RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Application des réseaux de neurones à l'ordonnancement de tâches temps réel sur une architecture multiprocesseurs hétérogènes

Imène Benkermi, Sébastien Pillement, Olivier Sentieys

ENSSAT – Université de Rennes I,
IRISA,
6, rue de Kerampont - 22300 Lannion - France
prenom.nom@enssat.fr

Résumé
Ce papier présente les premières réflexions sur l'extension de l'utilisation des réseaux de neurones dans l'ordonnancement de tâches temps réel, à des architectures multiprocesseurs hétérogènes. Ainsi, une nouvelle règle de construction de réseau est proposée. Celle-ci va permettre au réseau de se stabiliser sur un état qui tient compte de l'hétérogénéité des processeurs tout en respectant les contraintes imposées aux tâches.

Mots-clés : ordonnancement temps réel, réseaux de neurones, modèle de Hopfield, multiprocesseurs hétérogènes.

1. Introduction

De nombreux algorithmes d'ordonnancement ont été développés pour la satisfaction des contraintes dans les systèmes temps réel, la plupart considère un ensemble spécifique et homogène de contraintes. Ceux-ci peuvent, par exemple, prendre en compte des tâches périodiques, sporadiques ou apériodiques, acceptant ou non la préemption, dans une architecture mono ou multiprocesseurs, mais ils combinent rarement ces contraintes. L'étude des différents algorithmes d'ordonnancement et leur classification montrent que ces derniers sont incapables de fournir des solutions optimales dans un temps polynomial si on prend en compte toutes les contraintes qu'on peut rencontrer dans les applications temps réel [1]. Ceci est encore plus complexe à obtenir lorsque le calcul de l'ordonnancement est à effectuer en-ligne. Ce qui a encouragé l'utilisation de techniques approchées, telles que les algorithmes génétiques, le recuit simulé, les réseaux de neurones, etc.

Par ailleurs, l'augmentation de la complexité des algorithmes de traitement du signal, de l'image et du contrôle, notamment, dans les applications embarquées, nécessite une grande puissance de calcul pour satisfaire les contraintes temps réel. Ce qui a conduit les concepteurs à s'orienter de plus en plus vers des architectures matérielles composées de plusieurs processeurs opérant en parallèle et pouvant présenter des capacités de calcul différentes. C'est le cas des architectures matérielles développées actuellement pour le domaine des télécommunications de troisième génération [5]. Celles-ci sont caractérisées par l'intégration de plusieurs composants (processeurs, accélérateurs et unités reconfigurables) sur une même puce dans le but d'accélérer l'exécution de ces applications sur les unités de traitement spécifiques ou généralistes, selon les besoins. L'émergence de ce type d'architectures nécessite l'emploi d'outils et de mécanismes qui leur sont spécifiques. Ainsi, des travaux commencent à voir le jour notamment, sur l'ordonnancement de tâches à travers des processeurs différents [8, 13].

D'autre part, étant donné le caractère souvent imprévisible dans les systèmes temps réel (arrivée d'une tâche apériodique, par exemple), les algorithmes en-ligne sont souvent utilisés. Ils permettent ainsi de lever la rigidité constatée au niveau des algorithmes hors-ligne qui nécessitent une connaissance a priori de tous les paramètres des tâches du système [4]. Cependant, en raison du surcoût qui peut être engendré par l'exécution d'un algorithme en-ligne, il est nécessaire que sa complexité soit "acceptable" pour être utilisé.

