SQRT,0,0,0). On remarque qu'il n'y a ni traduction ni transfert. Le calcul de $(\sqrt{(x^2+1)}+x)$ nécessite la primitive ADD , le transfert du registre Z dans le registre Y et sa traduction en flottant , ce qui fait appel à la procédure: Niveau2(0,1,0,0,ADD,2,0,0).

Enfin pour calculer $\ln(\sqrt{(x^2+1)}+x)$ on exécute la procédure niveau2(0,0,1,1,ln,0,0,0). Remarquons qu' il n'y a pas de transfert . Le résultat final, après traduction de la valeur qui se trouve dans le premier registre en format de sortie, va se trouver dans le registre tampon OP1.

On a donc l'algorithme suivant:

```
Argsh:  Début
            niveau2(1, 0, 0, 0, MULT, 4, 0, 0);

            niveau2(0,1, 0, 0, ADD, 1, 3, 0);

            niveau2(0, 0, 0, 0, SQRT, 0, 0, 0);

            niveau2(0, 1, 0, 0, ADD, 2, 0, 0);

            niveau2(0, 0, 1, 1, LN, 0, 0, 0);
        Fin.
```

Comme les autres fonctions peuvent être exprimées en fonction des primitives simples et que pour celles-ci on peut extraire les paramètres d'enchaînement, la façon de construire les algorithmes de décomposition des fonctions composées est similaire à celle de la fonction Argsh.

3.2.3 Troisième niveau d'interprétation:

Les algorithmes du Troisième niveau d'interprétation [ZYS87] sont implémentés dans une R.O.M. sous forme de micro-instructions qui sont organisées sur 48 lignes fois 4 colonnes. Ces micro-instructions (chaque micro-instruction est codé sur 78 bits) traduisent les 10 procédures de base auquelles on fait appel dans le deuxième niveau d' interprétation:

. Traduction de format d'entrée en flottant,

. Transfert entre registres à 80 bits,

. Exception en entrée,

. Traduction de format flottant en format fixe,

. Transfert entre registres à 128 bits,

. Réduction d' argument,

. Calcul et restitution,

. Traduction de format fixe en format flottant,

. Transfert entre registres à 80 bits,

. Traduction de format flottant en format de sortie.

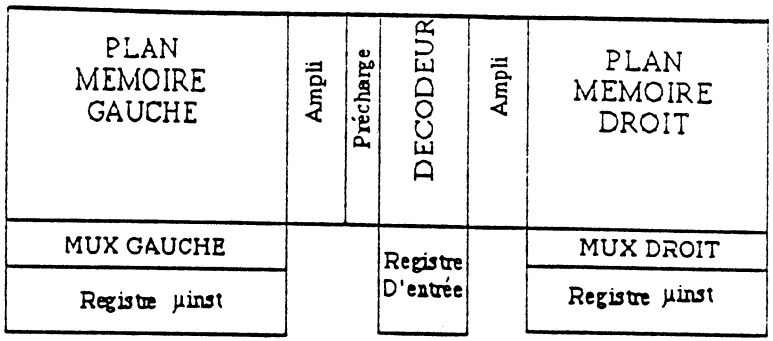
Celles-ci sont écrites en pseudo-pascal et sous forme de microinstructions) et décrites dans [ZYS87].

3.3 Description architecturale:

La partie contrôle [ZYS87] de la machine à calcul est divisée en 3 parties appelées premier et deuxième niveau d'interprétation et R.O.M des micro-instructions (figure1).

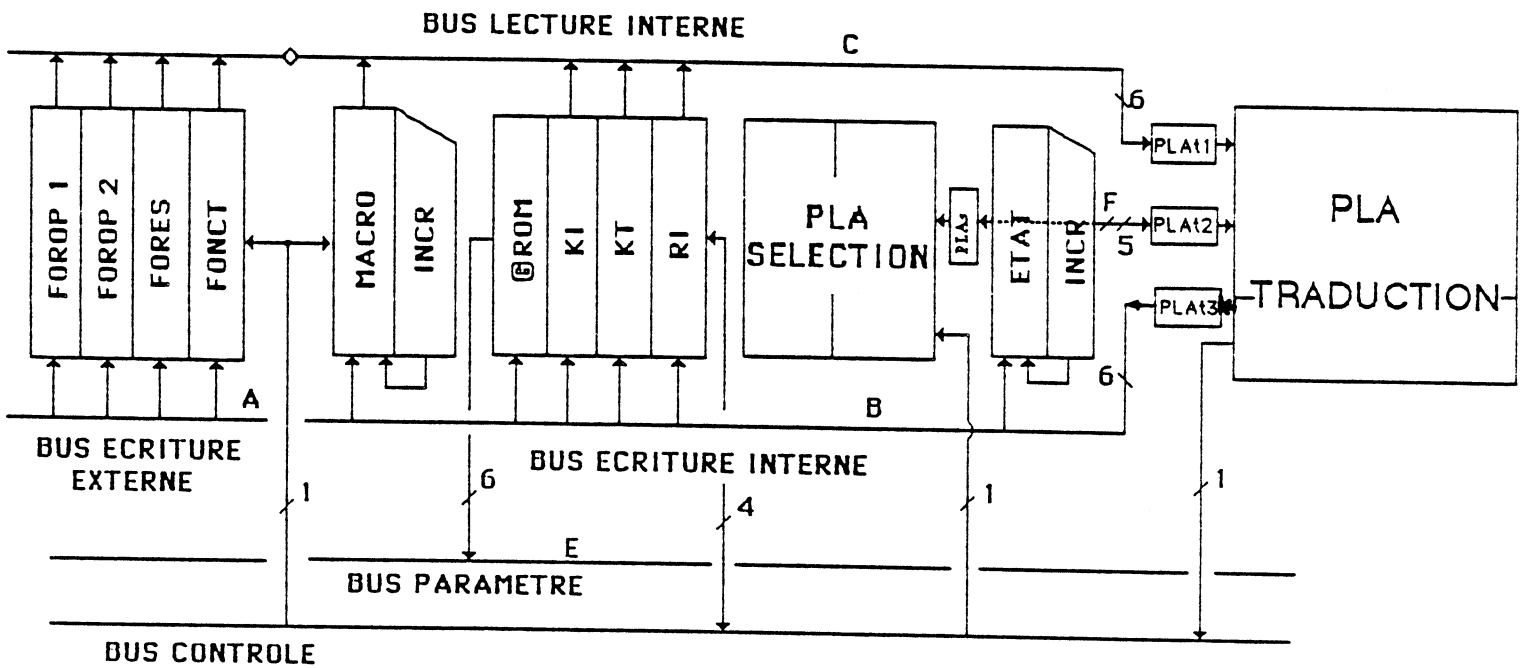
Le premier et le deuxième niveau d'interprétation sont constitués des éléments de base suivants:

-Quatre registres: FOROP1, FOROP2, FORES et FONCT qui indiquent respectivement le code du format du premier opérande, du deuxième opérande , du résultat de calcul et le code de la fonction à traiter. Ces quatres registres sont chargés par la machine de communication.



R.O.M. des micro-instructions.

6



Premier et Second niveau d'interprétation

-Un registre: MACRO qui reçoit le code de la première primitive de la décomposition d'une fonction.

-Quatre registres: ADROM, KI, KT et RI qui correspondent respectivement à l'adressage des 48 lignes de la R.O.M des micro-instructions, aux codes des trois paramètres K1, K2 et K3 concaténés (voir description algorithmique), à la valeur de la concaténation de K4 et (T1, T2, T3) compactés et au code de l'opération à exécuter.

-Un registre: ETAT qui indique l'état de l'automate. Ce registre n'intervient qu'en commentaire.

-Un flag: K qui permet au niveau inférieur de savoir si l'on traite le premier ou le deuxième opérande.

-Six bus: A: bus d'écriture externe, B: bus d'écriture interne, C: bus de lecture interne, D: bus paramètre, E: bus de contrôle et F: bus de lecture et d'écriture interne.

-Un PLA(Programming Logic Array) de selection [ANC 85] à 6 entrées et 17 sorties.

-Un PLA de traduction [ANC 85] à 11 entrées et 7 sorties.

-Un registre: CPLA qui permet de séléctionner le PLA de séléction.

La ROM des micro-instructions est constituée des éléments de base suivants:

- Un amplificateur.
- Un décodeur.
- Un registre de précharge.
- Un multiplexeur.
- Un registre qui reçoit le code des micro-instructions.
- Un plan mémoire dans lequel sont implantées les micro-instructions organisées sur 48 lignes et 4 colonnes.

3.4 Simulation:

Au cours de la simulation de la partie contrôle nous avons commencé par représenter les éléments de base par des variables, puis nous avons traduit le fonctionnement de la partie contrôle pendant les trois phases, phase de lecture, phase de traitement et phase d'écriture, par trois procédures. L'enchainement de celles-ci constitue le *simulateur de l'architecture*. Sur ce simulateur nous avons exécuté les algorithmes en tenant compte de leur découpage en phases. Les résultats obtenus sont vérifiés à la fin de chaque exécution.

Au début nous avons représenté les 10 registres FOROP1,FOROP2, FORES, FONCT, MACRO, ADROM, KI, KT, RI,ETAT et le flag K respectivement par un tableau Ω à dix éléments et une variable K. Les six bus par un autre tableau S à 6 éléments et la ROM des micro-instructions par un tableau R rectangulaire à 4 fois 48 éléments.

Le fonctionnement de la partie contrôle pendant un cycle est ensuite traduit par le tableau ci dessous et la procedure ROM(Ω_6, R, CPLA).

	phase 1	phase 2	phase 3
CYCLE	$\Omega_i \xrightarrow{C}$ PLAt1	(PLAt 1, PLAt 2) → PLATRAD	PLAs $\xrightarrow{F}$ Ω_j (selection)
		ensuite	
	ETAT $\xrightarrow{F}$ PLAt2	PLATRAD → F → PLAs; B → PLAt 3; K	PLAt3 $\xrightarrow{B}$ Ω_j

Au cours de la première phase on envoie la valeur du registre Ω_i par le bus C dans le PLAt1 lié à l'entrée du PLA de traduction, ainsi que la valeur du registre ETAT, par le bus F, dans le registre PLAt2. En phase 2, les valeurs des deux registres PLAt1 et PLAt2 sont traitées par le PLA de traduction qui par la suite fournit simultanément la valeur du flag K, du registre PLAt3 lié à sa sortie et celle du registre PLAs de l'entrée de PLA de séléction. Au cours de la troisième phase le PLA de séléction, après le traitement de la valeur du registre PLAs, séléctionne l'un des dix registres dans lequel on écrit la valeur du registre PLAt3 transportée par le bus B.

Chacune des trois phases est traduite par une procédure. La procédure PHASE1 correspondant à phase1 décrit deux transferts par un bus d'un registre à un autre.

Procédure PHASE1 (S, Ω_i, ETAT, PLAt1, PLAt2)

Début
AFFECTE2 (PLAt1, S(3), Ω_i);
AFFECTE2 (PLAt2, S(6), ETAT);
Fin.

où AFFECTE2 est la procédure qui traduit le transfert par un bus de la valeur d'un registre à un autre registre.

La phase2 est représentée par la Procédure PHASE2 qui décrit la fonction du PLA de traduction. PLAt1 et PLAt2 sont ses variables d'entrée et PLAs, PLAt3 et K ses variables de sortie:

procédure PHASE2(S(6), PLAt1, PLAt2, PLAs, PLAt3, K)

Début
PLAT(S, PLAt1, PLAt2, PLAs, PLAt3, K) ;
Fin.

La procédure PHASE3 correspondant à phase3 décrit la fonction du PLA de sélection et l'écriture de la valeur de PLAt3 dans le registre sélectioné par celui-ci.

procédure PHASE3(S, PLAt3, Ω);

Début
PLAS(PLAs1,Ω_i);
AFFECTE2(Ω_i, S(2), PLAt3);
Fin

où PLAS(PLAs1,Ω_i) est la procedure qui traduit la séléction du registre Ω_i à partir de la valeur du registre PLAs1 lié à l'entrée du PLA de séléction.

Les variables de sortie de la procédure PHASE1 sont des variables d'entrée de la procédure PHASE2 . Les variables de sortie de celle-ci sont des variables d'entrée pour PHASE3.

L'enchaînement de ces trois procédures constitue la procédure CYCLE(Ω,S,K) qui traduit le fonctionnement du premier et second niveau d'interprétation au cours d'un cycle.

En ce qui concerne la R.O.M des micro-instructions son fonctionnement est traduit par la procédure ROM(Ω_6, R, CPLA) qui fait appel aux trois procédures suivantes: CHAL, CHALC et CHPLA.

La procédure CHAL (Ω_6, R(i,j)) traduit la recherche de l'adresse R(i,j) de la

ligne et de la colonne, dans la ROM, de la micro-instruction à traiter à partir de la valeur, transportée par le bus E, du registre Ω_6.

La procédure CHALC (R(i',j'), R(k,l)) correspond au choix de l'adresse R(k,l) de la ligne et de la colonne suivante de la micro-instruction à traiter à partir de l'adresse R(i',j') de la ligne et de l'adresse colonne précédente.

La procédure CHPLA (R(k',l'), CPLA) traduit le choix du PLA, CPLA='TRUE', à partir de l'adresse R(k',l') de la ligne et de la colonne de la dernière micro-instruction correspondant à l'une des 10 procédures du troisième niveau d'interprétation en cours de traitement.

L'enchainement des deux procédures CYCLE(Ω,S,K) et ROM(Ω_6,R,CPLA) traduit le fonctionnement de la partie contrôle au cours d'un cycle et constitue le *simulateur de son architecture.*

La simulation logicielle de la partie contrôle est réalisée en transposant en langage PASCAL l'ensemble des procédures précédentes. Les procédures du simulateur sont basées essentiellement sur les quatre procédures suivantes: AFFECTE2, PLAS, PLAT et CHALC.

Les procédures du logiciel de simulation forment trois groupes:

- Le premier constitue le noyau de base et contient les procédures qui traduisent la fonction des deux PLAs, le transfert ainsi que les procédures CHLA, CHLC et CPLA .

- Le second est composé de celles qui correspondent aux phase1, phase2 et phase3.

- Le troisième, utilisant le premier et le second groupe, est formé de la procédure qui traduit le fonctionnement pendant un cycle et des procédures qui traitent l'enchaînement des micro-instructions.

Dans ce qui suit, nous présentons le type et les déclarations des variables utilisées ainsi que les procédures et leurs explications.

```
Type
    cd= array[1..6] of byte;

    alc= array[1..2] of integer;
var
  registres:  array[1..10] of cd;
  bus:        array[1..6] of cd;
  R:          array[1..4 1..48] of alc;
```

Procédures et leur explication:

Groupe1:

procédure **PLAS**(PLAs1:cr; var Ω:array[1..10] of cd);
(* sélectionne l'un des Ω_i à partir de la valeur de PLAs1 *)

procédure **PLAT**(S: bus;PLAt1,PLAt2: cd;var PLAs,PLAt3: cd;var K:integer);
(* calcule à partir des deux valeurs de PLAt1 et PLAt2 celles de PLAs, PLAt3 et de k*)

procédure **AFFECTE2**(var Y: cd; B,X: cd);
(* affecte X à Y par le bus B *)

procédure **CHAL**(ADROM: cd; var c: alc);
(*détermine la valeur du tableau à 2 éléments c à partir de celle de ADROM*)

procédure **CHALC**(c1:alc; var c2:alc);
(* calcule à partir de c1 la valeur de c2 *)

procédure **CHPLA**(c:alc; var CPLA: boolean);
(* affecte "TRUE" à CPLA si c est la bonne valeur *)

Groupe2:

procédure **PHASE1**(A,B,ETAT,X: cd; var Y,Z);
(* affecte X à Y et ETAT à Z respectivement par les deux bus A et B *)

procédure **PHASE2** (X,Y,A,B: cd; var Z, T: cd; var K: integer);
(* connaissant les valeurs de X et Y , elle calcule et affecte les résultats à K,Z, T par les 2 bus A et B *)

procédure **PHASE3**(A,X: cd; var Y:cd);
(* sélectionne le registre Y et par le bus A lui affecte la valeur X *)

Groupe3:

procédure **CYCLE**(var G: registres; var B: bus; var k: boolean);
(* calcule la nouvelle valeur de G_i à partir de celle de G_j en utilisant les bus B et sort la valeur de k*)

procédure **ROM**(ADROM: cd; var CPLA: boolean);
(* donne la liste de tous les couples correspondant aux adresses de la ligne et de la colonne des micro-instructions d'une procédure ,du troisième niveau ,microprogrammée et ensuite celle de CPLA connaissant celle de ÀDROM *)

La simulation du fonctionnement de la partie contrôle pour le traitement d'une fonction composée est réalisée en simulant respectivement celui du traitement de chacune des fonctions simples qui interviennent dans sa décomposition. Dans ce qui suit nous allons expliquer le programme de simulation correspondant au traitement de la fonction sinus.

Dans le programme de celle-ci on commence par transférer la valeur du registre Ω_4 (égale au code de la fonction sinus) et par le bus C, au registre PLAtl et on transfert de la valeur du registre ETAT, par le bus F, vers le registre PLAt2. Le transfert est réalisé par la procédure AFFECTE2(Y,A,X). Par la procédure PLAT(S(6),PLAt 1, PLAt 2, PLAs, PLAt3), on obtient la valeur du registre lié à la sortie du PLA de sélection qui est transportée par le bus F, la valeur de flag K et celle du registre PLAt 3, PLAs qui va être écrite dans le registre MACRO.

Celui-ci est ensuite séléctionné par le procédure PLAS(PLAs, Ωi) pour recevoir, par le bus B, la valeur du registre PLAt3 en utilisant la procédure AFFECTE2 (MACRO, PLAt 3, B).

L'enchaînement de ces procédures forment la procédureCYCLE(Ω,S,K).

A partir de la valeur du registre MACRO et de celle de ETAT, nous obtenons successivement les valeurs des paramètres RI, KT et KI. Le déroulement de toutes ces procédures représente le fonctionnement du premier niveau d'interprétation.

Pour traduire le fonctionnement du deuxième niveau d'interprétation, nous utilisons aussi la procédure CYCLE (Ω , BUS, K) qui à partir des valeurs des paramètres RI, KT et KI et des registres FOROP1, FOROP2 et FORES, fournit successivement les valeurs au registre ADROM.

A partir de la valeur du registre ADROM, la procédure CHAL(ADROM:cd;var c:alc), donne c l'adresse de la ligne de la première micro-instruction dans la ROM correspondant au traitement des traductions de formats, de transferts, d'exeption, de réduction d'argument ou de calcul proprement dit et restitution. Par la procédure CHALC(c1:alc, var c2:alc) nous obtenons c2,adresse de la ligne et de la colonne suivante de la micro-instruction à traiter à partir de c1 adresse de la ligne et de la colonne de la dernière micro-instruction du traitement en cours. Par CHPLA(c: alc; var CPLA: boolean) nous obtenons la valeur de CPLA qui permet de séléctionner le PLA de séléction (à partir de c).

L'ensemble de ces trois procédures constitue la procédure ROM(ADROM,R,CPLA) qui décrit la fonction de la ROM des micro-instructions pour réaliser un traitement de l'une des dix procedures.

L'éxécution des algorithmes sur le simulateur de l'architecture a conduit à des résultats qui traduisent l'état de fonctionnement de la partie contrôle au bout d'un cycle et par la suite à l'ensemble des cycles nécessaires au traitement d'une fonction.

A partir de ces résultats nous vérifions le fonctionnement du premier et deuxième niveau au cours d'un cycle en examinant respectivement si en phase 1 les bus C et F ont reçu les valeurs des 2 registres Ω_4 et ETAT respectivement et s'ils ont ensuite envoyé celles-ci dans les 2 registres PLAt1 et PLAt2. Puis nous vérifions en phase2 si les valeurs, calculées par le PLA de traduction, écrites dans le bus F, dans le registre PLAt3 et dans le flag K sont exactes et si ensuite le registre PLAs a reçu la valeur du bus F. En phase3 nous examinons si le bus B a reçu la valeur de PLAt3, si le PLA de séléction a séléctionné le registre Ω_5 et si celui-ci a reçu la valeur du bus B.

Pour la R.O.M. des micro-instructions nous vérifions si le décodeur, à partir de la valeur du registre ADROM, a fourni l'adresse de la ligne et de la colonne de la première micro-instruction correspondant à l'une des 10 procédures du troisième niveau d' interprétation qui est en cours du traitement. Puis nous examinons si le décodeur a fourni toutes les adresses des lignes et des colonnes des micro-instructions de la procédure précédente microprogrammée. Ensuite on vérifie si le PLA de séléction est séléctionné (PLAS:=" TRUE").

Ainsi on arrive à vérifier ces résultats pour un cycle et par suite pour les autres cycles.

Moyennant ces vérifications successives, les résultats obtenus de la simulation de la partie contrôle ont permis de rendre compte de son bon fonctionnement.

Chapitre 4

SIMULATION

DE LA MACHINE DE COMMUNICATION

4.1 Introduction:

Les algorithmes de la machine de communication, étudiés en détail par DUPRAT [DUP86], consistent à traiter l'entrée des données et la sortie des résultats, le transfert, des registres d'entrée dans les 2 R.A.M.s, des opérandes et des instructions, le calcul et enfin le transfert des résultats dans les registres de sortie.Ces algorithmes sont écrits en pseudo-Pascal et expliqués dans [PAU86].

L'algorithme d'entrée traduit l'écriture des données, opérandes et instructions, dans 2 registres appelés registres d'entrée de la machine de communication si celle-ci est séléctionnée par le processeur hôte et si la commande de l' écriture est activée. L' algorithme de sortie est semblable à celui d' entrée.

L'algorithme de transfert décrit le transfert simultané de l'opérande et de l'instruction des 2 registres d'entrée dans la file d'attente opérande et dans la file d'attente instruction qui sont stockées respectivement dans 2 RAMs de 15 mots.

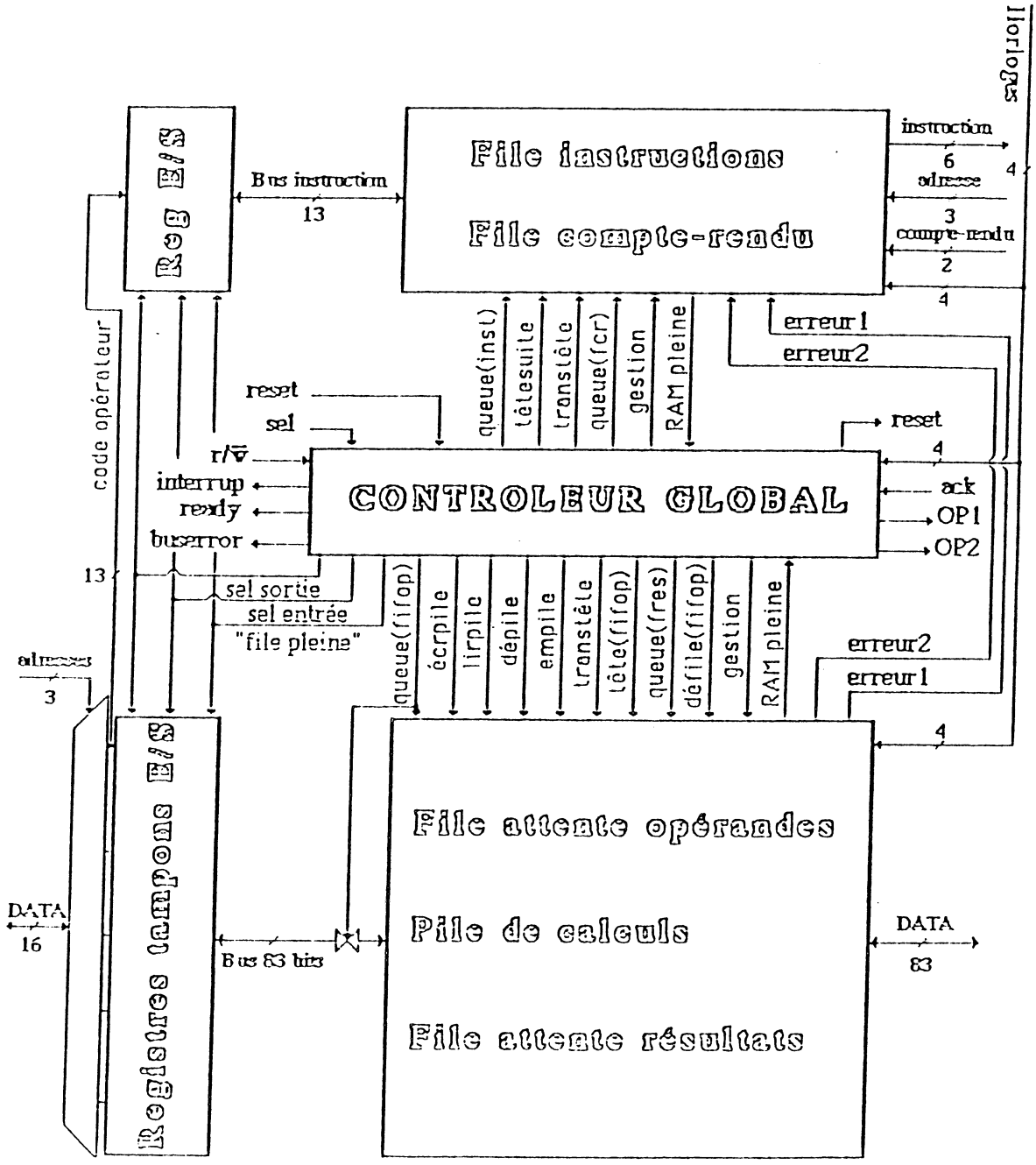
L'algorithme de calcul traduit l'envoie des opérandes et des opérateurs à la machine à calcul et le fonctionnement de la file des instructions, de la file des opérandes, de la pile d'évaluation d'expressions et le placement des résultats dans la file résultats et dans la file comptes rendus. Toutes les files ainsi que la pile se trouvent dans les 2 RAM des opérandes et des instructions.

