CRASY

une architecture de calcul reconfigurable pour la simulation de réseaux neuromimétiques

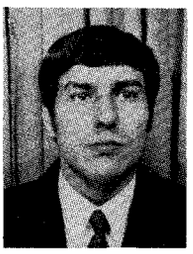
CRASY

a reconfigurable arithmetic architecture for neural networks analysis

Anne GUERIN

Institut National Polytechnique de Grenoble, École Nationale Supérieure d'Électronique et de Radioélectricité, Laboratoire de Traitement d'Images et Reconnaissance de Formes, 46, avenue Félix-Viallet, 38031 GRENOBLE CEDEX.

Maître de Conférence, ENSER, Grenoble. Axe principal de recherche depuis 1984 : Architecture de machines parallèles adaptées aux calculs neuromimétiques.



Jeanny HERAULT

Institut National Polytechnique de Grenoble, École Nationale Supérieure d'Électronique et de Radioélectricité, Laboratoire de Traitement d'Images et Reconnaissance de Formes, 46, avenue Félix-Viallet, 38031 GRENOBLE CEDEX.

Professeur à l'Institut Joseph-Fourier de Grenoble. Axe principal de recherche depuis 1970 : Modélisation de réseaux de neurones formels, aspects physiologiques et algorithmiques.

RÉSUMÉ

Depuis de nombreuses années, l'analyse de réseaux neuronaux suit un essor fantastique. L'étude suivant cette approche des fonctions postulées dans le système nerveux requiert de puissants outils de simulation. En nous inspirant à la fois des caractéristiques générales en traitement des signaux et des tendances actuelles vers le parallélisme, en matière de structures de calculateurs, nous proposons une architecture d'« array » processeur performante pour l'étude de réseaux récursifs adaptatifs en particulier et pour l'analyse de données (signal, image) en général: c'est le processeur « CRASY » (Calculateur de Réseaux Adaptatifs SYstolique).

MOTS CLÉS

Réseau neuromimétique, filtrage adaptatif récursif, « array » processeur, architecture « pipe-line » parallèle.

SUMMARY

Since a few years, neural networks analysis rouses great interests. According to this approach, the study of postulated functions in the nervous system demands some powerful simulation tools. Taking inspiration from general features of signals processing and from present tendances toward parallelism in computer architecture, we propose an efficient array processor architecture for recursive adaptive networks analysis and more generally for data (signal, image) analysis: it's the processor named CRASY (a systolique calculator for adaptive networks).

KEY WORDS

Neuromimetic network, recursive adaptive filtering, array processor, pipe-line parallel architecture.

Introduction

Dans les domaines du traitement d'images et des signaux, le besoin en calculateurs spécifiques répondant à une architecture non Von Neuman est très marqué. L'objectif principal est de briser le goulot d'étranglement inhérent aux voies de communication restreintes dans une structure classique. Déjà l'utilisation d'une architecture de type Harvard apporte une réelle amélioration sur la vitesse de traitement des données. Les processeurs de signaux présentent des performances de calcul très remarquables. Des circuits plus originaux comme le GAPP de NCR ou le Transputer d'INMOS, dont l'emploi est maintenant très répandu, offrent des outils très puissants pour la conception de systèmes parallèles de type SIMD ou MIND.

Cependant dans le domaine du Calcul Neuromimétique, en forte expansion, cette problématique des architectures parallèles à haut débit d'entrées/sorties n'est pas résolue. Par exemple la fonction de perception des signaux multidimensionnels, parfaitement maîtrisée par le système nerveux, l'est très difficilement sur un ordinateur; elle sous-entend, pour les informations perçues, le contrôle du cordage, du filtrage, du traitement lui-même... Toutes ces tâches sont exécutées en parallèle, elles mettent en jeu un très grand nombre de cellules nerveuses connectées en réseaux très denses. Ainsi tissés, ces réseaux confèrent au système nerveux une extraordinaire richesse, se traduisant par une grande diversité et complexité des traitements effectués. S'inspirer de ce modèle, c'est concevoir des structures dans lesquelles cohabiteront la puissance des traitements et la vitesse d'exécution.

Ainsi, l'effort n'est plus porté vers l'intégration d'une fonction unique très complexe, mais plutôt d'un grand nombre de cellules simples (« neuromimes ») à l'image des cellules nerveuses. Par exemple, concevoir un réseau de 1 000 cellules de traitement élémentaire, suppose la gestion de 1 million de points de communication possibles (synapses). Cette évolution quadratique de la complexité est une limite technologique à l'intégration de ces réseaux. Disposer de cellules élémentaires est simple, les faire toutes communiquer, là est le défi. On parle à ce propos de « Neurocalculateurs » ou de calculateurs de la sixième génération. Si l'intégration des réseaux neuromimétiques pose de nombreux problèmes, pour contourner ces difficultés et profiter pleinement des progrès technologiques, une nouvelle voie apparaît: la conception de machines spécifiques de simulations.

Actuellement la diversité des « Neurocalculateurs » met en évidence le fait que le choix d'une architecture est très complexe; il dépend fortement des objectifs recherchés et du degré de généralité des réseaux à étudier. Cet axe de recherche, par les réalisations concrètes qu'il engendre, soumet des idées originales d'architectures pour des machines intégrées plus performantes. L'objectif étant toujours de concilier les besoins en « megaflops » et en densité de communication.

Dans cet article, nous montrerons l'intérêt d'une structure de calcul modulable en fonction du volume de données et reconfigurable selon les différentes phases de calcul, pour présenter un maximum d'efficacité de traitement. A cet effet, nous décrirons un calculateur répondant à ces critères, particulièrement bien étudié pour le traitement de réseaux récursifs adaptatifs, et plus généralement pour l'analyse de données. Dans une première partie, seront décrites les grandes lignes directrices de l'architecture du calculateur: sa configuration doit s'adapter pour être performante à la fois en calcul vectoriel et en calcul scalaire.

Le calculateur prototype réalisé, cablé, appelé CRASY (Calculateur de Réseaux Adaptatifs SYstolique) [1] sera décrit plus en détail au second paragraphe.

Pour permettre d'évaluer les possibilités de l'architecture, nous présenterons en troisième partie, la simulation d'un algorithme de séparation de sources dans un environnement multidimensionnel [2]. Ce problème de séparation de sources, très important en traitement du signal, est résolu [3] selon une approche neuromimétique. L'implantation de cet algorithme sur CRASY montre toute la versabilité des configurations de calcul permises et la puissance du calculateur. Ces deux qualités en font un outil de travail très intéressant dans le domaines précités et ceux des réseaux neuronaux.

