SIRENE-F : système pour la construction et l'exécution de réseaux neuro=flous en vue d'applications en temps réel

Christophe Wolinski, Pierre-Yves Glorennec

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

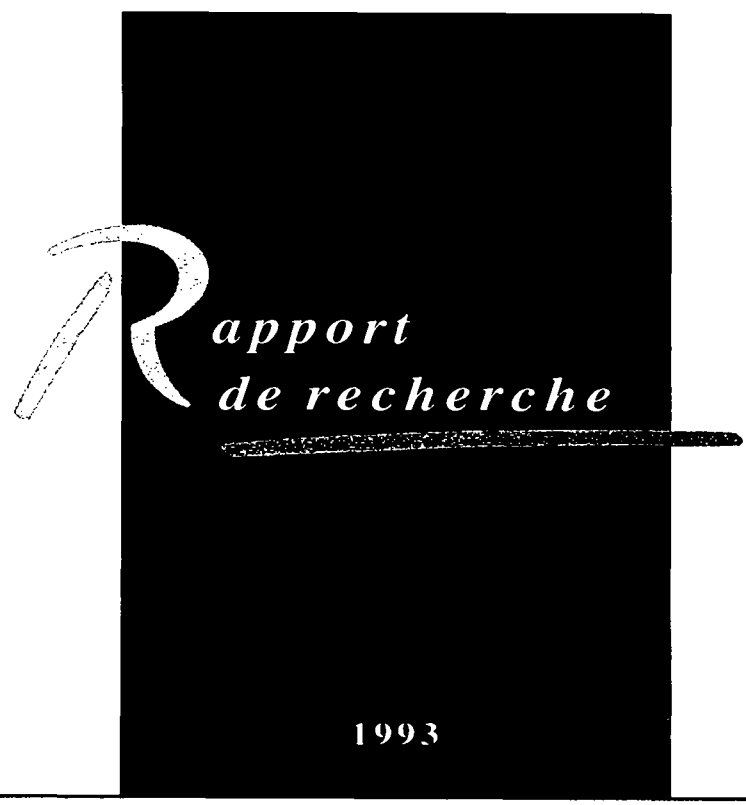
SIRENE-F : système pour la construction et l'exécution de réseaux neuro-flous en vue d'applications en temps réel

Krzysztof WOLINSKI
Pierre-Yves GLORENNEC

N° 1963
Juillet 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués



IRISA

INSTITUT DE RECHERCHE
EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 RENNES CEDEX FRANCE

SIRENE-F : système pour la construction et l'exécution de Réseaux Neuro-Flous en vue d'applications en temps réel.

Publication Interne n°705 - Février 1993 - Février 1993 - 14 pages

Krzysztof Wolinski(*), Pierre-Yves Glorennec(**)
(*) IFSIC, (**) INSA
wolinski@irisa.fr
glorenne@irisa.fr

résumé

Un environnement matériel et logiciel intégré est proposé pour des réseaux neuro-flous. Cet environnement permet le développement et la conduite d'applications à partir d'un compatible PC. L'utilisation de réseaux neuro-flous permet, de plus, de diminuer le temps d'apprentissage par la possibilité d'introduire de la connaissance a priori dans les poids synaptiques.

mots clés

réseaux neuronaux, logique floue, environnement intégré, carte accélératrice.

SIRENE-F : Building and Running Neuro-Fuzzy Networks for Real Time Oriented Applications

abstract

A software and hardware integrated environment is proposed for neuro-fuzzy networks. This tool can be used for developing and driving an industrial application from a Personal Computer. Moreover, with neuro-fuzzy networks, the learning process is greatly reduced thanks to the possibility to introduce prior knowledge into the synaptic weights.

keywords

neural networks, fuzzy logic, integrated environment, accelerating board.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(U.R.A. 227) UNIVERSITÉ DE RENNES I I.N.S.A. DE RENNES

INRIA
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE
ET EN AUTOMATIQUE - UNITÉ DE RECHERCHE DE RENNES

1. Introduction

Dans certaines applications, comme l'identification ou la commande de processus, l'intégration de la connaissance a priori permet de simplifier considérablement la phase de mise au point. Cette connaissance est souvent exprimée sous une forme qualitative, de manière imprécise ou incomplète. Pour l'affiner, il faut interroger des experts et/ou en faire des mesures complémentaires.

L'intégration, dans une base de règles, de données linguistiques et numériques peut être alors réalisée par une approche neuro-floue : le formalisme de la logique floue permet en effet de manipuler des prédicats vagues (par exemple : la température est élevée), tandis que des techniques d'apprentissage inspirées du connexionnisme permettent l'extraction automatique de connaissances à partir de données numériques. Cependant, à la différence des réseaux neuronaux, la connaissance acquise à la fin de l'apprentissage reste toujours interprétable, sous la forme de règles floues, en langage naturel.

Par rapport aux autres approches, les réseaux neuro-flous (RNF) présentent les avantages suivants :

- rapidité et facilité de mise en œuvre,
- robustesse,
- apprentissage en ligne,
- parallélisme important,
- implémentation possible sur des circuits VLSI neuronaux ou des processeurs performants permettant en particulier des traitements en temps réel.