Le dernier algorithme décrit le transfert des résultats des deux files résultat et compte rendu dans les 2 registres liés à la sortie de la machine de communication.

L'ensemble de ces algorithmes permet de déterminer la structure interne de la machine de communication.

4.2 Structure interne de la machine de communication:

La machine de communication est constituée de 4 blocs et des bus de connections [DUP86] [PAU86] (figure1):


Registres BIS
Bus instruction
13
File instructions
File compte-rendu
instruction
6
4
adresse
3
compte-rendu
2
4
signaux
queue(inst)
tête suite
transtête
queue(fcr)
gestion
RAM pleine
erreur1
erreur2
reset
sel
r/v
interrup
ready
buserror
CONTROLEUR GLOBAL
reset
4
ack
OP1
OP2
code opérateur
13
sel sortie
sel entrée
"file pleine"
queue(fifop)
écrpile
lirpile
dépile
empile
transtête
tête(fifop)
queue(fres)
défile(fifop)
gestion
RAM pleine
erreur2
erreur1
4
adresse
3
Registres tampons BIS
DATA
16
Bus 83 bits
File attente opérandes
File de calculs
File attente résultats
DATA
83

- Un contrôleur global dans lequel sont implémentés les 4 algorithmes précédents.
- 4 registres, 2 liés à l'entrée des instructions et des opérandes et les 2 autres à la sortie des résultats et des comptes rendus.
- Un bloc composé d' une file d' attente d' opérandes, d' une pile de calcul et d' une file d' attente de résultats.
- Un bloc constitué d' une file d' attente d' instructions et d' une file d'attente de comptes rendus.
- Des bus de connection entre ces 4 parties.

File: (*FIFO: First In First Out*)

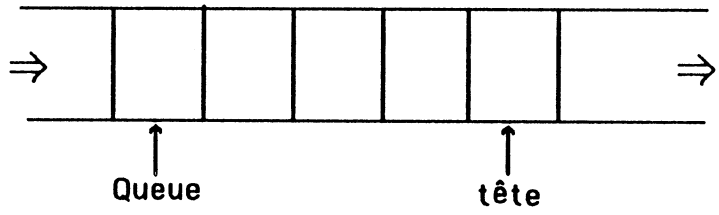


figure2

On appelle pointeur l'information qui permet de localiser directement un élément dans une structure de données.
Une file d'attente est généralement gérée par 2 pointeurs (figure2):

- position du premier élément (tête),

- position du dernier élément (queue).

La première donnée qui entre dans la file est la première qui sera traitée.

Chaque élément de la file est formé de 2 champs (figure3):

- objet,

- adresse de l'objet suivant.

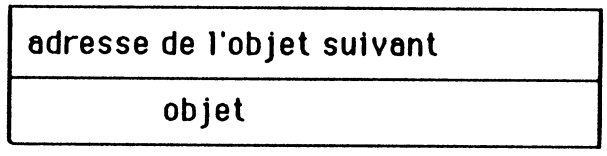


figure3

Gestion d' une file:

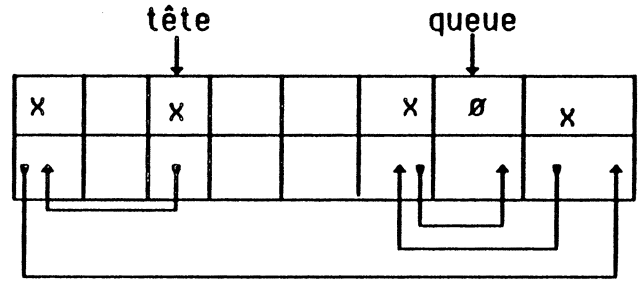


figure4

Une file est gérée au moyen de 2 opérations: opération enfile et opération défile suivies d'un test:

Enfile (x):

objet(pointé par queue) ← x;

adresse de l'objet suivant ← adresse d'un élément libre;

Queue ← adresse de l'objet suivant.

On place la donnée x dans l'élément pointé par queue dans le champ objet puis on écrit dans le champ adresse l'adresse de l'élément suivant qu'on affecte à queue.

Défile(x):

x ← objet(pointé par tête);

libérer tête;

tête ← adresse de l'objet suivant.

La variable x reçoit le champ objet de l'élément pointé par tête; celui-ci reçoit ensuite le champ adresse.

Tests: On teste si la file est pleine ou si elle est vide (tête = queue) .

Pile *(LIFO: Last In First Out)*

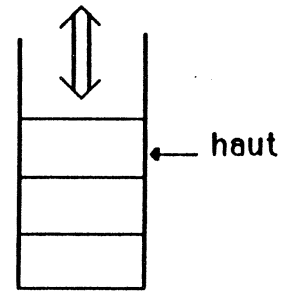


figure 5

Une pile est généralement munie d'un pointeur (figure 5).
Dans la pile la dernière donnée arrivée est la première sortie. Chaque élément de la pile est formé de 2 champs (figure 3):

- objet;

- adresse de l'objet suivant.

Gestion d' une pile:

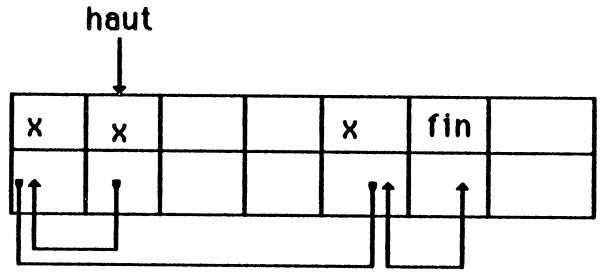


figure 6

Une pile est gérée au moyen de 2 opérations: opération empile et opération dépile suivies d'un test.

Empile (x):

aux ← nouvelle adresse;

aux.adresse ← haut;

aux.objet ← x;

haut ← aux.

On cherche l'adresse, du nouveau élément de la pile dans lequel on va placer la donnée x, que l'on met dans 'aux' , puis on écrit respectivement dans le champ adresse et dans le champ objet de l'élément de position 'aux' la valeur de 'haut' et x. Ensuite on affecte au pointeur 'haut' la valeur de 'aux'.

Dépile (x):

x ← haut.objet;

libérer haut;

haut ← haut.adresse.

Le champ objet de l'élément pointé par 'haut' est envoyé dans x puis le champ adresse est affecté au pointeur 'haut'.

Tests: on teste si la pile est pleine ou si elle est vide (haut = fin).

4.3 Simulation de la machine de communication:

La simulation de la machine de communication est divisée en deux parties. La première correspond à la simulation du fonctionnement des parties opératives opérande et instruction. La seconde traite la simulation de la machine de communication.

Dans un premier temps nous décrivons le schéma de la partie opérative opérande et montrons comment nous avons modélisé le fonctionnement de chaque partie de circuit avant de simuler et donner quelques résultats numériques.

4.3.1 Description architecturale de la partie opérative opérande:

La partie opérative est constituée des éléments de base suivants (figure 8):

- Huit registres:
  - REGHP: registre haut de pile;
  - REGTP: registre tampon;
  - TETFO: registre tête de file opérande;
  - QUEFO: registre queue de file opérande;
  - QFRES: registre queue de file résultat;
  - TFRES: registre tête de file résultat;
  - REGCO: registre comparateur;
  - REGDE: registre décodeur.

- Cinq portes: P1, P2, P3, P4, P5.
- Un comparateur.
- Un décodeur.
- Cinq bus: A, B, C, D, E.
- Une R.A.M. (Random Acces Memory) [ANC85] de 15 mots de 83 bits.
- Une partie occupation de place qui fournit l'adresse d' une place libre.
- Une partie pointeurs de 15 mots de 4 bits.

De la RAM sortent 2 bus: le bus format de l'opérande de 3 bits et le bus des données de 80 bits , et entre un bus de 83 bits composé de 3 bits du format de l'opérande et 80 bits des données. Par les 2 premiers bus on écrit de la RAM dans l'un des 2 registres de l' interface entre la machine à format et la machine à calcul et par le deuxième bus , on écrit des registres d' entrée dans la RAM.

4.3.2 Simulation de la partie opérative opérande:

Pour modéliser le fonctionnement des différents parties du circuit, nous avons procédé ainsi:

Nous avons représenté la RAM, partie occupation de place et partie pointeurs par un tableau de 15 éléments. Chaque élément est un record composé de 3

Pile/Files:

La pile/files (figure 7) est constituée d'une pile et deux files: file attente résultat et file attente opérande. Dans cette pile/files, 5 pointeurs sont utilisés:

- QFO: queue file opérande
- TFO: tête file opérande
- HP: haut pile opérative
- QFR: queue file résultat
- TFR: tête file résultat.

Gestion de la pile/files:

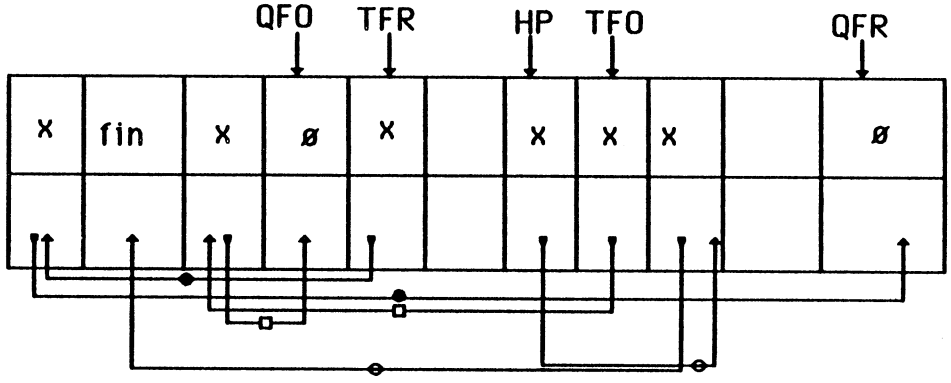


Figure 7

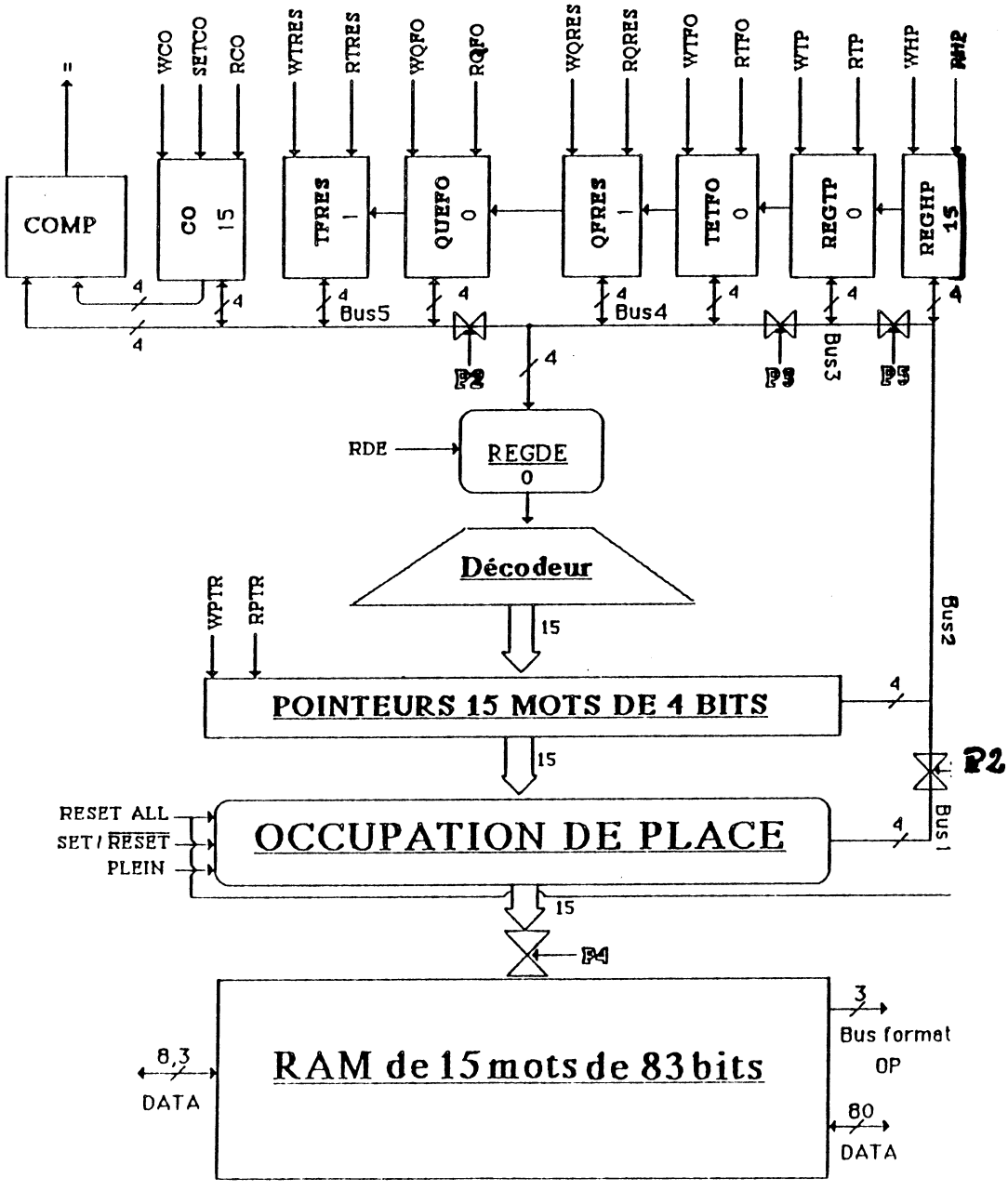
La gestion de la pile/file se fait au moyen de 8 opérations et l'initialisation des pointeurs:

Opérations:

. opqueue(x): insère x en queue de file opérande;
. optête(x): sort x de la tête de file opérande;
. empile(x): transfère la tête de file opérande au sommet de la pile;
. écrpile(x): écrit dans le sommet de pile (sans empiler);
. lirpile(x): lit x au sommet de pile(sans dépiler);
. dépile(x): sort x de sommet de pile;
. resqueue(x): dépile un emplacement qui est inséré en queue de la file résultat et y écrit x;
. restête(x): sort x de la tête de file résultat.

Initialisation:

. QFO = TFO;
. QFR = TFR;
. HP = fin.


=
WCO
SETCO
RCO
WTRES
RTRES
WQFO
RQFO
WQRES
RQRES
WTFO
RTFO
WTP
RTP
WHP
RHP
COMP
CO 15
TFRES 1
QUEFO 0
QFRES 1
TETFO 0
REGTP 0
REGHP 15
4
Bus5
Bus4
Bus3
P2
P3
P5
RDE
REGDE 0
Décodeur
WPTR
RPTR
15
POINTEURS 15 MOTS DE 4 BITS
Bus2
P2
RESET ALL
SET / RESET
PLEIN
OCCUPATION DE PLACE
Bus1
P4
RAM de 15 mots de 83 bits
8,3
DATA
3
Bus format OP
80
DATA

parties: la donnée à traiter, la place occupée ou non et le pointeur pour sélectionner ce record quand il est choisi par le décodeur. Les 8 registres cités dans la partie description du circuit sont représentés aussi par un tableau de 8 éléments qui sont des entiers, les bus et les portes sont aussi représentés chacun par un tableau à 5 éléments: entiers.

Dans les 3 phases nous avons 42 commandes qui activent la partie opérative opérande.

Nous avons 10 procédures à traiter [DUP86]:

QUEUE (FIFO,ENTBUS);
ECRPILE(HP,BUS);
LIRPILE(HP,BUS);
DEPILE(HP,BUS);
EMPILE(HP,BUS);
TRANSTETE(FRES,BUS);
TETE(FIFO,BUS);
QUEUE(FRES,BUS);
DEFILE(FIFO,BUS);
GESTION(POINTEURS).

Chacune de ces Procédures est décomposée en un ensemble de commandes s'exécutant pendant 3 phases $\varphi 1$, $\varphi 2$ et $\varphi 3$.

Nous avons dressé un tableau de 11 colonnes dont la première est constituée de toutes les commandes intervenant dans la décomposition de toutes les Procédures en différentes phases.

Chaque commande prend la valeur "1" si elle est activée ou la valeur "0" si elle ne l'est pas.

Pour simuler le fonctionnement de tout le circuit, nous avons fait correspondre à chacune des 3 phases de la décomposition des 10 procédures en phases une procédure PHASE(I) (Iϵ { 1,2,3 }).

Dans chaque procédure PHASE(I) on traite les 7 points suivants:

. Remise de tous les bus à zéro .
. Initialisation du registre CO à la valeur "15" si la commande SETCO est activée.
. Lecture des registres.
. Calcul de l'adresse d'une place libre (partie occupation de place) et son placement dans bus1.

. Transfert des bus.
. Ecriture dans les registres.
. Mise à jour de la table :
Gestion;
Set/Reset;
resetall.

Nous avons traduit chacune des 3 procédures en langage PASCAL. Les paramètres d'entrée et de sortie sont les valeurs de toutes les commandes intervenant dans chaque phase et de toutes les variables représentant différents éléments de base du circuit de la partie opérative opérande.

En langage PASCAL, nous avons représenté les éléments de base du circuit comme suit:

Les 3 parties: pointeurs, occupation de places et RAM:

```
article= Record
   adresse: integer;
    occupe: byte;
   element: real;
end;

var
Les éléments de la RAM:
     element: array[1..15] of article;

Les bus:
     bus: array[1..15] of integer;
```

Les valeurs des bus sont des entiers car ils reçoivent les valeurs des six registres REGHP,...,TFRES qui sont des entiers.

```
Les registres:
     r: array[1..8] of integer;

Les portes:
     p: array[1..5] of byte;
```

Dans chacune des 3 phases si la commande qui active l' une des portes P[i] prend la valeur "1" on dira que P[i] est ouverte, c' est à dire que les 2 bus séparés par cette porte communiquent sinon P[i] sera dite fermée.

Pour vérifier le fonctionnement de la partie opérative opérandes pour l'exécution des 10 Procédures de base nous avons validé le tableau des valeurs de

toutes les commandes correspondant à chacune des procédures.

Illustrons ceci par l'exemple de la procédure QUEUE(FIFO,ENTBUS). Cette procédure permet de mettre à la queue de la file opérande la donnée présente sur le bus entrée. Le chaînage se traduira par le pointage d'une nouvelle case vide.

Les commandes correspondant prennent les valeurs suivantes:

SET/RESET.φ1:=1	P4. φ_2:=0	P4. φ 3:=1	PCO. φ 4:=0
SETCO. φ 2:=1	WCO. φ 2:=0	WTRES.φ 4:=0	RTRES. φ 2:=0
RTRES. φ 4:=0	RQFO. φ 2:=1	RQFO. φ 3:=0	RQFO.φ4:=1
WQFO. φ 3:=1	P2. φ 2:=1	P2. φ 3:=1	P2. φ 4:=1
WDE. φ 2 :=1	WDE. φ 4:=1	WQRES.φ3:=0	RESETALL.φ 2:=0
RQRES.φ 2:=0	RQRES.φ3:=0	RQRES.φ4:=0	RTFO.φ 2:=0
WTFO.φ3:=0	P3.φ2:=0	P3.φ3:=1	P3. φ 4:=0
RTP.φ3:=0	RTP.φ4:=0	WTP.φ2:=0	P5. φ 2:=0
P5.φ3:=1	P5.φ4:=0	RHP.φ2:=0	WHP. φ 3:=0
WHP.φ4:=0	RPTR.φ2:=0	RPTR.φ3:=0	RPTR. φ 4:=0
WPTR. φ 3:=1	P1. φ 3:=1.		

Sa décomposition en phases est telle que:

phase2(φ2)	(RQFO,WDE) // SET	$\overline{P1}$ P2 $\overline{P3}$ $\overline{P4}$ $\overline{P5}$
phase3(φ3)	(WQFO,WPTR) // SET	P1 P2 P3 P4 P5
phase4(φ4)	(RQFO, WDE) // SET	$\overline{P1}$ P2 $\overline{P3}$ P4 $\overline{P5}$

Toutes ces commandes sont actives c'est à dire prennent la valeur "1" . Pour les autres commandes qui ne figurent pas aux 3 phases, elles prennent la valeur "0" (inactives).

Comme les 7 points vus précédement sont les mêmes pour chaque procédure PHASE(I),nous ne traitons que la procédure PHASE(2) correspondant à phase2.

Dans un premier temps on remet tous les bus à zéro:

```
FOR i:=1 TO 5 DO bus [i]:=0;
```

Après, puisque la commande SETCO. φ_2=0, le registre CO du comparateur ne sera pas initialisé à la valeur15:

IF C[5]=1 THEN r[7]:=15;

Puis on traite la lecture des registres de la manière suivante:
chaque fois qu' on rencontre l' ordre de lecture d' un registre et si cette commande est égale à "1" alors on envoie la valeur du registre dans le bus qui lui est lié:

IF C[8]=1 (C[8]=RTRES. φ_2) THEN bus[5]:=r[6] , ...

On exécute tous les ordres de lecture car si une commande vaut "1" alors qu'elle ne doit pas être activée ou inversement le déroulement de la procédure ne sera pas bon.

La procédure OCCUPLACE(X, bus[1]) cherche une case libre d'un élément de la RAM puis met son adresse dans le premier bus, bus[1].

Par la procédure TRANSBUS(bus,p), on transfère les valeurs des bus en deux étapes: dans un premier temps on fixe le sens de la circulation des données de bus[5] vers bus[1] (figure 8) . On égalise alors 2 bus si la porte qui les lie est fermée. Dans un deuxième temps on fixe le sens inverse.

Cette procédure nous permet de connaître les bus dans lesquels vont circuler les données pour le premier sens, et pour les deux sens de signaler qu'il y'a une erreur si 2 bus consécutifs sont chargés par 2 valeurs différentes et si la porte qui les lie est ouverte c'est à dire que les deux bus séparés par cette porte communiquent.

Pour l'écriture dans les registres, on procède comme dans le cas de la lecture traitée précedement.

Pour la mise à jour de la table (RAM, occupation de places et pointeurs) nous avons procédé ainsi: par la valeur sortante du décodeur, on séléctionne par un pointeur l'élément à traiter et après on traite l'un au moins des 2 cas suivants:
si la commande RESET est activée on libère la case séléctionnée c'est à dire on l'initialise à zéro. Si la commande qui active l'ouverture de la porte P4 est à "1" en phase3 alors on écrit la valeur de l'élément séléctionné dans "bus données" , si elle est égale à "1" en phase 4 alors c'est l'élément séléctionné dans la RAM qui prendra la valeur de "bus données".

Pour les 2 autres procédures PHASE(3) et PHASE(4) on traite les commandes de la même façon que pour la procédure PHASE(2). La seule différence est que pour PHASE(3), il faut traiter seulement les commandes .

Pour chacune des 10 procédures, nous avons enchainé les 3 procédures PHASE(2), PHASE(3) et PHASE(4) et comme paramètre d' entrée nous avons introduit les valeurs des commandes citées avant et nous avons corrigé les valeurs qui induisent des erreurs. Le résultat de la fin de chaque procédure a été vérifié.

4.3.3 Exemple de fonctionnement:

Pour vérifier le bon enchainement de ces procédures, nous avons traité beaucoup d' exemples. Dans ce qui suit, nous allons traiter l'exemple suivant:

$y = a * \sin(w)$.

En notation polonaise inversée y s'écrit:

a enter w sin *

Dans les algorithmes de premier niveau d'interprétation, pour chaque opérateur, nous traitons les Procédures auquelles il fait appel.

Pour l'opérateur "Enter" la seule procédure appelée est EMPILE(HP,FIFO). Pour "Sin" on appelle TETE(FIFO,OP1). Comme l' opérateur "*" est sans opérandes (ce qui correspond à "bitop=0"), les Procédures utilisées sont: DEPILE(HP,OP2) et LIRPILE(HP,OP1).

Nous avons ensuite écrit pour la première procédure les valeurs des 8 registres r[i] ($1 \leq i \leq 8$). Nous avons vérifié l'exactitude de ces valeurs.

Nous avons fait la même chose pour la seconde procédure et comme variables d'entrée des registres, nous avons introduit les valeurs r[i] de sortie de la première procédure, ...

L'enchaînement de toutes ces procédures a donné de bons résultats finaux. Ces résultats sont vérifiés aussi pour les valeurs écrites dans la RAM.

4.3.4 Simulation de la partie opérative instruction:

La structure d' acceuil de la partie opérative instruction est semblable à celle de la partie opérative opérande. La seule différence est qu'il faut remplacer dans r[3], r[4], r[5] et r[6] opérande par instruction et résultat par compte rendu.

La modélisation du fonctionnement des différentes parties du circuit est faite de la même façon que pour la partie opérative opérande.

Dans ce cas nous avons à traiter seulement 5 Procédures:

. QUEUE1
. TETESUITE
. TRANSTETE
. QUEUE2
. GESTIONPOINTEUR.

Il y'a 40 commandes qui activent cette partie. Elles sont légérement différentes de celles vues précédement.

Le bon déroulement des 7 points de chacune des 3 Procédures PHASE(I) (Ie {2,3,4}) est réalisé.

En conclusion pour chacune des 5 procédures, nous avons enchainé les 3 Procédures PHASE(1), PHASE(2) et PHASE(3) et comme paramètres d'entrée, nous avons introduit les valeurs des 40 commandes citées avant et nous avons corrigé les valeurs qui introduisent des erreurs. Le résultat de fin de chaque procédure a été vérifié.

Pour la vérification du bon enchaînement de ces procédures, nous avons traité beaucoup d'exemples.

4.3.5 Simulation de la machine de communication:

Dans ce qui a précédé, nous avons montré comment nous avons simulé le fonctionnement des 2 parties opératives opérande et instruction. Dans ce qui suit nous allons nous interesser à la simulation de la machine de communication.