Nous considérons dans nos travaux un algorithme d'ordonnancement en-ligne sur une architecture multiprocesseurs hétérogènes (pouvant présenter différentes puissances de calcul pour une même tâche). Pour ceci, nous proposons d'étendre les travaux de Cardeira et Mammeri [1, 2] sur l'application des réseaux de neurones, en particulier le modèle de Hopfield [9], pour l'ordonnancement de tâches temps réel. Dans leurs travaux, ces auteurs s'appuient sur les résultats obtenus par Tagliarini et *al.* [12] qui construisent une base théorique pour la conception de réseaux de neurones pour des problèmes d'optimisation. L'apport majeur de Tagliarini et *al.* réside dans la proposition d'une règle qui facilite la construction d'équations d'évolution dans le temps du comportement du réseau. Cette règle spécifie les poids des connexions et les entrées externes à appliquer aux neurones pour forcer aussi bien des contraintes d'égalité que des contraintes d'inégalités.
Ainsi, Cardeira et Mammeri démontrent dans leurs travaux l'efficacité de l'application des réseaux de neurones à l'ordonnancement en-ligne de tâches temps réel en prenant en compte des contraintes diversifiées. La diversité des contraintes est rendue possible par le caractère additif du réseau de Hopfield. Les résultats de l'utilisation de cette approche pour l'ordonnancement temps réel ont montré son efficacité étant donné qu'elle est progressive et qu'elle offre une vitesse de convergence remarquable, ce qui rend attrayante son utilisation en-ligne.

Dans cet article, nous commençons d'abord par présenter le modèle de Hopfield et les principaux résultats qui lui sont liés, notamment la règle de construction de Tagliarini et *al.* Nous présenterons par la suite l'application de ce modèle à l'ordonnancement de tâches temps réel. Enfin, nous expliquons nos travaux permettant la prise en compte de l'hétérogénéité des processeurs dans l'architecture considérée.

2. Travaux existants

2.1. Modèle de Hopfield

Cohen et Grossberg [3] ont montré l'existence d'une fonction de Lyapunov pour certains modèles de réseaux de neurones qui évolueront vers des états stables si certaines contraintes sont respectées : - la matrice représentant les poids des connexions entre les neurones est non positive et est symétrique, - les éléments de la diagonale sont nuls, - et la fonction d'activation est toujours croissante. L'idée centrale est que le système converge toujours de manière à ne pas augmenter la valeur de la fonction de Lyapunov.
Le modèle de Hopfield [10] est l'un des modèles considérés par leur étude. Ainsi, Hopfield a dérivé, à partir des résultats de Cohen et Grossberg, une fonction de Lyapunov spécifique, appelée fonction d'énergie de telle façon à ce que le réseau de neurones évolue vers un état stable sans l'incrémenter. Cette fonction est exprimée comme suit

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij}x_i x_j - \sum_{i=1}^{N} I_i x_i \qquad (1)$$

où : T_{ij} = Poids de la connexion entre les neurones i et j
x_i = Etat du neurone i
I_i = Entrée extérieure appliquée au neurone i

L'idée de Hopfield est que, puisqu'un réseau neuronal va chercher à minimiser la fonction d'énergie, on pourra construire un réseau pour la minimisation de cette fonction en associant les variables du problème d'optimisation aux variables de cette fonction.
Pour ceci, une démarche systématique pour la résolution des problèmes d'optimisation en général et des problèmes d'ordonnancement en particulier, par les réseaux de neurones, a été proposée et consiste à résoudre successivement les trois sous problèmes suivants [10] :
– trouver un schéma de représentation (topologie du réseau) dans lequel les sorties des neurones sont interprétées comme solution au problème ;
– construire une fonction d'énergie du problème dont le minimum correspond à la meilleure solution (les conditions de convergence de Cohen et Grossberg doivent bien sûr être vérifiées) ;
– déduire les poids des connexions entre les neurones et les entrées à appliquer à chaque neurone à partir de la fonction d'énergie trouvée.

Le réseau de neurones correspondant au problème d'optimisation étant trouvé, il suffira de le laisser évoluer jusqu'à stabilisation.

2.2. Règle de construction *k-de-N*

La règle de construction *k-de-N* proposée par Tagliarini et *al.* dans [12] permet de construire un réseau de N neurones dont l'évolution converge, à partir d'un état initial aléatoire, vers un état stable avec exactement "k **neurones actifs** (sortie égale à 1) parmi N".
La fonction coût (fonction qu'il faut minimiser pour résoudre le problème d'optimisation) suivante

$$E = (k - \sum_{i=1}^{N} x_i)^2 \tag{2}$$

va être minimale (égale à 0) lorsque la somme des neurones à l'état 1 est égale à k, et elle est strictement positive dans tous les autres cas.
Après transformations mathématiques sur cette fonction, une fonction équivalente à (1) est obtenue :

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} (-2) x_i x_j - \sum_{i=1}^{N} (2k-1) x_i \tag{3}$$

Ainsi, les paramètres suivants sont obtenus.

$$T_{ij} = \begin{cases} 0 & \text{si } i = j \\ -2 & \text{si } i \neq j \end{cases}$$
$$I_i = 2k - 1 \qquad \forall i$$

Le réseau correspondant à ces résultats respecte tous les critères de convergence, et peut donc être utilisé.