Le système SIRENE-F a été développé pour exploiter ces différents avantages et pour créer facilement des applications basées sur un système de règles floues. Il offre un environnement matériel et logiciel pour la construction et l'exécution d'un RNF en vue d'applications en temps réel, comprenant :

- une interface graphique permettant de créer des règles floues et d'interpréter celles qui sont issues de l'apprentissage,
- une bibliothèque logicielle permetant en particulier la création et l'apprentissage de réseaux neuro-flous,
- une carte accélératrice avec un microprocesseur performant.

Le choix du processeur s'est porté sur un processeur de traitement du signal, le DSP TMS320C30, pour les raisons suivantes : vitesse et précision des calculs, arithmétique en virgule flottante, jeux d'instructions permettant de paralléliser l'exécution des algorithmes flous, traitement de signaux numériques en temps réel (filtrage, FFT), souplesse d'utilisation.

Enfin, le tout est conçu pour pouvoir être utilisé sur un ordinateur hôte des plus courants.

Les RNF sont présentés au paragraphe 2 et le système SIRENE-F au paragraphe 3. Nous détaillons la carte accélératrice au paragraphe 4. Enfin nous concluons par des perspectives de travail.

2. Les Réseaux Neuro-Flous.

Un RNF est fondamentalement un système basé sur des règles floues, qui utilise le formalisme et les méthodes du connexionnisme pour l'extraction des connaissances. Une règle floue s'exprime par une relation du type :

si situation **alors** action

L'élaboration des règles comporte donc deux aspects :

- comment apprécier une situation ? Pour cela, le domaine de chaque variable d'entrée est découpé en sous ensembles flous (e.g. "grand", "assez chaud"...), de manière plus ou moins intuitive. Le choix de ce découpage influe sur la performance globale du système.
- que faire dans telle situation ? Il s'agit alors de déterminer l'"action" correspondante.

Dans le cas général, la réponse à ces deux questions n'est pas toujours évidente, car il n'existe pas de méthode systématique pour le choix des fonctions d'appartenance qui apparaissent dans les prémisses et les conclusions des règles. L'intérêt des RNF est alors d'introduire de façon naturelles des méthodes d'apprentissage permettant de mélanger des données symboliques et numériques.

Dans les différents réseaux neuro-flous déja proposés, les auteurs ont utilisé une analogie très forte entre un réseau neuronal multicouche et un système de commande floue. Comme on le voit sur la figure 1, le calcul de la sortie du système pour une entrée donnée passe par trois étapes que nous allons détailler en adoptant la terminologie usuelle des réseaux neuronaux.

1- La première couche cachée calcule les degrés d'appartenance de chaque entrée à différents sous ensembles flous définis sur les domaines de variation. On pose : $\mu_A(x)$ = degré d'appartenance de x au sous ensemble flou A. Les paramètres de cette couche définissent les fonctions d'appartenance $x \rightarrow \mu_A(x)$.

2- La deuxième couche cachée évalue en parallèle les prémisses des règles : α_i = ET(x est A_i , y est B_i , ...), où A_i , B_i , ... sont des sous ensembles flous. Les paramètres de cette couche sont fixes et caractérisent l'opérateur ET choisi.

3- La couche de sortie évalue les conclusions de toutes les règles, fait l'aggrégation (i.e. réalise un OU entre les règles) et donne la réponse du système à une entrée donnée.

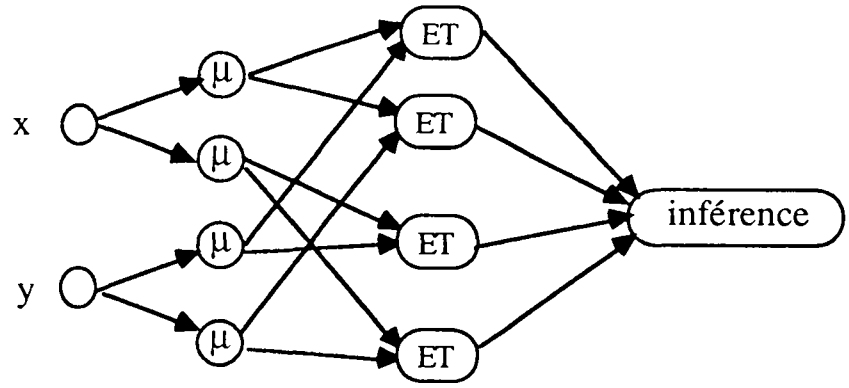


fig. 1 les trois étapes pour l'inférence floue

Selon le choix des fonctions d'appartenance, de l'opérateur logique ET figurant dans les prémisses, et du mode de raisonnement flou, l'analogie formelle avec un réseau neuronal sera plus ou moins poussée. Le réseau neuro-flou évolutif, proposé dans [4], permet l'implémentation totale d'un système de règles floues dans un réseau neuronal ordinaire. Les choix évoqués précédemment sont les suivants, cf. figure 2 :

1- fonctions d'appartenance gaussiennes : $x \rightarrow \exp[-(ax + b)^2]$. On peut aussi prendre la différence de deux sigmoïdes.

