Ordonnancement de tâches par réseaux de neurones pour architectures de SoC hétérogènes

Real-Time Scheduling on Heterogeneous SoC Architectures Using A Neural Network

Daniel Chillet, Sébastien Pillement et Olivier Sentieys

ENSSAT – Université de Rennes 1 – IRISA – Équipe CAIRN, BP 80518, 6 Rue de Kerampont, 22305 Lannion, France
Daniel.Chillet@irisa.fr, Sebastien.Pillement@irisa.fr, Olivier.Sentieys@irisa.fr

Manuscrit reçu le 25 février 2008

Résumé et mots clés

Les technologies de conception de circuits intégrés permettent aujourd'hui de concevoir des systèmes complets et complexes sur une seule et même puce. On parle alors de systèmes sur puce, ou encore de System-on-Chip (SoC). Ces systèmes ont en charge l'exécution d'applications complexes, composées de nombreuses tâches, le tout étant orchestré par un système d'exploitation dont l'un des rôles principaux consiste à ordonnancer les tâches et à les allouer aux ressources de calcul.
L'une des particularités de ces architectures concerne l'hétérogénéité des cibles d'exécution qui rend le problème de l'ordonnancement particulièrement délicat et complexe. Notons de plus que le critère temps réel des applications s'exécutant sur ce type de plate forme nécessite l'étude de solutions d'ordonnancement efficaces, notamment en terme de temps de calcul.
Dans ce papier, nous présentons nos travaux de modélisation du problème de l'ordonnancement pour architectures multi-processeurs hétérogènes par utilisation de réseaux de neurones. Des travaux précédents ont montré qu'une structure de réseaux de neurones suivant le modèle de Hopfield peut être définie pour ordonnancer des tâches sur une architecture homogène. Une extension à ces travaux a montré qu'il était possible de prendre en compte l'hétérogénéité de l'architecture mais au prix d'un grand nombre de neurones supplémentaires. De plus, ces solutions posent un problème de convergence important qui se traduit par un temps de convergence assez long et le besoin de ré-initialiser le réseau de neurones lorsque celui-ci se stabilise dans un état qui n'est pas une solution valide.
Pour contrer ces principaux inconvénients, nous proposons une nouvelle structure basée sur la mise en place de neurones inhibiteurs. Ces neurones particuliers permettent de limiter le nombre de neurones nécessaires à la modélisation et permettent surtout de se passer de ré-initialisations pour atteindre la convergence.
Nous illustrons l'apport de notre proposition en comparant les solutions classiques à base de réseaux de neurones de Hopfield avec notre proposition. Nous montrons que le nombre de neurones est assez largement réduit et surtout qu'il n'est plus nécessaire de ré-initialiser le réseau pour assurer sa convergence, ce qui laisse envisager une implémentation efficace de ce type de structure.

Ordonnancement de tâches, réseaux de neurones artificiels, architectures hétérogènes, système sur Puce.

Abstract and key words

Introduction

Several scheduling algorithms have been developed for constraint satisfaction in real-time systems. Optimality is difficult to reach, and the problem becomes NP-hard when a large set of constraints must be satisfied. To solve this type of problem, approximate methods are used, such as Artificial Neural Networks (ANNs). Neural networks have demonstrated their efficiency in optimization problems. They converge in a reasonable time if the number of neurons and connections between neurons can be limited. Another limitation concerns the need to regularly re-initialize the network when it converges towards a stable state which does not belong to the set of valid solutions. On the other hand, embedded applications are usually implemented on complex System-on-Chip (SoC) which are built around heterogeneous processing units. On such platform, task instantiation on execution resources is realized by using the scheduling service of an OS. As each task can be defined for several targets, this service must decide, on-line, on which resource the task should be instantiated.
In this work, we propose an on-line scheduling based on a neural network for heterogeneous system-on-chip (SoC) architectures with a limited number of neurons.