1. Traits généraux d'une architecture de calcul vectoriel et scalaire

1.1. Calcul vectoriel

Les applications en traitement numérique des signaux et des images, très voraces en temps de calcul, mettent en jeu des opérations sur des données très structurées. Celles-ci se présentent sous forme d'un flot de données: échantillons des signaux, pixels des images,... Chaque traitement est à itérer sur toutes les composantes du flot de données, il en est de même pour le traitement matriciel ou vectoriel.

Les « array » processeurs ou les processeurs vectoriels répondent favorablement à ces exigences: ils opèrent sur des matrices ou des vecteurs comme s'il s'agissait de données unitaires [4]. Ces processeurs sont très performants pour des algorithmes récursifs et parallélisables.

De plus pour ces calculateurs parallèles composés d'un ensemble de processeurs élémentaires (PE) associés à des modules de mémoires, la connexion d'un grand nombre de PE selon une technique « pipe-line » améliore considérablement les qualités du système. Mais chaque PE pris individuellement doit être aussi performant qu'une machine de type SISD. A ce propos, la loi d'Amdahl (1967) met en évidence l'effet désastreux, sur l'afficacité d'un système parallèle, d'une fraction de code ne pouvant s'exécuter que séquentiellement sur un PE [5].

En conséquence, pour les algorithmes difficilement parallélisables et contenant une partie de code séquentiel, il est indispensable de posséder des PE très performants. En d'autres termes, un système à architecture parallèle, pour être efficace, doit avoir des aptitudes aux calculs vectoriels mais aussi scalaires.

1.2. Calcul scalaire

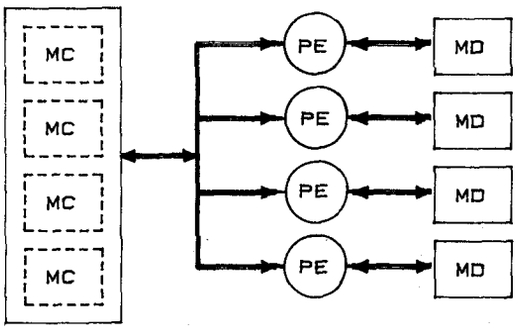
L'utilisation de la technique « pipe-line » améliorant considérablement la vitesse d'exécution d'un calcul vectoriel, a un effet contraire pour des vecteurs de faible dimension (à comparer avec le nombre de pas de « pipe-line ») ou pour des scalaires. Dans ce cas-là, une architecture parallèle où chaque PE travaille de façon autonome, est préférable.

Ainsi le passage d'un mode de calcul vectoriel à un mode de calcul scalaire impose la gestion d'un réseau de communication entre les PE.

1.3. Communication entre les processeurs élémentaires

Dans le domaine d'étude considéré, si les opérations vectorielles (sur un flot de données très ordonné), peuvent se décomposer en opérations élémentaires comme la multiplication de matrices ou plus simplement de matrices par des vecteurs, comme c'est souvent le cas, alors le réseau d'interconnexion peut être très rudimentaire: l'utilisation pour le calcul vectoriel, d'une chaîne unidirectionnelle de PE mises en « pipeline », est le réseau minimal à gérer. Ainsi, dans cette configuration, la communication entre les opérateurs élémentaires résultera de la propagation des résultats issus de chaque PE depuis le début de la chaîne jusqu'à sa fin.

A chaque PE est associé un module de mémoire; l'ensemble forme une entité de calcul élémentaire. La totalité de la structure de données (matrices, pixels, échantillons de signaux) est stockée de façon distribuée sur tous les modules de mémoire. Cependant, que ce soit dans une configuration vectorielle ou scalaire, pour exécuter l'ensemble des traitements, un PE peut avoir besoin d'accéder séquentiellement à toutes les informations (en lecture ou écriture) réparties sur tous les PE. Pour établir cette communication, une solution est préconisée. Chaque mémoire, associée aux PE, contenant cette information est déclarée en ressource commune, et peut être accessible en séquence par n'importe quel PE. Cette configuration est schématisée à la figure 1. Sur celle-ci n'est pas représenté le réseau d'interconnexion entre les PE.



MD: mémoire déclarée distribuée.
MC: mémoire déclarée commune.
PE: processeur élémentaire.

Fig. 1. – Communication par les mémoires déclarées communes.

Le processeur CRASY présenté ici, a été réalisé suivant ces principes d'architecture et de communication. Il présente donc une structure de calcul « pipe-line » ou parallèle, s'adaptant aux traitements à effectuer et un double mode de communication, soit par chaînage des PE, soit par les mémoires déclarées en ressource commune.

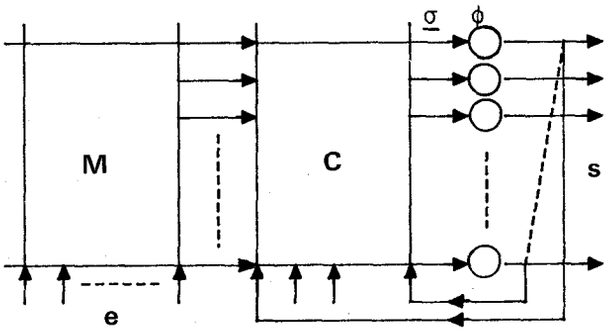
2. Description du processeur CRASY

2.1. Formalisme

Le processeur CRASY a été conçu pour l'étude et la modélisation de réseaux neuromimétiques. Dans l'analyse de tels réseaux, deux phases se distinguent; d'une part la relaxation du réseau, les connexions entre les éléments peuvent être récursives, et d'autre part l'adaptation des coefficients de connexion des cellules entre elles en fonction des activités présentes et passées du réseau pour minimiser ou maximiser un critère. Si celui-ci représente une fonction d'énergie, de coût, les problèmes traités selon ce modèle seront des problèmes d'optimisation (problème type du voyageur de commerce [6]). Il peut aussi représenter une distance entre deux motifs (motif reçu et motif désiré), la fonction du réseau sera alors une mémoire

associative [7]. L'analyse de données peut être entreprise selon une approche neuromimétique, la fonction d'adaptation réalisera un test d'indépendance statistique entre les signaux traités [3].

Considérons la formulation généralement admise pour modéliser un réseau neuromimétique schématisé à la figure 2.



Formulation :

$$(1) \qquad \begin{cases} \boldsymbol{\sigma}(t)=\mathbf{M}.\mathbf{e}(t)-\mathbf{C}.\mathbf{s}(t-1) \\ \mathbf{s}(t)=\mathrm{o}(\boldsymbol{\sigma}(t)) \end{cases}$$

$$(2) \qquad \begin{cases} \Delta\mathbf{C}=\mathrm{F}(\mathbf{C}, \mathbf{s}, \mathbf{e}, \boldsymbol{\sigma}, t, \ldots) \\ \Delta\mathbf{M}=\mathrm{G}(\mathbf{M}, \mathbf{s}, \mathbf{e}, \boldsymbol{\sigma}, t, \ldots) \end{cases}$$

Fig. 2. — Modèle de réseau entièrement connecté.