2- le ET logique est une version lissée (sigmoïdale) du ET de Lukasiewicz : $(u,v) \rightarrow \{1 + \exp[-a(u + v - 1.5)]\}^{-1}$.

3- le raisonnement flou utilisé correspond à la méthode de Sugeno, dans laquelle les règles sont de la forme (pour un système à deux entrées et une sortie pour simplifier) :

règle n° i : si x est A_i et y est B_i alors z est w_i

où A_i et B_i sont des sous ensembles flous mais w_i une valeur réelle.

Dans ce cas, la conclusion de l'ensemble des règles est la somme des w_i pondérés par la valeur de vérité relative de chaque règle :

$$(1) \quad z = \frac{\sum \alpha_i \, w_i}{\sum \alpha_i}$$

Les coefficients w_i peuvent être considérés comme des poids synaptiques et le numérateur et le dénominateur de (1) sont alors les sorties de deux neurones linéaires.

Avec ces hypothèses, **tous les paramètres d'un système flou sont des poids ajustables d'un réseau neuronal ordinaire** (où les neurones transmettent en aval une fonction de la somme pondérée de leurs entrées). De plus, ce réseau initial peut évoluer de façon continue vers un réseau neuronal ordinaire, en créant dynamiquement des liaisons entre tous les neurones des couches cachées 1 et 2, cf [5].

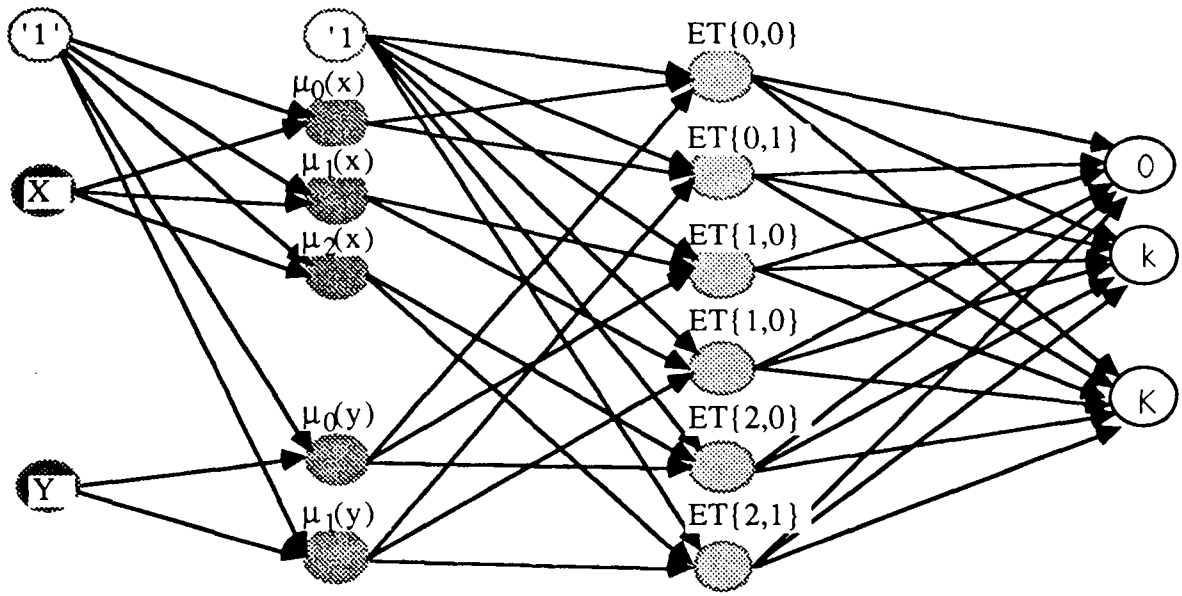


fig. 2 implémentation neuronale de règles floues

L'avantage majeur de cette présentation est que tous les résultats et propriétés des réseaux multicouches non récurrents sont directement applicables.

3. Présentation de SIRENE-F

3.1. Généralités

Une vue globale du système SIRENE-F est présentée en figure 3.

