LABORATOIRE D'INFORMATIQUE THéORIQUE
& APPLICATIONS DE MARSEILLE
L.I.T.A.M
Département de Mathématiques-Informatique
Luminy
UNIVERSITé AIX-MARSEILLE II

ISSN 0291 - 5413

INFORMATIQUE FONDAMENTALE ET APPLICATIONS



## SOMMAIRE



DéCEMBRE 1986

Adresse postale : FACULTé DES SCIENCES DE LUMINY
Mathématiques-Informatique LITAM BAT TPR 2 9e etage
Case 901 70 route Léon LACHAMP 13 288 MARSEILLE Cedex 9
91 26 90 81

# EDITORIAL

e. bianco

## INFORMATIQUE ET TERRORISME

Le terrorisme est une arme de pouvoir à double tranchant,c'est bien connu.Déja les Romains avaient pour emblème le faisceau dont émergeait une hache à deux fers.Et ce n'est pas tout-à-fait un hasard si le vieux Maréchalnousvoila avait adopté le même,mais que pour faire plus gaulois il appelait la Francisque.

Quand on a la loi pour soi,il est vraiment bien facile de se livrer à quelques petits excès sur la peau des autres,très rares sont ceux qui se sont retrouvés au pouvoir,même parmi les plus "démocrates",et qui n'ont pas un peu joué de ce registre.Mettons cela au compte de la faiblesse humaine et n'en parlons plus.

Ne parlons plus de la faiblesse humaine,mais que cela ne nous empêche pas de parler des conséquences des abus de pouvoir.

Attention,là nous semblons opérer un glissement vers le plan politique ...Pas vraiment car il nous faut observer que pour bien réussir une opération d'envergure,il est nécesseire d'abord,de réussir une bonne opération d'information,car il faut préparer les masses,et il existe d'excellents algorithmes destinés à préparer,à maintenir et à assurer une conviction.

Ces algorithmes sont des algorithmes de transfert d'information,c'est par là,finalement que nous rejoignons nos préoccupations essentielles de l'informatique.

Il ne me paraît pas tout-à-fait inutile ni hasardeux d'étudier un peu ces algorithmes,sans trop pénétrer dans le détail car cela deviendrait un ouvrage d'importance,mais juste un peu pour s'en faire une idée.Quelques petits exemples devraient nous permettre d'y voir un peu plus clair.

Quand on a décidé,période de crise oblige,de baser son économie sur de la vente d'armes il devient clair qu'on prend des risques.Rarement cette industrie marche bien en temps de paix,car les

clients renouvellent et modernisent certes le matériel,mais comme il s'agit en général de pays pauvres,les rentrées d'argent sont plutôt difficiles.Alors c'est le contribuable qui fait l'avance et comme ça finit par se voir,le découvert,alors on le minimise en mettant à côté un trou qui parait plus gros : le trou des halles,le trou de la sécurité sociale.

Peut-être même que c'est comme ça que les crises se forment.

Mais entre-parenthèses,si le veau est un Français-étalon au sens de notre Jeanne d'Arc numéro deux,la baleine devient un assuré-social-étalon au sens d'une majorité déchue...y-at-il vraiment de quoi se marrer ?

Le commerce n'est pas un métier d'amateurs,ni de super diplômés,qui n'ont jamais que des vues partielles du problème,nous l'avons bien remarqué chaque fois que l'occasion s'en est présentée.Les gens sérieux et capables se distinguent sur le terrain.Ainsi pour revenir à nos ventes d'armes rien n'est meilleur qu'un bon conflit,bien mijoté.

Alors vous êtes assuré d'avoir au moins un bon client.

Mais pourquoi pas deux ? Ah là,la manoeuvre devient délicate car il faut vendre à l'un en faisant semblant de ne pas vendre à l'autre.Hélas ! aujourd'hui avec les barbouses et les moyens modernes d'espionnage,tout se sait.Mais comme pas plus les acheteurs que les vendeurs n'ont interêt à étaler le débat sur la place publique...problèmes moraux,un peu...chemin de l'argent,beaucoup...

Imaginez l'Irangate que cela ferait !

Alors les acheteurs essaient de faire comprendre au vendeur qu'il ne faut pas trop déconner,et comme par hasard,quelques pétards pètent ( ils sont fait pour ) .

Mettez vous dans la peau des Grands Chefs - eux,c'est pas eux les vendeurs,eux ils sont responsables de la **sécurité** ( soulignée trois fois ) - mais il faut bien gagner des élections n'est-ce-pas ? et où trouver beaucoup d'argent ? car il en faut beaucoup,je vous le demande.

En 39 c'est le pastis qui nous a fait perdre la guerre,en 88 c'est peut-être lui qui nous la fera gagner,la vraie,celle des bonnes affaires,malgré le SIDA.En bref,les média n'ont plus qu'à broder ; pour peu que de ci de là quelques petits groupes d'idéalistes se livrent à quelques opérations que l'Ordre Public réprouve,l'amalgame est fait.

Et vous qui ne pouvez jamais lire la vérité qu'entre les lignes,allez vous y retrouver !

Bien sur quelques voix s'élèvent parfois pour dénoncer des choses surprenantes,mais le jeu des démentis gradués,des silences pleins de signification,des secrêt-défense,des demi-aveux assortis de demi-menaces vagues mais terrifiantes : chômage,sécurité,santé ...présente un effet dilatoire,et en même temps prépare l'opinion.Ce n'est que bien plus tard qu'on saura officiellement que l'Iraq a sa bombe atomique.La politique fonctionne un peu comme le courrier,elle est à "deux vitesses" , il y a celle qu'on étale beaucoup,comme la culture quand il y en a peu,et puis celle qui fait le lit de l'avenir.

Avec un bon substrat de terreur bien réparti,il devient plus facile de faire passer en douceur quelques idées-force.

Peur du chômage,peur de l'étranger envahisseur,peur du SIDA.Comment alors ne pas avaler des bavures de plus en plus graves,qui,elles,ne sont jamais classées au chapitre du terrorisme par les média,pourquoi ?

Comment alors ne pas avaler les vertus de la Vertu,génératrice d'encore plus d'angoisse,donc de peur.Que des Censeurs très XIX$^{e}$ siècle puissent allègrement exercer leurs talents au sein d'un gouvernement "libéral",ne laisse pas de poser quelque question.

Quand on prend le point de vue du philologue - que l'on me pardonne un peu de langue de bois - on pourrait imaginer que pratiquer le terrorisme c'est soumettre une communauté à la terreur à l'aide de tous moyens adéquats.

Appliquée comme peut l'être cette formule,me semble souligner combien on néglige que l'existence d'une terreur inconsciente et bien entretenne,est utile et nécessaire à la création d'une bonne terreur consciente,au bon moment.Il me paraît curieux que l'on ne se demande pas pourquoi on limite l'apellation de terrorisme au seul domaine de la terreur consciente et fabriquée à la demande.

Ces procédés ne sont pas nouveaux,déja notre bon Maréchalnousvoila évoqué plus haut,classait terroristes des opposants à son régime,qu'on classa plus tard dans la catégorie des Héros.

Hitler en son temps fit brûler le Reichsteg en utilisant habilement un anarchiste,cela bien sur dans une période d'insécurité qui rendait la chose commode et efficace,l'effondrement économique : le timbre-poste à cinq milliards de marks,le pays sillonné par les hordes de SA de l'amiral Roem,remplacées bientôt par les SS,le bon peuple perdu,éperdu de gratitude envers le Sauveur qui lui rend sa dignité Aryenne - que lui restait-il d'autre ?

On pourrait de la sorte citer mille exemples parmi les plus grandioses,et je fais référence aux sept couleurs de Brasillac,et d'autres encore plus modestes telles que l'organisation de la sécurité dans les usines d'automobiles,ce n'est pas Overney qui me contredira.

Mais ce qui me fascine dans tout ceci ( le mot fascisme est de même origine ) c'est la simplicité autant que l'efficacité de l'algorithme sous-jacent.Je ne parlerai pas,car c'est encore autre chose de la façon dont l'informatique peut intervenir pour créer un climat.Le citoyen est un peu comme un octet balloté au gré d'un algorithme dont il ignore tout ,bien qu'il puisse éventuellement comme élément de programme participer à son déroulement.

Pas de panique,mes Amis,courage.

# FAUT-IL RENDRE REELLES LES MACHINES VIRTUELLES.

roger dupuy

C.R. Subject classification informatics. : C53 C54 D31 D41

Résumé.

L'Auteur montre comment les solutions logicielles de la gestion des ordinateurs qui se sont développées dans le passé en même temps que les ordinateurs grossissaient,ont été brutalement appliquées au micro-processeur,dont on attendait précisément la libération du carcan ainsi obtenu.L'Auteur propose une solution qu'il montre d'abord plus simple et plus efficace,et enfin plus adaptée au micro,et qu'il a lui-même appliquée dans l'HYPERCUBE.

# FAUT-IL RENDRE RÉELLES LES MACHINES VIRTUELLES ?

roger dupuy

La micro-informatique connait actuellement un développement très important et la voie empruntée par la recherche s'oriente vers la construction de micro-processeurs de plus en plus puissants,de manière à les rendre aussi performants que les gros ordinateurs.Il conviendrait pourtant,de se souvenir que la naissance et le succès des microordinateurs sont dûs à leur plus grande simplicité par rapport aux gros ordinateurs.La complèxité des gros ordinateurs provenant du fait qu'ils sont constitués d'un ensemble de machines virtuelles,alors que la particularité d'un microordinateur est d'être personnel.

Or,on cherche actuellement à implanter des systèmes à machines virtuelles sur microordinateurs,c'est-à-dire à reproduire la configuration existant sur les gros ordinateurs.

Nous voudrions donc poser la question suivante: la recherche,dans ce domaine,s'est-elle engagée sur la bonne voie ou ne risque-t-on pas plutôt de recopier sans efforts d'innovation des techniques dont on peut se demander si elles sont réellement efficaces ?

Pour pouvoir répondre à cette question,il convient tout d'abord,d'expliciter le concept de machine virtuelle ? D'où provient ce concept et quel a été son intérêt dans le développement de l'informatique ?

S'il n'y avait qu'un seul utilisateur,on peut dire que la machine passerait son temps à l'attendre ; en effet,le temps d'utilisation de l'unité centrale,pour une machine en monoprogrammation est évalué à 10% à peine.

On a donc cherché à utiliser au mieux les ressources de l'ordinateur et le système de machines virtuelles permettant de faire fonctionner un même ordinateur en temps partagé pour plusieurs utilisateurs a paru le mieux adapté.

Ce système était jusqu'à présent irréalisable sur microordinateur,car la vitesse de l'unité centrale était trop faible pour pouvoir en permettre l'implantation.Les progrès technologiques ayant permis d'augmenter considérablement la densité d'intégration des transistors,il est devenu désormais possible de construire des microprocesseurs très rapides aux performances comparables à celles d'un gros ordinateur.

A la question: comment se servir de microordinateurs si puissants ? la réponse a

été,comme l'ont déja montré les microprocesseurs 32 bits Motorola,National Semi-conductors,etc ,par l'implantation d'un système de machines virtuelles.

Nous voudrions,pour notre part,envisager le problème sous un autre aspect.N'y aurait-il pas un autre moyen d'aboutir au même résultat,à savoir utiliser au mieux les ressources d'un ordinateur,qui permettrait par la même occasion,de briser le cercle de la surpuissance des microordinateurs entraînant nécessairement la création de systèmes de machines virtuelles ?

Nous pensons que ce moyen existe et nous l'avons démontré par la construction de l'ordinateur Hypercube en 1980.Il suffirait,en effet,d'utiliser plusieurs microprocesseurs de puissance moyenne connectés en parallèle,plutôt que de créer un système de machines virtuelles à partir d'un seul microprocesseur très puissant.

Ainsi,lorsque n terminaux se trouvent connectés dans un système de machines virtuelles sur un seul microprocesseur très puissant,on peut dire que chaque machine virtuelle associée à un terminal a un vitesse $v = V / n$ .Mais,de même,si un microprocesseur n fois moins puissant est connecté sur chaque terminal,chaque machine réelle associée à un terminal aura aussi une vitesse $v = V / n$ .