Related works

Most of scheduling algorithms consider very specific and homogeneous set of characteristics. In the context of SoC architecture, specific scheduler have been proposed to take parallel and heterogeneity characteristics into account. These service implementations are often complex, and are not always appropriate to real-time systems ; they are generally time costly and do not consider the dynamic behavior of applications.
Some works have concentrated on the problem of scheduling for multi-processor architectures and solutions have been developed and published. Algorithms producing optimal solutions have been proposed [8], as well as solutions sharing the tasks between multi-processor machines [9, 10], but there are principally developed for homogeneous machines. As a general rule, these solutions are not easily applicable in the context of SoCs because of the implicit heterogeneous nature of the latters. The complexity of the problem of scheduling as well as the time constraints (linked to the real time aspects of the applications running on the SoCs) have encouraged a number of works studying the implementation of services in hardware form [11, 12, 13]. Among the numerous solutions which have been suggested to try to resolve the optimisation problem, one uses neural networks [14, 15]. The underlying model is based on Hopfield's proposition [16]. This defines a neural network completely connected capable of producing solutions to an optimisation problem by minimising an energy function. The definition of a Hopfield neural network goes traditionally through three stages which are :
- Modelling the problem in such a way that the neurons' states defines a possible solution ;
- The definition of the energy function expresses a correct solution from the neurons' states ;
- The search for the values of the weight of the connections and of the inputs to each neuron.

From Hopfield's model, the authors of [1] suggest a modelling of the scheduling problem for homogeneous architectures. They ensure the existence of the *energy function* which permits the network to converge towards a stable state for which the optimization constraints are respected. The ability to take different constraints into account is made possible by the additive character of the Hopfield model. Based on this model, the *k-out-of-N* design rule facilitates the neural network construction. This rule allows the construction of a network of *N* neurons for which the evolution leads to a stable state with "exactly *k* active neurons among *N*". The results for this real-time scheduling solution exhibit interesting convergence speed which makes ANN suitable for on-line utilization. But this technique has two major drawbacks. The first is the number of necessary additional neurons needed to model the problem. The second drawback is the presence of several local minima when several rules are applied to the same set of neurons. These local minima are particular points of the energy function which represent invalid solutions. To ensure convergence towards valid solutions, these points must be detected and the network needs to be re-initialized.

In our work, the number of slack neurons is drastically reduced. Moreover no re-initialization is necessary due to the fact that there is no local minimum in the proposed energy function.

Scheduling problem modelling through neural network

In the case of monoprocessor architecture, the scheduling problem is modelled through ANN by the following representation :
- Neurons $n_{i,j}$ are arranged in a $N_T \times N_C$ matrix, where line i corresponds to the task i and the column j corresponds to schedule time unit j. The number of time units N_C depends on the hyperperiod of tasks (*i.e*. the least common multiple of all the task periods) and N_T is the number of tasks.
- An active neuron $n_{i,j}$ indicates that during the corresponding time unit j, the task i is being executed.
- One line of neurons is added to model the possible inactivity of the processor during the schedule times. These neurons are called *slack neurons* since they are not used to represent the solution.
In the case of homogeneous multiprocessor architecture, several matrix organized in layers are required. New slack neurons are then necessary to manage the exclusive execution of each task on resources. So the total number of slack neurons can be very important, especially when the system is composed of a large number of application tasks and execution resources.
In order to reduce the number of required neurons, we propose to modify the neural network structure. Our proposition consists in adding a specific neuron (called *inhibitor neuron*) for each task and each type of execution resource. The main idea consists in creating a mutual exclusion between the possible task instantiations on resources. A set of N_C neurons is called $S_{i,r}$ and represents the possible scheduling cycles of task i on the resource r. For each resource r, the worst case execution time (WCET) of task i is defined as $C_{i,r}$. The set of neurons $S_{i,r}$ is configured to converge towards $C_{i,r}$ active neurons among N_C. When one set has the required number of active neurons, the state of the corresponding inhibitor neuron is set to one. So, all the neurons included in the other sets will be forced to the inactive state. The main characteristic of this neural network is its capacity to converge to a stable state from any initial state. No local minimum exists in the energy function, *i.e*. no network re-initialization is needed to ensure the convergence.