La règle *k-de-N* peut s'appliquer à plusieurs ensembles de neurones à la fois pour la prise en compte de différents types de contraintes d'optimisation. Les nouveaux poids et entrées imposés par chaque contrainte peuvent être directement additionnés à ceux déjà existants (puisque le modèle est additif) et le réseau évoluera vers un état satisfaisant toutes les contraintes simultanément (si un tel état existe) [10]. Dans ce cas, la fonction d'énergie vaut zéro. Elle est différente de zéro si le réseau se bloque sur un minimum local de cette fonction, ou encore s'il n'y a pas de solution pour le problème.
Il a été également prouvé que la règle *k-de-N* peut s'appliquer à la résolution des problèmes du type "au moins 1 et au plus k neurones parmi N" [10], et ce en ajoutant k-1 neurones aux N neurones considérés et en appliquant la règle à l'ensemble des neurones, c.à.d., aux N+k-1 neurones. Le nombre de neurones actifs, après stabilisation, variera entre 1 et k. Les neurones additionnels sont appelés neurones cachés (*slack neurons*), puisqu'ils seront ignorés lors de l'interprétation des résultats.

2.3. Application à l'ordonnancement de tâches temps réel

Cardeira et Mammeri [1, 2] montrent, à partir des résultats de Hopfield, qu'en effectuant un bon choix de la topologie du réseau de Hopfield et en calculant la fonction d'énergie correspondant aux contraintes des modèles matériel et logiciel, le réseau de neurones pourra évoluer, à partir d'un état aléatoire, vers un état stable de la fonction d'énergie construite pour la configuration considérée. Cet état correspond à une solution (si elle existe) d'ordonnancement des tâches respectant les contraintes qui leur sont imposées. La méthode a été utilisée pour des problèmes d'ordonnancement de tâches temps réel en prenant en compte les contraintes de temps (les périodes, les échéances, etc.), la préemption et la non-préemption de tâches avec ou sans contraintes de précédence, dans une architecture monoprocesseur ou encore une architecture multiprocesseurs homogènes (i.e. les processeurs ont la même puissance de calcul).

Le schéma de représentation du réseau de neurones suivant a été adopté :
– les neurones sont rangés sous forme d'une matrice dont les lignes correspondent aux tâches et les colonnes correspondent à la longueur de l'ordonnancement, généralement considérée comme le plus petit multiple commun (PPCM) des périodes des tâches à ordonnancer ;

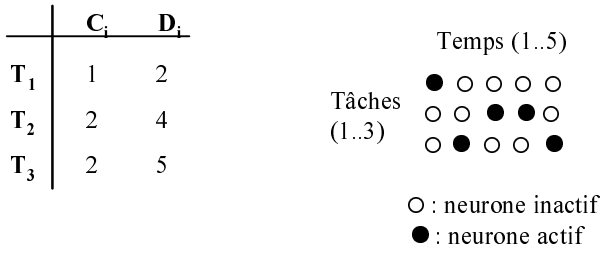

FIG. 1 – Résultat de l'évolution d'un réseau de Hopfield pour un exemple de tâches préemptibles et utilisant le processeur à 100%.

– un neurone dont la sortie est active (égale à 1), signifie que la tâche correspondante est active pendant l'unité de temps correspondante.

Après construction de ce schéma, la fonction d'énergie correspondant au problème doit être calculée, ce qui permettra d'obtenir les différents poids des connexions et les entrées externes des neurones. Cette fonction est la somme des diverses fonctions d'énergie qui correspondent aux différentes contraintes à prendre en compte.