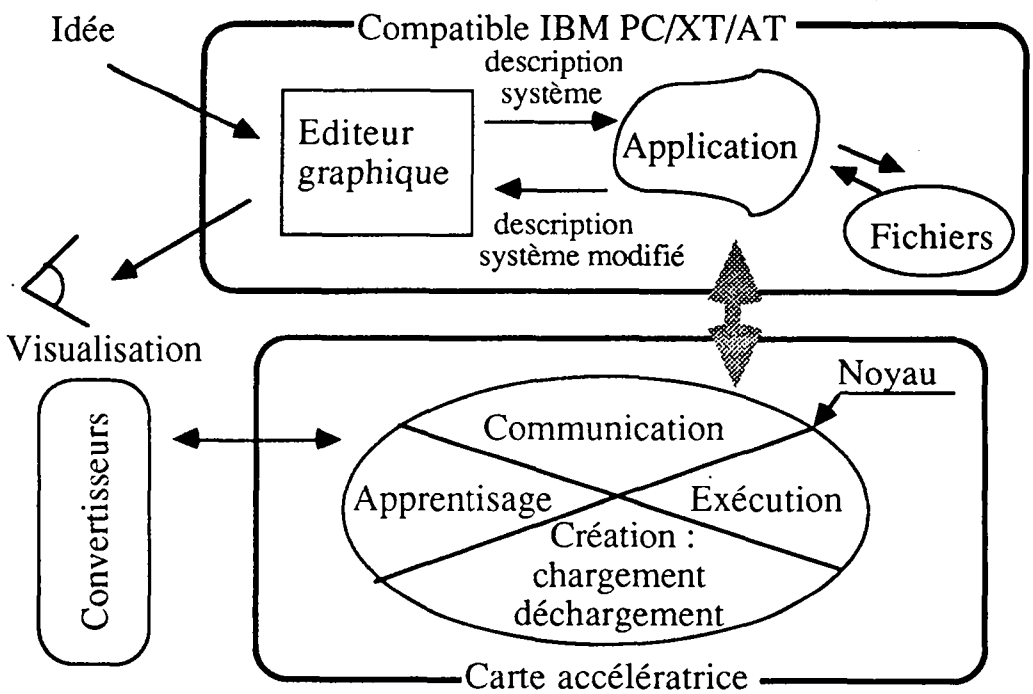


fig. 3 synoptique de SIRENE-F

La construction d'une application utilisant un RNF commence toujours par la définition d'un système de règles : il faut au minimum spécifier le nombre de variables d'entrée, leurs domaines de variation et le nombre de fonctions d'appartenance par entrée. Cette tâche est supervisée par un éditeur graphique travaillant sur la machine hôte sous environnement Windows-3. Le fichier textuel résultant contient les informations suivantes :

- le nombre d'entrées,
- le nombre de sorties,
- le nombre de fonctions d'appartenance pour chaque entrée et chaque sortie,
- les paramètres qui caractérisent les fonctions d'appartenance,
- les règles.

Le fichier ainsi obtenu est ensuite transformé par le traducteur en un fichier de paramètres qui spécifie complètement le RNF correspondant.

A partir de cet instant, il est possible de créer physiquement un RNF en utilisant des commandes faisant partie d'une bibliothèque (voir figure 4). Une fois le réseau créé sur la carte accélératrice, l'application peut utiliser d'autres commandes comme :

- le chargement d'un réseau
- les calculs
- l'apprentissage par l'exemple,
- le déchargement du réseau

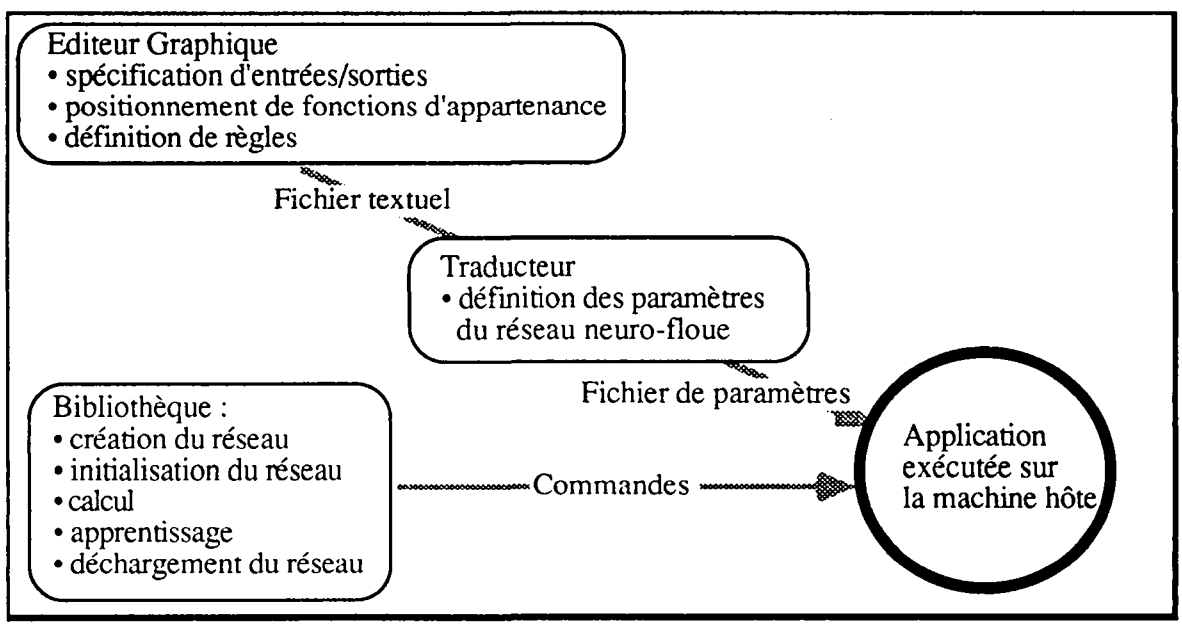


fig. 4 création d'une application

Il est possible de créer dynamiquement plusieurs RNF.Après avoir créé un réseau, le noyau rend un identificateur à l'application, ce qui permet d'accéder à un réseau particulier. Le nombre de réseaux qui cohabitent dépend de la taille de la mémoire disponible dans le système. Chaque réseau est considéré comme un processus, fig. 5.