Ainsi, la première étape [équations (1)] est une phase itérative de relaxation du réseau, la seconde [équations (2)] est une phase d'apprentissage.

Les signaux notés **M.e** sont des entrées directes ou externes, ceux notés **C.s** ont des données récursives ou internes. Dans la matrice de connexion **C**, sont stockés tous les poids de connexion c_{ij} entre une cellule i et une cellule j. La fonction notée ϕ modélise la caractéristique de transfert, généralement non linéaire, de l'opérateur «neurone» simulé. Pour un filtrage linéaire, cette fonction est l'identité.

Le système d'équations (1) décrit la phase de relaxation. Si la matrice **C** est différente de la matrice nulle, les équations (1) sont itérées jusqu'à l'équilibre, c'est-à-dire la stabilisation des sorties **s**. Remarquons que d'un point de vue du calcul arithmétique, la résolution de ce système met en jeu des opérations de multiplications matrice × vecteur et d'addition (soustraction) de vecteurs.

Le système d'équations (2) décrit très généralement les équations d'adaptation des coefficients de connexion directe **M** et de connexion récursive **C**. Soit **C***, l'état stable du réseau à atteindre, minimisant le critère choisi, alors $dc_{ij}/dt=0$ en $c_{ij}=c^*_{ij}$. Dans cette phase le système résout par itérations successives n^2 équations différentielles du type :

$$(3) \qquad dc_{ij}/dt=f(c_{ij}, e_i, \sigma_i, s_i, s_j, t, \ldots)$$

Il en est de même si la matrice **M** est adaptative. L'implantation de cette loi requiert pour les calculateurs, des puissances de calcul très importantes, comme pour résoudre en parallèle les n^2 équations différentielles citées.

Ces deux phases de calcul font appel à deux environnements arithmétiques différents, ainsi décrit au paragraphe précédent.

2.2. Phase de relaxation

2.2.1. *Utilisation d'une chaîne de processeurs élémentaires*

La résolutions du système d'équations (1) est caractérisée par un enchaînement constant de multiplications matrice × vecteur. Le choix d'un algorithme systolique de multiplication matricielle s'impose quant aux caractéristiques de cette phase. En effet, la régularité d'un réseau systolique contribue à donner au système un fort degré de modularité. Cela est important pour l'extension de ce dernier.

Une aire systolique simple et très efficace (rapport du nombre de PE actifs/nombre de PE total) pour le traitement matriciel ci-dessus, est une aire linéaire unidirectionnelle («one-way linear array») [8]. Elle est composée de cellules identiques, chacune effectuant une multiplication et une addition. Considérons la multiplication d'une matrice **A** de dimension $p \times n$ et d'un vecteur **x** de dimension n :

$$y_i=\sum_{j=i}^{n} a_{ij}.x_j \qquad i=1, \ldots, p.$$

Cette somme de produits se réalise pas à pas dans chaque PE (cellule systolique) et ce pour toutes les composantes du vecteur résultat **y**. La figure 3 illustre cet exemple pour $n=5$ et $p=4$.

On remarque que le flot des composantes de sortie est disponible après n cycles de base. Ainsi en enchaînant les multiplications les unes à la suite des

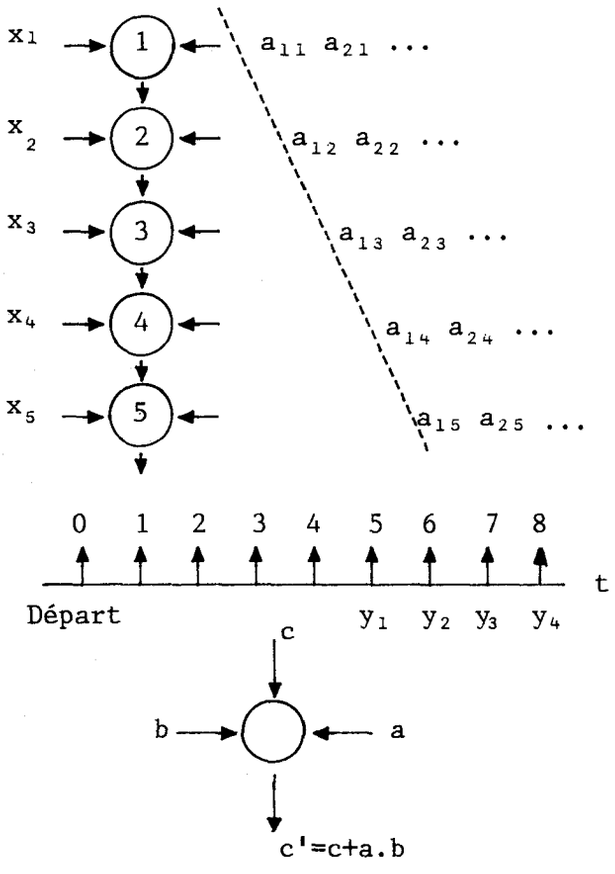
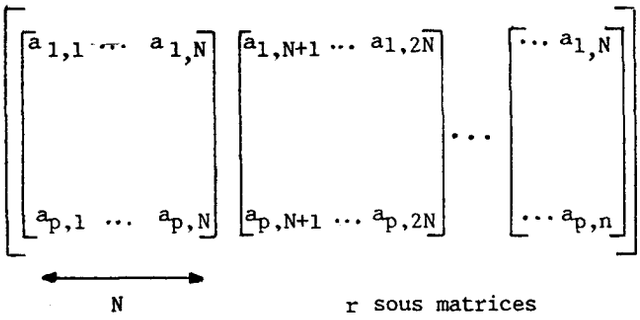


Fig. 3. — Algorithme systolique de multiplication matrice × vecteur.

autres, ce temps mort n'apparaît qu'une seule fois, à la première multiplication. Cette structure est très adaptée à la phase de relaxation.

Sa simplicité en fait une structure aisée à mettre en œuvre. Pour les opérations matricielles mettant en jeu deux matrices, le temps d'exécution est évidemment plus long avec une aire unidimensionnelle qu'avec une aire bidimensionnelle. Mais par contre le taux d'utilisation des PE est plus élevé avec une aire unidimensionnelle [9]. De plus cette structure étant minimale pour les opérations vectorielles, elle offre ainsi beaucoup de souplesse: un seul paramètre est à déterminer, c'est le nombre N de PE. En effet les dimensions des vecteurs ou des matrices à traiter peuvent être grandes vis-à-vis de N. Il paraît donc nécessaire que la taille de l'aire de calcul s'adapte à ces dimensions. Cependant il existe une organisation particulière des données qui peut rendre indépendantes les dimensions vectorielles et la taille de l'aire de calcul.