Pour modéliser le fonctionnement des différentes parties du circuit, nous avons procédé ainsi:

Nous avons représenté les quatres registres d'état (registres d'entrée et de sortie de la machine de communication) par un tableau R de quatre éléments. Les bus de connection parvenant de la partie opérative opérande et de la partie opérative instruction vers le contrôleur global sont représentés respectivement par un tableau FO de deux éléments et par un tableau FI de dix éléments. Les bus partant du contrôleur global vers la partie opérative instruction, la partie opérative opérande et les registres d'état sont aussi représentés respectivement par le tableau EFI de quatre éléments, le tableau EFO de dix éléments et le tableau ERE de cinq éléments.

Le fonctionnement de la machine de communication est traduit par la procédure suivante:

```
CGPIPORE(RO,RI,P,B,E,R, FO, FI, EFI, EFO, RE);
Début
        (* cycle1 *)

        (* phase1: lecture *)

            AFFECTE(ECG1, FO);
            AFFECTE(ECG2, FI);

        (* phase2: traitement *)

            CONTGLOB(ECG1, ECG2, SCG1, SCG2, SCG3);

         (* phase3: écriture *)

            AFFECTE(EFI, SCG1);
            AFFECTE(EFO, SCG2);
            AFFECTE(ERE, SCG3);

       (* cycle2 *)

            PLARE(RE, ERE );
            PLAPO( CO, EFO);
            PLAPI( CI, EFI);

       (* cycle3 *)

            PAROPERAN(RO, P, B, CO);
            PAROPINST(RI, P, B, CI);
Fin.
```

Au cours de la première phase du premier cycle on envoie la valeur des deux tableaux FO et FI dans les deux registres, liés à l'entrée du contrôleur global, ECG1 et ECG2. En phase2, les valeurs de ces deux derniers registres sont traités par le PLA du premier niveau d'interprétation (contrôleur global) qui par la suite fournit simultanément les valeurs des trois registres SCG1, SCG2 et SCG3 liés à sa sortie. Au cours de la troisième phase, on écrit les valeurs de ces registres dans EFI, EFO et ERE.

Au cours du second cycle, en phase1 on transfère les valeurs de EFI, EFO et ERE dans les trois registres EPLAI, EPLAO et EPLAR liés respectivement à l'entrée des PLAs de la partie opérative instruction, la partie opérative opérande et des registres d'état. En phase2 les valeurs des trois registres sont traités par les trois PLAs qui par suite fournissent simultanément les valeurs des registres SPLAR, SPLAI et SPLAO liés à leurs sorties. Au cours de la troisième phase, les valeurs de

ces deux derniers sont écrites dans les registres CI et CO qui représentent respectivement les quarantes et les quarantes deux commandes qui activent la partie opérative instruction et la partie opérative opérande(voir paragraphes 3.3. et 3.4.) . La valeur de SPLAR permet de connaître les registres d'état qui seront traités.

Au cours du troisième cycle, les deux parties opératives s'activent par les deux valeurs de CI et CO; le traitement se termine en fin du cycle.

Remarquons qu'au cours du cycle2 les trois PLAs liés aux deus parties opératives et aux registres d'état doivent travailler en parallèle et qu'au cours du cycle3 les deux parties opératives le doivent aussi. Pour pouvoir séquentialiser ces partie parallèles nous avons adjoint les registres logiques EFI, EFO, ERE à l'entrée des trois PLAs et les registres CI et CO à l'entrée des deux parties opératives et dans lesquels nous avons dupliqué les variables (les variables de sortie des quatres PLAs).

La procédure CGPIPORE(RO, RI, P, B, E,R, FO, FI, EFI, EFO, RE); traduit le fonctionnement de la machine de communication et constitue le *simulateur de son architecture.*

La simulation logicielle de la machine de communication, est réalisée en transposant, en langage PASCAL l'ensemble des procédures précédentes. Les procédures du simulateur sont basées essentiellement sur les Procédures suivantes:

AFFECTE(Y,A,X), CONTGLOB(U, V, Q, S, T), PLARE(A, B), PLAPO(C, D), PLAPI(E,F), PAROPERAN(RO,P,B,E) et PAROPINST(RI,P,B,E) qui traduisent respectivement le transfert d'une donnée, par un bus, d'un registre à un autre, les fonctions des quatres PLA et le fonctionnement des deux parties opératives.

Les procédures du logiciel de simulation constituent deux groupes:
Le premier contient celles citées ci-dessus et constitue le noyau de base. Le second est composé des Procédures correspondant à chacune des trois phases et aux trois cycles vus précédement.

L'exécution des algorithmes sur le simulateur de l'architecture avec prise en compte de leur découpage en phases et de la séquentialisation des parties parallèles a conduit à des résultats qui traduisent l'état de fonctionnement de la machine de communication au bout d'un cycle et par suite à l'ensemble des cycles nécessaires au traitement d'une expression algébrique. Ces résultats sont reportés sous la forme suivante:

A partir de ces résultats nous vérifions le bon fonctionnement de la machine de communication en examinant respectivement si en phase1 du premier cycle les deux registres ECG1 et ECG2 ont reçu les valeurs de FO et FI respectivevent. Puis nous vérifions en phase2 si les valeurs, calculées par le PLA du contrôleur global,

écrites dans les trois registres SCG1, SCG2 et SCG3 sont exactes. En phase3 nous éxaminons si les trois registres EFI, EFO et ERE liés à l'entrée des trois PLAs des deux parties opératives et des registres d'état ont reçu les valeurs de SCG2, SCG3 et SCG1 respectivement.

Pour le deuxième cycle, nous vérifions si tous les registres intervenant au cours des trois phases de ce cycle et liés à l'entrée et sortie des trois PLAs et à l'entrée des deux parties opératives contiènnent les bonnes valeurs.

Pour le troisième cycle , les vérifications effectuées sont menées d'une façon identique à celle de la partie opérative instruction et de la partie opérative opérande traitées dans les paragraphes 3.3 et 3.4.

Ainsi on arrive à vérifier ces résultats pour l'ensemble des cycles d'une exécution.

Moyennant ces vérifications successives, les résultats obtenus de la simulation de la machine de communication ont permis de rendre compte de son fonctionnement.

4.3.6 Conclusion:

Dans cette étude nous avons élaboré une méthodologie de simulation qui consiste d'abord à construire un simulateur de l'architecture du circuit dans lequel nous avons séquentialisé les parties parallèles par l'adjonction de registres logiques et par la duplication des variables, ensuite à l'exécution des algorithmes sur ce simulateur d'architecture avec prise en compte de leur découpage en phases et de la séquentialisation précédente, enfin à la vérification des résultats des exécutions.

Cette simulation s'est révélée efficace au cours de ces différentes étapes et notament pour la vérification du fonctionnement logique de la machine de communication.

Conclusion

Dans ce travail nous avons élaboré une méthodologie de simulation qui consiste d'abord à construire un simulateur de l'architecture du circuit dans lequel nous avons séquentialisé les parties parallèles par l'adjonction de registres logiques et par la duplication des variables, ensuite à l'exécution des algorithmes sur ce simulateur d'architecture avec prise en compte de leur découpage en phases et de la séquentialisation précédente, enfin à la vérification des résultats des exécutions.

Cette méthodologie de simulation pourrait être appliquée à la simulation d'autres architectures (architecture virgule flottante de la partie opérative de FELIN,...).

Cette simulation s'est révélée efficace au cours de ses différentes étapes et notamment pour la vérification du fonctionnement logique. Par ailleurs elle nous a permis de minimiser le nombre de cycles d'exécutions des algorithmes. Ainsi elle a constitué une contribution pour une bonne conception architecturale et algorithmique des différentes parties du coprocesseur FELIN: partie opérative, machine à calcul et machine de communication.

Le logiciel de simulation de FELIN élaboré pourrait être utilisé pour la vérification du fonctionnement logique d'autres architectures.

BIBLIOGRAPHIE

[AGA68] Agard et al. , *Méthodes de simulation*, Paris Dunod 68.

[ANC86] F.Anceau, *The architecture of micro-processors*, Addison-Wesley Publishing Company 86.

[BER77] J.M. Bernard et J.Hugon, *De la Logique cablée aux Microprocesseurs*, Tomes 1,2,3,4, Editions Eyrolles, 1977.

[BOR76] D. Borrione, *LASCAR un langage pour la simulation et l'évaluation des architectures des ordinateurs*, thèse de troisième cycle, informatique, Grenoble 1976.

[BOU77] G.Boulaye, *La Microprogrammation, maîtrise d'informatique*, 1977.

[BUT84] Butterfield, *CORDIC chip data-path*, IMAG R.R. N° 462 sept. 84.

[COD80] W.Cody et W. Waite, *software manual for the elementary functions*, Prentice-Hill Inc. Englewood Cliffs, New-Jersey, 1980.

[COO80] J.J. Coonen, *An implementation guide to a proposed standard for floating- point arithmetic*, IEEE Computer, janv. 80.

[COS85] M. Cosnard, A. Guyot, B. Hochet, J.-M. Muller, H. Ouaouicha, E.Zysman, *FELIN: An elementary functions cruncher*, actes du collogue "N.Gastinel... Le calcul demain" , Grenoble, dec. 85, édité par P. Chenin, C.Dicrescenzo et F. Robert dans computers and computing, Ed. Wiley&Son et Masson, Janvier 1986.

[COS85] M. Cosnard, A. Guyot, B. Hochet, J.-M. Muller, H. Ouaouicha, E.Zysman,*The FELIN arithmetic coprocessor chip*, actes du "8th Symposium on computer arithmetic", Italie, Mai 1987.

[CRO77] J. Crozet et R. Lyon-Caen, *Microprocesseurs et Micro-ordinateurs*, 1977, Masson.

[CRA85] M. Crastes de Paulet, *Spécification et simulation fonctionnelles de circuits complexes: le systéme CADOC*, thèse de troisième cycle, microéléctronique,Grenoble, nov. 85.

[DUP86] J. Duprat, *communication privée*, 1986.

[DES84] A.M. Despain, *pipe-line and parallel pipe-line FFT processors for VLSI implémentation*, IEEE trans. on computers, vol c 33 n° 5 may 84.

[EHR76] R.Ehrmann, *Les Langages de Simulation*, Dunod 1976.

[FEU76] C.-V. Feuvrier, *La Simulation des Systèmes*, Dunod 1976.

[HAM85] V.C. Hamacher, Z.G. Vranisic et S.G. Zaky, *Structure des ordinateurs* , Ed. franç: Daniel Etiemble et Michel Israël, Mc Graw-Hill 85.

[HB84] K. Hwang, F. Briggs, *Parallel processing and computer architecture*, Mc Graw Hill 1984.

[HOC87] B. Hochet, *Conception de VLSI et Application au calcul numérique*, thèse de doctorat, INPG, janvier 1987.

[HOC85] B. Hochet et J.M. Muller, *A way to build efficient carry-skip adders*, IMAG R.R. N° 583, 1986, à paraître dans IEEE Trans. on computers.

[HOC86] B. Hochet, A. Guyot, C. Mauras, J.M.Muller et Y. Robert, SCALA: *une cellule systolique programmable pour l'algèbre linéaire et le traitement du signal*, Colloque C3, Angoulême, juin 1987.

[KOM85] A. Komal et al., *An IEEE standard floating point chip*, IEEE Intern. Solid-State Circuits Conference, février 85.

[KUL81] U.V. Kulish and W.L. Miranker, *Computer arithmetic in theory and practice*, Acad. Presse, New-York, 1981.

[KUN] J. Kuntzmann, *Simulation CAB 500* , polycopié.

[LEQ86] R. Lequette, *Simulation de modèles de quasi-cristaux*, IMAG rapport technique N°3, 86.

[MAU78] Th. Maurin et M.Robin, *Les Systèmes, Microprogrammés, Automates, Mini et Microprocesseurs*, 1978, Dunod technique.

[MEA80] C. Mead et L. Conway, *Introduction to VLSI systems*, Addison-Wesley Publishing Company 80.

[MUL85] J.-M. Muller, *Méthodologie du calcul des fonctions élémentaires*, thèse de doctorat, INPG, septembre 1985.

[MUL] J.M. Muller, *Hardware computation of elementary functions including range reduction*, à paraitre dans IEEE Transactions on computers.

[MUL85] J.-M. Muller, *Discrete basis and computation of elementary functions*, IEEE Transaction on Computer,Sept. 1985.

[NAY68] T.H. Naylor, *Computer simulation techniques*, New York, Londres, Wiley&Sons 68.

[OUA85] H. Ouaouicha, *Simulation de la partie opérative du coprocesseur arithmétique FELIN*, 18ième congré d'Analyse Numérique, Mai 1985.

[OUA86] H. Ouaouicha, *Simulation du fonctionnement logique de la partie contrôle de la machine à calcul du coprocesseur arithmétique FELIN*, 19ième congré d'Analyse Numérique, Mai 1986.

[PAU86] Ph. Paul, *Conception de la machine de communication du coprocesseur FELIN*, stage de DEA, USMG, juin 86.

[PIC76] M. Pichat, *Contribution à l'étude des erreurs d'arrondi en arithmétique à virgule flottante*, thèse d'état, Grenoble, France, 1976.

[ROH76] J. Rohmer, *SSH: Un outil et des techniques simples à implémenter pour construire et simuler des modèles hiérarchises de systèmes*, thèse de troisième cycle, informatique, Grenoble 76.

[SAU] G. Saucier, *Conception de circuits*, première année ENSIMAG.

[SAU] G. Saucier et C. Bellon, *Structure des ordinateurs*, IPRO et MIAG première année.

[VOL59] J. Volder, *The CORDIC Computing technique*, IRE Trans. on computers, Sept. 1959.

[WAL71] J. Walther, *A Unified algorithm for elementary functions*, Joint Computer Conférence Proceeding, Vol. 38, septembre 1971.

[ZYS87] E. Zysman, *Conception de parties controles-applications*, thèse INPG à paraitre.

[ZYS86] E. Zysman et B. Hochet, *Concepçao de um coprocessador arithmetico: FELIN, 6 congresso da sociedade Brasileira de computaçao*, Juillet 86 Recife.

Deuxième partie

Algorithmes de calcul simultané de racines de polynômes

Chapitre 5

ETUDE DE QUELQUES METHODES DE RECHERCHE SIMULTANEE DE RACINES DE POLYNOME

5.1. Introduction:

Le besoin en algorithmes efficaces pour déterminer les zéros d'un polynôme à coefficients complexes a été souligné dans plusieurs applications.

Le problème de la recherche simultanée des racines d' un polynôme a été abordé par plusieurs auteurs: Durand [DUR60], Kerner [KER66], Ehrlich [EHR67], Aberth [ABE73], Nourein [NOU75] et Petkovic [PET83].

Dans ce qui suit nous présentons au deuxième paragraphe la méthode itérative de Durand-Kerner,simple mais très efficace, pour la recherche simultanée des racines d' un polynôme à coefficients complexes. Nous étudions dans le deuxième sous-paragraphe les propriétés théoriques et pratiques de cette méthode et donnons une borne de l'erreur des approximations successives ainsi qu'un résultat de convergence locale en utilisant le théorème de Gerschgorin. Trois variantes de cette méthode et quelques propriétés sont présentées dans le quatrième sous-paragraphe. Puis nous nous interessons à la méthode d'Ehrlich, basée sur les termes de correction de la méthode de Newton, et à deux améliorations de cette méthode. Enfin dans le quatrième paragraphe nous menons une étude comparative de toutes ces méthodes du point de vue du nombre d' opérations arithmétiques, d'ordre de convergence ainsi que d'occupation de place mémoire.

5.2. Methode de Durand-Kerner:

5.2.1. Présentation de la méthode:

La formule itérative de Durand-Kerner est un résultat classique intoduit par Weierstrass [WEI03] en 1891, pour démontrer le théorème fondamental de l'algèbre. C'est une méthode itérative pour la recherche simultanée de toutes les racines d' un polynôme à coefficients complexes. Elle a été proposée par Durand [DUR60], et indépendement plus tard par Kerner [KER66]. Dans ce qui suit nous donnons une dérivation de cette méthode.

Soit P(z) un polynôme de degré n à coefficients complexes:

$$P(z)= z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n ,$$

z_i, i=1, 2, ..., n des approximations des zéros de P(z) juste avant de commencer un cycle d'itération, et $z_i + \Delta z_i$, i=1,2,...,n les approximations à la fin du cycle. De manière idéale nous voudrions avoir:

$$P(z) = \prod_{i=1}^{i=n} (z - (z_i + \Delta z_i)) \qquad (2.1)$$

et alors $z_i + \Delta z_i$ seraient égaux aux racines.

Developpons le second membre de (2.1) en fonction des puissances de Δz_i et posons $z = z_r$:

$$P(z_r) = - \sum_{i=1}^{n} \Delta z_i \cdot \prod_{k=1,k\neq i}^{n} (z_r - z_k) + \sum_{i,j=1,i<j}^{n} \Delta z_i . \Delta z_j \prod_{k=1,k\neq i,j}^{n} (z_r - z_k) - \ldots \qquad (2.2)$$

Lemme ([NOU73]).

$$\sum_{i=1}^{n} \Delta z_i . \prod_{k=1,k\neq i}^{n} (z_r - z_k) = \Delta z_r . \prod_{i=1,i\neq r}^{n} (z_r - z_i) .$$

Démonstration:

$$\Sigma \Delta z_i \cdot \Pi_{k\neq i} (z_r - z_k) = \Sigma_{i\neq r} \Delta z_i \cdot \pi_{k\neq i} (z_r - z_k) + \Delta z_r \cdot \pi_{k\neq r} (z_r - z_k)$$

$$= \Sigma_{i\neq r} \Delta z_i \cdot \pi_{k\neq i} (z_r - z_k) + \Delta z_r \cdot Q'(z_r)$$

($Q(z) = \pi_{i=1,n} (z - z_i)$ et $Q'(z_r) = \pi_{i\neq r} (z_r - z_i)$).

Dans la première sommation dans le deuxième membre i≠r et :

$$\Sigma_{i\neq r} \Delta z_i \cdot \pi_{k\neq i} (z_r - z_k) = \Sigma_{i\neq r} (\Delta z_i / (z_r - z_i)) \cdot \pi (z_r - z_k)$$

$$= \Sigma_{i\neq r} (\Delta z_i / (z_r - z_i)) \cdot Q(z_r) = 0$$

puisque $Q(z_r) = 0$ et $z_i \neq z_j$ si $i \neq j$.

Tronquons (2.2) après le terme d'ordre un. Nous obtenons:

$$P(z_r) \approx -\sum_{i=1}^{n} \Delta z_i \cdot \prod_{k=1,k\neq i}^{n} (z_r - z_k) \quad (2.3)$$

L'utilisation du lemme précédent conduit à l'équation:

$$P(z_r) \approx -\Delta z_r \cdot \prod_{k=1,k\neq r}^{n} (z_r - z_k)$$

et par suite à la formule itérative de Durand -Kerner:

$$\Delta z_r = -P(z_r) / \prod_{k=1,k\neq r}^{n} (z_r - z_k) \quad r=1,2,\ldots,n. \quad \textbf{(DK)} \qquad (2.4)$$

Posons $Q(z)= \prod_{i=1}^{n} (z-z_i)$. En différentiant Q(z) en z_r nous obtenons

$Q'(z_r) = \prod_{k=1,k\neq r}^{n} (z_r-z_k)$. Alors la formule (2.4) s'écrit:

$$\Delta z_r = -P(z_r) / Q'(z_r) \quad r=1,2, \ldots , n.$$

Partant de n approximations des zéros, la formule itérative de Durand-Kerner modifie, à chaque itération, toutes les approximations.

Si w_i, i=1,...,n sont les zéros de P(z) nous remarquons que si $z \rightarrow w$ c'est à dire $z_1 \rightarrow w_1$, $z_2 \rightarrow w_2$, ... , $z_n \rightarrow w_n$ simultanément alors $Q(z) \rightarrow P(z)$ et $Q'(z) \rightarrow P'(z)$.

5.2.2. Propriétés théoriques:

Soit la formule itérative de Durand-Kerner:

$$\Delta z_r = - P(z_r) / Q'(z_r) , \quad r=1,2,\ldots,n;$$

ou sous la forme suivante:

$$z_r^{k+1} = z_r^{k} - P(z_r^{k}) / Q'(z_r^{k}) \quad r =1,\ldots,n; \; k = 0,1,\ldots$$

Proposition([KJE82]). *Soit P(z) un polynôme de degré n à coefficients complexes:*

$$P(z)= z^n + a_1.z^{n-1} + \ldots + a_n,$$

et soient z_r^k $r=1,\ldots,n$; $k=1, 2, \ldots$ *les itérés obtenus par la méthode de Durand-Kerner, en mode parallèle, alors:* $\Sigma_{r=1,..,n}\, z_r^k = -a_1$, *C'est à dire que la somme des approximations après chaque cycle d'itération coincident avec le centre de gravité des racines.*

Démonstration:

$$\sum_{i=1}^{n} z_i^k = \sum_{i=1}^{n} z_i^{k-1} - \sum_{i=i}^{n} \frac{P(z_i^{k-1})}{Q'(z_i^{k-1})} = \sum_{i=1}^{n} z_i^k - \frac{1}{2}\pi i \int_C \frac{P(z)}{Q(z)} dz$$

où C est un cercle contenant tous les points z_i^k.

Developpons P(z) / Q(z) en fonction des puissances de (1 / z):

$$P(z) / Q(z) = 1 + (a_1 + \sum_{i=1}^{n} zi).(1/z) + (\ldots)\,(1/z)^2 + \ldots$$

En intégrant terme à terme la série, toutes les intégrales sont nulles sauf la deuxième et elle est égale à $(a_1 + \Sigma\, z_i^k)$.

D'où $$\sum_{i=1}^{n} z_i^{k+1} = \sum_{i=1}^{n} z_i^k - (a_1 + \sum_{i=1}^{n} z_i^k) = -a_1.$$

Proposition. *Soit P(z) un polynôme de degré n, et soit* z_r^0, $r=1,2,\ldots,n$ *des approximations des racines* w_r *de P(z)=0.*
Si

$$|z_r^0 - w_r| < \frac{\frac{1}{2^{n-1}} - 1}{\frac{n}{2^{n-1}} - 1}, \quad d, r = 1, \ldots, n,$$

$d=min/w_r-w_p/ > 0$ *alors le processus itératif de Durand-Kerner converge i.e. les n suites* $\{z_r^k\}$ $k=0,\ldots,\infty$ *détérminées par la formule récurrente de Durand-Kerner convergent respectivement vers les racines de l'équation P(z)=0.*

Mieux encore nous avons le théorème suivant:

Théorème ([DOC62]). *Soit P(z) un polynôme de degré n ,w_i, i=1,...,m ses zéros exacts et q ∈]0,1[et soient $z_1^0,\ldots,z_m^0$ des approximations distinctes des zéros de P(z)=0. Si*

$$|z_r^0 - w_r| \le \mu = \frac{(1+q)^{\frac{1}{m-1}} - 1}{2(1+q)^{\frac{1}{m-1}} - 1} d \ , \ d = \min |w_r - w_p|$$

alors:

$$|z_r - w_r| \le \mu q^{2^n - 1} \quad \text{d'où} \quad \lim_{n\to\infty} z_r^k = w_r \ , \ r = 1, \ldots , m$$

5.2.3 Propriétés pratiques:

Propriété 1. *L'invariante de la relation* $\sum_{r=1}^{n} z_r^k = \sum_{r=1}^{n} w_r = -a_1, k = 1,2,\ldots$

permet de contrôler l'exactitude de calcul après chaque itération.

Propriété 2. *La relation* $\sum_{i=1}^{n} z_i^k = -a_1$ *donne différentes explications du comportement des approximations autour des zéros multiples. Considérons, par exemple, un polynôme avec une racine multiple et des racines simples. Les approximations convergentes vers les racines simples atteignent vite leurs buts avec la précision de l'ensemble. Les approximations qui convergent vers la racine multiple ont leur centre de gravité égal au zéro multiple et se positionnent autour de celui-ci.*

Propriété3. *On montre que si toutes les suites* $\{ z_i^k \}$, $k=0,..,\infty$, $i=1,2,\ldots,n$ *convergent alors le nombre de suites qui tendent vers chaque racine est égal à la multiplicité de la racine (on montre dans le chapitre 2 que la méthode de Newton appliquée au système de Viète est équivalent à la méthode de Durand-Kerner).*

Propriété4. *Des études numériques montrent que la méthode de Durand-Kerner converge d'une façon normale avec des approximations initiales suffisament proches des racines.*

Une itération de la méthode de Durand-Kerner nécessite [(3/2n+1/2)A+(2n-3)M + D].n secondes où A, M, D sont respectivement les temps nécessaires en secondes pour effectuer une addition(soustraction), une multiplication et une division.
La méthode de Newton éxige approximativement le même nombre d'opérations arithmétiques pour calculer toutes les racines [MAS84].

En pratique la formule de Durand-Kerner peut-être utilisée de 2 manières :

a) Partant des approximations initiales z_i^0 (i=1,...,n) on calcule simultanément z_i^1, i = 1,..., n en utilisant seulement les z_i^0 (mode parallèle: approche de Jacobi [ROB]), ...

b) Partant des approximations initiales z_i^0,i=1,...,n, à chaque fois qu' on calcule une nouvelle approximation on l'utilise pour calculer les suivantes(mode série: approche de Gauss-Seidel [ROB]):
supposons qu' on ait calculé z_1^1 à partir de $(z_1^0, \ldots, z_n^0)$, pour calculer z_2^1 on utilise les approximations $z_1^1, z_2^0, \ldots$ et $z_n^0, \ldots$.

De nombreux exemples numériques ont été testés, ils ont montré que le processus de Gauss-Seidel donne de meilleurs résultats que le processeur de Jacobi. Nous verrons dans le paragraphe 2.5. que le deuxième processus(G.S.) converge plus vite que le premier processus (J.).