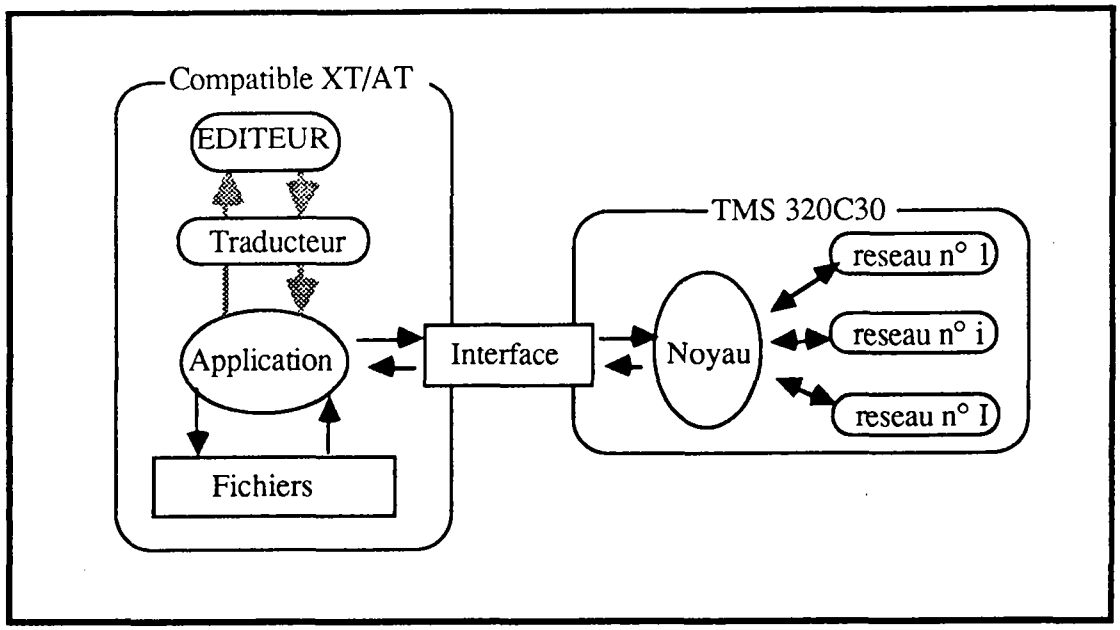


fig. 5 création de plusieurs RNF.

3.2. L'éditeur graphique

L'éditeur graphique permet de définir un système à base de règles floues de façon supervisée. Pour chaque entrée ou sortie, il faut préciser le nombre, la répartition et la forme des fonctions d'appartenance. Ce choix est fait a priori par l'utilisateur. Les fonctions d'appartenance seront éventuellement modifiées pendant l'apprentissage.

L'utilisateur peut aussi donner, sous une forme symbolique, des relations entre variables d'entrées et de sorties.

3.3. Le traducteur

Ce logiciel traduit le fichier textuel produit par l'éditeur graphique en un fichier contenant les paramètres d'un RNF. Il complète la base de règles floues en mettant à zéro les parties conséquence de toutes les règles qui n'ont pas été définies lors de l'étape précédente.

Réciproquement, à l'issue de l'apprentissage, le traducteur permet de visualiser le jeu de règles obtenu en envoyant les paramètres correspondant à l'éditeur graphique.

3.4. Le noyau

Le noyau du système utilise les fonctions de la bibliothèque. Il traduit les commandes venant de l'application par des actions effectuées directement par le processeur de la carte accélératrice. Les actions sont resynchronisées dans le temps pour bien adapter :

- la vitesse du traitement de la partie de l'application qui tourne sur la machine hôte,
- à la vitesse d'exécution des calculs sur la carte accélératrice.

Toute émise par l'intermédiaire de la machine hôte est confirmée par le noyau.

3.5. Les processus réseau

Chaque processus réseau est composé de données comme :

- un descripteur du réseau,
- les poids synaptiques,
- les entrées,
- les sorties des neurones,
- le code commun.

Ces données sont allouées dynamiquement au fur et à mesure de la création des réseaux. La taille de la mémoire exigée pour chaque nouveau réseau ne dépend que de son architecture, d'où l'intérêt de compacter au maximum les données.

4- La carte accélératrice

4-1 présentation

Cette carte a été conçue autour d'un microprocesseur de traitement du signal (DSP), le TMS 320C30, cadencé par un oscillateur à 25 Mhz. Le processeur est connecté par son bus primaire à une mémoire statique RAM 32 K x 32 bits, avec un temps d'accès de 25 ns et à une mémoire EPROM de 32 K x 32 bits.