Results

The first example concerns the schedule of three tasks on one execution resource. To solve this problem, the proposition in [15] needs to add one neuron line to model the processor idle cycles. The authors indicate that convergence is obtained after more than 600 fired neurons. For the same scheduling problem, our model requires less neurons than the previous solution. Moreover, for all simulations from random initial state, our network always converges to a valid solution. This convergence is obtained with an average number of fired neurons equal to 79. Contrary to the classical energy function evolution, we can notice that our modified energy function decreases monotonously with the number of fired neurons.
Other examples with two execution resources show that classical solution converges to a valid solution with more than 300 fired neurons. Our proposition limits the number of fired neurons to approximatively 70. With these kind of examples the number of added neurons explodes with the classical model and is limited in our approach. The number of slack neurons removed is directly proportional to the hyperperiod which can be important.
We also made some experiments on a SoC architecture. The considered platform is composed of 5 different resources (1 GPP, 3 Intellectual Property blocks IP and a Dynamically Reconfigurable Accelerator DRA). We consider that several tasks are defined for several heterogeneous resources while other tasks are defined for one specific resource. The complete application is composed of 10 tasks. Seven tasks have been described for the GPP resource, three tasks have been defined for the three IP blocks. Finally, four tasks have also been described for DRA. The experiments show that the number of fired neurons evolves linearly with the number of tasks considered, i.e. linearly with the neural network complexity. Another important result is the relatively constant number of evaluations of each neuron. In those examples, each neuron is evaluated between 8 and 12 times.

Conclusion and future work

In this paper, a scheduling service modelled by a neural network is presented. An adaptation of the Hopfield model is proposed. The major contributions of our work concern the limitation of slack neurons and the simplification of the network control thanks to the absence of re-initializations. To limit the number of neurons, we replace numerous slack neurons of classical solutions by several inhibitor neurons. Based on these inhibitor neurons, a new structure of a neural network is presented. Furthermore, in classical solutions, a lot of neuron network re-initialisations are necessary to converge towards a valid solution. This is due to the existence of local minima of energy function. We propose to replace the addition of two *k-out-of-N* rules by the addition of k_1-*out-of-*N_1 and *at-most-*k_2-*among-*N_2 rules. We have shown that this specific additivity of rules always ensures the convergence and yet limits the number of slack neurons. These two contributions are important advances for our future works which consist in dening a hardware implementation of the scheduling service in the context of SoC architecture. The major problem for the hardware implementation concerns the large number of connections between neurons.

Task Scheduling, Artificial Neural Networks, Heterogeneous Architecture, System-on-Chip.

1. Introduction

L'évolution de la complexité des applications, notamment de traitement du signal et de l'image, a progressivement amené les concepteurs de circuits intégrés à développer des architectures de type SoC (System-on-Chip). Comme le montre la figure 1, un SoC intègre en général un cœur de processeur généraliste, des blocs spécifiques (IP : Intellectual Properties) pour des traitements critiques, un ou plusieurs cœurs de processeurs de traitement du signal et de plus en plus fréquemment des éléments reconfigurables.

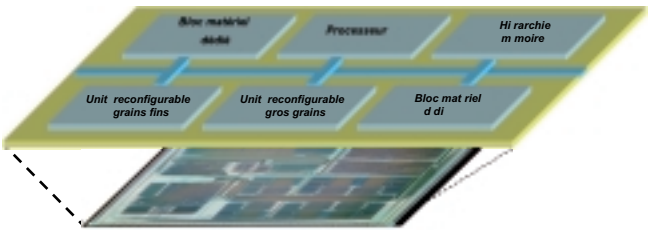


Figure 1. Exemple d'architecture de SoC. Ce type de système est assimilable à une architecture multi-processeurs hétérogènes. Les nombreuses ressources de calcul permettent de prendre en charge l'exécution de l'ensemble des tâches de l'application.