On obtient donc le même résultat,et de plus,on transforme le système de machines virtuelles en système de machines réelles.Notre propos sera donc se montrer qu'une telle architecture utilisant des machines réelles est considérablement plus simple qu'un système de machines virtuelles,tant du point de vue matériel que du point de vue logiciel,et qu'en outre,elle ne réduit pas les possibilités d'utilisation de la machine,puisque l'on obtient globalement la même puissance de calcul que dans le cas d'un système de machines virtuelles.

Nous montrerons d'autre part,que la généralité de notre méthode permet de l'appliquer à des microprocesseurs plus puissants,c'est-à-dire de construire aussi des systèmes de machines virtuelles à n dimensions ( chaque machine réelle connectée en parallèle pouvant à son tour être décomposée en machines virtuelles ) .

Mais revenons d'abord aux systèmes de machines virtuelles tels qu'ils existent actuellement et tentons d'expliquer pourquoi leur mise en place a connu un tel succès.Autrefois,les consoles n'existant pas,les machines travaillaient en batch à partir d'un fichier de cartes perforées et les résultats étaient imprimés sur listing.Ce système comportait toutefois certains inconvénients.

La mise à jour des fichiers n'étant pas assurée en temps réel,l'informatisation ne se suffisait pas à elle-même et nécessitait toute une organisation manuelle (classer les nouveaux listings,détruire les listings périmés ),ce qui n'était pas toujours simple

étant donné la masse de papier parfois considérable qui devait être manipulée.

Le rapport à la machine était très rigide et soumis à des règles bien précises dont l'utilisateur ne percevait pas toujours la nécessité,mais qui pouvaient entraîner des résultats erronés,parfois inéversibles lorsqu'elles n'étaient pas respectées.A cette époque,l'informatique restait donc peu employée en raison des catastrophes que pouvait provoquer une mauvaise utilisation d'un système informatique.

Avec l'apparition de l'interactif,l'utilisateur est devenu l'interlocuteur direct de la machine.Les données sont entrées en temps réel et les risques d'erreur sont considérablement diminués.L'aspect conversationnel donne désormais une part active à l'utilisateur et rend son travail moins fastidieux.Il est donc certain que le développement des systèmes interactifs a entraîné l'essort de l'informatique en créant une nouvelle manière de travailler.Mais si le confort de l'usager s'est trouvé de cette façon amélioré,les programmes informatiques ont dû ,pour leur part,tenir compte des exigences de l'utilisateur et sont devenus non seulement plus lourds ( en raison des écritures nécessitées pour l'affichage d'écran,de messages d'erreur,etc...) mais aussi d'une complèxité un peu plus grande ( puisqu'il ne s'agit plus de traiter des fichiers en séquenciel,comme pour le système batch,mais d'avoir un accès direct à l'information par l'intermédiaire des bases de données ).

Ceci étant,et bien que les avantages du système interactif soient incontestables du point de vue de l'utilisateur,on ne peut pas dire qu'il s'agisse d'un réel progrès du point de vue informatique,car en utilisant un bon système batch,accompagné d'une bonne organisation,il est très probable que l'on obtienne des résultats comparables,voire même meilleurs,que ceux obtenus par un système interactif.

En effet,dans un tel système la multiplicité des consoles connectées à une même machine,selon le principe des machines virtuelles,nécessite un espace mémoire gigantesque qui doit pouvoir contenir simultanément les programmes des différents utilisateurs.Par ailleurs,l'implantation des bases de données devant être effectuée sur des espaces disques *en-ligne* ,cela entraîne obligatoirement la multiplication des unités de disques.

La machine doit donc devenir très puissante pour pouvoir satisfaire les exigences de l'interactivité,et l'on peut,dans ces conditions,se poser la question de la rentabilité d'un tel investissement.

L'intérêt du système interactif consiste donc,essentiellement,à pouvoir entrer des données,les modifier,les supprimer en temps réel et à consulter des fichiers.Mais en général,même lorsqu'on utilise l'interactif,certains traitements continuent de

s'effectuer en batch et ces deux systèmes étant d'un usage incompatible ( ou bien le batch monopolise les accès aux bases de données et cela entraîne des temps d'attente très importants pour les utilisateurs,ou bien on décide de rendre prioritaires les terminaux ,auquel cas le batch fonctionne de manière très ralentie ),bon nombre d'installations ont choisi de faire travailler leur machine en interactif le jour et en batch la nuit.

En définitive,il reste à se poser la question suivante: quelle est donc la différence essentielle entre le batch et l'interactif ?

A cette question nous répondrons qu'à notre avis il n'y en a pas; l'interactif peut être considéré comme un système batch pour lequel l'organe d'entrée se trouve être la console.

En conclusion,nous dirons donc,qu'en ce qui concerne une machine purement batch,les batchs s'exécutent les uns après les autres et c'est le même moniteur qui assure l'enchaînement des travaux,ce que l'on appelle aussi un traitement par lots.Tandis qu'en ce qui concerne une machine à système interactif,il y a autant de moniteurs qu'il existe de consoles connectées à la machine,et chaque moniteur se déroule en temps partagé sur chacune des consoles.Autrement dit,un système interactif est un système multibatch.

Puisque nous considérons qu'un moniteur interactif est un moniteur multibatch, nous commencerons donc par rappeler ce qu'est un moniteur en général.Cela nous permettra de préciser ( forcément d'une manière succincte ) le fonctionnement d'une machine pour justifier les architectures que nous proposons.

L'utilisateur d'un ordinateur veut pouvoir manipuler des programmes et des données.Voyons donc comment cela est rendu possible.

## LE SYSTEME DE GESTION DE FICHIER (SGF) DANS UN SYSTEME DE MONOPROGRAMMATION.

Un disque est découpé en secteurs numérotés de 1 à n .Les premiers secteurs servent à enregistrer le catalogue qui est une sorte de table dont chaque entrée est un descriptif de fichier.La structure de ces entrées comporte entre autres:

- Le nom du fichier,
- Le numéro de secteur du début du fichier,
- Le nombre de secteurs alloué au fichier.
- Etc...

Il y a deux sortes de fichiers: les fichiers programmes,directement exécutables, et les fichiers données.Du point de vue du système de gestion de fichier,il n'y a aucune différence entre ces deux espèces de fichiers: en effet,pour accéder à un enregistrement il suffit d'indiquer le nom du fichier,et le numéro de secteur que l'on veut atteindre.Le SGF se contente de vérifier que le fichier existe bien dans le catalogue,et que la référence secteur demandée se situe dans l'espace alloué au fichier.

Par contre,au niveau utilisateur,la différence entre fichier programme et fichier données est fondamentale.

On accède à un fichier de programme par son nom: c'est-à-dire qu'il y a identité entre le nom d'un programme et le nom du fichier dans lequel est stocké ce programme.

On accède à un fichier de données à l'aide d'un mécanisme d'assignation. Préalablement à toute utilisation d'un fichier de données,on doit l'assigner,c'est-à-dire établir une correspondance entre un nom logique et le nom physique qui se trouve dans le catalogue.

Le nom logique est le nom par lequel sera référencé le fichier dans le programme.Ces mécanismes sont maintenant généralement admis dans tous les systèmes.

## LE MONITEUR DANS UN SYSTEME DE MONOPROGRAMMATION.

Il considère que chaque ligne d'information qui lui est soumise est le nom d'un

fichier programme,la syntaxe en étant la plupart du temps: nom de fichier programme,liste de paramètres.

Son rôle consiste à lire le fichier programme secteur par secteur en s'adressant au système de fichier,et à implanter ces secteurs en mémoire à partir de l'adresse de chargement inscrite en début de fichier.Une fois le programme chargé en mémoire et les paramètres transmis,le moniteur lui donne le contrôle.

Si,au cours de son exécution,le programme doit effectuer une entrée-sortie, il s'adresse à nouveau au moniteur en lui fournissant le nom logique du fichier.Le moniteur examine sa table d'assignation,et s'il ne trouve pas de correspondance entre le nom logique fourni et le nom physique,il abandonnera le travail en cours en allant lire une nouvelle commande sur le fichier d'entrée.Ainsi,par exemple la mise en oeuvre d'un programme de gestion de facture (GFAC) dont le source se trouve dans le fichier SGFAC se traduira par la liste des primitives ci-après,enchaînées par le moniteur:

créer OBJFAC
créer GFAC
assigner F1,SGFAC
assigner F2,OBJFAC
COBOL
assigner F1,OBFAC
assigner F2,GFAC
CATALOG
créer MFAC
assigner F1,AFAC
assigner F2,MFAC
assigner F3,NFAC
GFAC
Etc...

Le moniteur,lorsqu'il va rencontrer la primitive "créer OBJFAC" implante en mémoire le programme "créer" ,lequel va inscrire dans le catalogue le nom de fichier "OBJFAC",et réserver de la place pour ce fichier ( objet de SGFAC ).Puis il va créer "GFAC" de la même manière.Naturellement on pourra fournir au programme "créer" la taille du fichier,et d'autres paramètres si nécessaire.

Ensuite le moniteur rencontre la primitive "assigner".Il implante ce programme

en mémoire qui aura pour effet de construire une table de correspondance entre F1 et SGFAC,de même pour F2.La primitive suivante est COBOL: ce programme est chargé en mémoire comme les autres.Il a été programmé avec des lectures du source sur F1 et des écritures de l'objet sur F2.Chaque fois qu'il adressera des requêtes d'entrée-sortie,il y aura conversion unité logique,unité physique grâce à la table d'assignation.

Ensuite le programme de catalogueage,c'est-à-dire de transformation du fichier objet en fichier exécutable est à son tour appelé avec au préalable ses mécanismes d'assignation.

Enfin GFAC ,le programme exécutable de gestion des factures sera chargé en mémoire.Il met à jour le fichier des anciennes factures (AFAC) ,à l'aide du fichier MFAC (modification des factures ) pour créer le fichier NFAC des nouvelles factures.

Au travers de cet exemple on met en évidence le mécanisme d'enchaînement du moniteur,et le mécanisme d'assignation qui permet de rendre indépendant le support d'information par rapport au programme.Finalement le fonctionnement du moniteur peut se résumer par le schéma S1.

En assignant le fichier d'entrée soit au terminal,soit à un fichier par un ordre "assigner",l'ordinateur fonctionnera soit en interactif,soit en batch,et l'on pourra passer d'un mode de fonctionnement à l'autre en changeant uniquement l'assignation.

Il n'y a donc à notre sens aucune différence entre un moniteur interactif et un moniteur batch: il suffit d'assigner l'unité d'entrée soit au terminal soit à un fichier. On peut dans ces conditions écrire un système de conduite de l'ordinateur,résidant, tout-à-fait convenable en 4 Ko de mémoire,les fonctions de ce système sont alors:

* La fonction moniteur (MON),
* La fonction gestion de fichier (SGF).

Cette dernière fonction gère le disque,le terminal,l'imprimante,.La gestion du disque comprend la gestion du catalogue avec les primitives "créer","détruire" et la lecture et l'écriture d'un secteur.

Naturellement il faudra enrichir ce système de programmes non résidents,tels éditeur de texte,assembleur,éditeur de liens,compilateurs... qui,d'un point de vue strictement système sont des programmes utilisateurs.