La mémoire morte contient le chargeur qui permet de télécharger et exécuter un programme d'application.

La communication entre le processeur et la machine hôte est assurée par une interface parallèle réalisée en technologie Xilinx. La synchronisation avec la machine hôte se fait par attente active, tandis que la synchronisation vers le processeur se fait par interruption.

Le processeur peut être relié au monde extérieur, par l'intermédiaire de convertisseurs analogique/numérique et numérique/analogique, soit par sa liaison série soit par son bus secondaire.

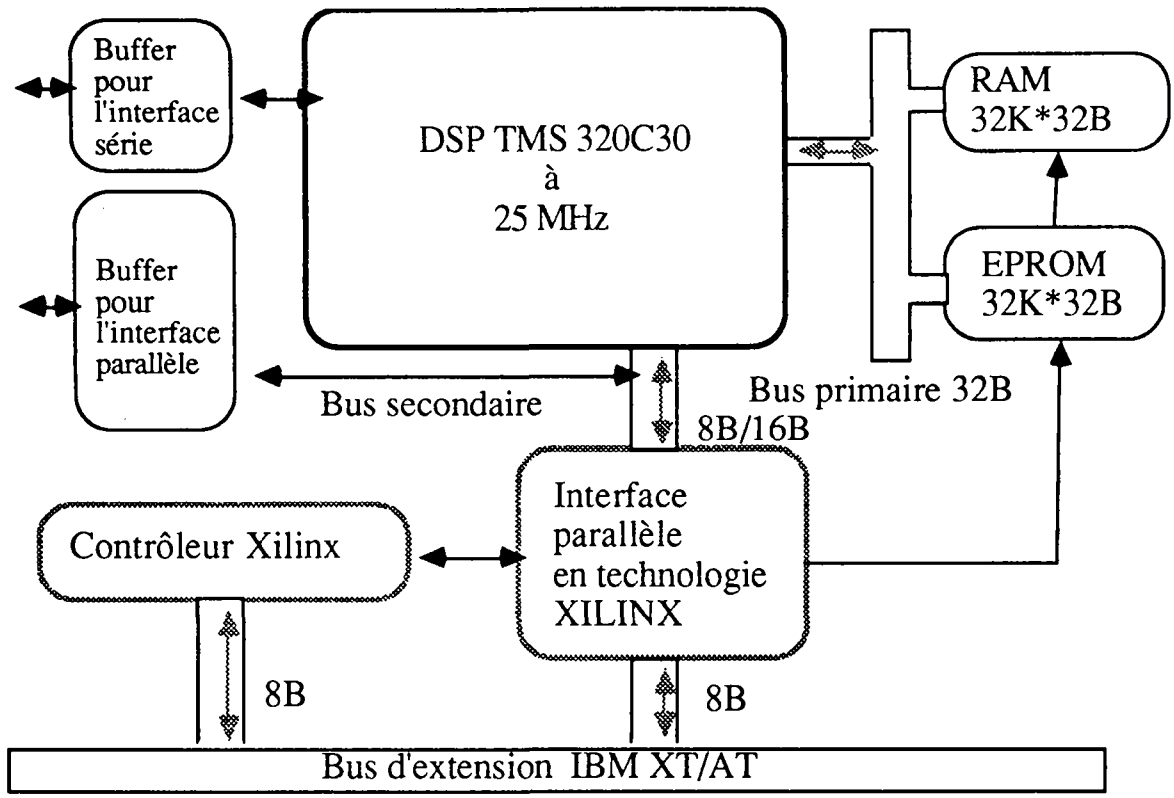


fig. 6 schéma de la carte accélératrice.

4-2 réduction de la taille mémoire

Le seul endroit où il est possible de réduire la taille des matrices synaptiques se situe entre les couches cachées 1 et 2. En effet, nous avons là des connexions incomplètes, dans la mesure où chaque prémisse de règle comprend I prédicats (I étant le nombre d'entrées). Pour un système à deux entrées, avec trois sous ensembles flous, notés a, b et c, pour l'entrées 1 et deux, notés d et e, pour l'entrée 2, il y a un nombre maximum de six prémisses, du type :

(1) x est a et y est d
(2) x est a et y est e
...
(6) x est c et y est e

La matrice synaptique correspondante est donc incomplète, comme le montre les figures 2 et 7, et peut donc être réduite en une matrice de taille I*J où J est le nombre de règles.