La figure 1 représente le résultat de l'application de la règle *k-de-N* au problème d'ordonnancement de trois tâches préemptibles sur une architecture monoprocesseur, où C_i représente la durée d'exécution au pire cas de la tâche T_i et D_i son échéance.
La règle *k-de-N* est appliquée pour chaque tâche (à chacune des lignes). Par exemple, pour la tâche T_1, la règle est appliquée aux deux premiers neurones de la première ligne (puisque son échéance est égale à 2) avec $k = 1$ (puisque $C_1 = 1$). Pour les neurones restants de la première ligne, la règle est appliquée avec $k = 0$. Enfin, pour gérer l'utilisation du processeur à chaque instant de l'ordonnancement en exclusion mutuelle entre les différentes tâches, pour chaque colonne, la règle *k-de-N* est appliquée avec $k = 1$. Pour cet exemple, la figure 1 montre les résultats après stabilisation ; T_1 a le processeur durant la première unité de temps, T_2 a le processeur durant la 3e et la 4e unité de temps et T_3 a le processeur durant la 2e et la 5e unité de temps.

Les résultats de simulation de l'application de cette technique sur différentes configurations de tâches présentés dans [1] montrent que l'exécution du réseau peut être relancée plusieurs fois (le réseau se stabilise en quelques centaines de nanosecondes) dans le cas où la fonction est différente de zéro pour déterminer s'il s'agit ou non d'un minimum local.

3. Application des réseaux de neurones à l'ordonnancement de tâches sur architectures multiprocesseurs hétérogènes

Le problème de l'ordonnancement en-ligne d'un ensemble de tâches avec échéances sur une architecture à m processeurs (avec $m \geqslant 2$) est un problème complexe dans le sens où il n'existe pas d'algorithmes optimaux pour ce type de problèmes [7] ; c'est un problème *NP*-Complet. De plus, des résultats d'utilisation d'algorithmes connus, tels que l'Earliest Deadline First [6] ou le Rate Monotonic [11], pour des configurations de tâches s'exécutant sur une architecture monoprocesseur, ne peuvent plus être directement appliqués au cas multiprocesseurs ; l'algorithme EDF, par exemple, optimal dans un environnement monoprocesseur, ne l'est plus dans un environnement multiprocesseurs [4].

3.1. Modèles logiciel et matériel considérés

Dans ce travail, nous proposons d'étendre les travaux sur l'utilisation des réseaux de neurones pour l'ordonnancement en prenant en compte des architectures matérielles composées de processeurs hétérogènes (pouvant présenter des capacités de calcul différentes pour une même tâche). Dans un environnement multiprocesseurs, un algorithme d'ordonnancement de tâches temps réel est valide si toutes les

échéances des tâches distribuées à travers les processeurs sont respectées. Aussi, les conditions qu'un processeur ne peut exécuter qu'une seule tâche à un instant donné et qu'une tâche ne peut être traitée que par un seul processeur (il n'y a pas de migration de tâches), sont imposées dans notre modèle.
Par ailleurs, les tâches présentes dans ce système sont supposées, dans un premier temps, indépendantes et les coûts de communication sont pris en compte dans les pires temps de réponse de ces tâches.
Nous supposons également que l'ordonnanceur est implanté sur un processeur où le système d'exploitation se charge de la gestion de toutes les ressources de l'architecture en ayant une vue globale de l'état du système accessible à tout moment. Ainsi, cet ordonnanceur va fournir un ordonnancement global pour les tâches arrivant au système et devant être dispatchées à travers les différents processeurs tout en respectant les contraintes qui leur sont imposées.
Nous partons d'une spécification des paramètres des tâches considérées dans le système qui précise en particulier les durées d'exécution au pire cas de chacune des tâches sur les processeurs sur lesquels elle peut éventuellement s'exécuter. Par exemple, une tâche T_i peut être représentée par $T_i(R_i, C_{i0}, C_{i1}, \ldots, C_{im}, D_i, P_i)$ où R_i est la date au plus tôt, C_{ij} la durée d'exécution de la tâche T_i sur le processeur j (elle est nulle si cette tâche ne peut pas être exécutée sur ce processeur), D_i son échéance et P_i sa période. Une fois cette spécification effectuée, il s'agira par la suite de calculer la séquence d'ordonnancement de ces tâches sur les m processeurs qui permettrait à toutes les tâches de voir leurs contraintes respectives satisfaites.