Sur la base de cette organisation générale, ces architectures offrent désormais des capacités d'intégration de systèmes complexes. La complexité globale de fonctionnement de ces architectures repose en grande partie sur l'hypothèse de disponibilité d'un système d'exploitation permettant de gérer la répartition des tâches au sein du système. Ces architectures peuvent être assimilées à des systèmes multi-processeurs hétérogènes pour lesquels l'un des rôles du système d'exploitation est d'assurer l'ordonnancement des tâches de l'application. D'une façon générale, il s'agit de déterminer les instants d'exécution ainsi que les placements des tâches sur les différentes cibles d'exécution. Ces deux phases que sont l'ordonnancement et le placement des tâches sont parfois simplement regroupées sous le terme d'ordonnancement. Dans la suite de cet article, nous engloberons ces deux phases sous ce terme. Dans ce contexte, l'ordonnancement est une problématique difficile non résolue, pour laquelle l'obtention d'une solution optimale n'est pas garantie. De nombreux travaux ont été publiés sur cette problématique. Les travaux de Cardeira *et al.* ont montré qu'il est possible de définir une structure de réseaux de neurones permettant de résoudre le problème de l'ordonnancement des tâches pour un système homogène [1]. Le problème d'ordonnancement se ramène alors à un problème d'optimisation sous contraintes. Les principaux inconvénients de la structure proposée résident dans le nombre de neurones nécessaires pour modéliser le problème ainsi que dans la nécessité d'effectuer un nombre de réinitialisations important pour s'assurer de la convergence du réseau vers un état représentant une solution non valide. La non validité des solutions ainsi produite doit être détectée et le réseau doit alors être ré-initialisé afin d'espérer atteindre une solution correcte. Cela se traduit par un contrôle du réseau relativement complexe et peu compatible avec une implémentation au sein d'un SoC.

Dans cet article, nous proposons une nouvelle structure de réseaux de neurones qui permet, d'une part une limitation importante du nombre global de neurones, et d'autre part une

convergence systématique vers une solution d'ordonnancement correcte. Ces deux points sont importants par rapport à notre objectif final consistant à proposer une implémentation efficace de structures matérielles d'ordonnancement.
Le papier est organisé de la façon suivante. Nous présentons dans la section 2 les travaux relatifs au problème d'ordonnancement et plus particulièrement les travaux de modélisation au travers de l'utilisation de réseaux de neurones. La section 3 présente notre solution qui s'appuie sur la mise en place de neurones inhibiteurs. La construction du réseau est expliquée et la convergence systématique d'un tel réseau est présentée. La section 4 illustre notre proposition à partir d'un ensemble d'expérimentations pour lesquelles nous faisons varier le nombre de tâches s'exécutant sur la plate forme d'exécution. Nous comparons les résultats obtenus par notre solution avec ceux obtenus par Cardeira *et al.* et nous montrons le gain significatif obtenu notamment sur la convergence du réseau. Finalement, la section 5 conclue ce papier et présente les perspectives de nos travaux.

2. État de l'art et modélisations classiques

De nombreuses propositions d'ordonnancement ont été publiées depuis de nombreuses années. On peut citer les algorithmes *rate monotonic* [2], *deadline monotonic* [3] ou encore *earliest deadline first* encore connu sous le nom *deadline driven* développé conjointement au *rate monotonic*. Ces algorithmes ont ensuite fait l'objet d'extensions et on compte de nombreux dérivés permettant notamment de prendre en compte des contraintes particulières. Le calcul de l'ordonnancement par ces algorithmes est réalisé soit avant l'exécution des tâches elles même, on parle alors d'ordonnancement hors ligne ou statique [4], soit lors de l'exécution des tâches, on parle alors d'ordonnancement à chaud, en ligne ou encore dynamique [5, 6]. Dans le contexte des SoC, le besoin de réactivité et d'adaptabilité du système aux événements extérieurs conduit les concepteurs à favoriser des ordonnancements en ligne ou encore des approches mixtes [7]. Cependant, ces approches en ligne souffrent d'une complexité importante qui limite leur utilisation, faute de puissance de calcul suffisante.
Certains travaux se sont concentrés sur la problématique de l'ordonnancement pour architectures multi-processeurs, et des solutions ont été développées et publiées dans la littérature. Des algorithmes proposant des solutions optimales ont été proposés [8], de même que des solutions de partitionnement pour machines multi-processeurs [9, 10], mais il s'agit principalement de solutions pour des machines homogènes. En règle générale, ces solutions sont difficilement applicables au contexte des SoC à cause de la nature implicitement hétérogène de ces dernières. La complexité du problème d'ordonnancement ainsi que les contraintes temporelles (liées aux aspects temps réel des applications s'exécutant sur les SoC) ont encouragé un certain nombre d'équipes à étudier l'implantation de services sous forme matérielle [11, 12, 13]. Parmi les nombreuses solutions qui ont été proposées pour résoudre ce problème d'optimisation, l'une d'elle repose sur les réseaux de neurones [14, 15]. Le modèle sous-jacent s'appuie sur la proposition de réseaux de neurones de Hopfield [16]. Cette proposition définit un réseau de neurones entièrement connecté capable de produire les solutions d'un problème d'optimisation par minimisation d'une *fonction d'énergie*. La définition d'un réseau de neurones de Hopfield passe traditionnellement par trois étapes qui sont :
- la modélisation du problème de telle sorte que les états des neurones définissent une solution possible ;
- la définition de la fonction d'énergie exprimant une solution correcte à partir des états des neurones ;
- la recherche des valeurs des poids des connexions et des entrées de chaque neurone.