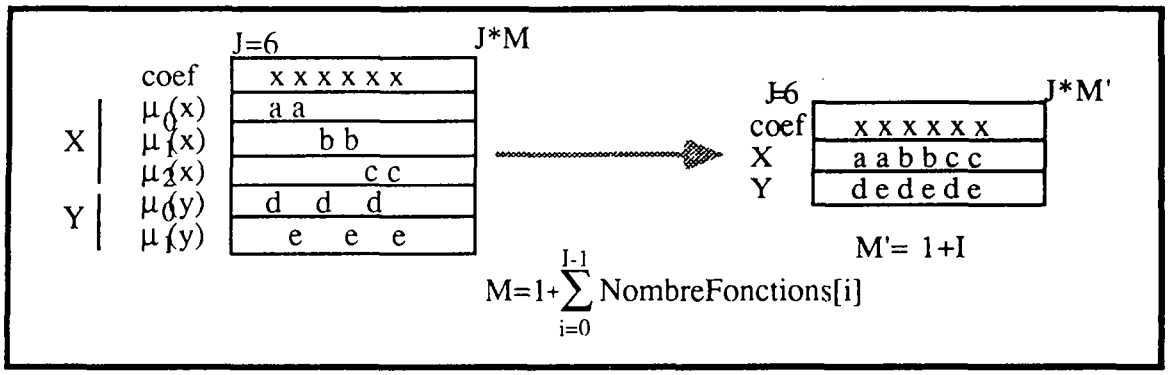


fig. 7 compactage des données : liens entre les couches cachées 1 et 2

En tenant compte cette remarque, la taille totale nécessaire pour un réseau donné est exprimée par la formule suivante :

$$\text{taille(réseau)}:=6*(I+1)+3*K+J*(K+I+5)+I+6*\sum_{i=0}^{I-1} \text{Nombre Fonctions}[i]$$

avec

I : nombre d'entrées
K : nombre de sorties
J : nombre de règles
NombreFonction[i] : nombre de sous ensembles flous pour l'entrée i

4-3 exploitations des possibilités du TMS

Le processeur TMS 320C30 est capable d'effectuer en parallèle des opérations de type addition et multiplication en virgule flottante ainsi que des opérations sur des pointeurs. La figure 8 présente en particulier les calculs pour la partie conséquence des règles.

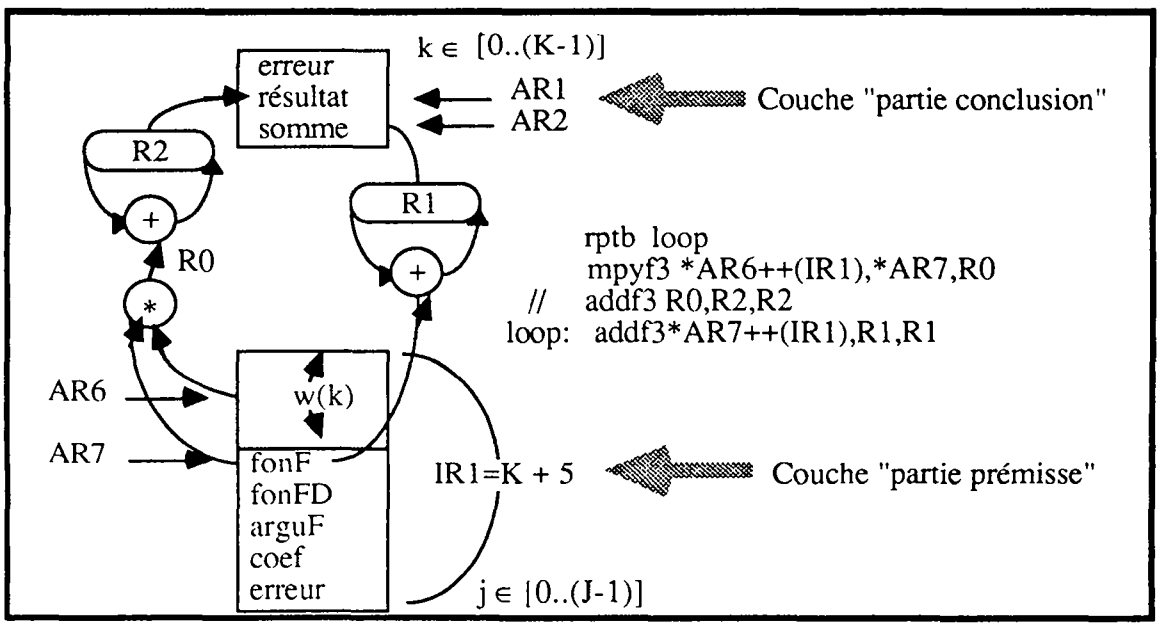


fig. 8 boucle principale pour le numérateur et le dénominateur de la formule (1)

Le code assembleur montre le calcul en parallèle, dans les registres R1 et R2, du numérateur et du numérateur de l'équation (1).

4-4 Performances

Les performances obtenues sont présentées en fig. 9. Les calculs sont faits sur 32 bits en virgule flottante, ce qui permet une bonne précision. Le système est ainsi capable d'évaluer 55 000 règles par seconde. A titre de comparaison, la référence [3] présente un contrôleur flou basé sur un autre DSP, qui peut évaluer entre 62 000 et 85 000 règles par seconde, selon le mécanisme d'inférence choisi (min-max ou prod-max). L'avantage de SIRENE-F est la possibilité de réaliser un apprentissage supervisé et donc d'optimiser les différents poids en ligne.