Remarquons que le système gère l'information en ce qui concerne:

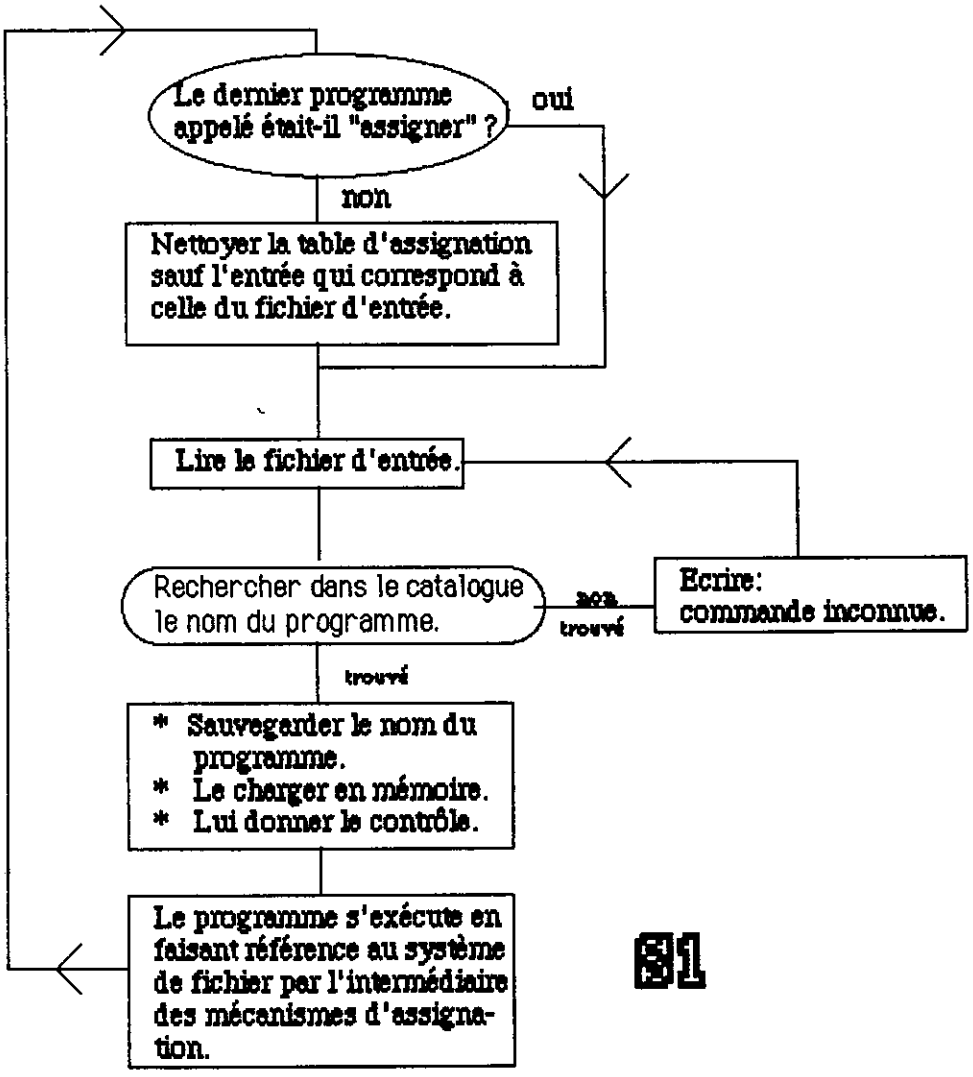

Le dernier programme appelé était-il "assigner" ?
oui
non
Nettoyer la table d'assignation sauf l'entrée qui correspond à celle du fichier d'entrée.
Lire le fichier d'entrée.
Rechercher dans le catalogue le nom du programme.
non trouvé
Ecrire: commande inconnue.
trouvé
* Sauvegarder le nom du programme.
* Le charger en mémoire.
* Lui donner le contrôle.
Le programme s'exécute en faisant référence au système de fichier par l'intermédiaire des mécanismes d'assignation.

Fig 1

* Un terminal,caractère par caractère,
* Une imprimante,caractère par caractère,
* Un disque secteur par secteur (on dit un bloc d'information).

Ce mode de gestion correspond au mode de fonctionnement des équipements: on appelle cela la gestion physique des entrées-sorties.

Dans la pratique l'information manipulée par le programmeur est de nature différente: pour un terminal ce peut être une ligne d'écran ou une page,pour un disque un enregistrement.Il est donc nécessaire de disposer d'un mécanisme de gestion logique de l'information,qui transforme les enregistrements logiques en blocs physiques si l'on est en écriture,ou l'inverse si l'on est en lecture.Il est d'usage d'inclure cette gestion logique de l'information dans les programmes systèmes non résidents précédemment cités.

Cependant on peut faire la remarque suivante,par exemple en ce qui concerne la disquette.Elle tourne à 10 tours/seconde environ.Le bras de la disquette se déplace lentement,il faut peut-être aussi $1/10^{6}$ de seconde pour positionner le bras sur une piste.Bref il faut compter au moins 200 ms pour accéder à un secteur.Pendant ce temps le microprocesseur doit attendre.Supposons qu'il lui faille 2 micro-seconde pour exécuter une instruction,il attend donc pendant un temps équivalent à l'exécution de 100.000 instructions,chaque fois qu'il référence le disque.Que dire alors lorsqu'il effectue une entrée-sortie sur terminal ou imprimante,organes encore plus lents ?

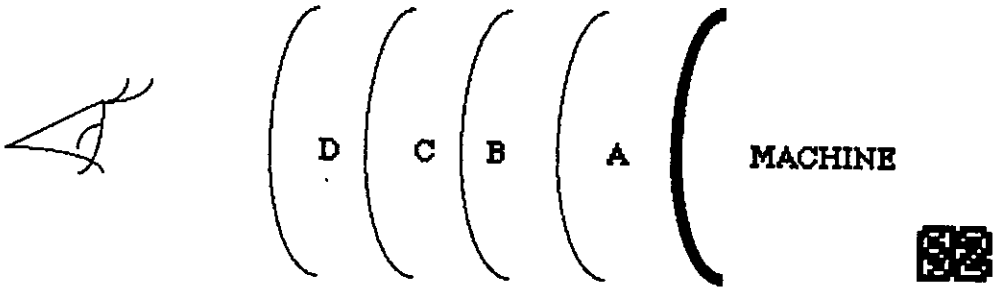
Il ne sert à rien d'augmenter la vitesse du processeur,car il est sans arrêt en train d'attendre après les entrées-sorties.Il vaudrait mieux augmenter la vitesse des périphériques,mais cela est quasiment impossible car on est limité par des problèmes de mécanique.

Avec l'organisation précédemment décrite,il est illusoire d'espérer faire fonctionner le microprocesseur à plus de 10% de sa vitesse intrinsèque.On ne se rend pas compte de ces phénomènes en interactif car le temps de réaction humain est encore beaucoup plus lent.En batch ces problèmes ne sont sensibles que sur des programmes qui effectuent beaucoup d'entrées-sorties.

La seule manière d'améliorer l'efficacité des ordinateurs est donc de pouvoir mettre en place un mécanisme qui permette aux entrées-sorties de se dérouler simultanément aux activités du processeur.Naturellement il sera nécessaire qu'il y ait plusieurs programmes en mémoire.Quand l'un demandera une entrée-sortie,le contrôle du processeur sera rendu à l'un des programmes qui effectue des calculs.

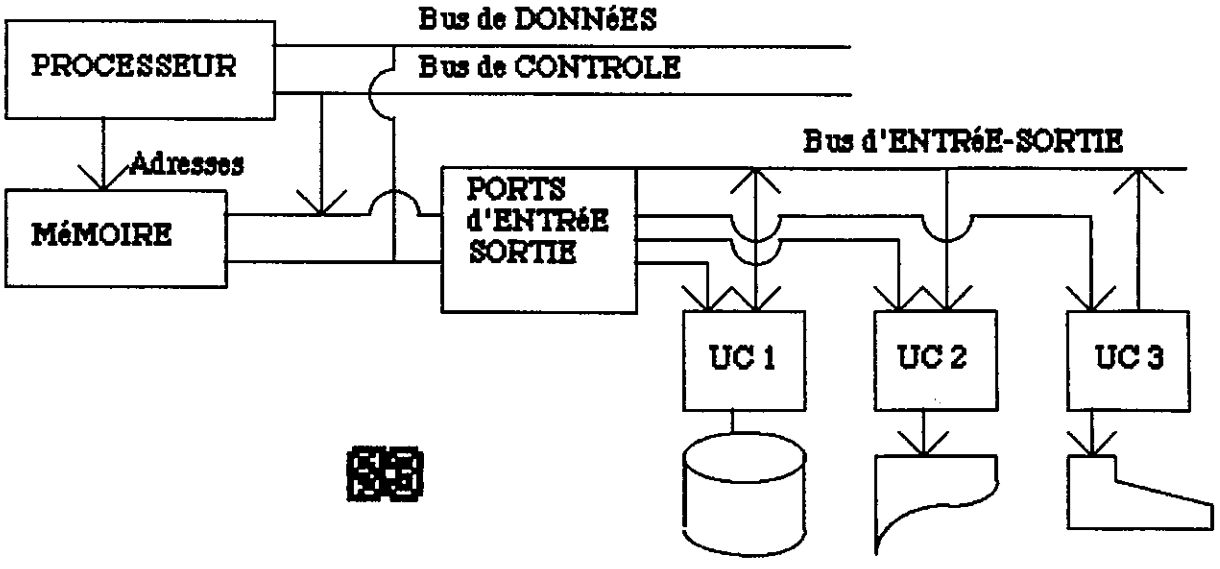
Ainsi,globalement la machine sera mieux utilisée.Tout cela n'est pas facile à mettre en oeuvre,et il ne semble pas que les solutions existantes mises en place actuellement,soient les plus efficaces.

Un utilisateur d'ordinateur voit donc la machine au travers de différentes couches de logiciel.



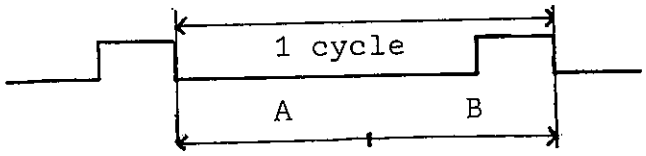

* A Gestionnaire de périphérique ( exemple: lire/écrire un secteur disque )
* B Système de gestion physique des entrées-sorties.
* C Moniteur.
* D Langages applicatifs et utilisateurs.

D'un point de vue matériel,la conception d'une telle machine est très rudimentaire et ne pose aucune difficulté.Les unités de contrôle sont connectées sur un bus d'entrée-sortie et dialoguent par *handshake* avec l'ordinateur.



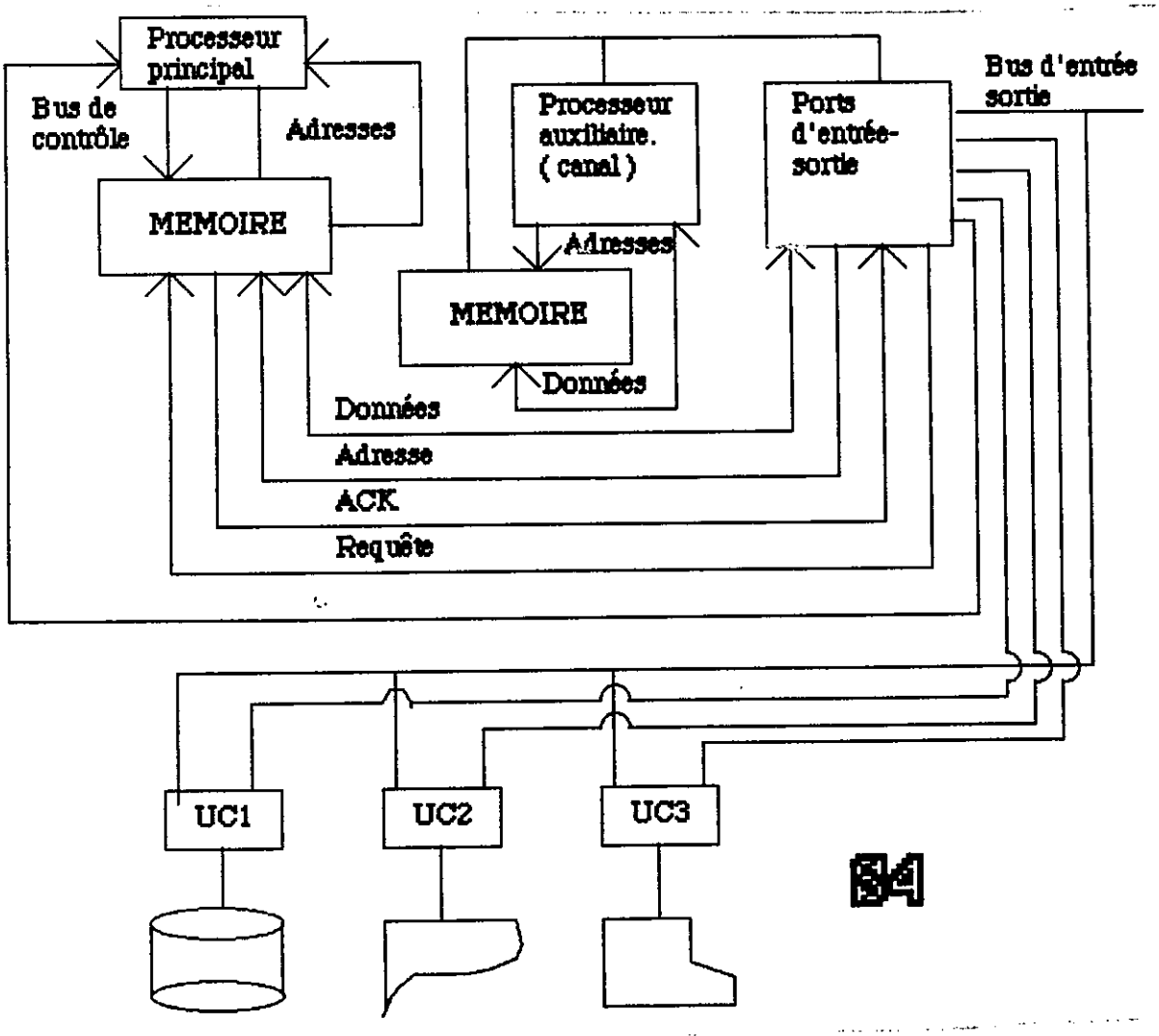
## LE PARALLELISME ENTRE CALCULS ET ENTREES-SORTIES.