5.2.4 Borne de l'erreur des approximations basée sur le théorème de Gerschgorin:

Soit P un polynôme de degré n et $z_1, z_2, \ldots, z_n$ des approximations suppoées distintes deux à deux, des zéros de P(z) et $Q(z)=(z-z_1)(z-z_2)\ldots(z-z_n)$. P(z) est identique avec son polynôme d'interpolation de Lagrange aux points $z_1, z_2, \ldots,$ et z_n:

$$P(z) = Q(z) * [\, 1 + \sum_{j=1}^{n} [\, P(z_j) / ((z - z_j) * Q'(z_j) \,] \quad (4.1)$$

Soit w un zéro de P différent des z_i pour i=1 (1), n. En posant z = w dans l'equation (4.1) nous obtenons:

$$\sum_{j=1}^{n} (P(z_j) / Q'(z_j)) * 1/(w - z_j) = 1 \quad (4.2)$$

Dans ce qui suit nous allons transformer l'équation (4.2) en un problème de valeurs propres. En appliquant le théorème de Gerschgorin à la matrice du système nous obtenons une borne de l'erreur des approximations successives des racines, ensuite nous définissons n disques de convergence pour la méthode de Durand-Kerner deux à deux disjoints et contenant chacun une seule racine de P.

Définissons $\Delta_j = -P(z_j) / Q'(z_j)$ le terme de correction de Durand-Kerner au point z_j. Alors (4.2) s'écrit:

$$w / (z_j - w) = (z_j + \Delta_j) / (z_j - w) + \sum_{i=1, i\neq j}^{n} \Delta_i / (z_i - w) \quad j=1\ (1)\ n$$

ou sous forme matricielle:

$$Au = wu \quad (4.3)$$

où $u^t = (1/(z_1 - w), \ldots , 1/(z_n - w))$ et $A = \mathrm{diag}(z_i) - e\, \Delta^t$

avec $\Delta^t=(\Delta_1,\ldots,\Delta_n)$ et $e = (1, \ldots , 1)$.

Theorème de Gerschgorin ([CIA82]). *Soit* $A = (a_{i,j})_{i,j\ =1\ (1)\ n}$ *une matrice d'ordre n, et* $(H_k)_{k=1,\ldots,n}$ *les régions circulaires suivantes:*

$$H_k = \{ z \in \mathbb{C} / |z - a_{k,k}| \leq \sum_{i=1, i\neq k}^{n} |a_{i,k}| \} \quad k=1,\ldots,n$$

alors les valeurs propres de A sont inclus dans la réunion de ces disques et chaque réunion connexe de k régions contient k valeurs propres.

Le théorème de Gerschgorin appliqué à (4.3) donne immédiatement:

Théorème. *Soient* $\mu=(\mu_1 ,\ldots, \mu_n)$ *un vecteur positif de* $\mathbb{R}^n$ *et*

$$G_j(\mu)=\{z \in \mathbb{C} / |z - (z_j + \Delta_j)| \leq 1/\mu_j * \sum_{i\neq j} \mu_i . |\Delta_i|\} \quad j=1\ (1)\ n$$

alors $G(\mu) = \cup G_j(\mu)$ *contient tous les zéros de P et chaque réunion connexe de k disques contient k racines.*

Pour la méthode de Durand-Kerner un choix spécial des μ_j donne les disques suivants:

$$D_j = \{ z \in \mathbb{C} / |z- (z_j + \Delta_j)| \leq (n-1) * |\Delta_j| \} \qquad 1\leq j \leq n .$$

Si ε est égal au maximum des rayons des disques D_j et si ces disques ne se chevauchent pas alors alors chaque point $(z_j + \Delta_j)$ centre de D_j, $1\leq j \leq n$, est une approximation d'un zéro avec une erreur inférieure ou egale à e. Si les disques D_j se chevauchent alors la borne $(2N-1).\varepsilon$ peut être utilisée.

Pour un choix spécial des μ_i dans le théorème précédent, tous les zéros de P appartiènnent à $G = \cup G_j$, avec G_j tel que

$$G_j = \{ z \in \mathbb{C} \ / \ |z - z_j| \leq R \} \quad \text{où } R = \sum |\Delta_k|.$$

Lemme. *Soient $D_1=\{c_1,r_1\}$ et $D_2=\{c_2,r_2\}$ deux disques de centres et de rayons respectivement c_1,r_1 et c_2,r_2. Alors D_1 et D_2 sont disjoints si et seulement si $|c_1-c_2|> r_1 + r_2$.*

Si $\min(|z_i - z_j|) > 2R$ pour i et j élémentsde {1, 2,..., n} et i≠j, d'après le lemme ci-dessus tous les disques G_j sont disjoints deux à deux.

Corollaire. *Si les n disques sont disjoints deux à deux alors chacun d'eux contient une et une seule valeur propre.*

Théorème. *Soit $q_r = max \{ q \in]\, 0, 1\, [\ / $ le disque $D_{r,\mu}$ de rayon μ du théorème de Docev du paragraphe 2.2 soit inclu dans $G_r\}$, alors pour tout point $v_{\sim}^{(0)}$de $D_{r,\mu}$:*

$$| v_r^{(k)} - w_r| \leq \mu * q_r^{(2^k-1)} \ , r=1, \ldots, n, \text{ où } v_r^{(k)} \text{ sont les itérés de Durand-Kerner.}$$

Démonstration:

En appliquant le théorème de Docev du paragraphe 2.2. à $D_{r,\mu}$ nous obtenons imédiatement l'inégalité du théorème.

5.2.5 Méthodes de Durand-Kerner modifiées et propriétés:

Soit P(z) un polynôme de degré n et w_i, i=1,...,n ses n racines supposées distinctes deux à deux:

$$P(z) = \prod_{i=1}^{n} (z - w_i) \qquad (2.1)$$

Soient $z_1, z_2, \ldots, z_n$, aussi supposées distinctes, respectivement des approximations de w_i, i=1 (1) n. Posons $z = z_k$ dans l'équation (2.1), on obtient:

$$P(z_k) = \prod_{i=1}^{n} (z_k - w_i) = \Delta z_k \cdot \prod_{i=1, i\neq k}^{n} [z_k - (z_i + \Delta z_i)] \qquad (2.2)$$

où $\Delta z_j = w_j - z_j$, j=1,2,...,n est à déterminer.

En réarrangeant l'équation (2.2) nous obtenons:

$$\Delta z_k = -P(z_k) / \prod_{i=1, i\neq k}^{n} (z_k - (z_i + \Delta z_i)) \qquad (2.3)$$

On pourra utiliser (2.3) comme un schéma itératif si une estimation de calcul de Δz_i à l'intérieur du produit est substituée.
En approximant Δz_i figurant dans le produit dans l'équation (2.3) par les termes de correction de Durand-Kerner $\Delta_i = -P(z_i) / Q'(z_i)$ nous obtenons:

$$\Delta z_k = P(z_k) / \prod_{i=1, i\neq k}^{n} [z_k - (z_i + \Delta_i)] \quad k=1,2,...,n \qquad \textbf{(DK1)} \qquad (2.4)$$

C'est la première méthode de Durand-Kerner modifiée.

Dans ce qui suit nous allons nous intéresser à l'ordre de convergence de cette méthode.

Considérons une fonction itérative F associée à la formule itérative

$$z^{k+1} = F(z^k) \qquad (2.1)$$

pour la solution du problème :

$$f(z) = 0 \qquad (2.2)$$

Si l'itération définie par (2.1) converge, donc z_k tend vers w, zéro de f, alors nous définissons l'ordre de convergence de la fonction F dans le sens suivant:
F est d'ordre de convergence p [OST60] si et seulement si:

$$F(w) = w, \quad F^j(w) = 0 \quad j=1\ (1)\ p-1 \qquad (2.3)$$

et

$$F^p(w) \neq 0.$$

Un système des équations non linéaires simultanées:

$$f_i(z_1, z_2, \ldots, z_n) = 0, \ i=1,\ldots,n \quad (2.4)$$

peut-être écrit:

$$z_i = F_i(z_1, z_2, \ldots, z_n),\ i=1,\ldots,n \quad (2.5)$$

pour des fonctions $F_i(z)$, $i=1,\ldots,n$.

En notation vectorielle:

$$f(z) = 0 \quad (2.4)' \quad \text{et} \quad z = F(z) \quad (2.5)'$$

Supposons que $F(z) \in C^k$. Une fonction itérative est d'ordre de convergence p si:

$$F(w) = w$$

$$\partial^k F_i(w) / (\partial x_{j1} \ldots \partial x_{jk}) = 0 \quad k=1,2,\ldots,(p-1) \quad i,j1,\ldots,jn=1,\ldots,n. \quad (2.6)$$

$$\partial^p F_i / (\partial x_{j1} \ldots \partial x_p) \neq 0 \text{ pour au moins une valeur de } i,j1,\ldots,jp \text{ où } f(w)=0.$$

Théorème. *Soit P(z) un polynôme de degré n à coefficients complexes:*

$$P(z) = z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n$$

La méthode de Durand-Kerner modifiée pour la recherche simultanée des racines d'un polynôme P(z) est d'ordre de convergence trois.

Démonstration:

Ecrivons la formule itérative de Durand-Kerner améliorée sous la forme:

$$z_i = F_i(z),$$

où

$$F_i(z) = z_i + \Delta^*_i(z), \qquad \Delta^*_i(z) = -(z_i - w_i) \cdot \pi^*_i(z),$$

$$\pi^*_i(z) = \prod_{p=1, p\neq i}^{n} (z_i - w_p) / (z_i - z^*_p), \qquad z^*_p = z_p + \Delta_p \text{ et}$$

$$\Delta_p = -(z_p - w_p) \cdot \pi_p(z) \quad \text{où} \quad \pi_p(z) = \prod_{q=1, q\neq p}^{n} (z_p - w_p) / (z_p - z_q)$$

Cette formule est d' ordre de convergence trois si et seulement si:

$$F_i(w) = w_i$$

$$\partial F_i(w) / \partial z_i = 0 = \partial^2 F_i(w) / \partial z_k \partial z_j \quad j,k=1,\ldots,n.$$

$$[\partial^3 F_i / \partial z_l \partial z_k \partial z_j] (w) \neq 0 \text{ pour au moins une valeur l,k,j de } \{1,..,n\}.$$

En effet, de (2.7) il est facile de voir :

$$\Delta^*_i (w) = 0 \quad (2.9)$$

$$F_i(w) = w_i \quad (2.10)$$

$$\pi^*_i(w) = 1 \quad (2.11)$$

on a $$F_i(z) = z_i + \Delta^*_i(z) \quad (2.12)$$

des équations (2.7)

$$\Delta_i = - (z_i - w_i). \prod_{p=1,p\neq i}^{n} (z_i - w_p) / (z_i - z_p) \quad (2.13)$$

Nous voyons qu' en calculant la dérivée partielle de Δ_i par rapport à z_j , en l'évaluant en w et en utilisant l'équation (2.9):

$$[\partial\Delta_i / \partial z_j] (w) = -\delta_{i,j} \quad (2.14)$$

où $\delta_{i,j}$ est le symbole de Kronecker.

En différentiant $\pi_i^*(z)$ partiellement par rapport à z_j, nous voyons que:

$$\partial\pi_i^*/\partial z_j = \{\delta_{i,j} \cdot \sum_{q=1,q\neq i}^{n} [(z_i - w_q)^{-1} - (z_i - z^*_q)^{-1}] + \sum_{q=1,q\neq i}^{n} (\delta_{j,q} \cdot \partial\Delta_q / \partial z_j)^* (z_i - z_q)\} \quad (2.15)$$

En évaluant ce résultat en w et en faisant intervenir les équation (2.9) et (2.14) la première et la deuxième sommation de l' équation (2.15) devient nulle. Alors:

$$[\partial\pi_i^* / \partial z_j] (w) = 0 \quad \text{pour tout i,j=1,2,...,n.}$$

Différentions l'équation (2.15) aussi par rapport à z_k et évaluons le résultat en w, c'est possible de montrer que:

$$[\partial \pi_i^* / \partial z_k . \partial z_j] (w) \text{ est finie.}$$

Ecrivons l'équation (2.12)

$$F_i = z_i + \Delta_i^*$$
$$= z_i - (z_i - w_i)\, \pi_i^* (z)$$

et $$F_i(w) = w_i \quad i=1,\ldots,n$$

En différentiant (2.12) partiellement par rapport à z_j nous avons:

$$\partial F_i / \partial z_j = d_{i,j} + \partial \Delta^*_i / \partial z_j$$
$$= \delta_{i,j} . (1 - \pi_i^*) - (z_i - w_i) . \partial \pi_i^* / \partial z_j \quad (2.16)$$

En évaluant (2.16) en w et en utilisant les équation (2.11) et (2.15) nous obtenons:

$$[\partial F_i / \partial z_j] (w) = \delta_{i,j} - \delta_{i,j} = 0 \quad i,j=1,2,\ldots,n.$$

Différentions (2.16) partiellement par rapport à z_k nous avons:

$$\partial^2 F_i / \partial z_k . \partial z_j = - \delta_{i,j} . \partial \pi_i^* / \partial z_k - \delta_{i,j} . \partial \pi_i^* / \partial z$$
$$-(z_i - w_i) . \partial^2 \pi_i^* / \partial z_k . \partial z_j .$$

mais $$[\partial^2 \pi_i^* / \partial z_k . \partial z_j] (w) \text{ est finie} \quad (2.17)$$

et $$[\partial \pi_i^* / \partial z_j] (w) = 0 = [\partial \pi_i^* / \partial z_k] (w) \quad \text{(par l'équation (2.15)*)}$$

et on a:

$$[\partial^2 F_i / \partial z_k . \partial z_j] (w) = 0 \quad j,k=1,\ldots,n.$$

$$[\partial^3 F_i / \partial z_i . \partial z_k . \partial z_j] (w) \neq 0,$$

ce qui finit la démonstration.

La formule itérative (2.4) est peu éfficace en pratique car elle utilise beaucoup d'opérations arithmétiques (le double de la méthode de Durand-Kerner) à cause des calculs supplémentaires de:

$Q'(zi) = \prod_{j=1, j\neq i}^{n} (z_i - z_j)$, i=1, ..., n. Ce point sera discuté au 4ième paragraphe.

Pour une deuxième modification de la formule itérative de Durand-Kerner introduisons $\Delta = \max \Delta_i$, i=1, .. , n et supposons que Δ est suffisament petit, en d'autres termes que toutes les approximations initiales sont prises de telles manière qu'elles soient proches des zéros.

Developpons $\prod_{i=1, i\neq k}^{n} (z_k - (z_i + \Delta_i))$:

$$\prod_{i=1, i\neq k}^{n} (z_k - (z_i + \Delta_i)) = \prod_{i=1, i\neq k}^{n} [(z_k - z_i) * (1 - \Delta_i / (z_k - z_i))]$$

$$= Q'(z_k) * [1 - \sum_{i=1, i\neq k}^{n} \Delta_i / (z_k - z_i) + 0(\Delta^2)] \qquad (5.5)$$

Si nous gardons seulement les termes linéaires de Δ_i dans (5.5), nous obtenons à partir de l'équation (2.4) le schéma itératif suivant:

$$z_k^* = z_k - \Delta_k * (1 - \sum_{i=1, i\neq k}^{n} \Delta_i / (z_k - z_i))^{-1} \qquad k=1\ (1)\ n\ \ (DK2) \qquad (5.6)$$

c'est la deuxième méthode de Durand-Kerner modifiée. L'ordre de convergence de cette méthode est trois.

Dans ce qui suit nous donnons un résultat de convergence locale de cette méthode.

Définissons d'abord quelques notations:

Considérons un polynôme de degré $n \geq 3$:

$$P(z) = \pi_{i=1}^{n} (z - w_i),$$

ayant des zéros simples, réelles ou complexes, $w_1,\ldots,w_n$. Supposons que les disques disjoints:

$$Z_i = \{ \text{cent}(Z_i); \text{ray}(Z_i)\} = \{ z_i; r_i\} \quad i=1,\ldots,n,$$

de centre z_i et de rayon r_i contiennent les zéros w_i , i = 1, ... , n .

$$\mu = \min_{i,j \text{ et } i\neq j} \{ |z| : z \in (z_i - Z_j) \} = \min_{i,j \text{ et } i\neq j} \{ | z_i - z_j| - r_j \} ,$$

$$r = \max_j r_j,$$

$$W_i = \Sigma_{j\neq i} \Delta_j/(z_j - Z_j).$$

Théorème. *Soient $w_i \in Z_i^{(o)}$ et la suite des disques $(Z_i^{(m)})$ définie par la formule :*

$$Z_i^{(m+1)} = z_i^{(m)} + \Delta_i/(1 - W_i^{(m)}) \quad (i=1,\ldots,n;\ m=0,1,\ldots),$$

alors sous la condition :

$$\mu^{(o)} > 3\,(n-1).r^{(o)}$$

pour tout i=1,...,n et pour tout m=0,1,..., nous avons:

(i) $w_i \in Z_i^{(m)}$.

(ii) $r^{(m+1)} < 7(n-1)\, r^{(m)\,3} \;/\; [\, \mu^{o} - \theta(n) r^{o} \,]^2$ *où* $\theta(n) = (n+4/3)/n-16/9.$

(ii) montre que la suite $(r^{(m)})$ converge vers zéro au moins cubiquement.

L'éfficacité du processus de Durand-kerner peut-être croissante, dans un certain degré, si en calculant les nouvelles approximations z_k^* nous utilisons les approximations déjà calculées z_i^* (i < k) dans la même itération (mode série: approche de Gausse-Seidel). Dans ce cas nous obtenons le processus itératif accéléré suivant:

$$z_k^* = z_k - P(z_k) \Big/ \prod_{i=1}^{k-1} (z_k - z_i^*) \cdot \prod_{i=k+1}^{n} (z_k - z_i), \quad k=1,\ldots,n \quad \text{(DKGS)} \quad (5.7)$$

La méthode (5.7) de Durand-Kerner-Gauss-seidel converge plus vite que la méthode de Durand-Kerner en mode parallèle(Jacobi).

Précisons la définition que nous utilisons pour le R–ordre de convergence[ORT70] avant de donner une borne inférieure de ce dernier pour la méthode (5.7):

Soit I un processus itératif avec un point limite x^* . Alors la quantité:

$$O_R(I, x^*) = \begin{cases} \infty \text{ si } R(I, x^*) = 0 \text{ pour tout } p \in [1, \infty[, \\ \inf \{p \in [1,\infty[\,/\, R(I, x^*) = 1\} \text{ sinon} \end{cases}$$

est appelée le R-ordre de I en x*.

R (I, x*) est appelé le R-facteur de I en x* et il est définie par:

$$R(I, x^*) = \sup \{R_p(\{x^k\}) / \{x^k\} \in C(I, x^*)\}, \quad 1 \le p < \infty$$

où

$$R_p(\{x^k\}) = \begin{cases} \lim\limits_{k \to \infty} |x^k - x^*|^{1/k} & \text{si } p=1 \\ \lim\limits_{k \to \infty} |x^k - x|^{1/p} & \text{si } p>1 \end{cases}$$

et C(I, x*) est l'ensemble de toutes les suites générées par I et convergentes vers x*.

Ostrowski [OST60] et Traub [TRA64] définissent l'ordre de convergence d'une suite itérative par un nombre p, s'il existe, tel que:

$$\lim \|x_{k+1} - x^*\| / \|x_k - x^*\|^p = c_p,$$

où la constante c_p de l'erreur assymptotique doit être $c_p \neq 0$ et $c_p \neq \infty$.

Ortega et Reindboldt [ORT70] montre que cette définition est plus restreinte que leur R-ordre.

Propriété ([ORT70]). *Soit $C(I, x^*)$ l'ensemble des suites engendrées par I et ayant même point limite x^* où I désigne un processus itératif. S'il existe un $p \in [1,\infty[$ et une constante c_1 tel que pour toute suite $\{x_k\} \in C(I,x^*)$ on a:*

$$||x_{k+1} - x^*|| \leq c_1 ||x_k - x^*||^p \quad \text{pour tout } k \geq k_o = k_o(\{x_k\}) \text{ alors } O_R(I, x^*) \geq p.$$

D'autre part s'il existe une constante c_2 et une suite $\{x_k\}$ élément de $C(I,x^*)$ tel que:

$$||x_{k+1} - x^*|| \geq c_2 ||x_k - x^*||^p > 0 \quad \text{pour tout } k \geq k_o = k_o(\{x_k\}) \text{ alors } O_R(I, x^*) \leq p.$$

Ainsi, si les deux conditions sont vraies, on a: $O_R(I, x^*) = p$.

Théorème ([ALH 74]). *Soit μ_n l'unique racine positive , $\mu_n \in]1, 2]$, de l'équation:*

$$P(\mu)=\mu^n - \mu - 1 = 0$$

alors le R-ordre de convergence de la méthode (DKGS) est au moins $(1+\mu_n)>2$.

5.3 Méthode d'Ehrlich:

5.3.1 Présentation:

En 1967 Ehrlich [EHR67] en appliquant la méthode de Newton à la fonction:

$R_k(z) = P(z) / \pi_{i \neq k} (z-z_i)$ au point $z = z_k$ obtient une méthode d'ordre de convergence trois pour la détermination simultanée de toutes les racines d' un polynôme.

Nous présentons cette méthode sous la forme suivante :

$$\delta z_k = \delta_k \cdot (1 + \delta_k . \beta_k)^{-1} \qquad \text{(EH)}$$

où $\delta_k = -P(z_k) / P'(z_k)$ est le terme de correction de Newton en z_k et

$\beta_k = \sum_{i \neq k} (z_k - z_i)^{-1}$.

Avec les mêmes notations définies pour le théorème de convergence locale de la méthode (DK2) la méthode d'Ehrlich converge cubiquement sous la supposition:

$$\mu^{(o)} > 2n.r^{(o)} .$$

5.3.2 Quelques améliorations:

Soit P un polynôme de degré n ayant pour racines w_i, i=1,...,n:

$$P(z)= \prod_{i=1}^{n} (z - w_i) \qquad (2.1)$$

En différentiant le logarithme de l'équation (2.1) et en évaluant le résultat en $z = z_k$ nous obtenons:

$$P'(z_k) / P(z_k) = \sum_{i=1}^{n} (z_k - w_i)^{-1}$$

$$= -(\delta z_k)^{-1} + \sum_{i=1, i\neq k}^{n} (z_k - (z_i + \delta z_i))^{-1} \qquad (2.2)$$

où $\delta z_j = w_j - z_j$, j=1,...,n est à déterminer. Un réarrangement de l'équation (2.2) donne:

$$\delta z_k = \delta_k . (1 + \delta_k . \sum_{i\neq k} [z_k - (z_i + \delta z_i)^{-1}]^{-1} \qquad (2.3)$$

où δ_k est le terme de correction de Newton.

L'équation (2.3) peut être utilisée comme un schéma itératif si une estimation de calcul de δz_i à l'intérieur du signe somme est substituée. Deux possibilités se présentent:

i) En ignorant complétement le terme de correction δz_i dans le membre de droite de l'équation (2.3) et en posant $\beta_k = \sum_{i\neq k} (z_k - z_i)^{-1}$ nous obtenons la méthode itérative d'Ehrlich:

$$\delta z_k= \delta_k . (1 + \delta_k . \beta_k)^{-1}, \quad k=1,...,n. \ (EH) \qquad (2.4)$$

ii) En approximant δz_i à l'intérieur du signe de sommation dans l'équation (2.3) par le terme de correction de Newton,nous obtenons la méthode d'Ehrlich modifiée :

$$\delta z_k = \delta_k.(1 + \delta_k.\beta_k^*)^{-1}, \qquad k=1,\ldots,n \qquad \text{(EH1)} \qquad (2.5)$$

où $\beta_k^* = \sum_{i \neq k} (z_k - (z_i + \delta_i))^{-1}$.

L'efficacité du processus d'Ehrlich peut être croissante, dans un certain degré, si en calculant les nouvelles approximations z_k^* nous utilisons les approximations déjà calculées z_i^* $(i < k)$, qui sont meilleures que les approximations de Newton $(z_i + \delta_i)$, dans la même itération (approche de Gauss-Seidel) nous obtenons:

$$z_k^* = z_k + \delta_k.\Omega_k, \qquad k=1\,(1)\,n \quad \text{(EHGS)} \qquad (2.6)$$

$$\text{où } \Omega_k = [\, 1 + \delta_k \,.\, (\sum_{i=1}^{k-1} 1/(z_k - z_i^*) + \sum_{k+1}^{n} 1/(z_k - z_i)) \,]^{-1}.$$

La méthode (2.6) d'Ehrlich-Gauss-Seidel (EHGS) converge plus vite que celle d'Ehrlich. Le théorème suivant précise une borne inférieure du R-ordre de convergence de la méthode (2.6).

Théorème([ALH74]). *Si β_n est l'unique racine positive, $\beta_n \in\,]\,1,2]$, de l'équation $\beta^n - \beta - 2 = 0$ alors le R-ordre de convergence de la méthode (EHGS) est au moins $2+\beta_n$ (> 3).*

La démonstration de ce théorème est similaire à celle du théorème sur le R-ordre de convergence de la méthode (DKGS).

Le R-ordre de convergence des deux méthodes (DKGS) et (EHGS) peut être augmenté si dans le deuxième signe du produit de l'itération de (DKGS) nous utilisons à la place de z_i, $z_i + \Delta_i$ et si dans le deuxième signe somme de l'itération de (EHGS) nous remplaçons z_i par $z_i + \delta_i$ où Δ_i et δ_i sont respectivement le terme de

correction de Durand-Kerner et celui de Newton. Nous avons les résultats suivants sur le R-ordre de convergence de ces deux méthodes:

Théorème. *Si β_n est l' unique racine positive $\beta_n \in (2,3)$, de l' équation suivante:* $\beta^n - \beta - \Sigma_{0,..,n-1}\beta^k = 0$ *alors le R-ordre de (DKGS) est* $> 1+\beta_n$ (> 3).

Théorème. *si β_n est l'unique racine positive $\beta_n \in (1,2)$,de l'équation suivante:* $x^n - x - 1 = 0$ *alors le R-ordre de (EHGS)* $>= 2(1+\beta(n))$ (>4).