Les réseaux de neurones ainsi définis vont alors converger vers un état qui minimise la fonction d'énergie, et qui représente une solution au problème.
À partir du modèle de Hopfield, les auteurs de [1] proposent une modélisation du problème d'ordonnancement pour architectures homogènes. Cette modélisation est schématisée sur la figure 2.a.

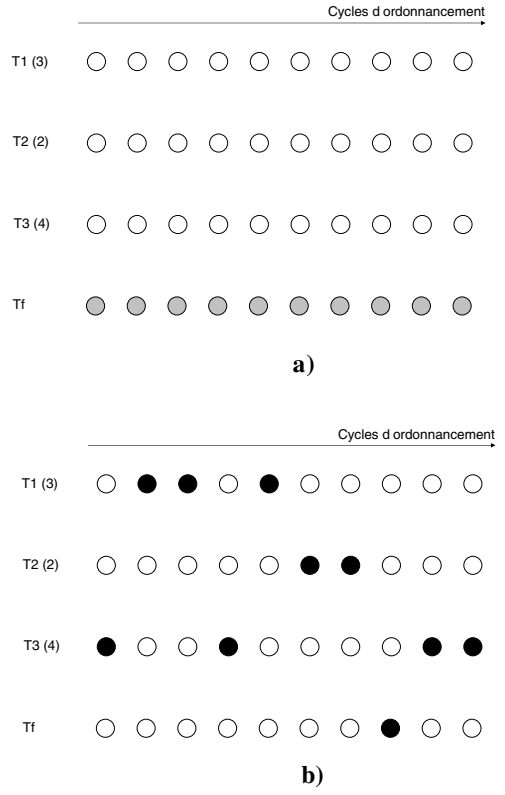


Figure 2. a) Représentation initiale du problème d'ordonnancement à résoudre. La ligne T f correspond à une tâche fictive dont l'objectif consiste à modéliser l'inactivité de la ressource lorsque celle-ci n'a pas de tâche à exécuter ; b) Exemple de convergence du réseau de neurones vers une solution correcte.

Sur cette figure, chaque point (ou neurone) représente un cycle d'ordonnancement. L'état du neurone (actif ou inactif) représente l'état de la tâche (« running » ou « suspended »). Un neurone actif indique que la tâche doit être placée dans l'état « running » et donc dispose de la ressource d'exécution, alors qu'un état inactif indique que la tâche ne possède pas la ressource d'exécution. La figure 2.b) présente un exemple de résultat de convergence du réseau. La modélisation proposée s'appuie en grande partie sur la définition d'une règle de construction du réseau dite règle $k-de-N$, permettant de spécifier que k neurones parmi N doivent être activés. Cette règle est définie par la fonction d'énergie suivante :

$$E_{k-de-N} = \left(k - \sum_{i=1}^{N} x_i\right)^2 \tag{1}$$

Avec x_i l'état du $i^{ème}$ neurone, cet état valant 1 si le neurone est actif et 0 sinon. Le minimum de cette fonction d'énergie correspond aux cas où k neurones sont actifs. Dans ce cas, ce minimum vaut 0.