On peut le réaliser grâce à un mécanisme d'accès direct mémoire.Un ordinateur fonctionne sur la base de cycle qui a la forme ci-après.




Le cycle se décompose en deux parties.La partie A dans laquelle le processeur demande un accès mémoire,la partie B pendant laquelle il traite l'information.La mémoire est donc libre pendant la partie B.On peut profiter de cette fenêtre B pour accéder à la mémoire sans perturber le processeur.

On dit que l'on accède directement à la mémoire.On se servira d'un processeur auxiliaire qui porte le nom de /canal/ pour gérer l'accès B.Ainsi la machine précédemment décrite revêtira l'architecture ci-après:



Le processeur canal accède à la mémoire de l'ordinateur principal ainsi:

* Il affiche sur ses ports d'entrée-sortie l'adresse du mot mémoire où il veut accéder dans l'ordinateur principal.

* Il émet une requête.

* Lorsque celle-ci a été satisfaite,le signal ACK est valide,la donnée est disponible.

Le temps d'attente est réduit à 1 cycle ( le signal ACK est donc pratiquement valide instantanément ).

L'émission de la requête peut provoquer l'incrémentation de l'adresse pour des transferts de bloc.Le canal et l'ordinateur principal connaissent l'adresse d'une case A dans l'ordinateur principal.Cette case A est l'adresse du programme canal.Lorsqu'elle a été mise à zéro par le canal,cela signifie que le canal est prêt à prendre une nouvelle commande.Quand l'ordinateur principal veut exécuter une entrée-sortie,il met dans A l'adresse du programme canal.Ainsi la programmation dans l'ordinateur principal peut être:

* Construire le programme canal ( c'est-à-dire la liste des opérations à effectuer,par exemple lire un secteur disque.)

* Attendre que A soit à zéro.

* Mettre en A l'adresse du programme canal.

* Emettre une interruption en direction du canal ( qui viendra lire la case A et la remettre à zéro ).

* Continuer le traitement .( l'entrée-sortie se déroulera simultanément à l'exécution du programme principal )

Dans l'ordinateur canal le programme est:

a) Dans le programme principal,traiter les programmes canaux les uns après les autres.

b) Dans le programme d'interruption,empiler le contenu de A,remettre A à zéro.

Pour améliorer le dialogue,on peut supposer que les deux ordinateurs connaissent une adresse B dans l'ordinateur principal.Si cette case est 0,le canal viendra mettre l'adresse du programme canal dont il vient de terminer l'exécution. Ainsi l'ordinateur principal recevant une interruption à chaque fin d'exécution de programme canal,pourra savoir laquelle des entrées-sorties en cours vient de se terminer.

Lorsqu'il a pris connaissance de cette adresse,il remet à zéro la case B.Dans cette architecture,on réalise un parallélisme effectif entre les traitements du processeur principal,et une entrée-sortie et une seule.Les autres sont mises en file d'attente grâce à la pile,mais vu du processeur principal,tout se passe comme si plusieurs entrées-sorties s'effectuent en parallèle.

Pour améliorer les performances,on peut chercher à rendre effectif le parallélisme avec toutes les entrées-sorties en cours,et non plus avec une seule entrée-sortie.Cela est relativement simple : remarquons en effet que le processeur canal est connecté au processeur principal par l'intermédiaire de ports; son accès direct mémoire est donc libre.Il suffit d'installer les unités de contrôle,non plus sur un bus d'entrée-sortie relié à des ports,mais sur un bus d'entrée-sortie relié à l'accès direct mémoire du canal.

Le processeur canal doit être un microprocesseur très rapide,par contre son jeu d'instructions peut être très réduit car il n'a à effectuer que des opérations très simples :un processeur en tranches bipolaires quatre bits convient tout à fait.

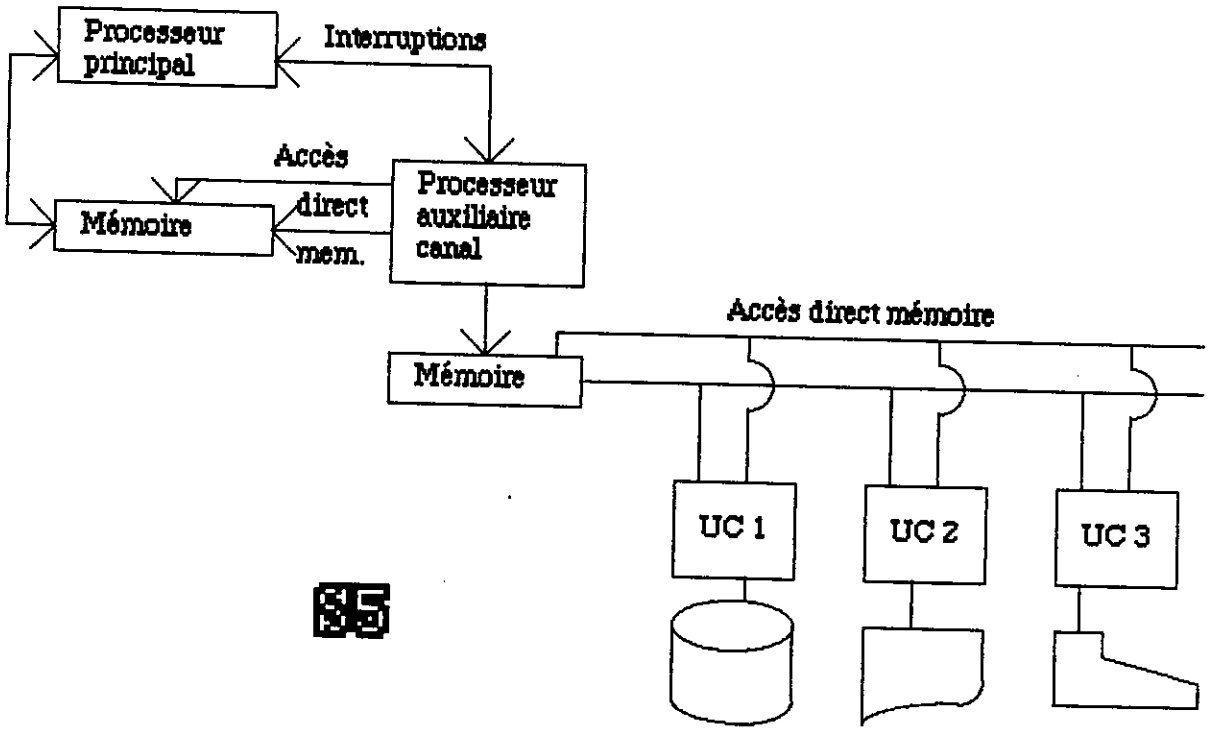
Les constructeurs d'ordinateurs dans leur quasi-totalité,intègrent les canaux dans des circuits spécialisés qui sont gérés par le processeur principal.Cela nous semble très lourd : il nous paraît beaucoup plus simple de faire gérer l'accès direct mémoire par un processeur auxiliaire comme expliqué précédemment.

Finalement nous disposons maintenant d'une machine à entrées-sorties toutes parallèles.

Le processeur canal transferre toutes les demandes dans sa mémoire,et les unités de contrôle viennent simultanément exécuter toutes ces requêtes par accès direct mémoire.

On peut alors réfléchir à la mise en oeuvre d'un système de conduite de l'ordinateur qui utilise efficacement cette architecture.Comme on l'a déja dit on peut positionner la mémoire du processeur principal,et dans chaque partition exécuter des programmes.Dès qu'un programme dans l'une des partitions effectue une entrée sortie,il perd le contrôle au profit d'un programme qui se trouve dans une autre partition,et ainsi de suite.

L'entrée-sortie du programme qui a perdu le contrôle s'effectuera pendant que la partition suivante calcule.C'est la multiprogrammation .




LA MULTIPROGRAMMATION

Pour illustrer l'amélioration des performances de la machine dûe à cette technique,prenons un exemple.

Soit un programme A qui,s'il s'exécute tout seul sur la machine dure 20 minutes.Sur les 20 minutes il utilise l'unité centrale pendant 2 minutes,le reste du temps il fait des entrées-sorties.De même,soit un programme B qui dure 10 minutes et utilise 5 minutes d'unité centrale,et un programme C qui dure 15 minutes et utilise 12 minutes d'unité centrale.

Si les programmes s'exécutent les uns après les autres sans parallélisme entre le calcul et les entrées-sorties,cela va durer 20 + 10 + 15 = 45 minutes.

Remarquons que l'exemple a été choisi pour que A fasse beaucoup d'entrées-sorties,que B n'en fasse que moyennement tandis que C en fait peu.On suppose aussi que les entrées-sorties sont réparties d'une manière homogène à l'intérieur des programmes.D'après le tableau ci-après,A utilise l'unité centrale à 10%, B à 50% et C à 80%.

| Programme | durée totale | temps unité centrale | temps entrée sortie | % d'utilisation de l'unité centrale |
|---|---|---|---|---|
| A | 20 | 2 | 18 | 10 |
| B | 10 | 5 | 5 | 50 |
| C | 15 | 12 | 3 | 80 |

Logeons maintenant en mémoire les trois programmes simultanément dans chacune des partitions,et supposons qu'il y ait du parallélisme entre calcul et entrées-sorties.

Pendant les 5 premières minutes A et B vont utiliser 60% du temps de l'unité centrale,n'en laissant que 40% à C qui en a besoin de 80% pour s'exécuter.C va donc s'exécuter pendant ce temps à la moitié de sa vitesse.Au bout de 5 minutes B sera terminé car il aura eu ses 5 minutes d'entrées-sorties et ses 5 minutes de calcul en parallèle.A aura eu 5 minutes d'entrées-sorties sur les 18 dont il a besoin.Il lui faudra donc encore 13 minutes pour se terminer.Pendant ces 13 minutes,l'unité centrale est utilisée à 90% de son temps : A et C s'exécutent donc à leur vitesse maximale.A se termine donc effectivement au bout de 13 minutes.Pendant les 5 premières minutes C tournait à la moitié de sa vitesse : il a donc eu l'équivalent de 2.5 minutes de calcul sur les 12 dont il a besoin.Il se termine donc au bout de 9.5 minutes.

Finalement B se termine au bout de 5 minutes,C au bout de 14.5 minutes ( 5 + 9.5 ) et A au bout de 18 minutes ( 5 + 13 ).Les trois programmes seront donc terminés au bout de 18 minutes au lieu de 45 minutes.Soit une amélioration des performances de la machine de 60% !

Le parallélisme entre calcul et entrée-sortie est donc un bon moyen d'accroître la puissance des ordinateurs.