3.2. Modèle neuronal considéré

Nous modifions la topologie du réseau de neurones proposée dans [1] en étendant le réseau avec autant de plans que de processeurs dans le système. Chaque plan de neurones représentera ainsi un processeur, où les lignes représenteront les tâches et les colonnes les instants d'ordonnancement. Tous les neurones seront évidemment totalement interconnectés pour pouvoir utiliser les résultats de Hopfield.
Les contraintes imposées au départ supposent que durant chaque période d'une tâche, celle-ci doit être exécutée sur un et un seul processeur. Pour respecter cette contrainte, nous proposons d'abord, d'appliquer aux neurones de chacune des périodes de la tâche sur les différents processeurs, la règle *k-de-N* avec k égal au maximum des durées d'exécutions de la tâche sur les différents processeurs et N l'ensemble de ces neurones étendu d'un nombre de neurones qui correspond à la différence entre le maximum et le minimum (différent de zéro) des durées d'exécution de cette tâche sur les différents processeurs. Cette application de la règle garantit d'avoir au moins une exécution de la tâche sur l'un des processeurs (sur lesquels elle peut s'exécuter) durant sa période d'exécution.
Prenons l'exemple simple de trois tâches périodiques T_1, T_2 et T_3 s'exécutant sur une architecture à deux processeurs hétérogènes. Appliquer le raisonnement énoncé ci-dessus implique l'application de la règle *k-de-N* pour T_1 sur les neurones lui correspondant sur les deux plans, étendus de 2 neurones $(C_{12} - C_{11})$, avec $k = \max(C_{11}, C_{12}) = 4$. Le même raisonnement est appliqué aux autres tâches. Ceci conduit à un schéma d'exécution tel que celui de la figure 2, dans lequel on peut se retrouver avec une exécution de la même instance de tâche sur deux processeurs différents.

	C_{i1}	C_{i2}	D_i=P_i
T_1	2	4	6
T_2	3	2	6
T_3	1	2	6

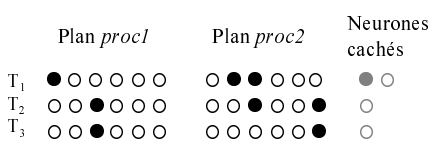

FIG. 2 – Exemple d'exécution sur deux processeurs hétérogènes.

Pour s'assurer que cette exécution se fasse sur un et un seul processeur, il faut forcer les neurones actifs de la même tâche à être tous sur le même processeur. Une nouvelle règle de construction, présentée ci-dessous, est nécessaire à introduire, on l'appellera, règle *(0-ou-k)-de-N*, et qui sera appliquée pour chaque

période de tâche T_i sur un processeur j avec $k = C_{ij}$.

3.3. Nouvelle règle de construction *(0 ou k)-de-N*

Pour que uniquement 0 ou k neurones soient activés parmi N, l'ensemble des neurones considérés devront respecter les conditions suivantes :

$$\sum_{i=1}^{N} x_i = k \qquad \text{ou} \qquad \sum_{i=1}^{N} x_i = 0$$

La fonction coût suivante

$$E' = \sum_{i=1}^{N} x_i (k - \sum_{i=1}^{N} x_i)^2 \tag{4}$$

va être minimale lorsque la somme des neurones à l'état 1 est égale à 0 ou à k, elle est positive dans les autres cas.
Il reste maintenant à démontrer que cette fonction peut être mise sous la forme d'une fonction d'énergie d'un réseau de Hopfield. La fonction (4) est équivalente à