5.4 Etude comparative des méthodes:

La méthode de Durand-Kerner (DK) et ses variantes ainsi que celle d'Ehrlich (EH) et ses variantes peuvent être utilisées pour chercher les racines multiples. En utilisant des approximatios distinctes deux à deux, nous obtenons toutes les racines avec leur ordre de multiplicité mais la convergence vers les racines multiples est lente.

Beaucoup d'exemples numériques ont été traités et ont montré que les approximations qui convergent vers les racines multiples se déplacent directement vers ces racines à chaque cycle d'itération à chaque fois en réduisant leurs distances de la fraction de 2/(m+1) pour, la première variante (DK1) et pour (EH), de 1/m pour la méthode (DK) et de 3/(m+2) pour la première méthode d'Ehrlich modifiée (EH1); m étant la multiplicité de la racine .

Pour comparer la méthode de Durand-Kerner (DK) et ses trois variantes (DK1), (DK2) et (DKGS) ainsi que celle d'Ehrlich (EH) et ses deux améliorations (EH1) et (EHDK), du point de vue du nombre d'opérations arithmétiques , nous présentons le tableau ci-dessous pour un polynôme de degré n à coefficients complexes. Le nombre d'opérations arithmétiques nécessaires au calcul des valeurs du polynôme $P(z_1)$, $P(z_2)$,...et $P(z_n)$ est le même pour toutes les méthodes.

Méth. / OPér.	(DK)	(DK1)	(DK2)	(DKGS)	(EH)	(EH1)	(EHGS)
Add/Sous.	n^2	$2n^2$	$n(3n-2)$	$2n^2-2$	$2n^2-n$	$3n^2-2n$	$2.5n^2-1.5n$
Mult.	$n(n-1)$	$2n(n-2)$	$n(n-2)$	$2n^2-5n+2$	n^2	n^2	n^2
Div.	n	$2n$	$n(n+1)$	$2n-1$	n	n	n
Ordre	2	3	4	entre 3 et 4	3	4	entre 4 et 5

Tableau 1

Ce tableau montre que la première méthode de Durand-Kerner modifiée (DK1) demande 2 fois plus d'opérations arithmétiques que la méthode de base (DK). Par suite la méthode (DK1) n'est pas applicable. La remarque suivante confirme cette conclusion. En utilisant le même nombre d'opérations, la méthode de base (DK) peut être appliquée successivement deux fois en donnant dans un certain sens, un processus d'ordre 4. A partir de tableau1 nous concluons aussi que la deuxième méthode de Durand-Kerner modifiée (DK2) utilise beaucoup de divisions.

Beaucoup d'exemples numériques montrent que les méthodes (DK1) et (DK2) présentent un comportement similaire sous les même conditions.

En plus de la convergence plus rapide, le processus itératif (DKGS) nécessite moins d'opérations arithmétiques et occupe moins d'espace mémoire (à cause de l'utilisation des approximations précédement calculées dans la même itération) que les méthode (DK1) et (DK2).

Contrairement à la méthode (DK1) l'accélération de la convergence de la méthode (EH1) est atteinte sans calculs supplémentaires puisque les termes de correction de Newton déjà calculés (qui apparaissent aussi dans la méthode (EH)) sont utilisés. Par conséquent la méthode itérative (EH1) est plus convenable en pratique que les méthodes simultanées (DK1), (DK2).

L'augmentation de l'ordre de convergence de la méthode (EHGS) est obtenue sans calculs supplémentaires. De plus cette méthode occupe moins d'espace mémoire et utilise moins d' opérations arithmétiques que la méthode (EH1) (voir tableau 1).

En conclusion les méthodes (DK) et (EH) en mode parallèle et en mode série sont plus performantes comparées aux autres méthodes car elles font la synthèse d'un ordre de convergence assez élevé et d'une itération peu coûteuse.

Chapitre 6

METHODE DE DURAND-KERNER: RESULTATS DE CONVERGENCE

6.1 Introduction:

Dans ce qui suit nous montrons au deuxième paragraphe que la méthode de Durand-Kerner est équivalente à la méthode de Newton dans l'espace de dimension n, $\mathbb{C}^n$ en considérant la fonction:

$$F: \mathbb{C}^n \to \mathbb{C}^n, \quad F(z) = (F_1(z), F_2(z), \ldots, F_n(z)),$$

où les fonctions F_k sont définies par:

$$F_k(z) = S_k(z) + (-1)^{k-1} a_k, \quad k=1,2,\ldots,n,$$

et S_k est la fonction symétrique élémentaire des racines [DUR60]. Au troisième paragraphe nous montrons que la méthode de Durand-Kerner converge presque partout dans $\mathbb{C}^2$ pour les polynômes quadratiques et nous caractérisons l'ensemble de non convergence. Ensuite, à l'aide de ce dernier résultat et deux remarques importantes, nous conjecturons que la méthode de Durand-kerner converge presque partout dans $\mathbb{R}^n$ quand elle est appliquée aux polynômes ayant leurs racines réelles et distinctes deux à deux. Enfin dans le dernier paragraphe nous présentons quelques graphiques illustrant, à partir de quelques points initiaux, le comportement itératif de la méthode de Durand-Kerner appliquées aux polynômes de degré 3 à coefficients complexes.

6.2 Equivalence de la méthode de Durand-Kerner et de la méthode de Newton:

Théorème. *Soit P un polynôme de degré n à coefficients complexes* $a_1, \ldots, a_n$: $P(x)=x^n + a_1x^{n-1}+\ldots+ a_{n-1}x +a_n$ *et* $F: \mathbb{C}^n \to \mathbb{C}^n$, $F(z)=(F_1(z),F_2(z),\ldots, F_n(z))$ *telle que* $F_k(z) = S_k(z) + (-1)^{k-1} a_k$, $k=1,2,\ldots,n$, *où* S_k *est la fonction symétrique élémentaire des racines.*
La méthode de Durand-Kerner appliquée à P est équivalente à la méthode de Newton appliquée à F.

Démonstration:

Considérons le système de Viète (V) des relations qui lient les racines du polynôme P à ses coefficients:

$$\begin{cases} F_1(x_1, \dots, x_n) = x_1 + x_2 + \dots + x_n + a_1 \\ F_2(x_1, \dots, x_n) = [\,(x_1x_2 + \dots + x_1x_n) + (x_2x_3 + \dots x_2x_n) + \dots + x_{n-1}x_n] - a_2 \\ \quad . \\ \quad . \\ \quad . \\ F_p(x_1, \dots, x_n) = \sum_{i,j,\dots,q}^{i<j<\dots<q} x_i x_j \dots x_q + (-1)^{p-1} a_p \\ \quad . \\ \quad . \\ \quad . \\ F_n(x_1, \dots, x_n) = x_1x_2\dots x_n + (-1)^{n-1} a_n \end{cases} \qquad (V)$$

La formule de Newton dans $\mathbb{C}^n$ est définie par:

$$X^{k+1} = X^k - \left[F'(X^k) \right]^{-1} \left[F(X^k) \right] \quad , k=0,1,\dots \quad (1.1)$$

où F est une fonction différentiable de $\mathbb{C}^n$ dans $\mathbb{C}^n$ et $F'(X^k)$ est la dérivée de F au point X^k (la matrice Jacobienne $F'(X^k)$ est supposée inversible en chaque X^k) .

(1.1) peut s'écrire sous la forme suivante:

$$\left[[\, F'(X^k) \right] . \left[[\, \Delta X^k \right] = - \left[F(X^k) \right] \qquad (1.2)$$

En appliquant la méthode de Newton au système de Viète (V), nous obtenons le système (1.3):

$$\begin{bmatrix} 1 & \dots & 1 \\ -(x_2+x_3+\dots+x_n) & \dots & -(x_1+x_2+\dots+x_{n-1}) \\ \quad . & & \\ \quad . & & \\ \quad . & & \\ (-1)^{n+1}x_2x_3\dots x_n & \dots & (-1)^{n+1}x_1x_2\dots x_{n-1} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ . \\ . \\ . \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} F_1(x) \\ F_2(x) \\ . \\ . \\ . \\ F_n(x) \end{bmatrix}$$

En calculant $(x_i^{n-1}F_1(X) + x_i^{n-2}F_2(X) + \ldots + F_n(X)$ pour i variant de 1 à n à partir des équations du système (1.3), nous obtenons le système équivalent:

$$\begin{bmatrix} A_1(X).\Delta x_1 \\ A_2(X).\Delta x_2 \\ \cdot \\ \cdot \\ \cdot \\ A_n(X).\Delta x_n \end{bmatrix} = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ & & \ldots & & \cdot \\ & & \ldots & & \\ & & \ldots & & \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} F_1(X) \\ F_2(X) \\ \cdot \\ \cdot \\ \cdot \\ F_n(X) \end{bmatrix}$$

où $A_p(X) = (x_p - x_1)(x_p - x_2) \ldots (x_p - x_{p-1})(x_p - x_{p+1}) \ldots (x_p - x_n)$, $1 \leq p \leq n$.

A partir des équations du système de Viète, nous obtenons:

$x_i^{n-1} F_1(X) + x_i^{n-2}F_2(X) + \ldots + F_n(X) = P(x_i)$ pour i variant de 1 à n. En effet:

$$\sum_{q=1}^{n} x_1^{n-q} F_q(X) = \sum_{q=1}^{n} x_1.a_q - \sum_{q=1}^{n} (-1)^q x_1^q \sum_{i,j,\ldots,p}^{i<j<\ldots<p} x_i.x_j \ldots x_p \quad \text{(q facteurs)}$$

$$= \sum_{q=1}^{n-1} a_q x_1^q - \sum_{q=1}^{n-1} (-1)^q.x_1^q \sum_{i,j,\ldots,p}^{i<j<\ldots<p} x_i.\, x_j \ldots x_p \quad \text{(q facteurs)}$$

$$= \left(\sum_{q=1}^{n-1} a_q x_1^q + x_1^n \right) - \sum_{q=1}^{n-1} (-1)^q .x_1^q. \sum_{i,j,\ldots,p}^{i<j<\ldots<p} x_i.x_j \ldots x_p \quad (x_1^n \text{ est suprimé})$$

Le deuxième membre de l'égalité est nul car si on multiplie par x_1 le $(s-1)^{\text{ième}}$ terme de la somme à l'étape suivante, on obtient l'opposé de ce résultat pour s variant de 1 à n-1. Ce qui finit la démonstration.

Comme la méthode de Durand-Kerner est équivalente à la méthode de Newton, il existe $d > 0$ telle qu'elle converge quadratiquement vers α pour tout point de départ choisi de telle manière que la distance entre ce dernier et α soit

inférieure à d si det(F'(α)) est différent de 0 i.e. si les zéros α_i , i=1,...,n, de P_n sont tous simples. En ce qui conserne sa convergence locale, nous appliquons le théorème d'Ostrowski à la fonction H telle que $H(z) = z - [F'(z)]^{-1}. F(z)$ (H est l'opérateur de Newton) où $F = (F_1, ...,F_n)$, $F_k(z) = S_k(z) + (-1)^{k-1} a_k$, k=1,2,...,n, et S_k sont les fonctions symétriques élémentaires des racines.

Théorème d'Ostrowski[ROB84]. *Soit H : $\mathbb{C}^n \to \mathbb{C}^n$. Si H admet un point fixe α. Si H est dérivable en $\alpha=(\alpha_1,..., \alpha_n)$ et admet en α une Jacobienne H'(α) de rayon spectrale inférieur à 1:*
alors il existe autour de α, dans $\mathbb{C}^n$, un voisinage V_α de α tel que:

* $H(V_\alpha) \subseteq V_\alpha$;

* *pour tout z^0 pris dans V_α l'itération:*
$z^{K+1} = H(z^k)$ *reste dans V_α et converge en α ;*

* α *est alors l'unique point fixe de H dans V_α .*

Notons que $(\alpha_i)_{i=1,...,n}$ sont les solutions de P si et seulement si α est point fixe de H et en plus que pour l'opérateur de Newton $H'(\alpha) = 0$.

Remarquons que la fonction H du théorème précédent est égale aussi à $(H_1,...,H_n)$ où H_i est telle que:

$$H_i (z_1,z_2,...,z_n) = z_i - P(z_i)/ Q'(z_i) \quad \text{avec} \quad Q(z) = (z-z_1).(z-z_2)...(z-z_n)$$
et $Q'(z_i)= \pi_{r\neq i}(z_i-z_r)$.

Vue que les fonctions F_k sont symétriques alors la méthode de Durand-Kerner converge vers toute permutation de l'ensemble des racines de P_n (n! permutations).

Le fait que l'on a le choix entre n! solutions, doit faciliter la convergence de la méthode.

Le théorème de convergence ci-dessus porte sur l'existence de α
toutes choses justement inconnues dans un exemple réel. Dans ce qui suit nous donnons un résultat "plus fort", connu sous le nom du théorème de Newton-Kantorovich:

Théorème de Newton-Kantorovich ([CIA82]). *Soit F une application de* $\Omega \subseteq \mathbb{C}^n \to \mathbb{C}^n$ *dérivable dans l'ouvert* Ω. *On suppose qu'il existe trois constantes a,b,c et un point* $x_0 \in \Omega$ *tels que:*

$0 < abc \leq 0.5$,

$B \subseteq \Omega$ où $B = \{ x \in \mathbb{C}^n ; \| x-x_0 \| < \mu^- = (1 - \sqrt{(1 - 2abc)})/bc \}$,

$F'(x_0)$ *inversible et* $\| (F'(x_0))^{-1} F(x_0) \| \leq a$,

$\| (F'(x_0))^{-1} \| \leq b$,

$\sup_{x, y \in \Omega^+} \| F'(x) - F'(y) \| \leq c \| x-y \|$,

où $\Omega^+ = \{ x \in \Omega ; \| x-x_0 \| < \mu^+ = (1 + \sqrt{(1-2abc)})/bc \}$

Alors

$F'(x)$ *est inversible pour tout* $x \in B$,

et la suite définie par:

$$x_{k+1} = x_k - (F'(x_k))^{-1} F(x_k) \quad , k \geq 0,$$

est entièrement contenue dans la boule B, et elle converge vers un zéro a de F, qui est d'ailleurs le seul zéro de F dans l'ouvert Ω^+. *Enfin, posant* $\theta = \mu^- / \mu^+$, *on a les majorations suivantes:*

$$\| x_k - \alpha \| \leq (2.\sqrt{(1-2abc)})/abc . (\theta^{2K}/(1- \theta^{2K}) . \| x_1 - x_0 \| \quad \text{si } a.b.c < 0.5$$

$$\| x_k - \alpha \| \leq \| x_1 - x_0 \| / 2^{(k-1)} \quad \text{si } abc = 0.5$$

$$2.\| x_{k+1} - x_k \| / (1 + \sqrt{(1 + 4.\theta^{2K} (1 + \theta^{2K})^{-2})}) \leq \| x_k - \alpha \| \leq \theta^{2K-1} \| x_k - x_{k-1} \|$$

On notera au passage que les deux dernières inégalités fournissent un encadrement de l'erreur $\| x_k - \alpha \|$ à l'aide de quantités entièrement calculables.

6.3 Convergence globale pour les polynômes quadratiques:

Soit $P_2(z)$ un polynôme de degré 2 : $P_2(z) = z^2 + a_1.z + a_2$.

Dans ce qui suit nous caractérisons l'ensemble de non convergence NC de la méthode de Durand-Kerner appliquée aux polynômes quadratiques et nous montrons que NC est de mesure nulle.

Par la transformation $z \rightarrow z - a_1/2$ $P_2(z)$ devient:

$$p_2(z)=z^2 + (a_2-a_1/4) = z^2 + a.$$

L'ensemble de non convergence NCt pour $p_2(z)$ est une translation de NC. Par conséquent si NCt est de mesure nulle alors NC l'est aussi et NC s'obtient à partir de NCt par translation inverse.
En supposant a=1 on ne restreint pas la généralité du problème. En appliquant la méthode de Durand-Kerner à $p_2(z)$ nous obtenons le schéma itératif suivant:

$$\begin{cases} z_1 = z_1 - (z_1^2 + 1) / (z_1 - z_2) \\ \\ z_2 = z_2 - (z_2^2 + 1) / (z_2 - z_1) \end{cases} \qquad (2.1)$$

Nous avons vu au premier chapitre que $z_1 + z_2=-a_1$ c'est à dire dans notre cas $z_1 + z_2 = 0$ puisque $a_1 = 0$, ceci s'obtient facilement à partir des deux équations ci-dessus. En effet:

$$(z_1 + z_2) = (z_1 + z_2) - [(z_1{}^2 + 1) - (z_2{}^2 + 1)] / (z_1 - z_2) \quad \text{donc}$$

$$(z_1 + z_2) = (z_1 + z_2) - [(z_1 - z_2)(z_1 + z_2)] / (z_1 - z_2) = 0$$

Le schéma itératif (2.1) est équivalent au schéma suivant:

$$\begin{cases} z_1 = z_1 - (z_1^2 + 1) / 2.z_1 \\ \\ z_1 + z_2 = 0 \end{cases} \qquad (2.2)$$

On remarque donc que le schéma itératif de Durand-Kerner est équivalent à un système dont la première équation est le schéma itératif de Newton appliqué à la première composante z_1 et la deuxième équation est une relation enter z_1 et z_2.
L' ensemble de non convergence NCt est tel que :

$$NCt = \{(z_1, z_2) \in \mathbb{C} \times \mathbb{C} \ / \ (z_1, z_2) \text{ ne converge pas vers les 2 racines de } p_2 \}.$$

D'après (2.2) NCt est égale à l'ensemble suivant:

NCt = $\{(z_1, z_2) / z_1$ appartient à NCN et $z_1 + z_2 = 0\}$; NCN est l'ensemble de non convergence de la methode de Newton appliquée à p_2.

Proposition ([MAS84]). *L'ensemble de non convergence NCN est égale à la médiatrice du segment $[z_1, z_2]$ où z_1 et z_2 sont les deux racines de p(z).*

Dans notre cas NCN = $\mathbb{R}x\{0\}$ et alors:

$$NCt=\{(z_1, z_2) / (z_1 - (z_1^2 + 1) / (z_1 - z_2) \in \mathbb{R}x\{0\}\}.$$

Cherchons à déterminer (z_1, z_2) tel que $z_1 - [(z_1^2 - 1) / (z_1 - z_2)] \in \mathbb{R}x\{0\}$.

$$z_1-(z_1^2-1)/(z_1 - z_2) \in \mathbb{R} \Leftrightarrow z_1-(z_1^2+1) / (z_1- z_2) = z_1-(z_1^2+1) / (z_1-z_2) \quad (2.4)$$

(2.4) est équivalent à:

$$(z_1.z_2 + 1) / (z_1 - z_2) = (z_1.z_2 + 1) / (z_1 - z_2) \quad (2.5)$$

De (2.5) nous déduisons que:

$$z_1z_1z_2 + (z_1 - z_2) - z_1z_2z_2 - z_1z_1z_2 - z_1 + z_1z_2z_2 + z_2 = 0$$

$$\Leftrightarrow (z_2 - z_2) + (z_1 - z_1) + z_1.z_1(z_2 - z_2) + z_2z_2(z_1 - z_1) = 0$$

$$\Leftrightarrow (z_2 - z_2)(1 + |z_1|^2) + (z_1 - z_1)(1 + |z_2|^2) = 0$$

$$\Leftrightarrow (z_2 - z_2) / (1 + |z_2|^2) = (z_1 - z_1) / (1 + |z_1|^2)$$

$$\Leftrightarrow \mathrm{Im}(z_1) / (1 + |z_1|^2) = \mathrm{Im}(z_2) / (1 + |z_2|^2) \quad (2.6)$$

Donc d'aprés (2.6), l'ensemble NCt est tel que:

$$NCt= \{ (z_1, z_2) / \mathrm{Im}(z_1) / (1 + |z_1|^2) = \mathrm{Im}(z_2) / (1 + |z_2|^2) \}$$

Soit $z_1 = (x, y)$, alors x et y vérifient:

$$y / (1 + x^2 + y^2) = k \quad (2.7)$$

$$\Leftrightarrow \quad x^2 + (y - 1/2k)^2 = (1/4k^2 - 1) \text{ si } k \neq 0 \quad (2.8) \text{ et } y = 0 \text{ si } k = 0.$$

si $k \neq 0$ et $1/4k^2 - 1 \geq 0$ c'est à dire si $k \in [-1/2, 0[\cup]0, 1/2]$ alors (2.8) est un cercle C_k de centre (0, 1/2k) et de rayon $\sqrt{((1/2k)^2-1)}$ qui ne touche pas l'axe des x.

L'ensemble de non convergence NCt est donc caractérisé comme suit:

$NCt = \{ (z_1, z_2) \in \mathbb{C}x\mathbb{C} / (z_1, z_2) \in (\mathbb{C}x\{0\})^2$ ou $(z_1, z_2) \in C_k x C_k / C_k$ est le cercle de centre (1/2k,0) et de rayon $\sqrt{((1/2k)^2-1)}$ pour $k \in [-0.5,0.5]$ et $k \neq 0$ }

Montrons que l'ensemble NCt est de mesure de Lebesgue nulle dans $\mathbb{C}x\mathbb{C}$.

Théorème. *Soit F une application différentiable de* $\mathbb{C}x\mathbb{C}$ *dans* $\mathbb{R}$ *et A un sous-ensemble de* $\mathbb{C}$ *. Si* $\mu_{\mathbb{R}}(F(A)) = 0$ *alors* $\mu_{\mathbb{C}x\mathbb{C}}(A) = 0$.

Théorème. *Soit NCt l'ensemble de non convergence de la méthode de Durand-Kerner appliquée aux polynômes quadratiques alors NCt est de mesure de Lebesgue nulle.*

Démonstration:

Construisons l'application F de $\mathbb{C}x\mathbb{C}$ vers $\mathbb{R}$:

$$F: \mathbb{C}x\mathbb{C} \rightarrow \mathbb{R} : (z_1, z_2) \mapsto Im(z_1) / (1 + |z_1|^2) - Im(z_2) / (1 + |z_2|^2)$$

F est une application différentiable. Comme F(NCt) = { 0 } (NCt est le noyau de F) alors, d'après le théorème précédent, $\mu_{\mathbb{C}x\mathbb{C}}(NCt) = 0$.

En conclusion, le schéma itératif de Durand-Kerner appliqué aux plynômes quadratiques converge presque partout dans l'ensemble $\mathbb{C}x\mathbb{C}$.

Caractérisons NCt pour les polynômes P_2 ayant leurs racines réelles:

$NCt=\{(x, y) \in \mathbb{R}x\mathbb{R} / (xy + 1)/(x - y) \neq 0 \text{ et } x \neq y \}$

NCt est la droite y=x et l'hyperbole y =1/x. La méthode de Durand-Kerner converge dans dans $\mathbb{R} \times \mathbb{R} - (\{ y = x \}, \{ y = 1/x \})$ vers l'un des deux points (1,-1) et (-1, 1).

La région de convergence vers le point (1, -1) est: $y \neq x$ et $(1 - xy) / (x - y) > 0$. Cette région est représentée sur la figure1.

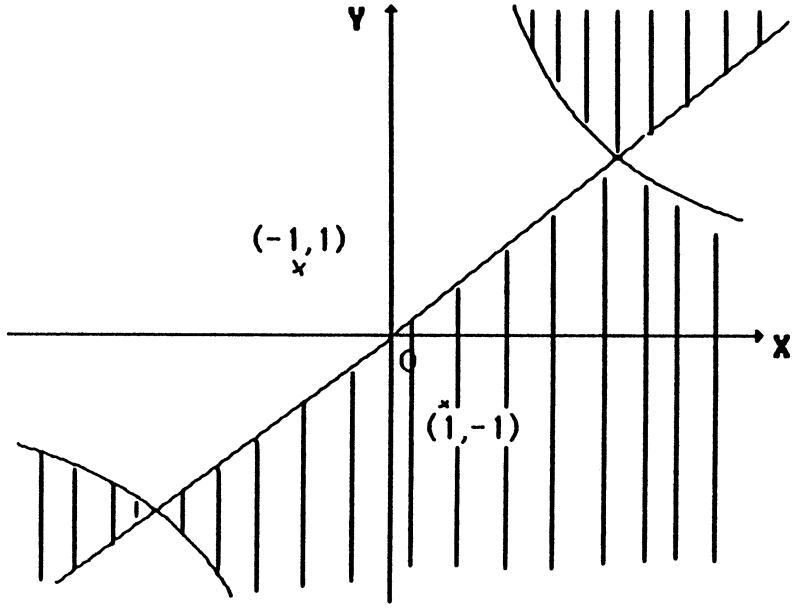


figure 1

Remarquons que la région de convergence vers le point (1,-1) est caractérisée par $y \neq x$ et $[(1 - xy) / (x - y)] < 0$.

La méthode de Durand-Kerner converge presque partout pour les polynômes quadratiques. Comme elle converge souvent en pratique(figure 2) pour n quelconque et que chaque composante d'une itération complète de Durand-Kerner est une itération d'une dérivée de la méthode de Newton, nous conjecturons quand elle est appliquée aux polynômes qui admettent leurs racines réelles et distinctes 2 à 2 qu'elle converge dans $\mathbb{R}^n$ sauf pour un ensemble de mesure de Lebesgue nulle dans $\mathbb{R}^n$.