Dans leurs travaux [1, 15], Cardeira *et al.* ont montré que la construction d'un réseau de neurones peut se faire par additivité de règles de type $k-de-N$. La construction du réseau est alors assurée par applications de cette règle de construction sur une succession d'ensembles de neurones. Les figures 3.a et 3.b illustrent la construction du réseau pour un ensemble de trois tâches dont les caractéristiques sont les suivantes :

- Tâche T1 : charge de calcul sur la ressource de calcul $C_1 = 3$;
- Tâche T2 : charge de calcul sur la ressource de calcul $C_2 = 2$;
- Tâche T3 : charge de calcul sur la ressource de calcul $C_3 = 4$;

La règle $k-de-N$ est d'abord appliquée horizontalement sur chaque ligne de neurones des tâches (voir figure 3.a.), avec $k = C_i$ pour l'application de la règle sur la tâche i, et N égal au nombre de cycles d'ordonnancement global (dans le cas présenté $N = 10$). L'application de cette règle assure que chaque tâche obtiendra le bon nombre de cycles d'ordonnancement pour s'exécuter.

La règle $k-de-N$ est ensuite appliquée sur chaque colonne du réseau afin de s'assurer qu'une seule tâche est ordonnancée à chaque cycle (voir figure 3.b). L'ordonnancement de l'ensemble des tâches pouvant nécessiter moins de cycles que la modélisation n'en prévoit, il est nécessaire de prévoir une tâche dite *fictive* (tâche Tf) correspondant aux instants d'oisiveté de la ressource de calcul. La règle $k-de-N$ est alors appliquée sur les colonnes, avec $k = 1$ ce qui correspond à une seule tâche exécutée sur la ressource à chaque cycle et N égal au nombre de tâches augmenté de 1.

La troisième étape consiste à mettre en forme la fonction d'énergie. Il s'agit de faire apparaître les termes $T_{i,j}$ et I_i correspondants aux poids des connexions, et aux entrées de chaque neurone. La forme à obtenir est alors la suivante :

$$E = -\sum_{i=1}^{N} I_i \cdot x_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{i,j} \cdot x_i \cdot x_j \tag{2}$$

L'état d'un neurone est calculé par la relation suivante :

$$x_i = \begin{cases} 1 & \text{si } I_i + \sum_{j=1}^{N} x_j \cdot T_{j,i} \geq 0 \\ 0 & \text{sinon.} \end{cases} \tag{3}$$

Il a été démontré que le réseau ainsi construit va naturellement évoluer vers un état tel que la fonction d'énergie est minimale et égale à 0 [17, 18]. On dit alors que le réseau converge vers un état stable.

La règle de construction $k-de-N$ qui est utilisée pour la modélisation du problème d'ordonnancement de la figure 2.a peut alors être transformée pour être écrite selon la forme de l'équation 2 :

$$\begin{aligned} E_{k-de-N} &= \left(k - \sum_{i=1}^{N} x_i\right)^2 \\ &= \left(\sum_{i=1}^{N} x_i\right)^2 - 2k\sum_{i=1}^{N} x_i + k^2 \\ &= \sum_{i=1}^{N} x_i^2 + \sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N} x_i \cdot x_j - 2k\sum_{i=1}^{N} x_i + k^2 \end{aligned}$$

Sachant que x_i ne peut prendre que les valeurs 0 ou 1, alors $\sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i$, donc

$$\begin{aligned} E_{k-de-N} &= \sum_{i=1}^{N} x_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}(-2) \cdot x_i \cdot x_j - 2k\sum_{i=1}^{N} x_i + k^2 \\ &= -\sum_{i=1}^{N}(2k-1) \cdot x_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}(-2) \cdot x_i \cdot x_j + k^2 \end{aligned} \tag{4}$$

Conformément à la relation 2, cette formulation permet de déterminer les poids des connexions et des entrées des neurones, ils sont donnés dans les expressions suivantes :

$$T_{i,j} = -2 \cdot \overline{\delta_{i,j}} \qquad \forall\, i = 1 \ldots N,\ \forall\, j = 1 \ldots N \tag{5}$$

$$I_i = 2k - 1 \qquad \forall\, i = 1 \ldots N \tag{6}$$

Avec $\overline{\delta_{i,j}}$ le complément du symbole de Kronecker qui prend la valeur 0 si $i = j$, et 1 sinon.