Reprenons l'exemple précédent du calcul de $\mathbf{y}=\mathbf{A}.\mathbf{x}$. Le nombre de cycles élémentaires pour le calcul des composantes de **y** est minimal ($n+p$ cycles) quand le nombre de PE est égal à la dimension du vecteur **x** à multiplier. Quand cette dimension est quelconque l'organisation adoptée est la suivante: la matrice **A** et le vecteur **x** sont décomposés en sous-matrices (cf. *fig.* 4) et sous-vecteurs.



Si n MOD N = 0, $r = n/N$.
Sinon $r = \text{INT}(n/N) + 1$.

Fig. 4. — Décomposition en sous-matrices.

A partir de cette décomposition, pour obtenir toutes les composantes du vecteur résultat, il faut accumuler les r résultats partiels issus des multiplications des r sous-matrices avec les r sous-vecteurs.

Ainsi sans perte d'efficacité (sauf lorsque n/N n'est pas entier), la dimension des structures de données est compatible avec la taille généralement plus petite du réseau de PE. La durée du traitement pour la multiplication donnée en exemple est alors de $N + r.p$ cycles élémentaires.

Les figures 5 et 6 illustrent l'agencement des données autour de l'aire systolique. L'exemple pris est celui de l'enchaînement de deux multiplications, **A1.x1** et **A2.x2,** avec dim(**xi**) = ni et dim(**yi**) = pi. Dans un premier cas, avec $N = n1 = n2$, la mémoire associée au i-ième PE stocke d'une part, les i-ièmes composantes des vecteurs **x1** et **x2** et d'autre part, les i-ièmes colonnes des matrices **A1** et **A2,** en formant ainsi deux blocs alimentant en données le i-ième PE. A la sortie de l'aire, les composantes de **y1** puis de **y2** sont disponibles et sont stockées dans les mémoires associées des PE, déclarées en utilisation commune.

Dans le cas réel où N est différent des dimensions des espaces vectoriels, en sortie de l'aire ne sont disponibles que des résultats partiels issus des multiplications avec les sous-éléments. Un accumulateur est placé à la suite du dernier PE composant la chaîne (cf. *fig.* 6), dans la boucle de retour des composantes à stocker dans les mémoires déclarées en utilisation commune. Le nombre de PE est réduit mais la capacité des mémoires associées à chaque PE est augmentée. Cette capacité et le nombre de PE fixent la taille maximale d'une structure de données pouvant être traitée dans l'aire, sans partition.

Similairement à l'architecture d'un «array» processeur, celle du calculateur, telle qu'elle est représentée à la figure 6, met en évidence la notion d'entité de calcul élémentaire. Nous appelerons «tranche» de processeur, l'ensemble constitué d'un PE et de ses différents modules de mémoires. Chaque PE est constitué de modules arithmétiques ($+$, $-$, $\times$).

En découpant la mémoire allouée à un PE en plusieurs modules, leur organisation (commune ou distribuée) n'est pas figée et peut évoluer suivant les traitements.

Regardons maintenant, comment dans CRASY suivant ce principe, la relaxation d'un réseau neuromimétique est établie.

2.2.2. *Équilibre du réseau*

On considère le système d'équations (1). Les matrices de connexion directe **M** et récursive **C** sont distribuées sur les modules de mémoire de chaque PE, de même pour le vecteur d'entrée **e**. Une fois la contribution des entrées **M.e** évaluée, l'aire de calcul formée par les PE en «pipe-line est alors constamment alimentée par deux flots de données: d'un côté les coefficients de connexion **C** et de l'autre les activités des sorties **s**. La figure 7 symbolise ce traitement. A chaque évaluation de **s**, les mémoires «côté vecteur» alternent leur rôle (déclaration MD-MC).

En présentant le vecteur **M.e** au premier PE de la chaîne, l'addition des deux vecteurs **M.e** et $-$**C.s** est effectuée en même temps que le produit $-$**C.s.** En sortie de l'accumulateur, les composantes du vecteur $\boldsymbol{\sigma} = \mathbf{M}.\mathbf{e} - \mathbf{C}.\mathbf{s}$ sont disponibles et sont appliquées en entrée d'une table de transcodage ϕ de la fonction de l'opérateur «neurone». L'élaboration des activités des sorties **s** par une table de transcodage offre ainsi pour l'utilisateur, une gamme très étendue de fonctions possibles.

2.3. Phase d'adaptation

2.3.1. *Configuration parallèle des PE*

Dans un «array» processeur, réaliser l'actualisation de n^2 coefficients revient à implanter la loi de modification dans son intégralité, sur tous les PE. Ils dérouleront ainsi en parallèle, de façon autonome, la suite des opérations décrivant la loi. Pour une «tranche» de processeur numéro i, tous les coefficients stockés dans le module numéro i, sont modifiés suivant la loi programmée, les uns à la suite des autres, par les résultats fournis par le PE numéro i.

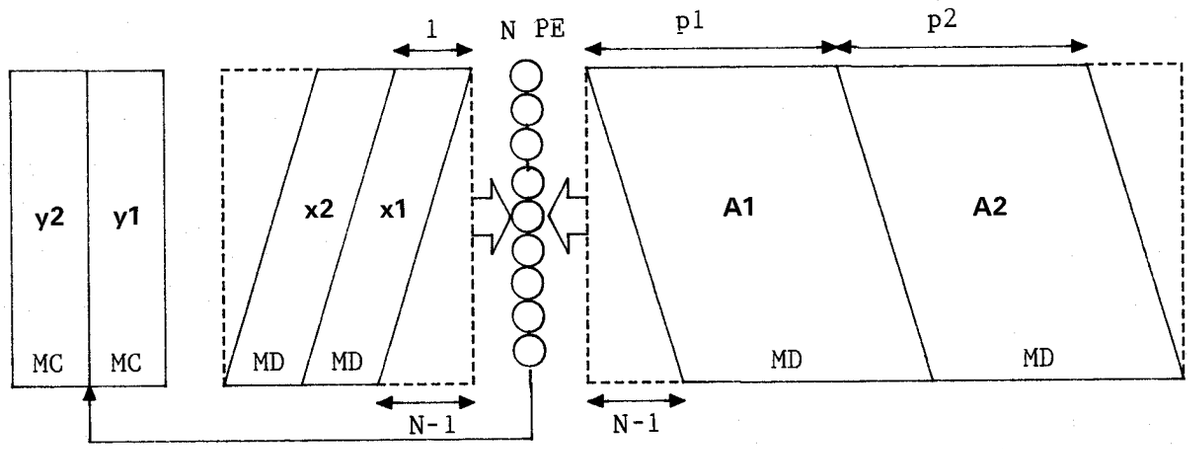


Fig. 5. — Implantation de l'algorithme de multiplication avec $N = n1 = n2$.