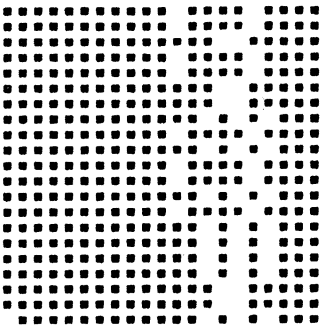


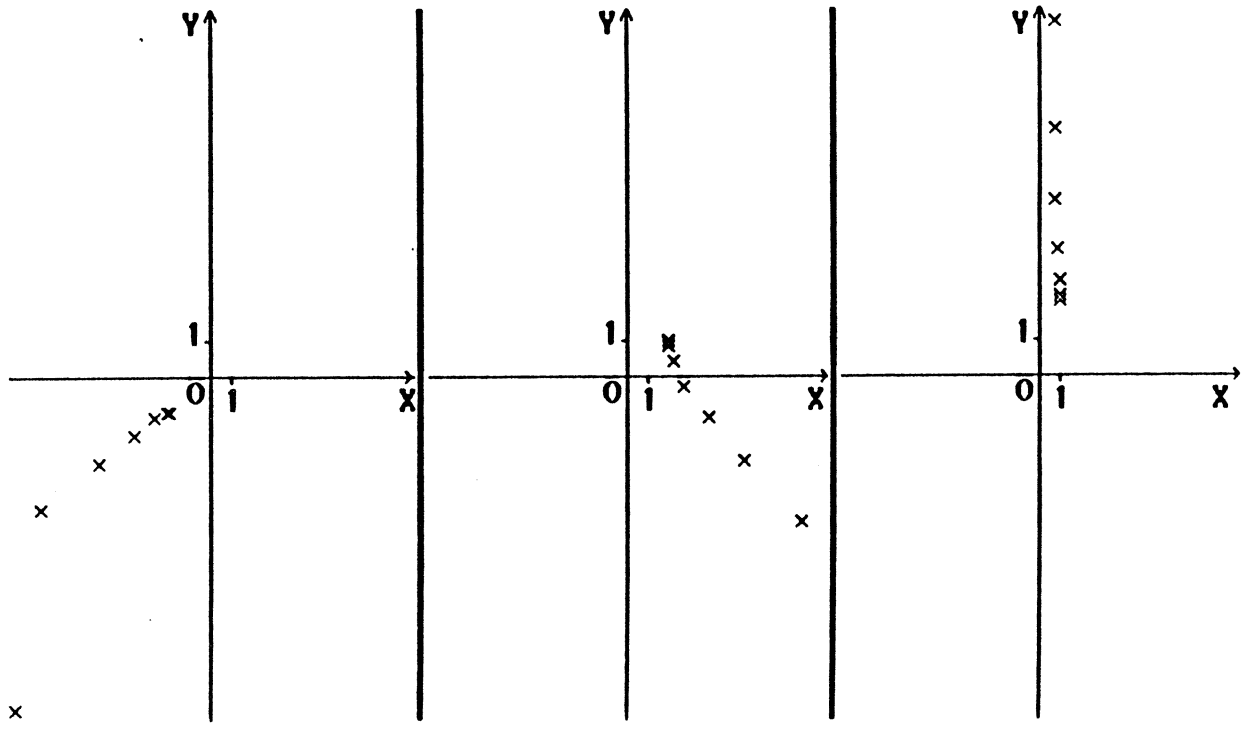
Figure 2

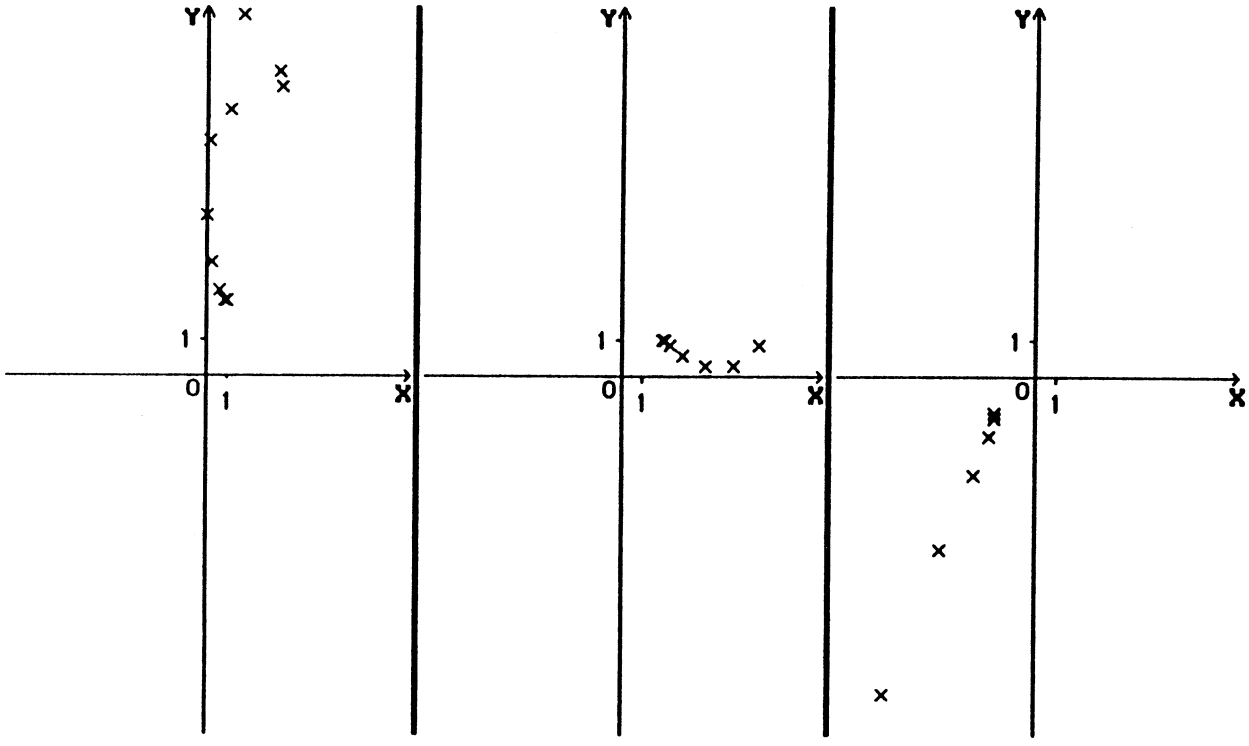
Par la relation $\Sigma_{i=1,\ldots,n} z_i^{(k)} = -a_1$, k=1,2,... où $z_i^{(k)}$ sont les itérés de Durand-Kerner, l'étude en dimension n se ramène en dimension (n-1).

6.4 Illustration du comportement itératif de la méthode de Durand-Kerner appliquée aux polynômes de degré trois:

Le comportement d'itérations d'applications en dimension supérieure à 1 est largement ouvert (notons cependant que dans $\mathbb{C}$,un travail important a été réalisé par Fatou et Julia, travail qui est repris actuellement grâce à l'utilisation de moyens de calculs modernes).

Dans ce qui suit nous présentons deux graphiques(figure 3 et 4) illustrant, à partir de deux points de départ, le comportement itératif de la méthode de Durand-Kerner appliquées aux polynômes de degré 3 à coefficients complexes.




Y
1
0
1
X
Y
1
0
1
X
Y
1
0
1
X

Chapitre 7

EXPERIMENTATION SUR UNE ARCHITECTURE VECTORIELLE

7.1 Introduction

Nous avons comparé sur le processeur vectoriel FPS264 (Floating point systems) les 6 méthodes itératives de la recherche simultanée des racines d'un polynôme à coefficients complexes (DK), (DK1), (DKGS), (EH), (EH1) et (EHGS) étudiées auparavant dans le premier chapitre. Cette comparaison expérimentale est menée de point de vue du nombre d'itérations et du temps de calcul sur des polynômes de degré variant entre 4 et 25. Les résultats obtenus pour chacune de ces méthodes, nous ont permis d'en séléctionner les bonnes.

Nous avons ensuite donné une procédure de choix du point de départ qui a amené à une convergence plus rapide des 6 méthodes que pour d'autres points de départ.

Par ailleurs, nous avons comparé pour les méthodes (DK) et (DKGS) les vitesses de traitement [COS86] du processeur vectoriel FPS264 pour un ensemble de polynômes de degré 4, 7, 9, 12, 20 et 25. Ensuite nous avons vectorisé les calculs des corps des boucles des programmes [COS86], [HWA84], [DGK84] correspondant à (DK) et (DKGS) pour utiliser efficacement les unités pipelines afin d'améliorer les vitesses de traitement pour ces dernières.

7.2 Comparaison des méthodes pour des polynômes de degré variant entre 4 et 25:

Nous avons comparé les méthodes de point de vue du temps de calcul et du nombre d'itérations sur des polynômes dont le degré varie entre 4 et 25. Dans ce qui suit, nous présentons dans les tableaux 1 et 2 les résultats obtenus pour un polynôme de degré 4 et pour un autre de degré 25.

Exemple: $P_4(z) = (z-6)\,(z-5)\,(z+3)\,(z+7)$ $\quad z^0=(-7.5, -2.5, 2.5, 7.5)$

méthodes	temps			itérat.	ordre
	MULTICS	FPS	COEF(Tm/Tf)		
GRANT	—	13.1ms	—	10	2
(DK)	0.052s	1.17ms	43	9	2
(DK1)	0.152s	4.89ms	31	7	3
(DKGS)	0.044s	1.07ms	38	8	Entre 2,3
(EH)	0.161s	5.33ms	31	7	3
(EH1)	0.173s	5.21ms	33	5	4
(EHGS)	0.057s	1.25ms	46	6	Entre 3,4

Tableau 1

$P_{25}(z)$ polynôme de degré 25, ses coefficients et un point de départ sont tels que:

(4,8) (34,5) (23,-12) (2,0.9) (1.9,-0.99) (2.1,0.95)
(6,-0.96) (2,4) (2,0.1) (1.1,4.1) (1.3,2.1) (2.1,5.3)
(1.2,2.3) (4.3,4.2) (2.12,3.2) (3.01,9) (5.6,0.1) (3.12,9)
(3.52,6.4) (12,0.6) (2.3,8.6) (1.2,4.7) (3.98,1.09) (3.98,41)
(0.99,6.5).

(-2,0) (-3,0) (10,0) (4,0) (-8,0) (-12,0)
(40,0) (1,0.9) (3.1,30) (3,0.4) (4.1,4.2) (1.8,0.4)
(3.2,2.1) (3.3,0.6) (1.32,5.1) (1.9,2.4) (3.08,5.3) (0.1,8.2)
(6,0.6) (2.1,0.9) (7.34,1.75) (3.45,4.66) (3.23,0.35) (4.3,3.6)
(9.8,3.76).

méthodes	temps			itérat.	ordre
	MULTICS	FPS	COEF(Tm/Tf)		
GRANT	—	9.7s	—	203	2
(DK)	69.3s	0.92s	75.3	191	2
(DK1)	241.18s	3.82s	62.1	153	3
(DKGS)	72.45s	0.89s	81.4	188	Entre 3,4
(EH)	264.74s	4.51s	58.7	156	3
(EH1)	261.40s	4.23s	61.8	127	4
(EHGS)	68.43s	0.94s	72.8	142	Entre 4,5

Tableau 2

Remarquons que le test de la (seule) méthode de la recherche des racines d'un polynôme de Grant et Hitchins de la bibliothèque du processeur vectoriel FPS264

montre qu'elle est 10 fois moins rapide que celle de Durand-Kerner (DK). La méthode de Grant et Hitchins est telle que:

Les racines sont localisées par une méthode basée sur l'utilisation simultanée de la méthode de Newton pour résoudre:

$$Re(P_n) = Im(P_n) = 0$$

et la méthode de la plus forte pente(steepest descent):

$$Re^2(P_n) + Im^2(P_n) = 0$$

où $Re(P_n)$ est la partie réelle de P_n et $Im(P_n)$ la partie imaginaire.

Le test d'un grand nombre de polynômes de degré 4 qui ont leurs racines entières et appartiennent à l'intervalle [-10,10] montre que si l'on choisit z1=(-7.5,-2.5, 2.5, 7.5), z2=(-10, -5, 5, 9) ou z3=(-9, -5, 5, 10) comme point de départ alors les 6 processus itératifs convergent et le nombre d'itérations varie entre 5 et 10 pour aboutir à une précision de 10^{-6}.

La méthode (DK) converge plus rapidement pour z1 que pour z2 et z3; le temps de calcul et le nombre d' itérations moyens sont de 1.2ms et 8 pour z1 et de 1.4ms et 10 pour z2 et z3. Par ailleurs les vitesses de convergence des méthodes (DK) (approche de Jacobi) et (DKGS) (approche de Gauss-Seidel) sont très voisines et sont 4 fois plus grandes que celle de la méthode (DK1) pour le point z1 et 5 fois plus grandes pour z2 et z3.

La méthode (EHGS) converge aussi plus vite pour z1 que pour z2 et z3 et sa vitesse de convergence est légérement inférieure à celle de (DK). Elle est presque 4.5 fois plus grande que celle de (EH) et (EH1) pour z1 et presque 5.5 fois plus grande pour z2 et z3.

Pour de nombreux polynômes de degré supérieur à 4 et inférieur à 25 les 6 méthodes convergent pour un grand nombre de points de départ qu'on a choisi. Les vitesses de convergence des processus itératifs (DK), (DKGS) et (EHGS) sont voisines et nettement meilleures que celles de (DK1), (EH) et (EH1) (tableau 2).

7.3 Procédure pour le choix de point de départ:

Dans ce qui suit nous allons nous interesser au choix de points de départ pour des polynômes de degré quelconque. Pour des polynômes de degré 7,9,12 aussi que pour le polynôme de WILKINSON [WIL65] de degré 15 nous donnons le nombre d'itérations pour chaque choix pour la méthode de Durand-Kerner en mode parallèle et en mode série pour obtenir une précision de 10^{-6} sur les racines.

Soit P un polynôme de degré n à coefficients complexes a_1, a_2, ... et a_n:

$$P(z) = z^n + a_1 z^{n-1} + \ldots + a_n$$

dans la propriété suivante nous présentons une borne interessante des racines:

Propriété([DEM73]): *Soit w_1, w_2, ... et w_n les racines du polynôme P, alors:*

$$|w_j| \leq (1 + \max |a_i|) \quad pour\ j \in \{1, 2, \ldots, n\}.$$

Une première proposition de points de départ est de les choisir sur le cercle de rayon r0 = (1 + max $|a_i|$) répartis d' une manière uniforme:

$$z_k = r_0 . e^{i2k\pi/n} \quad k=1,2,\ldots,n.$$

Des expérimentations numériques montrent que le nombre des itérations nécessaires pour la convergence dépend essentiellement et d'une manière irrégulière du choix de r_0.

Si r_0 est supérieur à la valeur absolue de la plus grande racine pour des polynômes de degré supérieur à 10 ou si r_0 est inférieur à la plus petite racine en valeur absolue alors le problème de l' instabilité connu se pose.
La procédure suivante [GUG85] pour choisir r_0 , non loin de l'optimum (à priori inconnu) produit des résultats numériques excellents.

Soit $P(z) = a_0 z^n + \ldots + a_{n-1} z + a_n$:

a) • Pour i := 1 jusqu'à n si $a_i \neq 0$ alors $u_i := 2\,|a_i / a_0|^{1/i}$,

• Pour j := 0 jusqu'à (n-1) si $a_j \neq 0$ alors $v_j := 1/(2\,|a_j / a_n|^{1/(n-j)})$,

b) • Calcul de $u = \sum_1^n u_i / n$ et de $v = \sum_1^n v_i / n$,

• Enlever u_i pour lesquels $(u_i - u)$ est maximal;

• Enlever v_j pour lesquels $(v - v_j)$ est maximal;

• Calcul à nouveau de la moyenne des u_i et de la moyenne des v_j.

c) $r_0 = (u + v) / 2$.

Pour le troisième choix de points de départ nous ramenons le centre de gravité des racines à l' origine par la transformation suivante $x \mapsto x+t$ où $t = -a_1/n.a_0$ et nous appliquons la procédure précedente aux coefficients du nouveau polynôme obtenu. Les points choisis sont tels que:

$$z_k = -a_1/na_0 + r_0.e^{i2k\pi/n} \quad k=1,\ldots,n.$$

Dans le tableau 3 nous présentons pour 3 polynômes de degré 7, 9, et 12 le nombre d' itérations obtenus pour les 3 choix de points de départ:

$P_7(z)=z^7-6.01z^6+12.54z^5-8.54z^4-5.5z^3+12.54z^2-8.03z+2.01 \rightarrow r_0=1.71.$

$P_9(z) = z^9 - 9z^8 + 9z^7 - 8z^6 + z^3 + 3z - 1 \rightarrow r_0 = 1.51.$

$P_{12}(z)=z^{12}-12z^{11}+10z^{10}+35z^9-120z^7+140z^5-43z^3+120z^2-35z+195 \rightarrow r_0 = 2.44.$

	P7		P9		P12	
choix	(DK)	(DKGS)	(DK)	(DKGS)	(DK)	(DKGS)
choix 1	86	81	93	89	149	139
choix 2	78	73	85	77	137	129
choix 3	75	71	81	75	132	124

Tableau 3

Nous remarquons que le nombre d'itérations diminue du choix1 au choix3 pour la méthode de Durand-Kerner en mode parallèle et en mode série.

Certains polynômes sont mal conditionnés en ce sens qu' une petite variation des coefficients entraine une grande variation des racines. On observe en général un tel phénomène pour des polynômes dont les racines sont groupées loins de l'origine ou des polynômes à gros coefficients.

Pour atténuer ces phénomènes nous ramenons le centre de gravité des racines de P à l'origine; dans un deuxième temps nous pouvons effectuer la transformation $z \mapsto \mu.z$ qui a pour effet de séparer les racines.

Dans ce qui suit nous présentons pour les trois choix de points de départ ci-dessus les racines et le nombre d' itérations obtenus par la méthode de Durand-Kerner pour le polynôme de WILKINSON de degré 15 . Ce dernier est tel que:

$$P_{15}(z) = \pi_{i=1,15}(z - i) = z^{15} - 120z^{14} + 6580z^{13} - 218400z^{12} + 4899622z^{11} - 78558480z^{10} + 928095740z^{9} - 8207628000z^{8} + 54631129553z^{7} - 272803210680z^{6} + 1009672107080z^{5} - 2706813345600z^{4} + 5056995703824z^{3} - 6165817614720z^{2} + 4339163001600z - 1307674368000.$$

Le premier choix de points de départ est tel que:

$$Z_k = r_0 e^{i2k\pi/n}, \; k=1,2,\ldots,n \text{ où } r_0 = (1 + \max |a_i|) = 6165817614721.$$

Le deuxième choix de points de départ est tel que:

$$Z_k = r_0 e^{i2k\pi/n}, \; k=1,2,\ldots,n \text{ où } r_0 = 12.228957$$

Le troisième choix de points de départ correspond à:

$$Z_k = -a_1/n + r_0 . e^{i2k\pi/n} = 8 + 12.228957 . e^{i2k\pi/n}, \; k=1,2,\ldots,n .$$

Pour ces trois choix de points de départ nous avons obtenu les résultats suivants:

1	(0.42,0.18)	(1.00,1.7E-12)	(1.00,1.9E-28)
2	(1.52,0.28)	(7.07,5.2E-9)	(7.04,6.9E-18)
3	(2.7,0.42)	(2.00,1.0E-15)	(2.00,3.0E-27)
4	(3.85,0.48)	(4.00,2.2E-15)	(4.00,-1.9E-28)
5	(4.9,0.43)	(3.00,2.7E-16)	(3.00,5.0E-26)
6	(5.88,0.41)	(5.01,-1.9E-10)	(5.01,1.0E-19)
7	(6.94,0.48)	(6.04,1.6E-9)	(6.02,6.3E-16)
8	(7.96,0.53)	(8.05,2.1E-5)	(8.03,1.7E-21)
9	(9.17,0.47)	(9.06,1.7E-5)	(9.04,9.2E-18)

10	(10.03,0.46)	(12.00,5.1E-16)	(12.00,6.4E-23)
11	(11.10,0.52)	(11.06,-1.0E-9)	(11.04,4.2E-16)
12	(12.30,0.66)	(10.07,-7.7E-9)	(10.04,-3.8E-17)
13	(13.23,0.68)	(14.00,-6.5E-22)	(14.00,-5.12E-28)
14	(14.02,0.70)	(15.00,-3.1E-26)	(15.00,-6.4E-27)
15	(14.99,0.47)	(13.00,5.1E-27)	(13.00,4.3E-28)
Nombre d' itérations:	501	161	156

(Le nombre maximum d'itérations que nous avons fixé est 500.)

Remarquons que la précision devient meilleure et le nombre des itérations diminue du choix1 au choix3.

7.4 Amélioration de la vitesse du traitement pour (DK) et (DKGS):

Dans ce qui suit nous présentons pour la méthode de Durand-Kerner en mode parallèle et en mode série les vitesses de traitement du processeur vectoriel FPS264 pour 6 polynômes de degré 4, 7, 9, 12, 20 et 25. Puis nous représentons la vitesse de traitement en fonction de degré du polynôme. Ensuite nous expliquons comment nous avons vectorisé les 2 algorithmes (DK) et (DKGS) pour augmenter les vitesses de traitement que nous comparons à celles des algorithmes non vectorisées.
Les 6 polynômes que nous avons testés sont :

$P_4(z) = (z-5)\,(z+9)\,(z-1)\,(z+3)$;

$P_7(z) = z^7-6.01z^6+12.54z^5-8.54z^4-5.5z^3+12.54z^2-8.03z+2.01$;

$P_9(z) = z^9 - 9z^8 + 9z^7 - 8z^6 + z^3 + 3z - 1$;

$P_{12}(z) = z^{12}-12z^{11}+10z^{10}+35z^9-120z^7+140z^5-43z^3+120z^2-35z+195$;

$P_{20}(z) = \pi_{i=1,\ldots,20}\,(z - i)$;

$P_{25}(z)$: (cf p. 100)

Dans le tableau 4 nous reportons pour les 2 méthodes (DK) et (DKGS) les résultats obtenus des vitesses de traitement. Celles-ci sont calculées de la façon suivante:

nous multiplions le nombre d'opérations arithmétiques par le nombre d'itérations. Le résultat obtenu est divisé par le temps total de calcul; ce qui correspond au nombre d'opérations flottantes traitées par seconde(i.e. vitesse de traitement).

Dans le cas des opérations dans le champ complexe nous remplaçons une addition complexe par 2 additions flottante, une multiplication complexe par 2 additions et 4 multiplications flottantes et une division complexe par 4 additions, 6 multiplications et 2 divisions flottantes.

Polyn. / Méth.	P4	P7	P9	P12	P20	P25
(DK)	1.488MF	1.62MF	1.78MF	1.89MF	1.97MF	2.002MF
(DKGS)	1.492MF	1.64MF	1.809MF	1.91MF	1.99MF	2.04MF

Tableau 4.

Le tableau 4 montre que la vitesse de traitement croît vite de P4 à P12 et lentement entre P12 et P25 aussi bien pour (DK) que pour (DKGS). Les vitesses de traitement de celles-ci sont légérement différentes pour chaque polynôme (figure1).

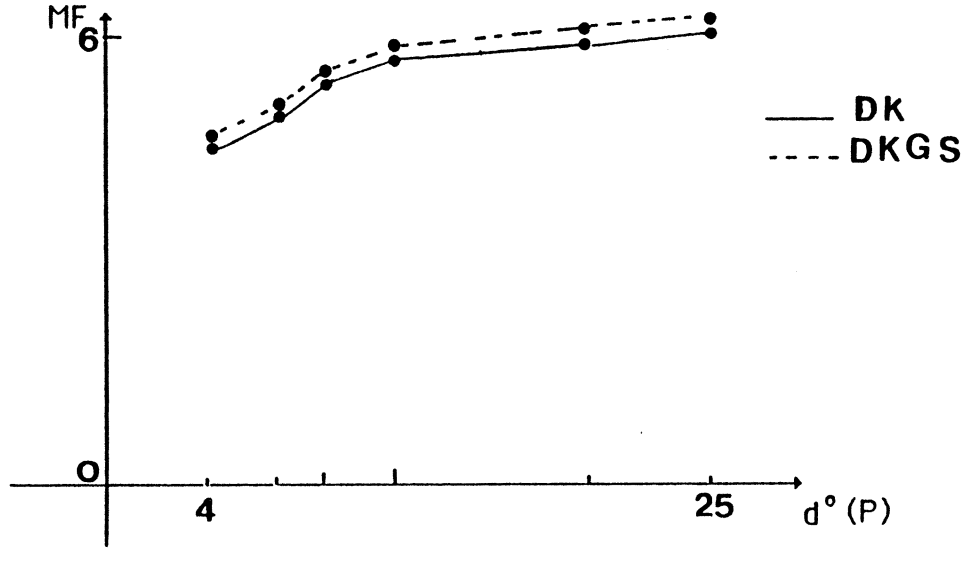


Figure 1 Performance en fonction du degré de polynôme.

Pour augmenter la vitesse de traitement, nous avons vectorisé les algorithmes (DK) et (DKGS). Pour celà nous avons vectorisé les calculs des corps des boucles des 2 programmes correspondant à ces dernières pour utiliser efficacement les unités pipelines, ensuite nous avons utilisé des procedures de la bibliothèque, APMATH64, du FPS. Nous expliquons ceci pour la méthode (DK).

La boucle principale du programme de cette méthode est telle que:

```
 DO 1 i=1,n

    t=z(i) + a(1)

    s=(1.,0.)

  (* évaluation de la valeur de P au point zi,par le schéma de Hörner *)

    DO 2 m=2,n
       t=t*z(i) + a(m)
2   Continue

  (* évaluation du produit des différences entre les z(i) *)

    DO 3 m=1,n
       IF ( m.ne.i ) s=s*(z(i)-z(m))
3   Continue

  (* calcul d'un nouveau itéré *)
    nz(i)=z(i) - t/s

1 Continue
```

Dans la boucle de l'évaluation de la valeur du polynôme ,au point z(i), suivante:

```
   DO  2  m=2,n
      t=t*z(i) + a(m)
2  Continue
```

Nous remarquons que l'évaluation de l'expression (t*z(i) + a(m)) dépend de la valeur de t du premier membre ainsi le calcul d' une nouvelle valeur de t ne pourra commencer que si sa valeur précédente est déjà évaluée [COS86].
Par ailleurs dans la boucle, du calcul du produit des différences entre les z(i), suivante:

```
   DO 3 m=1,n
      IF (m.NE.i) s=s*(z(i)-z(m))
3  Continue
```

Nous constatons qu'en plus de la dépendance entre s*(z(i)-z(m)) et s, à chaque pas de la boucle, nous devons faire un test.