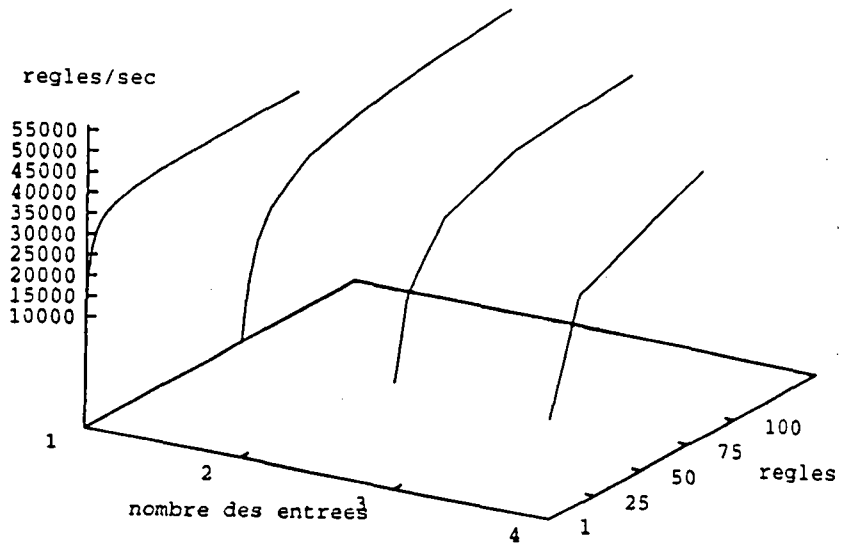


fig. 9 performances en 32 bits, virgule flottante.

5- Perspectives

Le travail s'oriente actuellement dans deux directions. La première consiste à rendre le RNF universel, pouvant utiliser toute sorte de fonctions d'appartenance et un plus grand choix de mécanisme d'inférence (la conjonction pourra être le Min, le produit ou toute autre T-norme). Les algorithmes d'apprentissage devront correspondre à ces modifications.

Une deuxième direction consiste évidemment à traiter des applications de type industriel, nécessitant de travailler en temps réel. Nous envisageons des applications en traitement du signal, pour exploiter au mieux les possibilités du DSP.

6- Conclusion

Nous avons présenté un système intégré permettant le développement et l'exécution d'applications utilisant une approche neuro-floue. Le cœur du système est une carte accélératrice utilisant un processeur de traitement du signal performant. Les temps de calculs sont suffisamment courts pour

permettre des applications de commande en temps réel, avec ou sans apprentissage en ligne, avec une bonne précision.

Ce type de carte accélératrice, utilisant comme hôte des ordinateurs très courants, doit pouvoir être utilisé facilement dans des applications industrielles.

7- références

[1] H. Berenji et P. Khedkar "Fuzzy rules for reinforcement learning", Proc. of IPMU'92, Palma de Majorque, juillet 92.

[2] Brown et Harris "A nonlinear adaptive controller", IMA J. of Math. Control, 1991.

[3] I. del Campo, J. Gonzalez, J. Garitagoita and J. Tarela, "A DSP-based fuzzy logic controller", Proc of 2th Int. Conf. on Fuzzy Logic and Neural Networks, Iizuka, Japon, juillet 92.

[4] P.Y. Glorennec "Un réseau neuro-flou évolutif", Actes de Neuro-Nîmes'91, novembre 91

[5] P.Y. Glorennec "A Neuro-Fuzzy Network designed for implementation on a neural chip", Proc. of Iizuka'92, juillet 92.

[6] Hayashi et Al. "Fuzzy neural controller", Proc. of FUZZ-IEEE'92, San Diego, mars 92.

[7] Horikawa et Al. "A fuzzy controller using neural network", Proc. of Iizuka'92, Iizuka, Japon, juillet 90.

[8] J-S.R. Jang "Rule extraction by generalized neural network", Proc. of 4th IFSA Congress, Bruxelles, juillet 91.

[9] Lin et Lee, "NN-based fuzzy logic control and decision systems" IEEE Trans. on Computers, Vol. 20, n° 12, déc. 91.

[10] Nomura et Al. "A self-tuning methos of fuzzy control", Proc. of 4th IFSA Congress, Bruxelles, juillet 91.

[11] H. Takagi, "Fusion technology of fuzzy theory and neural networks : survey and future directions", Proc. of Iizuka'90, Iizuka, Japon, juil 90.

[12] TMS 320C30, "User's Guide", Texas Instruments

[13] Xilinx "The programmable gate array Data Book"

[14] L.X. Wang, "Fuzzy systems are universal approximators", Proc. of FUZZ-IEEE'92, San Diego, mars 92.

[15] Windows 3, Microsoft.

[16] Krzysztof Wolinski, Pierre Yves Glorennec, Yves L'Azou, David Le Corre, Annick Orer et Pascal Rabier, " Environnement matériel et logiciel pour réseaux neuro-flous", Actes de Neuro-Nîmes, nov. 92.

Unité de Recherche INRIA Rennes
IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399