Examinons maintenant quelles sont les incidences d'une telle architecture, sur le système de conduite de l'ordinateur .Rappelons que dans une machine en monoprogrammation,nous avions les couches successives de logiciel :

* A Gestionnaire de périphérique.
* B Système de gestion physique des entrées-sorties.

* C Moniteur.

* D Langages applicatifs et utilisateur.

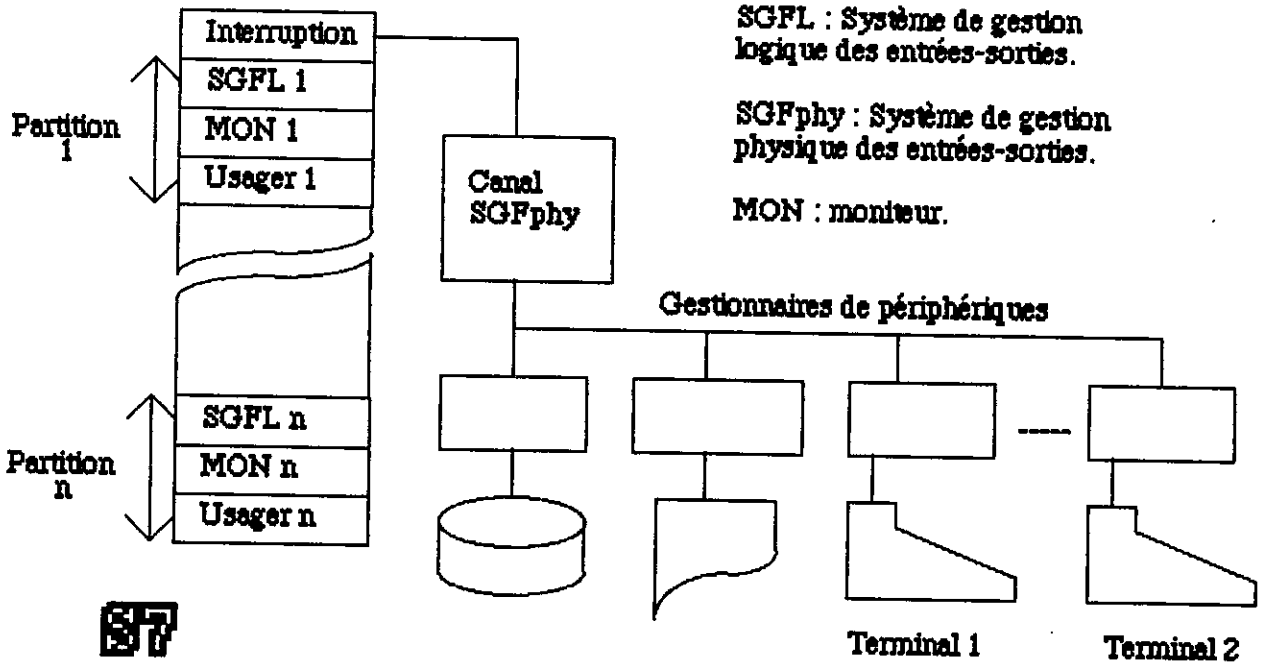
Dans une telle architecture,il n'y a qu'un processeur qui fait tout.Dans une machine fonctionnant en multiprogrammation,nous avons introduit des processeurs spécialisés. ( Un processeur canal,des processeurs qui gèrent les périphériques. ) Cela va être commode pour disperser les logiciels.Dans le processeur canal nous logerons le système de gestion physique des entrées-sorties.Dans les processeurs périphériques,les programmes de gestion des périphériques.

Dans le processeur principal nous aurons:

* un gestionnaire des interruptions.

* les différentes partitions.

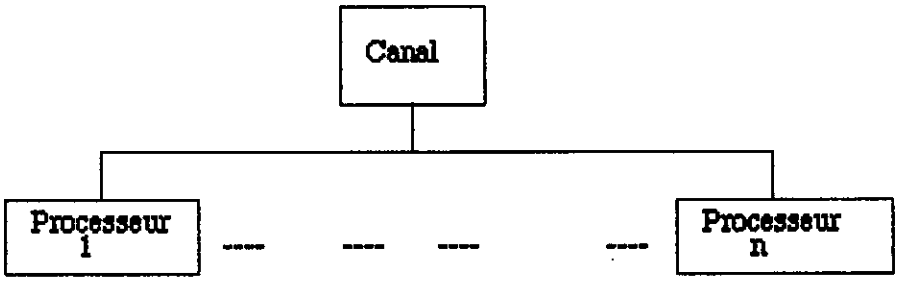
Dans chaque partition,il y aura un système de gestion logique des entrées-sorties,un moniteur,les langages applicatifs et utilisateur.La gestion logique des entrées-sorties,est un module qui convertit les enregistrements qui se trouvent être une notion de programmation au niveau de l'utilisateur,en bloc qui est,lui, une notion de programmation au niveau du système de fichier.Ainsi donc nous aurons la dispersion ci-dessous.



Le terminal 1 sera rattaché à la partition 1,...,le terminal n à la partition n.Il est évident que dans une telle organisation,le processeur principal doit être assez puissant ; il doit aussi disposer d'une capacité d'adressage assez grande,bien que l'on puisse toujours augmenter par des artifices câblés l'espace mémoire d'un ordinateur,mais surtout il est nécessaire d'assurer la protection mémoire entre les diverses configurations sans perturber la communication du canal avec d'autres partitions quand l'une d'elles est active.Toutes ces considérations font que la conception de l'ordinateur principal est assez complèxe.

## MACHINE MULTIPROCESSEUR A ARCHITECTURE DISPERSEE.

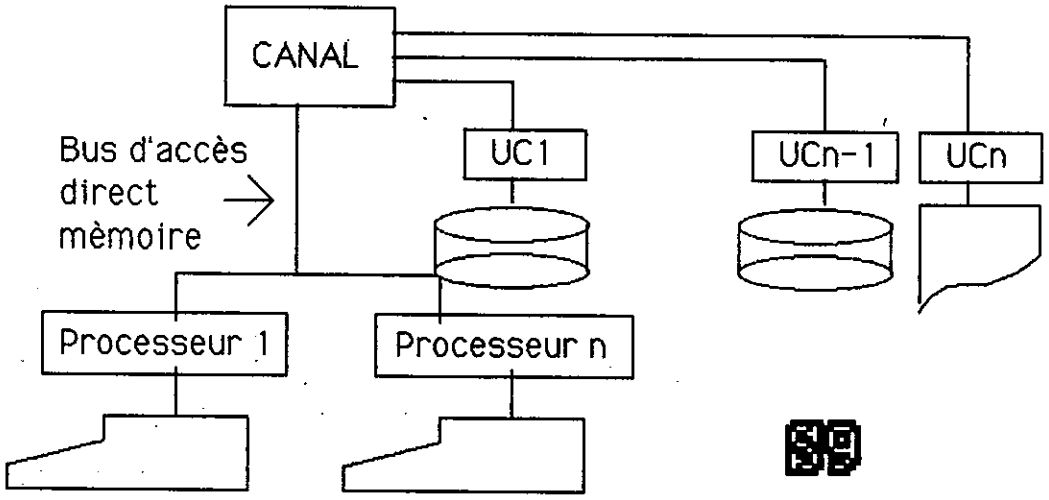
Les difficulté de construction de la carte principale nous ont amené à imaginer une autre architecture.Il n'est pas plus difficile d'attacher un canal à une seule machine,que d'attacher ce même canal à plusieurs machines,nous obtiendrons alors l'architecture ci-dessous,dans laquelle chaque processeur dispose de sa propre mémoire locale et à laquelle le canal accède directement.




Reste à savoir où allons-nous implanter les unités périphériques ? Puisque notre objectif est de considérer un ordinateur comme une fédération de machines réelles simples,et non pas comme un ensemble de machines virtuelles,les terminaux,unités non partageables,seront connectés sur les processeurs,tandisque les autres unités partageables seront reliées au canal.D'où l'architecture du schéma S9.

Le canal scrute en permanence les mémoires des processeurs 1 à n.Dès qu'il trouve une requête,il la stocke dans sa propre mémoire.Les unités de contrôle

scrutent la mémoire du canal,et si une entrée-sortie leur est destinée elles l'exécutent.

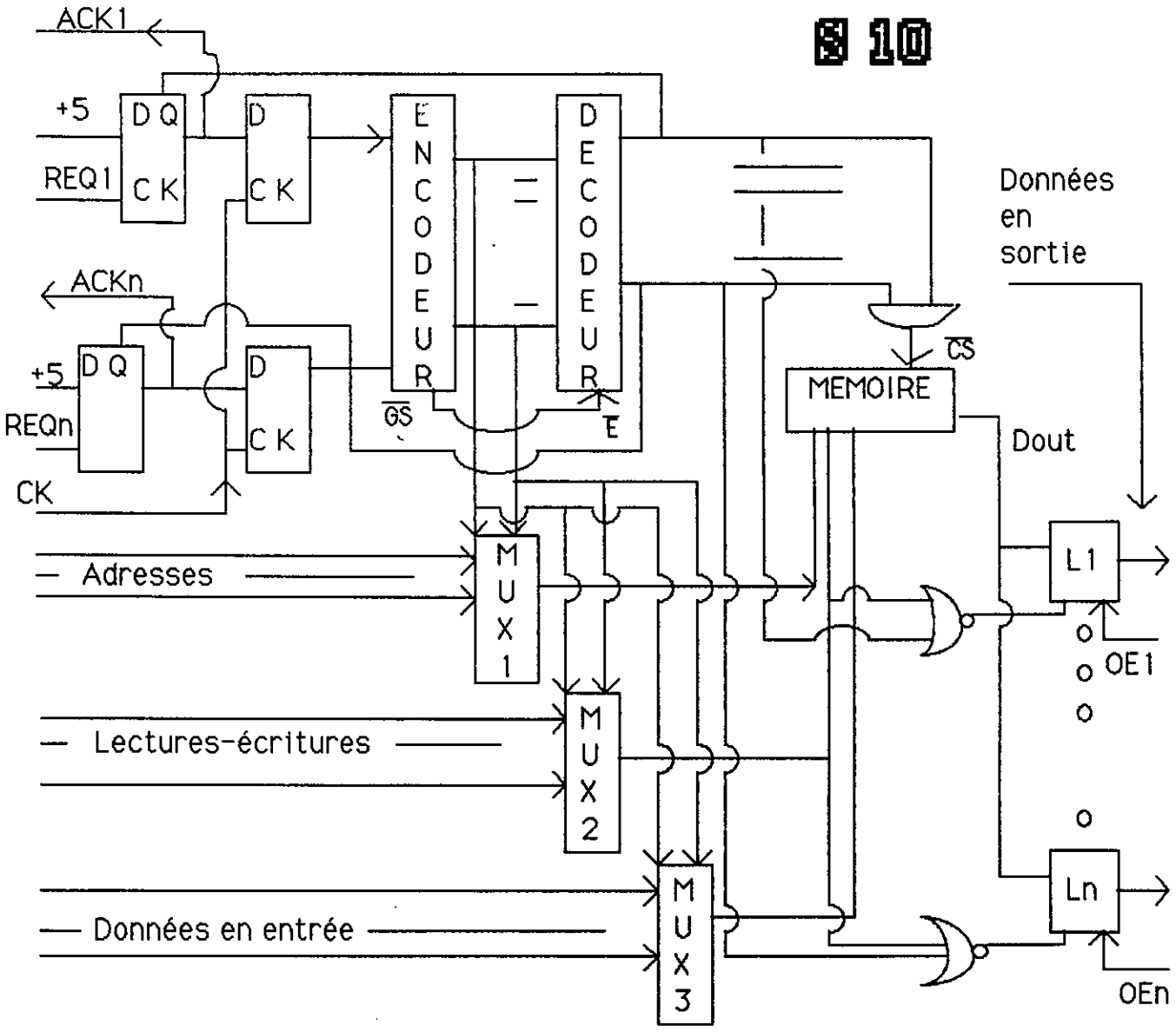



Le canal a un rôle :

1) de scrutation des processeurs,
2) de transfert de blocs d'information,
3) de gestionnaire de fichiers.