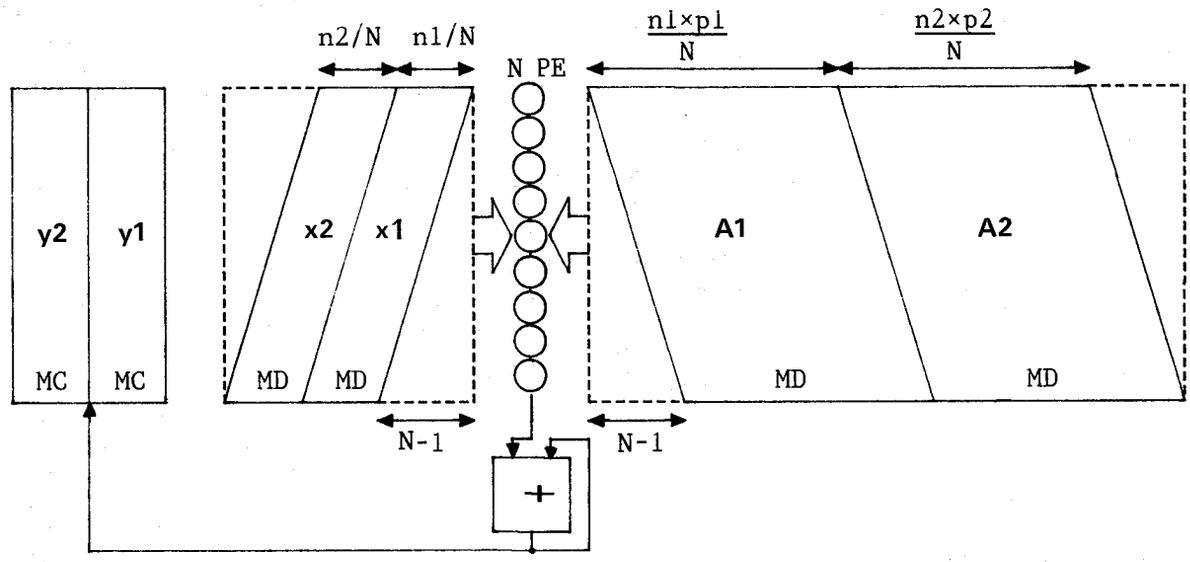


Fig. 6. — Implantation de l'algorithme de multiplication avec $N \langle\rangle n1 \langle\rangle n2$.

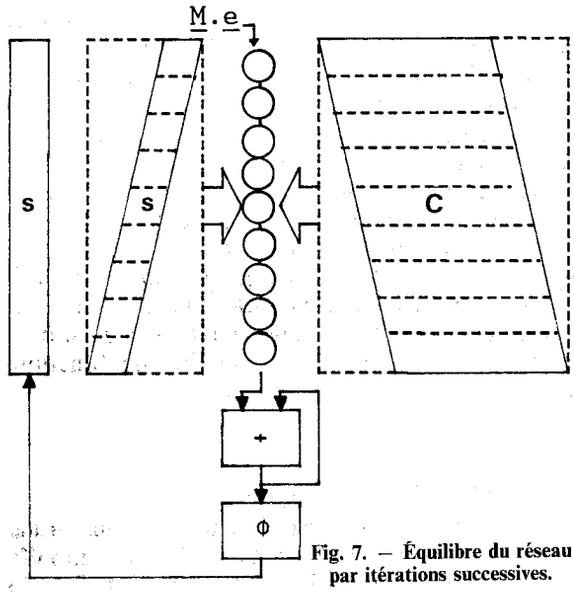


Fig. 7. — Équilibre du réseau par itérations successives.

Donc si la loi d'actualisation est locale (l'adaptation d'un coefficient ne dépend pas d'un autre coefficient), la distribution des n^2 équations différentielles sur les N « tranches » est alors identique à celle conduisant à la mémorisation des n^2 coefficients modifiables sur les N modules de mémoire. Dans ce cas-là, ces N modules sont déclarés en utilisation distribuée (cf. *fig.* 8). Si la loi n'est pas locale (l'exemple le plus simple : elle dépend d'une fonction de normalisation), la première étape est de calculer cette fonction en déclarant les N modules mémorisant les coefficients, en utilisation commune.

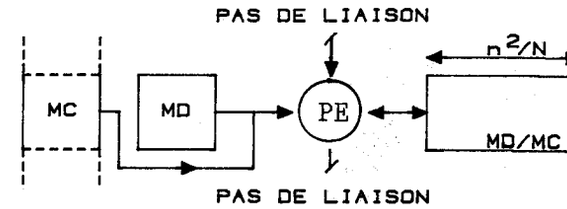


Fig. 8. — Cassure de la chaîne de PE dans la configuration parallèle.

Les limites à l'implantation d'une loi d'actualisation sont liées aux facultés de calcul des PE. Dans le processeur CRASY, un PE est une cellule de multiplication, d'addition et de soustraction en virgule flottante. Cette cellule très simple permet déjà la simulation de nombreux algorithmes neuromimétiques. L'utilisation d'un processeur de signal comme PE élargit la gamme des équations d'actualisation implantables et facilite la mise en œuvre de cette phase de traitement.

2.3.2. *Actualisation des coefficients dans CRASY*

Pour le prototype réalisé, la classe des lois de plasticité permises couvrent largement celles utilisées pour l'analyse des réseaux neuromimétiques. Cette classe peut être définie par une équation générale de la forme :

$$(4)\quad dc_{ij}/dt = a.c_{ij} + f_1.f_2 + \ldots + g_1.g_2 + \ldots + h_1.h_2 + \ldots + \ldots$$

où $f_1, f_2, \ldots, g_1, g_2, \ldots, h_1, h_2, \ldots$ sont des fonctions quelconques à une variable, dont le nombre est limité à 16 dans CRASY. Puisque les fonctions arithmétiques réalisées par un PE sont des multiplications, additions et soustractions, la classe des équations d'actualisation permises met en œuvre des sommes de produits.

Le temps d'exécution de cette phase à $k.n^2/N$ cycles élémentaires, où k représente le nombre de cycles pour exécuter une équation différentielle sur un PE.

2.4. SYNOPTIQUE GÉNÉRAL

2.4.1. *Présentation*

Le synoptique présenté à la figure 9 résume l'ensemble des voies de communication autour des PE. Il met en évidence la distinction à faire entre les « tranches » de processeur identiques, duplicables et les modules uniques de calcul particuliers pour une application donnée (accumulation, ϕ, f, g, h, ...). Ces modules s'insèrent à la suite du dernier PE de la chaîne.

On remarquera en trait plein, le bus commun à tout le processeur permettant d'une part la liaison avec le calculateur hôte et d'autre part l'implantation du mode de communication basée sur la mémoire, où sur ce bus, une donnée peut être diffusée à toutes les tranches du processeur.