Pour remédier à ces 2 problèmes nous avons inversé les boucles i et m de la boucle principale [COS86] et nous avons décomposé la boucle i (i=1,2,...,n et i≠ m) en deux boucles (i=1,2,...,m-1) et ensuite (i=m+1,m+2,...,n). La version vectorisée du calcul que nous avons obtenu est telle que:

```
   DO 1 i=1,n
      t(i) = z(i) + a(1)
      s(i)=(1.,0.)
1  Continue

   DO 2 m=2,n
      DO 2 i=2,n
         t(i) = t(i)*z(i) + a(m)
2  Continue

   DO 3 m=1,n

      DO 4 i=1,(m-1)
         s(i) = s(i) * (z(i) - z(m))
4     Continue

      DO 5 i=(m+1),n
         s(i) = s(i) * (z(i) - z(m))
5     Continue

3  Continue

   DO 6 i=1,n
      b(i) = z(i) - t(i)/s(i)
6  Continue
```

Nous avons ensuite utilisé des routines arithmétiques, qui existent dans la bibliothèque du processeur vectoriel FPS, écrites en assembleur, pour faire les calculs des boucles. Par suite nous avons obtenu la version vectorisée suivante de la boucle principale du programmme correspondant à la méthode (DK):

```
 DO 1 i=1,n
    t(i) = z(i) + a(i)
    s(i)=(1.,0.)
1 Continue

 DO 2 m=2,n
    cvmadd(t,1,z,1,t,1,a(m),n)
2 Continue

 DO 3 m=1,n
    cvsubm(z,1,z,1,s,1,s,1,(m-1))
    cvsubm(z,1,z,1,s,1,s,1,(m+1))
3 Continue

 cvdivs(t,1,s,1,z,1,nz,1,n)
```

La première routine "cvmadd" effectue le produit entre les 2 vecteurs t et z élément à élément et ajoute a(m). La seconde "cvsubm" effectue la différence entre les éléments du vecteur z et z(m) puis multiplie le résultat par s(i) pour le premier calcul de 1 à (m-1) et de (m+1) à n pour le second calcul. Enfin par la routine "cvdivs" nous effectuons pour chaque composante des 3 vecteurs z, t et s la différence entre une composante de z et le rapport entre une composante de t et de s. Ce qui permet de calculer les n nouveaux itérés nz.
Les résultats que nous avons obtenus pour les 2 algorithmes (DK) et (DKGS) vectorisés pour les polynômes P4, P7, P9, P12, P20 et P25 sont reportés dans le tableau 5 ainsi que ceux des algorithmes (DK) et (DKGS) non vectorisés.

POLYN. / Méth.	P4		P7		P9		P12		P20		P25	
(DK)	3.02	1.4	4.12	1.6	4.60	1.78	5.32	1.89	5.75	1.97	6	2
(DKGS)	3.13	1.5	4.32	1.6	4.8	1.8	5.49	1.9	5.79	1.99	6.05	2.04

tableau5

La comparaison de ces résultats montre d'une part que les vitesses de traitement sont voisines pour les 2 méthodes (DK) et (DKGS) vectorisées (figure 2) et d'autres part qu'elles sont nettement meilleures que celles de (DK) et (DKGS) non vectorisées (tableau 5).

Nous constatons aussi que la vitesse de traitement passe de 2.04 à 6.05 MFLOPS après vectorisation de (DKGS) et de 2.002 à 6 MFLOPS pour (DK).

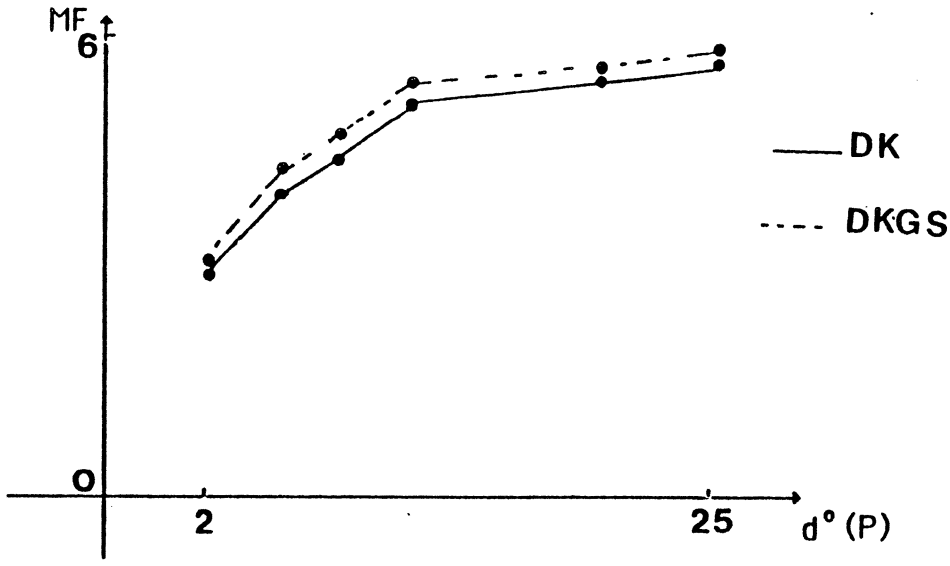


Figure 2 Performance en fonction du degré du polynôme.

Chapitre 8

ARCHITECTURE PARALLELE POUR LE CALCUL DES ITERATIONS DE LA METHODE DE DURAND-KERNER

8.1 Introduction:

Pour la recherche simultanée des racines d'un polynôme à coefficients complexes, nous avons étudié sept méthodes itératives. En comparant celles-ci de point de vue du temps de calcul et du nombre d'itérations, nous avons conclu que la méthode dite de Durand-Kerner est l'une des plus performantes car elle fait la synthèse d'un ordre de convergence assez élevé et d'une itération peu coûteuse.

Cette méthode est telle que:

Soit P un polynôme de degré n à coefficients complexes a_0, a_1, ... et a_{n-1}:

$$P(z) = z^n + a_{n-1}z^{n-1} + \ldots + a_1 z + a_0 .$$

Pour i:=1 jusqu'à n faire

Début

$$z_i^{k+1} := z_i^k - P(z_i^k) / \prod_{j=1, j\neq i}^{n} (z_i^k - z_j^k)$$

Fin.

Dans ce qui suit nous allons nous interesser à l'introduction d'une architecture parallèle [COS86] adaptée au calcul des itérations complètes de cette méthode.

8.2 Nouvelle formulation de l'algorithme:

Nous avons obtenu une nouvelle formulation de l'algorithme en explicitant les différentes étapes auquelles participent les variables.

Pour i := 1 jusqu'à n faire $z_i^0 := b_i$;
k := 0;
limite := N;
précision := ε ;

Tant que [(norme ≥ précision) et (k ≤ limite)] faire
Début
k := k + 1;
Pour i := 1 jusqu'à n faire

Début
$P_i := 1$; $\pi_i := 1$;
Pour j := 1 jusqu'à n faire
Début
$P_i := P_i * z_i^{(k-1)} + a_j$;
Si $(j \neq i)$ alors $\pi_i := \pi_i * (z_i^{(k-1)} - z_j^{(k-1)})$;
Fin
$z_i^k := z_i^{(k-1)} - P_i / \pi_i$;
Fin;
norme := max $| z_i^k - z_i^{k-1} |$;
Fin.

Dans la nouvelle formulation de l'algorithme on initialise les approximations initiales z_i^0 aux valeurs b_i, la précision de calcul des racines, precision, à la valeur ε et le nombre maximum d'itérations, limite, à N.

Pour chaque $z_i^{(k-1)}$, i=1 (1) n et k=1, 2, ... , on calcule $P_i = P(z_i^{(k-1)})$, la valeur du polynôme P au point $z_i^{(k-1)}$, par le schéma de Hörner ainsi que le produit π_i des différences entre $z_i^{(k-1)}$ et $z_j^{(k-1)}$ pour j=1 (1) n et j≠i. Pour cela, on initialise d'abord P_i à 1, coefficient de z^n, et π_i à 1. Ensuite , pour j=1 (1) n, on calcule ($P_i * z_i^{(k-1)} + a_i$) que l'on affecte à P_i, puis on calcule $\pi_i * (z_i - z_j)$ si j est différent de i et on le place dans π_i. A partir des valeurs P_i et π_i, on détermine le nouveau itéré $z_i^{(k)}$ par $(z_i^{(k-1)} - P_i/\pi_i)$.

Ensuite on calcule la valeur de la norme, qui est égale dans notre cas au maximum des $| z_i^k - z_i^{(k-1)} |$ (norme max.), et on l'affecte à norme. Si celle-ci est supérieur à e et le nombre d'itérations k est inférieur à N alors on itère le procédé du calcul décrit ci-dessous.

8.3 Adaptation d'une architecture parallèle au calcul des itérations de de la méthode de Durand-Kerner:

On distingue plusieurs types de reseaux parallèles [COS86]. Ils sont classés généralement en fonction de la géométrie des connections entre les cellules qui les composent; les reseaux linéaires et les reseaux toriques sont les plus simples. C'est ce dernier type de reseaux que nous allons adapter au calcul des itérations de Durand-Kerner.

On se donne un reseau circulaire de n cellules travaillant toutes en parallèle et dans lequel les données circulent dans la même direction d'une cellule à une autre (figure1). On suppose que toutes les valeurs des coefficients du polynôme a_i, des approximations initiales z_i, N le nombre maximum d'itérations et ϵ la précision de calcul entrent simultanément dans le réseau par un dispositif externe avant le début de calcul (figure 2).

La cellule (i) de la figure 2 reçoit par les canaux de communication nord can.N1, can.N2, can.N3, can.N4 respectivement les valeurs N, ε, z_i et a_{i-1}.

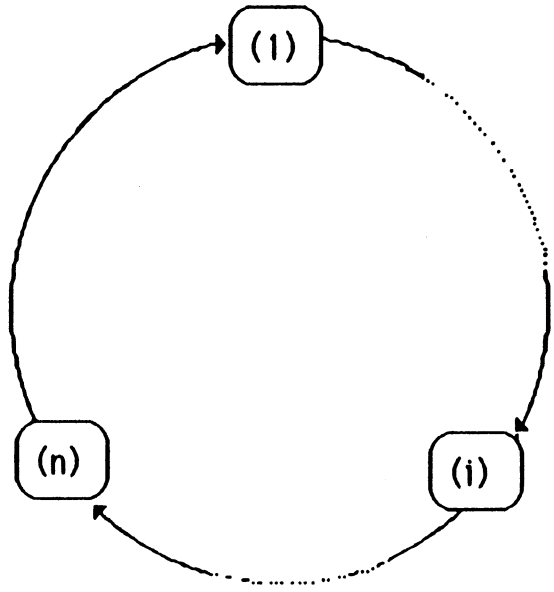


figure1 reseau torique de processeurs.

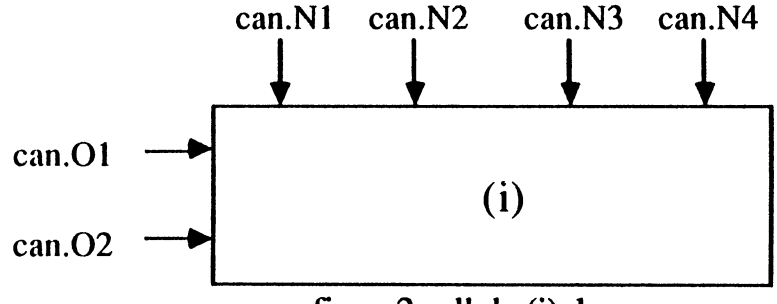


figure2 cellule (i) du reseau.

Soit P un polynôme de degré n à coefficients complexes b_0, b_1, ... et b_{n-1}:

$$P(z) = z^n + b_{n-1}z^{n-1} + \ldots + b_0$$

pour lequel on veut calculer toutes les racines en parallèle sur un reseau circulaire de cellules ou de processeurs.

Dans chaque cellule (i) du reseau, i=1, 2, ..., n on veut calculer par le schéma de Hörner la valeur du polynôme au point z_i, i=1, 2, ..., n. Pour cela chaque cellule doit recevoir à une étape i de calcul, la valeur a_{n-i}. Ceci ne peut pas se produire sur un reseau circulaire.

Pour remédier à ce problème nous proposons une décomposition de l'évaluation de la valeur du polynôme en z_i, par le schéma de Hörner, en trois étapes et une réorganisation de la circulation des a_i dans le reseau.

On transforme d'abord P(z) par le changement de variable $z \mapsto z-b_{n-1}/n$ en un autre polynôme p(z) de même degré et dont le coefficient de z^{n-1} est nul:

$$p(z) = z^n + a_{n-2}z^{n-2} + \ldots + a_0$$

que l'on peut écrire sous la forme suivante:

$$p(z) = (a_0 + a_1z + \ldots + a_{i-1}z^{i-1}) + (a_iz^i + a_{i+1}z^{i+1} + \ldots + a_{n-2}z^{n-2} + z^n)$$

ou encore

$$p(z) = (a_0 + a_1z + \ldots + a_{i-1}z^{i-1}) + z^i (a_i + a_{i+1}z + \ldots + a_{n-2}z^{n-2-i} + z^{n-i}).$$

Posons $A_1(z) = (a_{i-1}z^{i-1} + \ldots + a_0)$ et $A_2(z) = (z^{n-i} + a_{n-2}z^{n-2-i} + \ldots + a_i)$ alors:

$$p(z) = A_1(z) + z^i A_2(z).$$

Pour l'évaluation de la valeur de p en un point z nous procédons en trois étapes successives:

étape1:

$$A_1(z) = a_{i-1}z^{i-1} + \ldots + a_0$$
$$B = z^i$$

au cours de cette étape nous évaluons $A_2(z)$ par le schéma de Hörner et nous calculons $B=z^i$.

étape2:

$$A_2(z) = z^{n-i} + a_{n-2}z^{n-i-2} + \ldots + a_i$$

dans cette étape nous calculons aussi la valeur de $A_2(z)$ au point z , par le schéma de Hörner.

étape3:

$$p(z) = A_1(z) + B*A_2(z).$$

Pour calculer $p(z_i)$, i=1 (1) n, dans la cellule (i), i=1 (1) n, en parallèle nous proposons une circulation des données dans le reseau telle qu'au début du calcul, la valeur a_{i-2} soit dans le canal de communication can.O1 à l'entrée da la cellule (i) et la valeur a_{i-1} soit à sa sortie dans le canal can.E1 (figure 3)

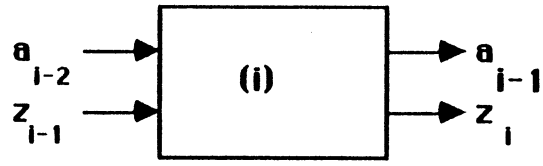


figure 3

Pour évaluer les valeurs $\pi_i = \pi_{j\neq i}(z_i - z_j)$, i=1, ..., n , dans les cellules (i) i=1, ..., n simultanément nous choisissons une circulation des z_i dans le reseau telle qu'au début de calcul z_{i-1} soit dans le canal can.O2 à l'entrée de la cellule (i) et z_i à sa sortie dans le canal can.E2 (figure 3).

Dans ce qui suit nous présentons les différentes étapes de calcul effectuée par le reseau de la figure4 pour une itération complète de Durand-Kerner:

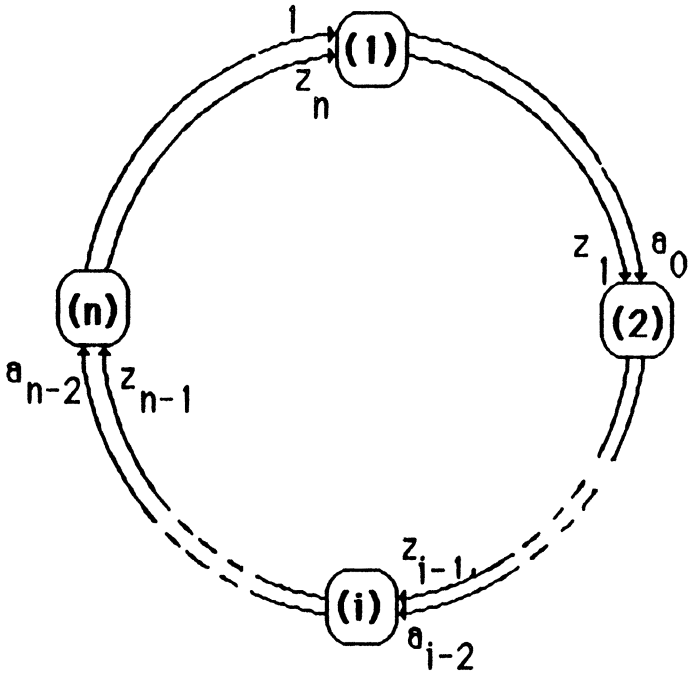


Figure 4

La cellule de base (i) de ce reseau est constituée des registres suivants:
R1, R2, R3: contiènnent respectivement la valeur N du nombre maximum d'itérations, la valeur e, précision de calcul et le coefficient du polynôme, a_{i-1}. Toutes ces valeurs restent stockées dans les registres jusqu'à la fin de calcul.

R4: Pour stocker la valeur de z_i au début de chaque itération.

R5, R6, R7, R8: contiènnent au début de chaque itération z_i, z_i,z_i et 1 respectivement. Au cours de chaque itération ces registres reçoivent respectivement les valeurs de π_i, B, A_1 et A_2; Le registre R_8 reçoit ensuite la valeur de $P(z_i)$ puis celle du nouveau itéré.

R9: contient la valeur 1, à la fin de calcul de nouveau itéré Z_i, si ($|Z_i-z_i|\leq\varepsilon$),sinon il reçoit la valeur 0.

étape1:

t=0:

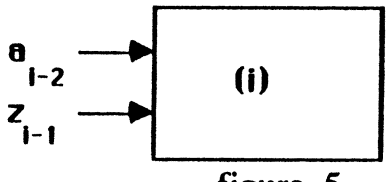


figure 5

A l'instant t=0, a_{i-2} et z_{i-1} entrent dans la cellule qui commencent à calculer $A_1(z_i)*z_i + a_{i-2}$ ($A_1(z_i)$ est initialisé à a_{i-1}), $B = B*z_i$ (B est initialisé à z_i) et $\pi_i = \pi_i*(z_i - z_{i-1})$ (π_i est initialisé à 1).

t=1:

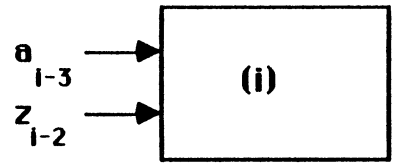


figure 6

a_{i-3} et z_{i-2} entrent dans la cellule. Celle-ci effectue le calcul de $A_1(z_i)=A_1(z_i)*z_i+a_{i-3}$, $B = B*z_i$ et $\pi_i=\pi_i*(z_i - z_{i-2})$.

.
.
.

t=i-2

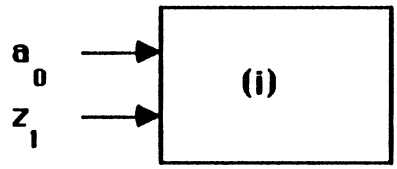


figure 7

A t=i-2 a_0 et z_1 entrent dans la cellule qui commencent à calculer

$A_1(z_i)=A_1(z_i)*z_i+ a_0$, $B=B*z_i$ et $\pi_i=\pi_i*(z_i - z_1)$.

étape2:

t=i-1

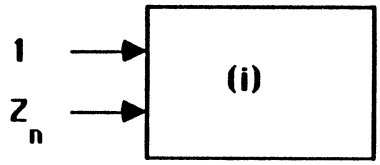


figure 8

1 et z_n entrent dans la cellule. Celle-ci effectue le calcul de $\pi_i=\pi_i*(z_i-z_n)$. Par contre $A_2(z_i)$ reste initialisé à z_i.

t=i:

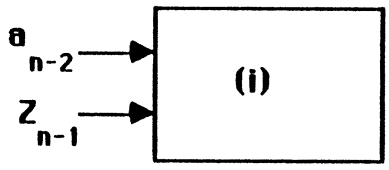


figure 9

La cellule calcule $A_2(z_i) =A_2(z_i)*z_i + a_{n-2}$ et $\pi_i=\pi_i*(z_i - z_{n-1})$.

.
.
.

t=n-2:

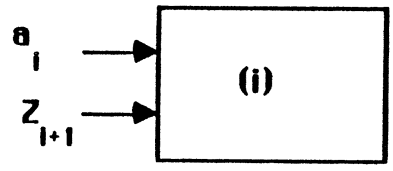


figure 10

A t=n-2, la cellule effectue $A_2(z_i)=A_2(z_i)*z_i + a_i$ et $\pi_i=\pi_i*(z_i - z_{i+1})$.

étape3:

t=n-1:

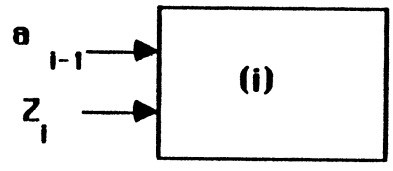


figure 11

a_{i-1} et z_i entrent dans la cellule. celle-ci calcule $P(z_i)=A_1(z_i)+B*A_2(z_i)$.

A t=n chaque cellule (i), i=1,...,n a fini le calcul de $P(z_i)$ et de $\pi_i=\pi_{j\neq i}(z_i-z_j)$, i=1, ...,n:

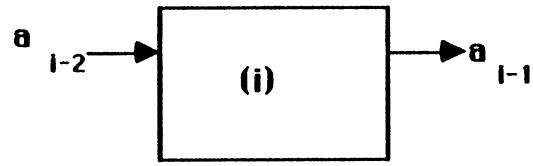


figure 12

ses canaux de communication can.O1 et can.E1 contiennent respectivement les valeurs a_{i-2} et a_{i-1} alors que les canaux can.O2 et can.E2 ne contiènnent pas de valeurs (chaque cellule ne sort pas de valeur sur le canal can.E2).

A la suite de ces étapes chaque cellule (i) calcule le nouveau itéré $Z_i=z_i-P(z_i)/\pi_i$ puis compare la $i^{ième}$ composante de la norme $|Z_i-z_i|$ à ε et sort Z_i sur le canal can.E2 et le boolean $VF=(|Z_i-z_i|\leq\varepsilon)$ sur le canal can.E3 (figure13).

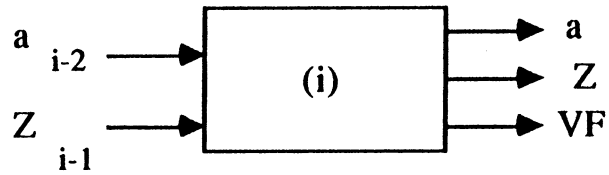


figure 13

Entre t=n+2 et t=(n+2)+(n-1)=2n+1, à chaque top d'horloge dans chaque cellule (i), entre la valeur du canal can.O3, qui est ensuite multiplié par la valeur du registre R9 (1*0=0 et 1*1=1). Le résultat de cette multiplication est stocké dans R9.

Si nous ajoutons un canal de communication entre la cellule (i) et la cellule (i-1) (can.E4) allant de la première vers celle-ci, et un registre R10 (la valeur de R9 est multipliée avec celle de can.E3 et ensuite avec celle de R10. Le résultat de cette multiplication est placé dans R9) pour stocker la valeur booleenne du registre R9 de la cellule (i+1), qui est dans le canal de communication can.O4 allant de celle-ci vers la cellule (i) (figure 15) alors la complexité en temps diminue de $\lceil n/2 \rceil -1$ et elle sera égale à $t=\lfloor(3n+4)/2\rfloor$.

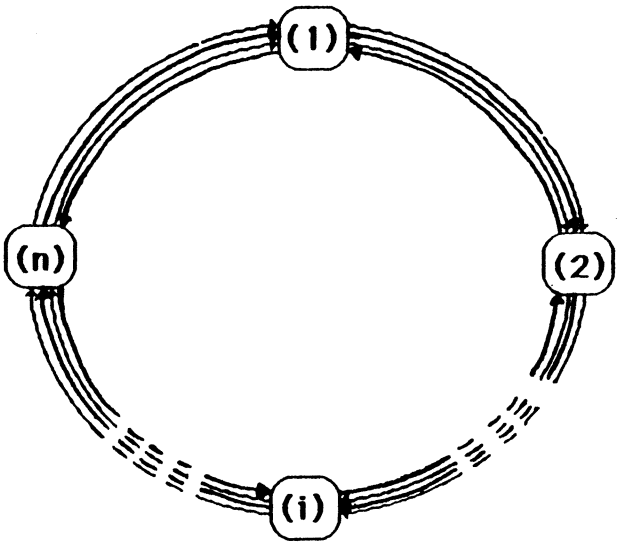


figure 14 reseau torique de processeurs adapté au calcul des itérations de Durand-Kerner.

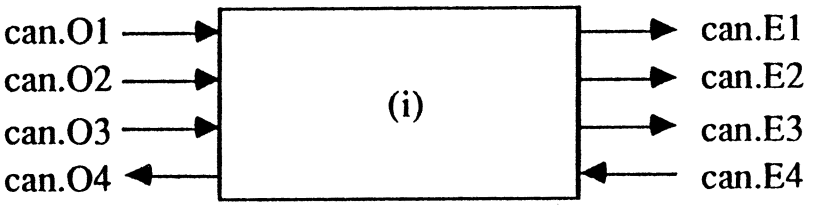


figure 15 cellule (i) du reseau.

Remarquons qu'à $t=\lfloor(3n+4)/2\rfloor$, les valeurs des registres R9 de toutes les cellules sont toutes égales à zéro ou toutes égales à un .

Chaque cellule de réseau (figure 15) reprend le calcul d'un nouveau itéré suivant les mêmes étapes ci-dessus si la valeur de R9 est égale à zéro et si le nombre d'itérations est inférieur à N.