Il est donc très chargé.Dans une architecture dispersée,il faut éviter de concentrer des activités partageables sur un même processeur,sous peine de créer des goulots d'étranglement.

Nous utilisons donc une mémoire à accès multiple pour régler ce problème. Il est assez surprenant qu'un tel dispositif n'ait pas fait l'objet de recherches approfondies,par exemple en créant des circuits spécialisés,car il est très simple à réaliser et semble assez efficace.


ACK1
+5
REQ1
D Q
C K
D
C K
ENCODEUR
DECODEUR
ACKn
+5
REQn
CK
GS
E
Données en sortie
CS
MEMOIRE
Dout
Adresses
MUX 1
L1
OE1
Lectures-écritures
MUX 2
Ln
Données en entrée
MUX 3
OEn

La mémoire est asynchrone par rapport aux demandes : elle dispose de sa propre horloge CK dont la fréquence doit être de l'ordre du temps de cycle de la mémoire (de l'ordre de 200 ns ) .Les bascules D d'entrée mémorisent un 1 à chaque front montant d'un signal REQi ( requête d'entrée-sortie en provenance du processeur i ) .Les requêtes peuvent donc survenir simultanément.Le deuxième rang de bascules D,mémorise ces requêtes à l'aide de l'horloge CK,et stabilise ainsi les requêtes pendant un cycle.L'encodeur de priorité choisit la plus prioritaire,et le décodeur permet de remettre à 0 la requête.La sortie de l'encodeur positionne aussi les multiplexeurs sur les paramètres (adresse,sens du transfert,données en écriture ) de la requête en cours d'exécution.

Dès l'apparition d'une requête à la sortie du décodeur,la mémoire est sélectionnée ( CS ).Si l'on est en lecture,le NOR correspondant à la requête en cours ouvre le latch de sortie LEi où la donnée sera stockée.L'utilisateur,après avoir testé ACKi qui est actif immédiatement,l'émission de REQi,peut lire la donnée mémorisée dans le latch en ouvrant OEi,après que ACKi soit inactif.

Le programme d'accès à cette mémoire est donc :

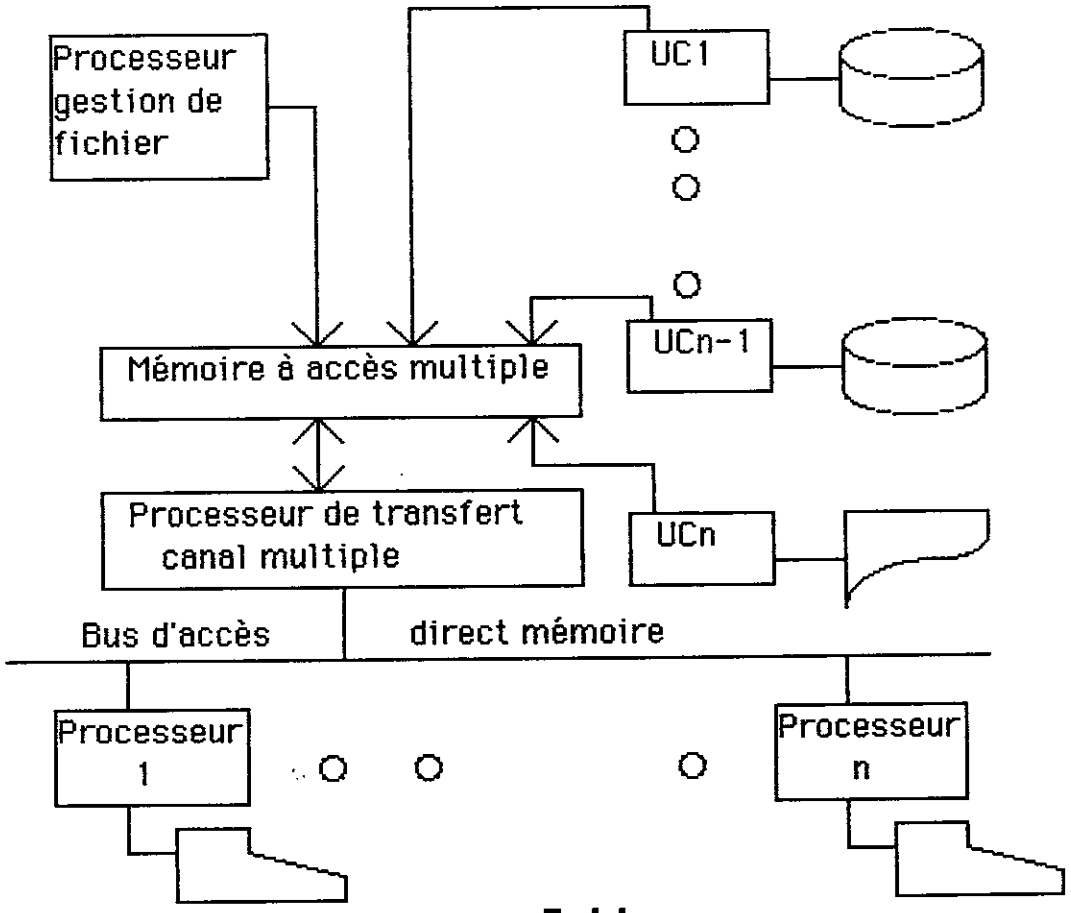
a) En écriture,
   * Afficher l'adresse,
   * Afficher le sens (R/W i = 1 ),
   * Afficher la donnée,
   * Faire monter REQi,
   *Faire descendre REQi.

b) En lecture,
   * Afficher l'adresse,
   * Afficher le sens ( R/Wi = 0 ),
   * Faire monter REQi,
   * Mettre 0 sur OEi,
   * Faire descendre REQi,
   * Attendre ACKi = 0 ,
   * LIre la donnée.

Remarquons que si l'on a un cycle de mémoire de 200 ns (ce qui est courant maintenant ) ,et 8 accès mémoire,et que l'on émette simultanément les 8 requêtes, celles-ci seront toutes traitées en 1,6 s ( 8 X 200 ns ).Il n'est donc pas nécessaire de tester les signaux ACKi ( dans le cas où on se limite à 8 accès par exemple ).Le sens

du transfert est le même pendant la transmission d'un bloc,il n'a pas besoin d'être réaffiché.De même l'adresse peut être automatiquement incrémentée à la fin de chaque requête.Les deux programmes précédents se réduisent alors à lire ou écrire la donnée et à faire monter et descendre REQi.Ils peuvent être réduits à une durée de 5 micro-s par octet au maximum,soit un débit de 200 Ko par seconde,ce qui est suffisant.Mais on peut encore de beaucoup augmenter cette vitesse en câblant la mémoire en mots de 16 ou 32 bits,puisque celle-ci ne manipule que des blocs.

Appliquons l'accès multiple à cette architecture multiprocesseur.




Σ 11

Le processeur de transfert scrute les processeurs de traitement pour savoir s'ils ont émis des demandes d'entrée-sortie. Si oui, il les transfère dans la **mémoire à accès multiple**.

Le processeur de gestion fichier contrôle sans cesse ces requêtes, et ne laisse les coupleurs accéder à cette information pour effectuer l'échange avec le périphérique que quand tout a été validé. Le processeur de transfert peut aussi faire communiquer les processeurs de traitement entre eux.

Remarquons que la technique d'accès à la **mémoire à accès multiple** est strictement la même que de l'accès direct mémoire. Elle peut être reliée au bus d'accès direct mémoire ( en pointillé sur le schéma ). Dès lors le processeur de transfert peut être réalisé en un seul boitier.

Les processeurs de traitement n'ont pas besoin d'être tous les mêmes ; ils peuvent être hétérogènes dès lors qu'ils respectent l'interface de communication.

## CONCLUSION

Nous avons réalisé une véritable machine multiprocesseurs à architecture dispersée, aux entrées-sorties rigoureusement parallèles. Toutes ces techniques ont été expérimentées à l'Université Pierre et Marie Curie, au **laboratoire d'informatique appliquée**.

Bon nombre de recherches sur les multiprocesseurs ont été entreprises. Elles consistaient à faire partager une mémoire commune par plusieurs processeurs. Cette approche est foncément vouée à l'échec car si deux processeurs font un accès simultané à la mémoire, l'un attendra l'autre. Que dire si n processeurs sont actifs ? Il y en a toujours n-1 qui attendent, car le propre d'un processeur est de faire à chaque cycle une référence mémoire. En général on ne dépasse jamais le bi-processeur.

A la limite on peut concevoir des systèmes à mémoire qui soit commune seulement pour les données, et surtout pas pour les instructions à condition que le processeur ait un mécanisme d'adressage qui s'y prête (l'adresse des données n'est pas dans les instructions mais dans des registres ).

Notre approche a été différente. Chaque processeur possède sa mémoire locale, et on utilise les **mémoires à accès multiples** uniquement pour les entrées-sorties c'est moins rapide qu'une mémoire de processeur, mais plus qu'un accès disque, et en outre cela permet le parallélisme. Donc cela accroit considérablement l'efficacité de l'ordinateur, dont la lenteur provient surtout du traitement des entrées-sorties.

Le **LABORATOIRE d'INFORMATIQUE APPLIQUéE**
est dirigé par le Professeur Roger DUPUY.

Les travaux sur les machines multiprocesseurs à architecture dispersée ont été effectués ou poursuivi par :

| | |
|---|---|
| B. Gossens | Assistant à Paris VII |
| D. Toupé | PDG Imega |
| J.R. Menand | Ingénieur à Thomson |
| S. Alsmadi | Assistant à Paris VI |
| M. Henni | Chercheur à Paris VI |
| S. Feltane | Ingénieur Imega |
| Akil | Professeur ESIEE |
| Bennis | Professeur à l'université de Rabat |

# NOTION D'INSERTION DANS LA MACHINE DE NOLIN.

edmond bianco

C.R. Subject classification informatics.: D31

Résumé.

La simplicité de forme de la machine de Nolin en masque un peu toute la puissance.L'Auteur montre ici comment la notion d'insertion concept classique s'il en est,se retrouve entière dans cette machine. Accessoirement ,et d'un point de vue pratique,on peut mesurer l'économie d'efforts que l'utilisation de cette notion représente pour le programmeur.Il est montré également l'élargissement du champ d'application d'une notion que permet l'utilisation de la notion de machine universelle.

# NOTION d'INSERTION DANS LA MACHINE DE NOLIN

edmond bianco

La machine de Nolin parait si simple qu'on peut raisonnablement se demander dans quelle mesure des structures algorithmiques complèxes pourraient être réalisées à l'aide de son langage.Une première forme algorithmique un peu complèxe,c'est l'insertion de procédure.Une autre encore plus complèxe c'est la machine universelle.Je vais essayer de montrer qu'il est possible,mais surtout comment il est possible de réaliser des algorithmes qui soient dotés de ces deux propriétés.

Il s'agit de bien poser le problème.Imaginons un langage qui permet de travailler sur des cases,telles qu'on en trouve dans une mémoire de machine de Nolin.Je réfléchis pour l'instant dans le domaine des machines simples,c'est-à-dire des machines définies par un algorithme et une mémoire,donc d'une machine qui ne sait traiter qu'un seul problème : celui qui est décrit par son algorithme.

Celui-ci construit,il fait partie de la machine et il devient indéformable en conséquence.

Je vais envisager les différentes constructions possibles,réalisables avec des insertions.Pour la commodité de l'exposé,je me donne un langage qui contient toutes les instructions de la machine de Nolin,et en plus l'instruction d'insertion.C'est avec ce langage que je me donne un ensemble complet de formes ,que j'essaye ensuite de traduire en termes de machine de Nolin elle-même.

## INSERTIONS EN PARALLELE

Je prends à titre d'exemple un algorithme qui a la structure suivante:

```
P:   début
        I1
        insérer Q ( a , b )
        I2
        insérer Q ( d , e)
        I3
        insérer Q ( f , g )
        I4
        fin

Q:   début
        I5
        fin
```

**A 1**

Les I1,I2,...,I5 ne contiennent que des instructions de la machine de Nolin :

```
ca := 0
ca := cb
si ca = 0 vers ei
ca := ca +k 1
vers ej
```

**T1**

Le problème se formule donc ainsi :

peut-on construire une machine de Nolin équivalente à celle qui vient d'être définie ? en d'autres termes,comment simuler l'insertion avec une machine qui n'a pas d'instruction d'insertion.