Sur ce synoptique sont figurés le bloc de séquencement et l'interface d'entrée/sortie reliant le processeur à un calculateur hôte.

2.4.2. *Description d'une tranche opérative sur CRASY*

Chaque tranche opérative sur CRASY d'un module arithmétique et de deux modules mémoire. Le module arithmétique est lui-même composé d'un additionneur-soustracteur point flottant et d'un multiplieur point flottant. A une fréquence d'horloge de 10 MHz, chaque tranche à un taux de calcul de 20 MFLOP/s. De part et d'autre de la cellule arithmétique, deux modules mémoire sont connectés. Leurs capacités sont réduites sur le processeur CRASY à 8 kmots pour le « côté vecteur » et à 16 kmots pour le « côté matrice ».

2.5. CONTRÔLE DES SÉQUENCEMENTS PAR MICROPROGRAMMATION

Le contrôle de cette structure est réalisé par une unité de séquencement central. Elle génère tous les microcodes de contrôle donc en particulier la validation et la dévalidation des voies de communication. Une « tranche » est microprogrammable et le même microcode est exécuté par toutes les « tranches ».

Dans la configuration « pipe-line, à chaque cycle d'instruction, le séquenceur contrôle l'opération que doivent exécuter tous les PE, organise le flot de don-

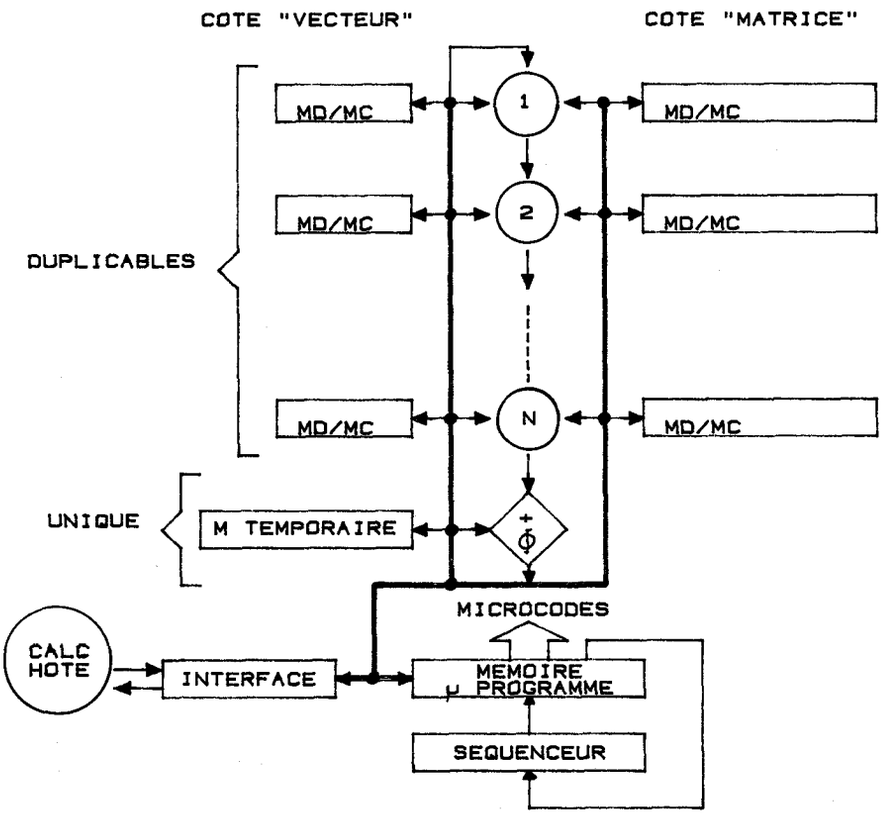


Fig. 9. — Synoptique général.

nées « biaisé » (cf. *fig.* 3) alimentant les PE et enfin valide les voies de communication utilisées.

Dans la configuration parallèle, le principe est identique, avec cette seule différence que le flot de données n'est pas « biaisé », c'est-à-dire qu'à chaque cycle tous les PE sont en phase.

Ainsi, le contrôle d'une chaîne de PE est identique à celui d'un PE unique.

2.6. Avantages du processeur CRASY

De l'architecture matérielle et logicielle choisie, se dégagent quatre avantages importants:

– La modularité: une « tranche » est une unité de calcul autonome. Le processeur CRASY en tant que prototype n'est composé que de deux « tranches ».

– La puissance de calcul augmente linéairement avec le nombre de « tranches » mises en jeu. Avec deux « tranches » la performance maximale de CRASY est de 2×20 soit 40 MFLOP/s.

– Chaque PE s'insère dans la structure de calcul de deux manières: avec l'ensemble des PE en « pipe-line, pour des calculs vectoriels et seul, indépendamment des autres PE en parallèle, pour des calculs scalaires.

Contrairement aux processeurs vectoriels (*cf.* loi d'Amdahl), la vitesse du calcul n'est pas réduite lors de la phase « calculs scalaires »: CRASY fonctionne toujours à sa puissance calcul maximale.

– La taille des réseaux simulés n'est pas limitée dans le principe: on peut soit augmenter le nombre de « tranches » soit augmenter la capacité des mémoires sur chacune d'elles.

Par contre, la mise en œuvre d'un système microprogrammé offrant beaucoup de souplesse d'utilisation est longue et délicate, nous envisageons d'écrire un langage de haut niveau compilable directement en microcodes. Ce compilateur peut être paramétré selon le nombre de PE de CRASY, ce qui représente un avantage incontestable.

3. Implantation d'un algorithme neuromimétique de séparation de sources

3.1. Présentation du problème

Cet algorithme a été mis au point au laboratoire de Traitement d'Images et de Reconnaissance de Formes de Grenoble par J. Herault, C. Jutten et B. Ans [2] pour simuler les fonctions d'analyse de données dans le système nerveux.

Considérons un réseau entièrement connecté, il reçoit sur ses entrées directes un mélange inconnu de grandeurs primitives indépendantes. Le réseau n'a accès qu'aux signaux issus de ce mélange inconnu: il ne connaît donc ni les signaux primitifs indépendants, ni les coefficients du mélange. Un exemple typique est l'analyse de l'environnement par un système multicapteurs où chaque capteur n'étant pas parfait, donne une image d'une grandeur principale superposée aux autres grandeurs mesurées.

Ce problème se formalise de la manière suivante: les données accessibles **e** sont des mélanges de grandeurs primitives **x**. Si l'on suppose que ce mélange est linéaire, pour un système à n composantes, on a:

$$e_i(t) = \sum_k m_{ik} \cdot x_k(t) \tag{5}$$

A partir de la connaissance seule des réalisations temporelles de $\mathbf{e}(t)$, le problème consiste à estimer les signaux indépendants $\mathbf{x}(t)$, en posant par hypothèse que la matrice **M** est non singulière.