$$\begin{aligned}
E' &= \sum_{i=1}^{N} x_i (k^2 - 2k \sum_{j=1}^{N} x_j + (\sum_{j=1}^{N} x_j)^2) \\
&= \sum_{i=1}^{N} x_i (k^2 - 2k \sum_{j=1}^{N} x_j + \sum_{j=1}^{N} x_j^2 + \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} x_j x_k) \\
&= \sum_{i=1}^{N} x_i (k^2 - (2k-1) \sum_{j=1}^{N} x_j + \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} x_j x_k) \\
&= k^2 \sum_{i=1}^{N} x_i - (2k-1) \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j + \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} x_i x_j x_k \\
&= k^2 \sum_{i=1}^{N} x_i - (2k-1) \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} x_i x_j - (2k-1) \sum_{i=1}^{N} x_i^2 + \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} x_i x_j x_k \\
&= \underbrace{(k-1)^2 \sum_{i=1}^{N} x_i - (2k-1) \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} x_i x_j}_{E'_1} + \underbrace{\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} x_i x_j x_k}_{E'_2}
\end{aligned}$$

La fonction E'_1 correspond à une fonction d'énergie d'un réseau de N neurones dont les connexions et les entrées externes ont les valeurs suivantes

$$\begin{aligned}
T_{ij} &= \begin{cases} 0 & \text{si } i = j \\ 4k-2 & \text{sinon} \end{cases} \\
I_i &= -(k-1)^2 \qquad \forall i
\end{aligned}$$

Cependant, le terme (4k-2) est positif lorsque $k \succ \frac{1}{2}$, ce qui ne vérifie pas les critères de convergence de Cohen et Grossberg énoncés précédemment. Ceci peut être résolu par l'ajout du terme

$$(4k-2)(k - \sum_{i=1}^{N} x_i)^2 \tag{5}$$

Cette fonction ne va pas affecter le résultat, puisqu'elle vaut zéro lorsque le nombre de neurones actifs est égal à k. La matrice de connexions et les entrées des neurones de la deuxième partie de ce terme ont été calculés pour la règle k-de-N et peuvent être réutilisés. Ainsi, le terme (5) conduit à

$$T_{ij} = \begin{cases} 0 & \text{si } i = j \\ -2(4k-2) & \text{sinon} \end{cases}$$
$$I_i = (2k-1)(4k-2) \qquad \forall i$$

Et puisque le modèle est additif, ces résultats combinés avec ceux calculés pour E_1 donnent

$$T_{ij} = \begin{cases} 0 & \text{si } i = j \\ -(4k-2) & \text{sinon} \end{cases}$$
$$I_i = -(k-1)^2 + 2(2k-1)^2 \qquad \forall i$$

Ces résultats respectent parfaitement les critères de convergence d'un réseau de Hopfield.

Le terme E'_2 équivalent à $-\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}\sum_{k=1}^{N} -2x_i x_j x_k$, contient un produit de trois neurones ce qui le rend plus complexe. En introduisant la notion de *neurone inhibiteur*, cette complexité peut être levée. Lorsqu'un neurone est inhibiteur pour une connexion donnée, il est neutre pour cette connexion s'il est actif. La connexion sera égale à zéro si ce neurone inhibiteur est inactif. Ainsi, si le neurone x_k du terme E'_2 est considéré comme inhibiteur, les connexions et les entrées des neurones de E'_2 seront comme suit

$$T_{ij} = \begin{cases} 0 & \text{si } x_k = 0 \\ -2 & \text{sinon} \end{cases}$$
$$I_i = 0 \qquad \forall i$$

Ce résultat est cumulé avec celui calculé pour E'_1.
Au niveau de la simulation du réseau de neurones, il suffit, pour une connexion donnée, d'effectuer un test de l'état de tous les neurones pour rajouter 0 ou -2 aux poids trouvés pour E'_1. D'un point de vue matériel, inhiber une connexion par un neurone revient à faire passer la connexion par un "et" logique avec l'état des neurones inhibiteurs. Ceci complique le circuit mais il reste réalisable [1].
L'exemple présenté précédemment peut ainsi être amélioré. L'une des solutions pouvant être trouvées est représentée par la figure 3.

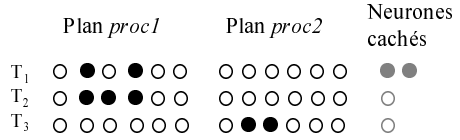

FIG. 3 – Exemple d'exécution sur deux processeurs hétérogènes en introduisant la règle *(0 ou k)-de-N*.