Pour envisager une solution ,je formule quelques hypothèses de construction : l'algorithme Q utilise ici 2 paramètres,j'imagine qu'il a besoin de 3 variables locales, je choisis les cases 0 et 1 pour les paramètres et 2,3,4 pour les variables locales.Je suppose que l'algorithme P utilise 5 variables,je lui réserve les cases 5,6,7,8,9 .

Que signifie alors :

```
insérer Q ( a , b )
```

Cela veut dire qu'il faut envoyer les contenus des cases a et b ,à prendre dans la liste des cases 5 à 9,dans les cases 0 et 1 respectivement.Ensuite il faut réaliser un aller à Q.Le retour sur I2 est un peu plus délicat.

J'utilise une case de plus,la 10,qui va servir pour un aiguillage.Je vais m'arranger pour que son contenu soit caractéristique de l'insertion qui se déroule,le programme précédent devient donc en termes de machine de Nolin :

```
P1:  début
        I1
        I4
        vers fin
```

```
c0 := ca
      c1 := cb
      c10 := 0
      vers Q1
e1:   ca := c0
      cb := c1
      I2
      c0 := cd
      c1 := ce
      c10 := 1
      vers Q1
e2:   cd := c0
      ce := c1
      I3
      c0 := cf
      c1 := cg
      c10 := 2
      vers Q1
e3:   cf := c0
      cg := c1
```

```
Q1:   I5
      si c10 = 0 vers e1
      c10 := c10 -k 1
      si c10 = 0 vers e2
      c10 := c10 -k 1
      vers e3
fin:  fin
```

**A2**

Bien entendu,les formes

```
c10 := 1
c10 := 2
c10 := c10 -k 1
```

sont des algorithmes que l'on sait écrire en machine de Nolin,et je n'utilise ici qu'une abréviation.

Il devient facile alors de se convaincre que,dès lors qu'un algorithme du type de A1 existe,il est toujours possible d'en déduire un algorithme du type de A2 qui,lorsqu'il se déroule produit bien la même modification de la configuration utile que ce que peut faire A1.

Le coût de l'opération se mesure ainsi:

a) Il faut transférer les paramètres effectifs ( quelqu'en soit le nombre ),et on constate qu'on peut toujours.

b) L'algorithme inséré doit être muni pour l'opération de "retour systématique", d'un aiguillage qui renvoit à autant de sorties qu'il y a de points de retour possibles.

Bien sûr si le nombre de ces retours dépassait k,le contenu maximum d'une case, on pourrait toujours prendre plusieurs cases,ce qui compliquerait le marquage de l'insertion,et l'aiguillage de retour systématique,tout en restant possible.

Remarque 1

Bien entendu,l'algorithme Q1 ne peut pas avoir de forme générale indépendante de celle de l'algorithme dans lequel il s'insère.Mais là on ne peut aller plus loin,car il faut voir que ces algorithmes sont considérés en fait comme câblés,ils constituent la partie algorithmique d'une machine,et prendre un élément d'un tel algorithme pour l'accoupler à un autre algorithme,n'a pas grand sens.

Imaginons alors le schéma suivant :

```
F:  début                    G:  début
    I1                           I6
    insérer G ( a , b )          fin
    I2
    insérer H ( d , e , m )
    I3                       H:  début
    insérer G ( f , g )          I7
    I4                           fin
    insérer H ( h , i , q )
    I5
    fin
```

**A3**

Je suppose que G travaille sur 2 paramètres et 3 variables locales,et H sur 3 paramètres et 5 variables locales,et aussi que F travaille sur 4 variables en tout ,par exemple.Je vais construire une machine de Nolin qui réalise un calcul équivalent.Pour cela,je constitue une configuration ainsi organisée :

| | | |
|---|---|---|
| Les cases | 0,1 | pour les paramètres de G |
| | 2,3,4 | pour les variables locales de G |
| Les cases | 0,1,2 | pour les paramètres de H |
| | 3,4,5,6,7 | pour les variables locales de H |

G et H interviennent de manière alternative donc je peux utiliser les cases à partir de 8 pour F.C'est ainsi que 8,9,10,11 sont disponibles pour F. Les cases 12 et 13 sont réservées à l'aiguillage ce qui me permet d'écrire A4 ; sous forme de deux colonnes,celle de droite venant à la suite de celle de gauche.

```
F1:  début
     I1
     c0 := ca
     c1 := cb
     c12 := 0
     vers G
e1:  ca := c0
     cb := c1
     I2
     c0 := cd
     c1 := ce
     c2 := cm
     c13 := 0
     vers H
e2:  cd := c0
     ce := c1
     cm := c2
     I3
     c0 := cf
     c1 := cg
     c12 := 1
     vers G
e3:  cf := c0
     cg := c1
     I4
     c0 := ch
     c1 := ci
     c2 := cq
     c13 := 1
     vers H
e4:  ch := c0
     ci := c1
     cq := c2
     I5
     vers fin
G:   I6
     si c12 = 0 vers e1
     c12 := c12 -k 1
     si c12 = 0 vers e3
H:   I7
     si c13 = 0 vers e2
     c13 := c13 -k 1
     si c13 = 0 vers e4
fin: fin
```

**A4**

On constate,pour le retour systématique,qu'en prenant autant de cases disponibles que d'algorithmes à insérer parallèlement,on obtient un algorithme encore équivalent.Il aurait été possible de n'utiliser qu'une seule case,par exemple 12,mais cela aurait un peu compliqué les aiguillages de retour systématique de G et H bien que cela demeure réalisable.

INSERTIONS EN SERIE

Il faut envisager également les séries d'insertions.Je me place devant le cas suivant :

```
W:  début                    X:  début
    I1                           I3
    insérer X ( a , b , d )      insérer Y ( e )
    I2                           I4
    insérer X ( h , i , j )      fin
    I8
    fin

Y:  début                    Z:  début
    I5                           I7
    insérer Z ( f , g )          fin
    I6
    fin
```

Je procède de la même manière en réservant la place pour chacun des jeux de variables :

| | | |
|---|---|---|
| Z | 2 par. | 3 var. loc. |
| Y | 1 par. | 2 var. loc. |
| X | 3 par. | 1 var. loc. |
| W | 5 variables en tout | |

Je réserve les cases suivantes :

| | paramètres | variables locales |
|---|---|---|
| Z | 7 , 8 | 9 , 10 , 11 |
| Y | 4 | 5 , 6 |
| X | 0 , 1 , 2 | 3 |
| W | 12 , 13 , 14 , 15 , 16 | |
| aiguillages | 17 , 18 , 19 | |

L'algorithme s'écrit alors ,toujours présenté en deux colonnes,mais mis sous forme bien évidemment,d'un algorithme à corps unique :

```
W1:  début
     I1
     c0 := ca
     c1 := cb
     c2 := cd
     c17 := 0
     vers X1
e1:  ca := c0
     cb := c1
     cd := c2
     I2
     c0 := ch
     c1 := ci
     c2 := cj
     c17 := 1
     vers X1
e2:  ch := c0
     ci := c1
     cj := c2
     I8
     vers fin
X1:  I3
     c4 := ce
     c18 := 0
     vers Y1
e3:  ce := c4
     I4
     si c17 = 0 vers e1
     c17 := c17 -k 1
     si c17 = 0 vers e2
Y1:  I5
     c7 := cf
     c8 := cg
     c17 := 0
     vers Z1
e4:  cf := c7
     cg := c8
     I6
     si c18 = 0 vers e3
Z1:  I7
     si c19 = 0 vers e3
fin: fin`
```

## LAMACHINEUNIVERSELLE

Tout naturellement sont apparues des limitations dûes à la spécificité des machines simples.Ainsi,un algorithme couramment utile ne peut pas être mis sous une forme suffisamment générale,pour être directement utilisable dans plusieurs machines simples différentes.

Ceci est évident,puisqu'on doit mettre dans l'algorithme un aiguillage qui réalise le retour systématique aux divers points d'appel,ce qui est faisable quand les points en question sont connus,au moment de la construction de cet aiguillage.

Pour accéder à un degré de généralité plus grand,il faut introduire le concept de machine universelle.Car la lecture et l'interprétation du code-programme par cette

dernière,lui permet de noter le point d'insertion,et donc de calculer le retour au moment du déroulement.C'est bien cela qui rend la procédure indépendante du programme dans lequel elle s'insère.

Afin de conserver à la machine universelle une forme relativement simple,j'adopte quelques conventions qui laissent à la charge du programmeur un certain nombre de manipulations.Je renvoie à la théorie de la Procédure Formelle pour l'étude dee la manière dont ces conventions peuvent être absorbées dans la structure de la machine universelle elle-même.Je veux simplement montrer ici que cette machine universelle existe.

CONVENTION 1

Toute procédure qui doit être insérée,doit commencer par transférer les contenus des dernières cases de la mémoire,dans les cases dont les contenus lui sont des paramètres.Je réserve donc ainsi aux dernières cases de la mémoire le rôle de cases de transmission des paramètres.

La machine universelle qui rencontre un code d'instruction d'insertion,dans lequel on trouve la liste des cases dont les contenus sont à transmettre - les paramètres effectifs - transmet ces contenus dans les dernières cases de la mémoire,puis effectue un aller-à la procédure à insérer,qui commence donc par récupérer ses paramètres.

Le code de l'insertion contient également la distance entre ce code lui-même et le code de l'instruction début de la procédure à insérer.

Pour constituer la configuration de la machine universelle,j'utilise les cases suivantes :

| p | p' | p'' | a | n | qi | | * |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

* = (k-1),ou (k-2),etc.

```
insert:c1 := c0
       c1 := c1 +k 1
       R^a1_d3                    qui met a en case 3
       c1 := c1 +k 1
       c2 := c1
       R^a2_d4                    qui met n en case 4
       c7 := 0
suit:  c7 := c7 -k 1
       c6 := c7
       c1 := c1 +k 1
```

```
        c2 := c1
        si c4 = 0 vers fins
        c4 := c4 -k 1
        R^a2_d5                          qui met qi en case 5

        S^a6_d5
        vers suit
fins:   S^a6_d1                          met c1 en réserve
        c0 := c0 +k c3
        S^a6_d1                  met en réserve le retour systématique
        vers entrée.
```

Je fais maintenant une autre convention :

CONVENTION 2

Pour réaliser le retour systématique,ayant laissé au programmeur le soin de mettre en place les paramètres à partir des cases (k-1),(k-2),...,(k-i) s'il y a i paramètres,je réserve la case (k- i- 1) pour l'adresse de retour.

Je crée,de plus,une instruction fin à laquelle je joins un numéro de case,à charge pour le programmeur de mettre dans la case de ce numéro,le contenu de la case (k-i-1).

```
fin:    c1 := c0
        c1 := c1 +k 1
        R^a1_d2
        c0 := c2
        vers entrée
```

Il ne me reste plus qu'à construire l'aiguillage de la machine universelle :

```
entrée: c1 := c0
        si c1 = 0 vers deb
        c1 := c1 -k 1
        si c1 = 0 vers fin
        c1 := c1 -k 1
        si c1 = 0 vers dpeg
        c1 := c1 -k 1
```

si c1 = 0 vers dpplun
c1 := c1 -$_k$ 1
si c1 = 0 vers vers
c1 := c1 -$_k$ 1
si c1 = 0 vers si
c1 := c1 -$_k$ 1
si c1 = 0 vers dpezr
c1 := c1 -$_k$ 1
si c1 = 0 vers insert

Ceci signifie que pour les divers opérateurs du langage de la machine de Nolin j'ai choisi les codes de la liste qui suit :

début : 0 ; fin : 1 ; := : 2 ; :=+1 : 3 ; vers : 4 ; si : 5 ; :=0 : 6 ; insérer : 7 .

Je laisse de côté les algorithmes : deb , dpeg , dpplun , vers , si , dpezn ,qui ne présentent aucune difficulté.