3.2. Solution neuromimétique

Le vecteur $\mathbf{e}(t)$ est appliqué à un réseau de n opérateurs auto-adaptatifs en interconnexion complète.

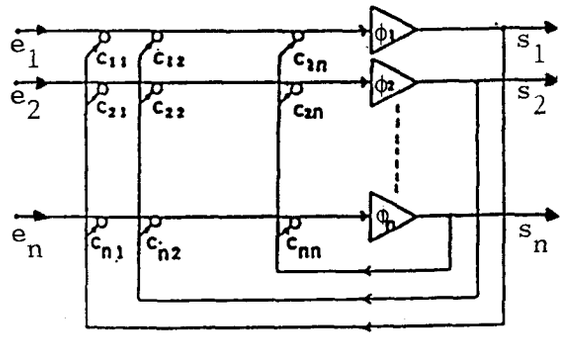


Fig. 10. – Réseau d'opérateurs en interconnexion complète.

Le réseau de la figure 10 est un réseau neuromimétique au sens où chaque opérateur ϕ_i reçoit une information externe e_i et une information interne $c_{ij} \cdot s_j$. Les coefficients c_{ij} sont les poids des j interconnexions analogues aux efficacités synaptiques d'un système nerveux. De plus l'évolution des coefficients c_{ij} n'est fonction que des données *locales* s_i et s_j.

Sur la figure 10, chaque triangle ϕ_i est un opérateur dont la sortie est s_i.

La ligne horizontale à gauche de l'opérateur est une ligne de sommation, elle réalise la somme: $e_i - \sum_j c_{ij} \cdot s_j$.

En utilisant la formulation matricielle de description du réseau, nous pouvons écrire (en supposant les opérateurs linéaires):

$$s_i = e_i - \sum_j c_{ij} \cdot s_j \quad \text{et} \quad c_{ii} = 0 \tag{6}$$

soit après relaxation $\mathbf{s} = \mathbf{e} - \mathbf{C} \cdot \mathbf{s}$, avec

$$\mathbf{e} = \mathbf{M} \cdot \mathbf{x} \quad \text{d'où} \quad \mathbf{s} = [\mathbf{I} + \mathbf{C}]^{-1} \cdot \mathbf{M} \cdot \mathbf{x} \tag{7}$$

Nous omettons, pour des raisons de simplicité d'écriture, la variable temporelle t. Les variables $\mathbf{x}(t)$ doivent rester stables durant un temps suffisant à la stabilisation des sorties du système bouclé. Cette dernière équation s'écrit à la condition que la matrice

$[\mathbf{I}+\mathbf{C}]$ soit régulière. Si l'on veut que les composantes de **x** soient isolées, chacune sur une composante de **s**, à un coefficient de proportionnalité près, il faut que le produit $[\mathbf{I}+\mathbf{C}]^{-1}.\mathbf{M}$ soit une matrice $n \times n$: **P**, qui ait un seul élément non nul par ligne et par colonne.

Par un algorithme adaptatif, il est possible de faire tendre la matrice des connexions récursives **C**, d'un état initial vers un état stabilisé tel que:

$$(8) \qquad \mathbf{P}=[\mathbf{I}+\mathbf{C}]^{-1}.\mathbf{M} \quad \text{et} \quad \mathbf{s}=\mathbf{P}.\mathbf{x}$$

La fonction de ce réseau est d'estimer les signaux primitifs indépendants à partir des réalisation des entrées externes et internes. Dans l'équation (6), la différence entre ces deux termes est la sortie du i-ième opérateur. Elle s'analyse comme étant l'erreur d'estimation des entrées externes **e** par les entrées internes **C.s**. Par hypothèse, un processus adaptatif doit tendre à minimiser cet estimateur. D'après l'équation (7) l'erreur obtenue sur cette estimation sera l'une des grandeurs x_k cherchée. Pour choisir une loi d'adaptation, on applique la méthode de la décroissance du gradient pour minimiser la moyenne quadratique de l'erreur d'estimation:

$$\frac{\partial (e_i - \sum c_{ij}.s_j)^2}{\partial c_{ij}}$$

Il en résulte une loi du type:

$$(9) \qquad dc_{ij}/dt = a.(s_i - \bar{s}_i).(s_j - \bar{s}_j)$$

Cette loi correspond à un test de non-corrélation des signaux. C'est un test d'indépendance uniquement si les signaux sont gaussiens.

La loi proposée étend la notion de test de covariance à celui de test d'indépendance en dissymétrisant les variations de dc_{ij} et de dc_{ji}:

$$(10) \qquad dc_{ij}/dt = f(s_i - \bar{s}_i).g(s_j - \hat{\bar{s}}_j)$$

avec f et g deux fonctions impaires.

Le développement de f et g en série de Taylor, donne pour dc_{ij}/dt, une somme de moments croisés d'ordre élevé; ces moments seront tous nuls quand s_i et s_j seront indépendants. Une loi simple, qui a donné de bons résultats, dans les simulations, est la suivante:

$$(11) \qquad dc_{ij}/dt = b.s_i^3.\mathrm{Arctg}[(s_j - \hat{\bar{s}}_j)]$$

où:

— $s_i - \bar{s}_i$, variable centrée, a été remplacée par s_i, sorties du réseau;

— la moyenne de s_j, $\bar{s}_j$, a été estimée par $\hat{\bar{s}}_j$ à l'aide d'un filtrage passe-bas du premier ordre: $s_j = s_j * e^{-t/\tau}$.

Le terme b est le gain d'adaptation. Avec un gain faible, les variations dc_{ij} seront petites et la vitesse de convergence sera réduite. Avec un gain plus fort, les coefficients se déplaceront rapidement vers l'état optimal, mais la boule de convergence aura un grand volume.