Cette application de la règle à elle seule n'est pas suffisante. En effet, il faut assurer que chacun des processeurs soit accédé en exclusion mutuelle par les tâches qui le sollicitent. Pour ceci, il suffit d'appliquer la règle *k-de-N* à chaque colonne représentant un instant de l'ordonnancement, avec k=1 et N le nombre de tâches dans le système étendu d'un neurone pour assurer la contrainte "au plus 1 neurone actif" à un instant donné. Ainsi, pour l'exemple introduit précédemment le schéma d'exécution de la figure 4 peut être trouvé.

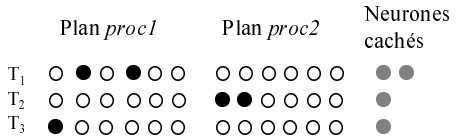

FIG. 4 – Exemple d'exécution sur deux processeurs hétérogènes avec utilisation de la règle *(0 ou k)-de-N* et prise en compte de l'exclusion mutuelle.

4. Conclusion

A travers cet article, nous avons proposé une extension de l'utilisation des réseaux de neurones, pour l'ordonnancement de tâches temps réel, à des architectures multiprocesseurs hétérogènes. Nous avons présenté une nouvelle règle qui permet d'obtenir des connexions et des entrées adaptés pour construire un réseau de neurones qui puisse se stabiliser sur un état respectant les contraintes du modèle considéré, notamment, l'hétérogénéité des unités de calcul sur lesquelles les tâches peuvent s'exécuter.

A l'état actuel, les résultats de simulation ne permettent pas encore d'affirmer la validité de cette approche. En particulier, le réseau se stabilise souvent sur certains états "indésirables" qui ne vérifient pas toutes les contraintes imposées au départ. Ainsi, nous avons remarqué au niveau de la nouvelle règle énoncée une tendance du réseau à ne se stabiliser que rarement sur l'état "0". Ceci pourrait être dû à l'ajout du terme (5) à la fonction d'énergie et qui favorise plutôt la stabilisation sur "k". Nous essayons actuellement de trouver un moyen pour rendre l'état "0" plus attracteur afin de pouvoir montrer l'efficacité de l'approche proposée dans ce travail en démarrage.

Bibliographie

1. C. Cardeira. *Ordonnancement Temps Réel par réseaux de neurones.* PhD thesis, INPL, Septembre 1994.
2. C. Cardeira and Z. Mammeri. Preemptive and non-preemptive real-time scheduling based on neural networks. *Proceedings DDCS'95*, pages 67–72, September 1995.
3. M. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. on Systems, Man and Cybernetics*, 13(5) :815–826, Sept. / Oct. 1983.
4. F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems.* Wiley, 2002.
5. R. David, D. Chillet, S. Pillement, and O. Sentieys. Dart : A dynamically reconfigurable architecture dealing with next generation telecommunications constraints. *9th IEEE Reconfigurable Architecture Workshop RAW*, April 2002.
6. M. Dertouzos. Control robotics : The procedural control of physical processes. *in Proc. IFIP Congr.*, pages 807–813, 1974.
7. S. Grossberg. *Studies of Mind and Brain : Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*, volume 70. Reidel Press Bonston, 1982.
8. B. Hamidzadeh, D. Lilja, and Y. Atif. Dynamic scheduling techniques for heterogeneous computing systems. *Journal of Concurrency : Practice and Experience*, 7 :633–652, October 1995.
9. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79 :2554–8, 1982.
10. J. J. Hopfield and D. W. Tank. Computation of decisions in optimization problems. *Biological Cybernetics*, 52 :141–52, 1985.
11. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. of the ACM*, 20(1) :46–61, 1973.
12. G. Tagliarini, J. F. Christ, and W. E. Page. Optimization using neural networks. *IEEE Trans. Comput.*, 40(12) :1347–58, December 1991.
13. P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins. Energy-aware runtime scheduling for embedded multiprocessor socs. *IEEE Design & Test of Computers*, pages 46–58, Sept.-Oct. 2001.