BIBLIOGRAPHIE

[ABE73] O. Aberth, *Iteration methods for finding all zeros of a polynomial simultaneously*, Math. Comp., vol. 27, N° 122, 1973.

[ALE74] G. Alefeld and J. Herzberger, *On the convergence speed of some algorithms for the simultaneous approximation of polynomial roots*, SIAM J. Numer. Anal. Vol.11,N°2,April 1974.

[BOR63] W. Börsch-Supan, *A posteriori error bounds for the zeros of polynomials*, Num. Math. 5, 380-398, 1963.

[BRA73] D. Braess and K.P. Hadeler, *Simultaneous inclusion of the zeros of a polynomial*, Num. Math. 21, 161-165, 1973.

[BRE73] R.P. Brent, *Some efficient algorithms for solving systems of nonlinear equations*, SIAM. J. Numer. Anal. Vol. 10, N° 2, April 1973.

[CIA82] P.G. Ciarlet, *Introduction à l'analyse numérique matricielle et à l'optimisation*, Masson 1982.

[CIA82] P.G. Ciarlet et J.M. Thomas, *Exercices d'analyse numérique matricielle et d'optimisation*, Masson, 1982.

[COS83] M. Cosnard et al., *Parallel algorithms & Architectures*, editors M.Cosnard et al., Elsevir Science Publishers B.V. (North-Holland), 1986.

[COS86] M. Cosnard, *Algorithmique numérique parallèle, cours DEA de Mathématiques Appliquées*, ENSIMAG troisième année, 1986.

[COS86] M. Cosnard et C. Masse, *Convergence presque partout de la méthode de Newton* , C.R Acad. Sc. Paris, 297 (nov 1983), p. 549-552.

[DOC64] K. Docev and P. Byrnev, *Certain modifications of Newton's method for the approximate solution of algebraic equations*, Zh. Vych. Mat. 4, N°. 5, 915-920, 1964.

[DGK84] J.J. Dongarra, F.G. Gustavson, A. Karp, *Implémentation linear algebra algorithms for dense matrix on a vector pipeline machine* , SIAM Reviev 26, 1, (1984), pp. 91-112.

[DUR60] E. Durand, *Solutions numériques des équations algébriques*, Tome1, Masson 1960.

[EHR67] L. W. Ehrlich, *A modified Newton method for polynomials*, Comm. ACM 10 (1967), 107-108.

[FAR75] M.R. Farmer and G. Loizou, *A class of iteration functions for improving, simultaneously, approximations to the zeros of a polynomial*, BIT 15, 250-258,(1975).

[FOX68] L. Fox, *Polynomial factorisation and the Q.D. algorithm*, Lin. Alg. Appl. 1,445, 63 1968.

[GAS66] N. Gastinel, *Analyse numérique linéaire*, Ed. Hermann, 1966.

[GRE76] M.W. Green, A.J. Korsak et M.C. Pease, *Simultaneous iteration towards all roots of a complex polynomial*, SIAM Rev, v. 18, 1976, pp. 501-502.

[GUG86] H. Guggenheimer, *Initial approximation in Durand-Kerner's root finding method*, BIT 26(1986), 537-539.

[HAC71] G. Hacques, *Algorithmique numérique*, Mathématiques pour l'informatique, collection U, A. Colin, 1971.

[HAN77] E. Hansen & Merrell Patrick, *A family of root finding methods*, Num.Math.27, 257-269(1977).

[HB84] K. Hwang, F. Briggs, *Parallel processing and computer architecture*, Mc Graw Hill 1984.

[HEN65] P. Henrici, *Finding zero of a polynomial by the Q.D. algorithm*, Communication ACM8, p.572-573, 1965.

[HEN70] P. Henrici, *Upper bounds for the abscissa of stability of a stable polynomial*, SIAM J. Numer. Anal, vol. 7, N° 4, Decembre 70.

[HOU70] A.S. Householder, T*he numerical treatment of a single non linear equation*, Mc Graw Hill Book Cie, New York, Londres, 1970.

[HOU53] A.S. Householder, *Principles of numerical analysis*, MC Graw Hill,1953.

[JEN70] M.A. Jenkins and J.F. Traub, *A three-stage algorithm for real polynomials using quadratic iteration*, SIAM J. Numer. Anal. Vol. 7, N° 4, december 1970.

[KER66] O. Kerner, Algorithm 283, *Simultaneous displacement of polynomial roots if real and simple*, Comm. ACM 3, 273, 1966.

[KER66] O. Kerner, *Ein Gesamtschrittverfahren zur berechnung von nullstellen von Polynomen*, Numer. Math; 8, 290-294, 1966.

[KJE84] G. Kjellberg, *Two observations on Durand-Kerner's roots finding method.* BIT 24 (1984), 556-559.

[MAS84] C. Masse, *L'itération de Newton: convergence et chaos*, thèse de troisième cycle, Université de Grenoble 1, oct. 1984.

[NOU75] A. Nourein, *An iteration formula for simultaneous determination of the zeros of a polynomial*, J. Comput. Appl. Math. 4 (1975) 251-254.

[ORT70] J.M. Ortega and W.C. Rheinboldt, *Iterative solution of non linear equations in several variables*, Academic press, New York, 1970.

[OST60] A.M. Ostrowski , *Solution of equation and systems of equation*, Academic Press, New York, 1960.

[OST70] A.M. Ostrowski, *A theorem on clusters of roots of polynomial equations*, SIAM J. Numer. Anal. Vol. 7, N°4, December 1970.

[PAS85] L. Pasquini and D. Trigiante, *A globaly convergence methods for simultaneously finding polynomial root*, Math. Comp. vol.44, N°169, pp.135-149, 1985.

[PET82] M.S. Petkovic, *on an iterative methods for simultaneous inclusion of polynomial complex zeros*, J; Comp. Appl. Math., vol. 8, N° 1, 1982.

[PET83] M.S. Petkovic and G.V. Milovanovic, *A note on some improvements of the simultaneous methods for determination of polynomial zeros*, J.Comp. Appl. Math. 9,p.65-69, 1983.

[ROB] F. Robert, *Discrète iterations*, Spinger Verlag.

[ROB84] F. Robert et Y. Robert, *Méthodes numériques pour la résolution de systèmes d'équations*, Polycopié juin 84, ENSIMAG.

[SCH84] U. Schendel, *Introduction to Numerical Methods for Parallel Computers*, Ellis Horwood Series, J.Wiley & Sons, New York, (1984).

[SHE67] G.S. Shedler, *Parallel numerical methods for the solution of equations*, Communication of the ACM vol.10, number 5, mai 1967.

[TRA64] J.F. Traub, *Itérative methods for the solution of equations*, Prentice Hall, Inc. Englewood Cliffs, N.J. 1964.

[WER77] W. Werner, *On the simultaneous determination of polynomial roots*,Int. J. Comp. Math., Ser. B, 1977.

[WIL59] J.H. Wilkinson, *The evaluation of the zeros of ill-conditioned polynomials*, Num. Math. 1, 150-180, 1959.

[WIL65] J.H. Wilkinson, *The algebraic eigenvalue problem*, Claerendson Press. Oxford. 1965.

Annexe 1

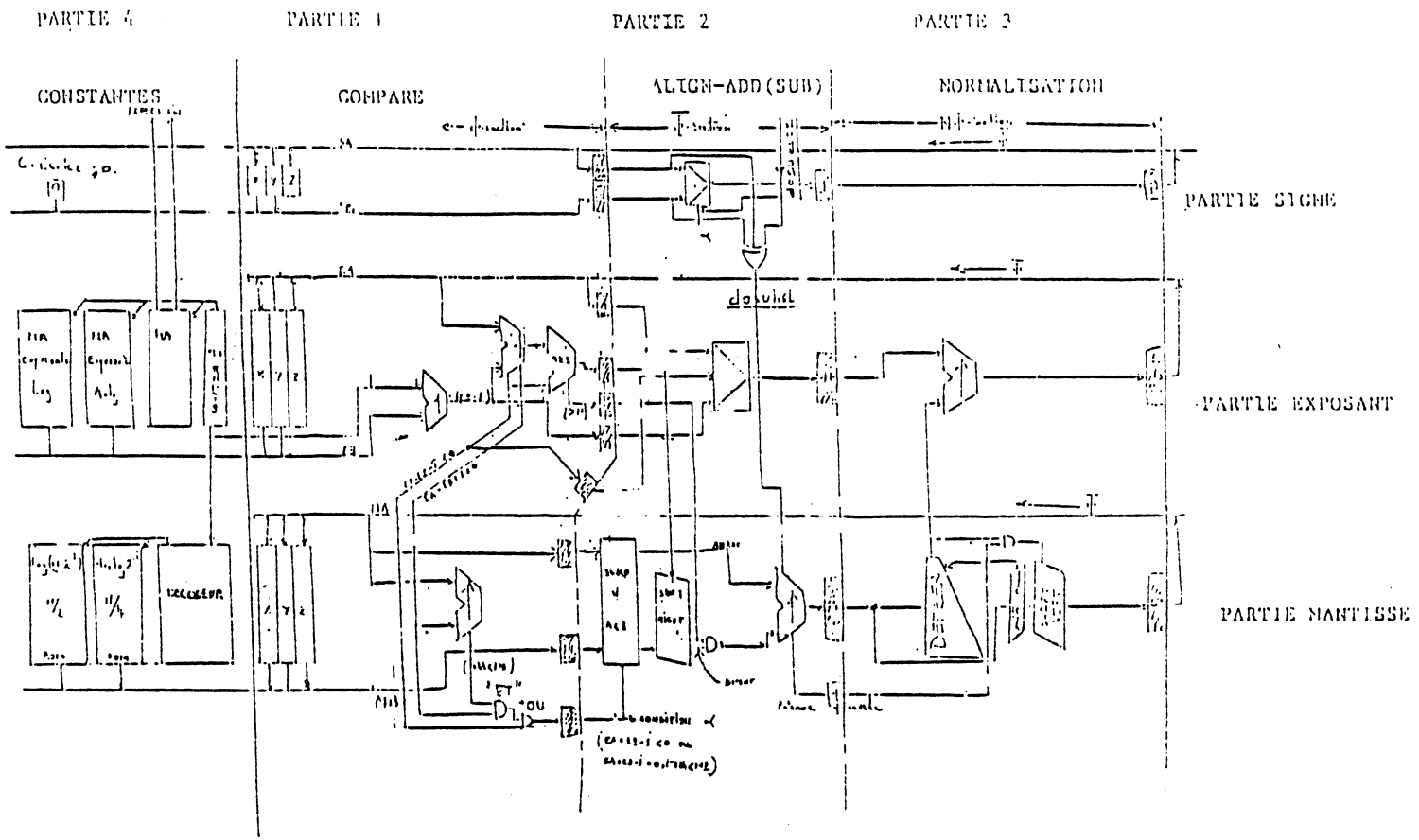

PARTIE 4
PARTIE 1
PARTIE 2
PARTIE 3
CONSTANTES
COMPARE
ALIGN-ADD(SUB)
NORMALISATION
PARTIE SIGNE
PARTIE EXPOSANT
PARTIE MANTISSE

Tableau de l'Igorithme cos/sin en cycles:

$t \in [-\pi/2, \pi/2]$ Algorithme(COS/SIN)

$t \longrightarrow X$
$X \longrightarrow Z$
$K \longrightarrow X$ $0 \longrightarrow Y$

$$\begin{cases} X_{n+1} = X_n - d_{Z_n} Y_n \cdot 2^{-n} \\ Y_{n+1} = Y_n + d_{Z_n} X_n \cdot 2^{-n} \\ Z_{n+1} = Z_n - d_{Z_n} \mu_n \cdot 2^{-n} \end{cases}$$

$$\begin{cases} d_{Z_n} = \text{signe}(Z_n). \\ \mu_n = 2^{-n} * \text{Arctan}(2^{-n}). \\ K = 0.607252935...; \end{cases}$$

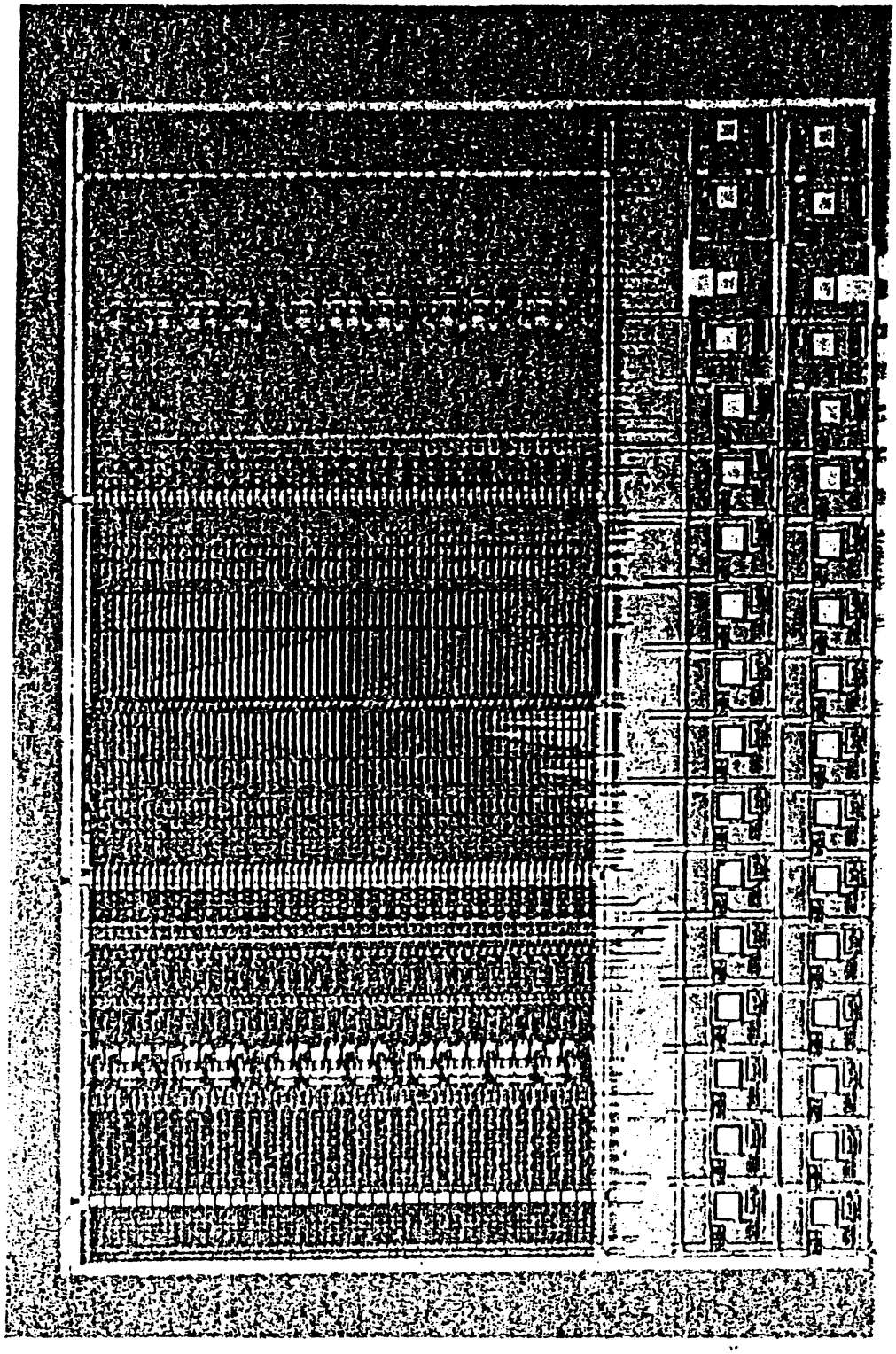
$X \longleftarrow \cos(X).$

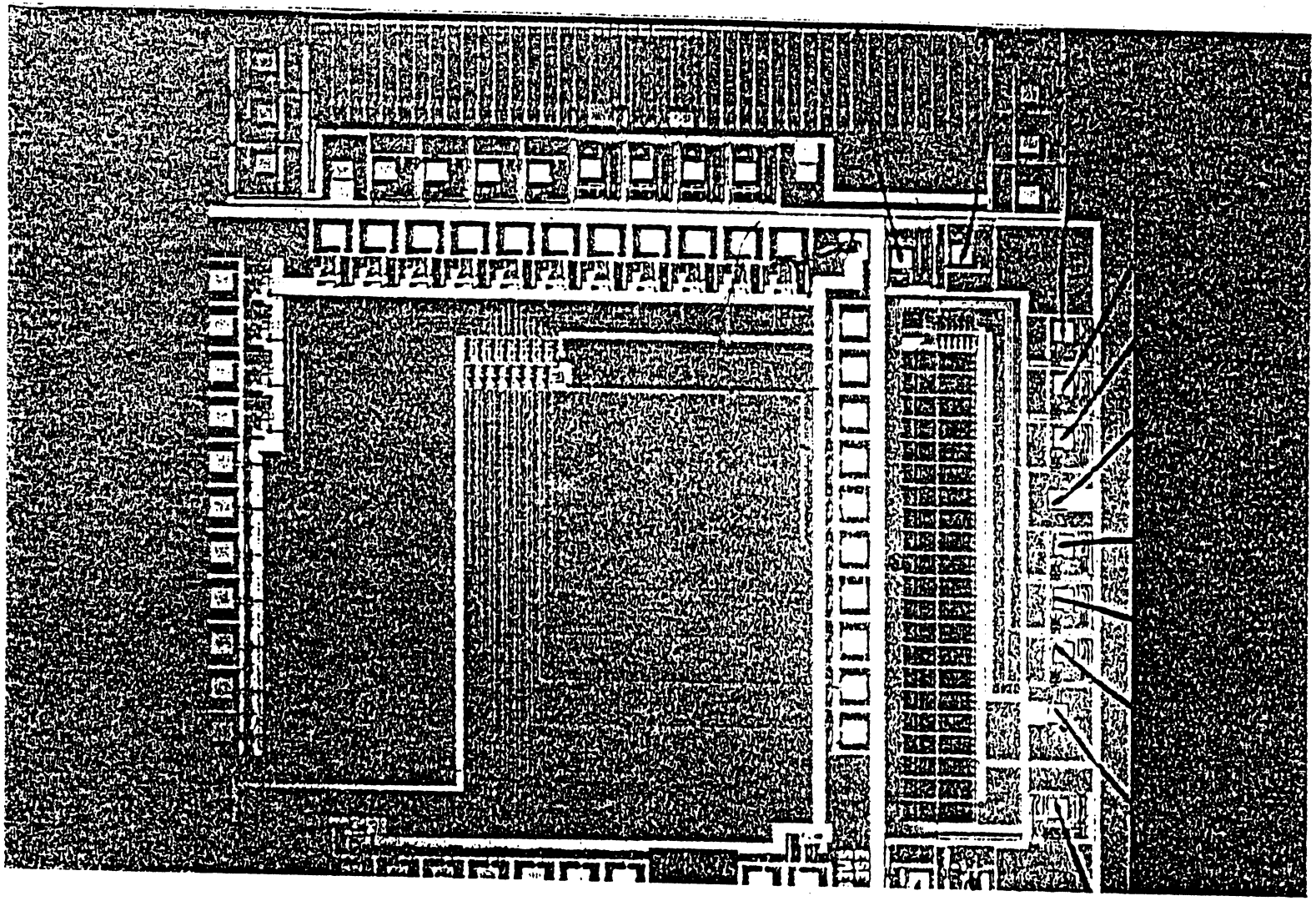
$Y \longleftarrow \sin(X).$

cos/sin	PHI	PHIbarre
cycle1	$.Z_{n+1}$ (PHI1) $.Y_n$ (PHI3)	$.Z_{n+1}$ (PHI2) $.Y \longleftarrow Y_n$ (PHI4)
cycle 2	$.Z_{n+1}$ (PHI3) $.X_{n+1}$ (PHI1)	$.X_{n+1}$ (PHI2) $.Z \longleftarrow Z_{n+1}$ (PHI4)
cycle3	$.X_{n+1}$ (PHI3) $.Y_{n+1}$ (PHI1)	$.Y_{n+1}$ (PHI2) $.X \longleftarrow X_{n+1}$ (PHI4)
cycle4	$.Y_{n+1}$ (PHI3) $.Z_{n+2}$ (PHI1)	$.Y \longleftarrow Y_{n+1}$ (PHI4) $.Z_{n+2}$ (PHI2)

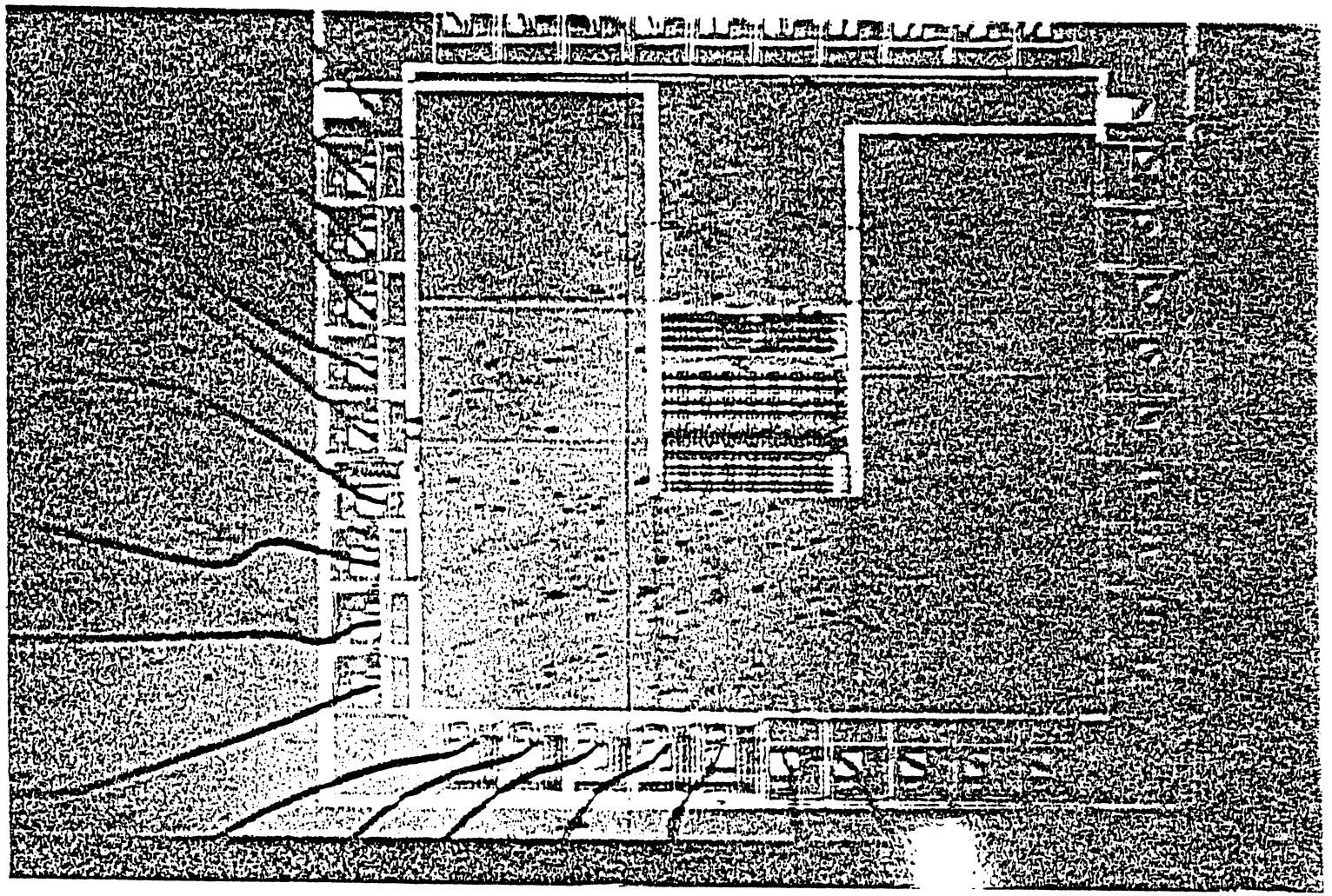
Annexe 2

PARTIE OPERATIVE DE FELIN





PARTIE CONTROLE DE LA MACHINE A CALCUL DE FELIN



MACHINE DE COMMUNICATION DE FELIN

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974

VU le rapport de présentation de Monsieur Michel COSNARD

Monsieur OUAOUICHA Hassan

est autorisé à présenter une thèse en soutenance en vue de l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Mathématiques appliquées".

Fait à Grenoble, le 22 mai 1987

Georges LESPINARD
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

Résumé:

Dans la première partie de ce travail, nous élaborons une méthodologie de simulation du fonctionnement logique du coprocesseur arithmétique FELIN (Fonctions Elémentaires Intégrées). Cette méthodologie consiste d'abord à construire un simulateur de l'architecture du circuit dans lequel nous avons séquentialisé les parties parallèles par l'adjonction de registres logiques et par la duplication des variables, ensuite à l'exécution des algorithmes sur ce simulateur d'architecture avec prise en compte de leur découpage en phases et de la séquentialisation précédente, enfin à la vérification des résultats des exécutions.

Dans la deuxième partie nous étudions tout d'abord la méthode de Durand-Kerner ainsi que celle d'Ehrlich pour la recherche simultanée de toutes les racines d'un polynôme à coefficients complexes, puis nous les comparons à cinq variantes algorithmiques.

L'étude comparative de ces différentes méthodes montre que les plus performantes sont celles de Durand-Kerner en mode parallèle (approche de Jacobi) et en mode série (approche de Gauss-Seidel). L'étude expérimentale de ces différentes méthodes est menée sur une architecture vectorielle (FPS264 : Floating Point Systems).

Une architecture parallèle adaptée au calcul des itérations de la méthode de Durand-Kerner est étudiée.

Mots-clés:

Simulation, Processeur Arithmétique, Fonctions Elémentaires, Racines de Polynôme, Méthode de Durand-Kerner, Algorithmique Parallèle.