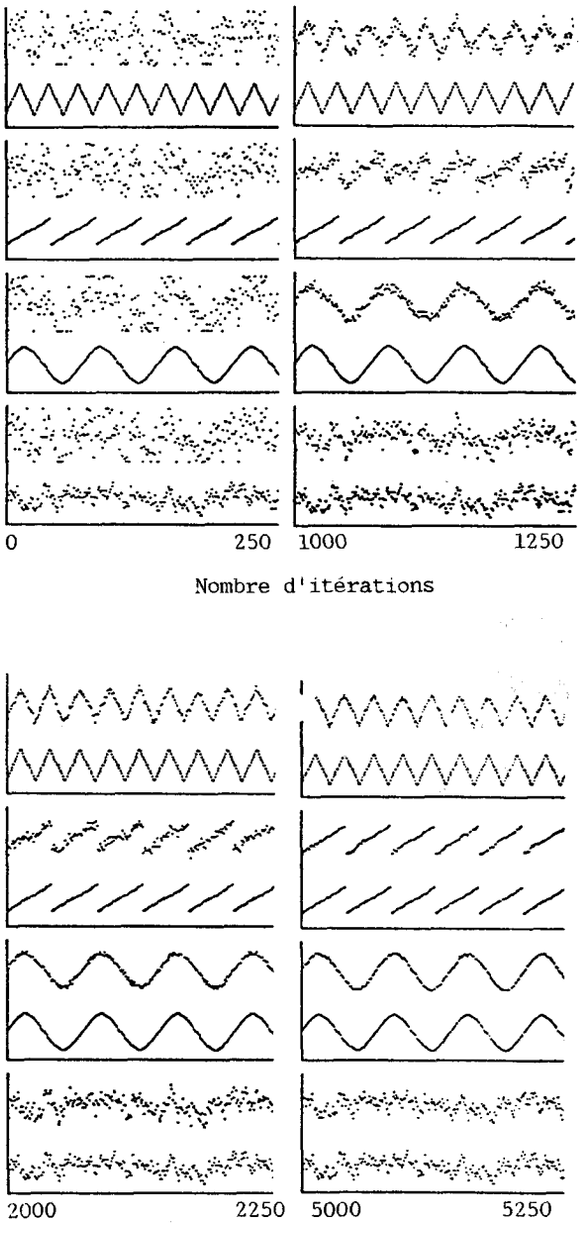


Fig. 11. — Activité de quatre cellules en cours d'apprentissage. Le signal de référence est le signal inférieur.

Enfin, il est possible de compléter la loi (11) en faisant intervenir un terme d'oubli, on a:

$$(12) \quad c_{ij}(t+1) = c_{ij}(t) - a.c_{ij}(t) + b.f(s_i).g(s_j - \hat{\bar{s}}_j).$$

Ce facteur d'oubli agit à la manière d'une force de rappel, appliquée au coefficient et dirigée vers l'origine. Le rôle et les effets des paramètres de la loi d'apprentissage choisie (11) sont développés dans les travaux de B. Biffi [10].

3.3. Résultats de simulations sur CRASY

Les simulations présentées mettent en évidence, qualitativement, la faculté du réseau, muni de cette loi d'apprentissage, à discriminer les signaux indépendants dans un mélange composite.

Le réseau étudié est formé de huit cellules entièrement connectées. Chacune des cellules reçoit sur l'entrée externe un mélange de huit signaux synthétisés, sinusoïde, dent de scie, carré, triangle, bruit filtré, bruit uniforme... A la figure 11, sont représentées l'évolution de quatre sorties à chaque pas d'échantillonnage, au cours de l'apprentissage. En référence, sous chaque de sortie, est représenté le signal primitif à extraire, vers lequel cette sortie doit tendre. La première séquence montre à partir de la première itération, le mélange initial des entrées primitives, la matrice **C** étant initialisée à 0. Les deuxième, troisième et quatrième séquences montrent respectivement l'état des sorties après 1 000, 2 000 et 5 000 itérations, c'est-à-dire ajustements élémentaires dc_{ij} des coefficients c_{ij}.

Ensuite, les coefficients c_{ij} restent en moyenne dans une boule de convergence autour de c_{ij}^*. L'apprentissage est toujours actif, et la matrice $[\mathbf{I}+\mathbf{C}]^{-1}$ est une estimation sans biais de la matrice solution **M**.

Comme l'apprentissage est permanent, si ce mélange dérive, l'état **C*** dérive et les coefficients c_{ij} suivent ce déplacement.

3.4. Temps d'exécution

Les temps mesurés pour l'exécution d'une itération sur CRASY (relaxation du réseau à 8 cellules et modification de 64 coefficients) est de 32 µs, soit 500 ns par actualisation. Avec un réseau de 128 cellules (c'est la capacité actuelle maximale de CRASY), la durée d'une itération serait environ de 10 ms. Il faut donc pouvoir alimenter le processeur à une cadence de 100 vecteurs de 128 composantes par seconde, soit environ 13 kmot/s.

Conclusions

La simulation numérique des réseaux neuromimétiques de grande taille impose l'utilisation d'outils de calcul adéquats. Les modèles de réseaux et de « neurones » étant très divers, le calculateur utilisé doit satisfaire à un compromis, entre une structure universelle et une structure dédiée. L'architecture modulaire de CRASY répond favorablement à cette alternative.

CRASY appartient à la famille des « array » processeurs. Mais ce n'est pas un calculateur universel, sa conception a été orientée vers l'analyse de flots de données. Une des conséquences importantes est l'extrême simplicité du réseau d'interconnexion entre les processeurs élémentaires. En corollaire avec ce réseau d'interconnexion très rudimentaire, réduit à une chaîne « cassable », le fait d'ajouter un processeur élémentaire (avec sa mémoire) ne modifie en rien la structure du réseau de PE et augmente la puissance de calcul.

De plus l'utilisation de processeurs élémentaires plus sophistiqués, comme des processeurs de signaux, augmenterait encore l'étendue du domaine d'applications.

Manuscrit reçu le 15 *décembre* 1987.

BIBLIOGRAPHIE

[1] A. Guerin, CRASY : un Calculateur de Réseaux Adaptatifs SYstolique. Application au calcul neuromimétique, *Thèse de Docteur-Ingénieur*, ENSER, Grenoble, juillet 1987.

[2] J. Herault, C. Jutten et B. Ans, Détection de grandeurs primitives dans un mélange composite par une architecture de calcul neuromimétique en apprentissage non supervisé, *GRETSI*, 20-24 mai 1985, p. 1017-1022.

[3] C. Jutten, Calcul neuromimétique et traitement du signal : analyse en composantes indépendantes, *Thèse de Docteur ès Sciences*, INP, Grenoble, juin 1987.

[4] M. J. Flynn, Some computer organization and their effectiveness, *IEEE Trans. on Comp.*, C21, n° 9, septembre 1972, p. 948-960.

[5] J. P. Riganati et P. B. Schneck, Supercomputing, *Computer*, octobre 1984, p. 97-112.

[6] J. J. Hopfield et D. W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetcis*, 52, 1985, p. 141-152.

[7] T. Kohonen, *Self organization and Associative memory*, Springer Verlag, 1984.

[8] R. B. Urqhart et D. Wood, Systolic matrix and vector multiplication methods for signal processing, *IEE Proceedings*, 131, n° 6, octobre 1984, p. 623-631.

[9] T. F. Kadela et J. H. Graham, VLSI Architectures for optimal state estimation, *IEEE Trans. on Comp.*, C34, n° 11, novembre 1985.

[10] B. Biffi, Séparation de sources par un réseau auto-adaptatif de neurones formels, *DEA Électronique*, ENSER, Grenoble, septembre 1985.