Il reste à montrer qu'on peut écrire des algorithmes de la forme :

$R^{ai}{}_{dj}$ et $S^{am}{}_{dn}$

cette écriture signifiant,donc,dans le premier cas,que la case N° i contient une adresse qu'on récupère le contenu qui se trouve à cette adresse,et qu'on le prote dans la case N°j. l'autre cas est l'opération inverse,en adresse dont la valeur est en case m ,j'envoie le contenu de la case N°n.

$R^{ai}{}_{dj}$ début — détruit le contenu de la case i

```
      ci := ci +k 1
      si cw = 0 vers e1
      cj := c(k-1)
      vers fin
e1 :  ci := ci +k 1
      si cw = 0 vers e2
      cj := c(k-2)
      vers fin
e2 :  ci := ci +k 1
      si cw = 0 vers e3
      cj := c(k-3)
      vers fin
e3 :  --------------
```

```
---------------------------------
---------------------------------
        e(k-1) : cj := c0
ek : fin :      fin
```

L'autre algorithme donne de la même manière :

```
S^af_dn         début
                cf := cf +_k 1
                si cw = 0 vers s1
                c(k-1) := cn
                vers fin
          s1 :  cf := cf +_k 1
                si cw = 0 vers s2
                c(k-2) := cn
                vers fin
          s2 :  cf := cf +_k 1
                si cw = 0 vers s3
                c(k-3) := cn
                vers fin
          s3 :  ---------------
          ----------------------
          ----------------------
          s(k-1) :  c0 := cn
          fin :  fin
```

Ceci achève de prouver la propriété :
dans le domaine de la machine de Nolin l'insertion de procédure existe en ce sens qu'on peut se munir de formes algorithmiques standard,réalisables dans le cadre d'abord spécifique des machines simples,et ensuite en toute généralité par le moyen de la machine universelle,qui elle-même existe,décrite en termes de machine de Nolin.

# RECURSIVITE

Il s'agit de montrer ici qu'une structure de procédure récursive générale peut être mise en oeuvre en toute généralité dans le système de la machine de Nolin.C'est une propriété d'existence qu'il faut établir,il n'est donc pas très important que la solution paraisse compliquée car elle n'a aucun caractère d'utilitarisme.Mais il est fondamental par contre,que l'algorithme soit complètement descriptible,et que ce fait puisse être prouvé.

Je fais quelques hypothèses sur la configuration,et je vais raisonner sur l'exemple simple d'une procédure qui s'insère elle-même.Je pars donc d'une forme algorithmique telle que celle-ci :

```
A:  début
    I1
    insérer A ( a , b )
    I2
    fin
```

Les I1 et I2 sont des suites d'instructions de la machine de Nolin prises exclusivement dans la liste des formes du tableau T1 .

J'ai besoin de quelques cases choisies à l'avance pour réaliser les manipulations sur les adresses afin d'assurer le passage d'une configuration à l'autre,et pour avoir dynamiquement les adresses des cases de chacune des configurations.En effet,à chaque insertion je crée une nouvelle configuration à la suite de la précédente.La case 0 contient l'adresse de la première case de la configuration en cours d'utilisation.

Cette adresse permet de calculer la mise en place des paramètres,puis le changement d'origine,en lui rajoutant le volume de la configuration,connu quand la procédure est construite.

Je suppose que les paramètres sont,en valeur,et dans l'ordre,dans les premières cases de la configuration.

Je suppose également que la procédure est écrite pour travailler sur une configuration (abstraite),dont la première case porte le N°0.

Pour préciser les idées,si je m'intéresse à la procédure A qui a besoin de 2 paramètres,et de 4 variables locales,la configuration se compose d'une suite de cases numérotées : 0,1,2,3,4,5

0 et 1,servant aux paramètres et les autres aux variables locales.

Je me réserve les véritables cases 0,1,2,3 du début de la mémoire pour les calculs d'adresses,de sorte que la première configuration coïncide avec les cases : 4,5,6,7,8,9 la configuration suivante avec 10,11,12,13,14,15 etc.

Il me devient impossible alors de programmer de façon directe,puisque si je dois travailler,par exemple avec la case 4,ce ne sera plus la 4 mais la 10,après déroulement d'une insertion,puis la 16 après encore une autre insertion,etc.

Ceci signifie qu'une instruction simple de la machine de Nolin,employée dans une procédure récursive devient un algorithme,et je vais montrer sur quelques exemples comment construire de tels algorithmes.Ainsi,

| l'instruction : | devient : |
|---|---|
| c5 := c1 | c1 := c0 |
| | $c1 := c1 +_k \underline{5}$ |
| | c2 := c0 |
| | $c2 := c2 +_k \underline{1}$ |
| | $R^{a2}_{d3}$ |
| | $S^{a1}_{d3}$ |

Les valeurs soulignées sont les numéros de cases de l'instruction de gauche.

| L'instruction : | devient : |
|---|---|
| $c5 := c5 +_k 1$ | c1 := c0 |
| | $c1 := c1 +_k \underline{5}$ |
| | $R^{a1}_{d2}$ |
| | $c2 := c2 +_k 1$ |
| | c1 := c0 |
| | $c1 := c1 +_k \underline{5}$ |
| | $S^{a1}_{d2}$ |

Si je nomme A la procédure récursive dont je me préoccupe,je peux imaginer une instruction qui s'écrirait : <u>Insérer</u> A ( 4 , 1 ) , les cases 4 et 1 sont celles de la configuration de A dont il faut transférer le contenu dans les cases 0 et 1 de la configuration suivante.

| L'instruction : | devient : |
|---|---|
| <u>Insérer</u> ( 4 , 1 ) | $c1 := c0 +_k \underline{4}$ |
| | $c2 := c0 +_k \underline{6}$ |

$R^{a1}_{d3}$
$S^{a2}_{d3}$
c1 := c0 +$_k$ 1
c2 := c0 +$_k$ 7
$R^{a1}_{d3}$
$S^{a2}_{d3}$
c0 := c0 +$_k$ 6
vers A

L'instruction : insérer A ( 4 , 1 ) ,ou plutot son algorithme,doit être suivi du texte suivant :

A' : c1 := c0 +$_k$ 4
c2 := c0 +$_k$ 6
$R^{a2}_{d3}$
$S^{a1}_{d3}$
c1 := c0 +$_k$ 1
c2 := c0 +$_k$ 7
$R^{a2}_{d3}$
$S^{a1}_{d3}$

Il est facile de généraliser ceci,pour une insertion avec n paramètres il faudrait répéter n fois la séquence du calcul des adresses en cases 1 et 2 suivi du R et du S.Il reste encore à préciser ce que devient l'instruction fin :

fin
c0 := c0 +$_k$ 6
vers A'

Tout ceci montre bien que,quelquesoit le texte de la colonne de gauche,on peut toujours lui faire correspondre un texte équivalent en colonne de droite,et qui,lui,n'utilise que les instructions élémentaires de la machine de Nolin.

Peut-être est-il bon de souligner ici encore le genre de simplification qu'apporte la notion d'index,qui permet de faire disparaitre les algorithmes du type $R^{ai}_{dj}$ et $S^{am}_{dn}$. On constate également,que la notion d'insertion de procédure se traduit en fait sous la forme d'une organisation particulière de la configuration,et là,je renvoie à la Procédure Formelle pour l'étude complète de la structure de cette configuration dans le cas général de la machine universelle.

## VOUZZAVEDIBISAR

**Langue de bois.**

Tentons une définition : Procédé d'expression de la pensée moderne au travers des moyens et des modèles perceptifs qui déterminent les réactions de l'être individuel,entité isolée de la société,dans la captation de l'information,sur la base de la communicatique.

**Voila un essai prometteur,choisissons encore quelques exemples, comment dire qu'il fait beau et que le vent va se mettre à souffler?**

Les configurations d'ordre nuageux s'éloignent de notre horizon de perception, et par le déclenchement d'un jeu anticyclonique de hautes pressions et d'un étalement dépressionnaire,amorcent un processus de déferlement d'un flux de compensation.

**Le candidat député fait un discours dans lequel il explique qu'il faut voter pour lui,car il va araser toutes les petites entreprises de la région qui ne permettent plus de développer vraiment celle-ci et implanter la Grande Société dont il est le représentant,pour cela il augmentera les impôts.**

Mes chers Concitoyens,je retrouve à votre contact la chaleur de mon bon vieux pays,accompagné de toutes les riches senteurs qui parlent à mon âme.Et c'est le soucis constant de participer au développement de cette région qui nous est chère qui m'amène,appuyé par le grand Parti que je représente ici,vous dire tout mon optimisme dans le rassemblement des forces essentielles qui sous-tendent le tissu tant social que culturel,de l'univers économico-productif de cette population laborieuse,que vous représentez et dont je ressens profondément en moi l'énergie bouillonnante. Mes Amis .

Le progrès c'est l'évolution .

Je suis venu puiser aux sources,ici,la conviction que notre développement passe par la prise en main de notre avenir ; tous ensemble , créons le renouveau ,

mettons en commun nos efforts pour faire de ce Vieux Pays le Pays Neuf qu'il mérite d'être, dans le cadre rénové d'une Nation jeune et dynamique au passé riche et au futur prometteur.

Je suis là.

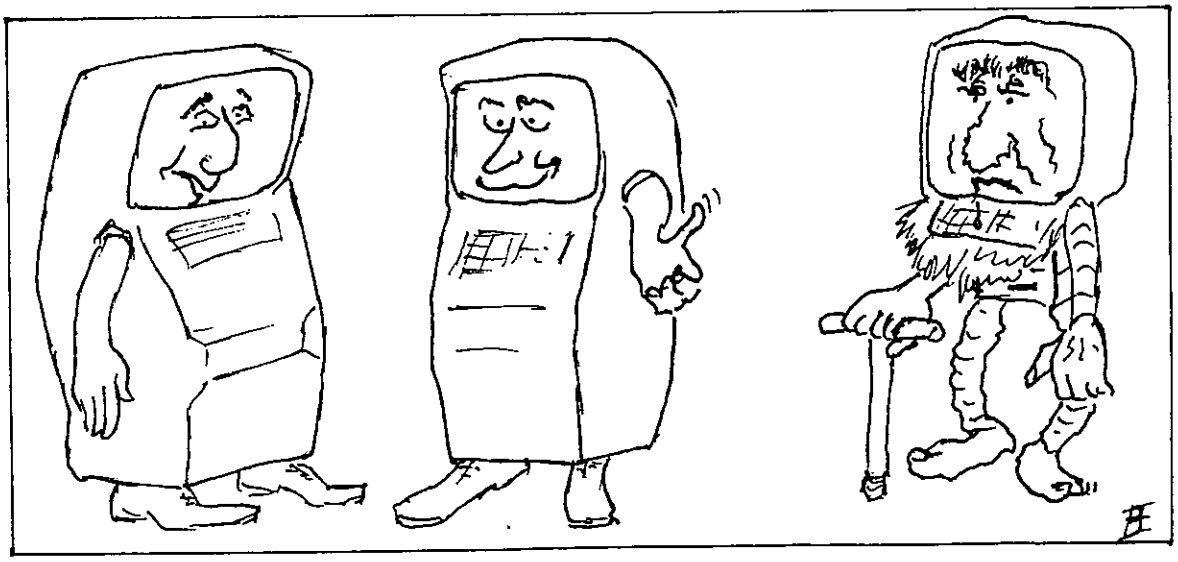
Je suis là pour vous appuyer dans vos efforts,Je suis là pour vous conseiller,Soyons un exemple et nous partagerons le fruit de ce digne effort.

Dans cet exemple là,visiblement la traduction en langue de bois est un peu abrégée,mais soyons modestes ce n'est qu'une petite illustration.

Bien maniée,la langue de bois peut-être,comme la musique militaire,génératrice de frissons,même si elle a quelquefois l'abstraction des mathématiques.Peut-être est-ce une forme moderne de la Poësie ?

Nous lançons un concours sur les plus beaux textes en cette langue , puisés dans la vie de tous les jours ou même imaginés pour l'occasion.Un beau clin d'oeil au gagnant.

B



Il est de la Cinquième génération....


Le premier ordinateur qui programme un homme.