Sur la base de la règle $k-de-N$ et en utilisant le caractère additif du modèle de Hopfield, le problème d'ordonnancement pour architecture mono-processeur est simple à modéliser. Toutefois, cette simplicité cache un problème délicat de convergence du réseau. En effet, comme l'ont montré les auteurs de [1], l'application successive de plusieurs règles de construction provoque l'apparition de minima locaux dans la fonction d'énergie. Ces minima conduisent alors le réseau à converger vers des états stables qui ne sont pas des solutions valides au sens du problème à résoudre. Pour parvenir à faire converger le réseau vers une solution satisfaisante, il est alors nécessaire

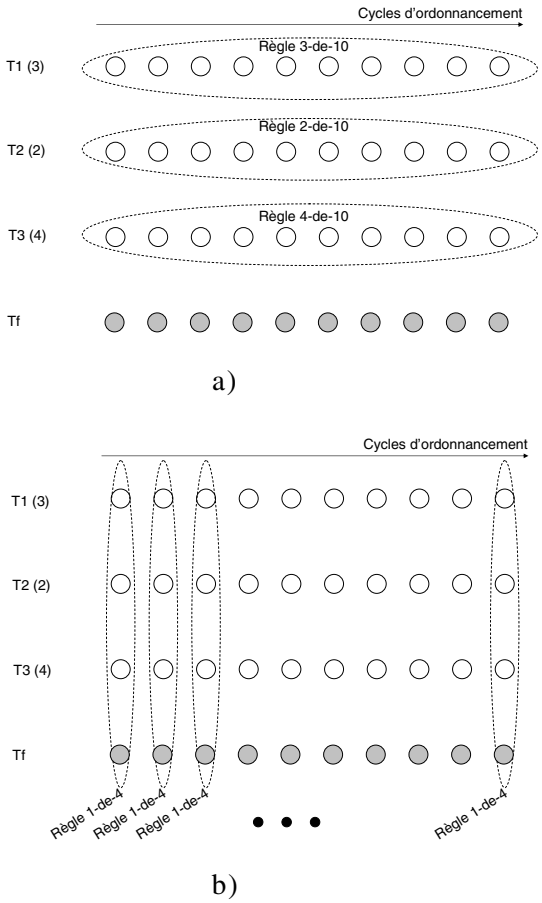


Figure 3. a) Application de la règle $k-de-N$ sur les lignes du réseau de neurones (avec $k = C_i$ charge de la tâche i sur la cible d'exécution) ; b) Application de la règle $k-de-N$ sur les colonnes du réseau de neurones (avec $k = 1$ dans le cas d'un processeur exécutant une seule tâche à la fois).

d'effectuer un apport d'énergie dans le réseau afin de l'extraire de ces minima. Cela se traduit par un mécanisme de contrôle complexe, mettant en œuvre une détection de stabilisation sur un minimum local, puis une ré-initialisation du réseau afin de relancer le cycle de convergence.

Ce modèle de réseau de neurones peut être utilisé dans le cas d'architectures disposant de plusieurs cibles d'exécution de même type : architectures dites homogènes. Pour cela, il suffit d'appliquer une règle $k-de-N$ verticale avec k égal au nombre de cibles d'exécution et en plaçant autant de tâches fictives qu'il existe de cibles d'exécution. Toutefois, ce modèle souffre des mêmes problèmes de convergence que dans le cas mono-processeur et nécessite donc un mécanisme de contrôle complexe.

Dans le cas des architectures disposant de plusieurs cibles d'exécution hétérogènes, ce modèle ne peut plus être utilisé. Or, comme nous l'indiquions dans l'introduction, les SoC sont très largement développés selon un modèle d'architectures hétérogènes. Pour prendre en compte ce type d'architectures, nous avons proposé, dans [19, 20], une extension à ce modèle de réseaux de neurones. L'idée générale consiste à formuler que si une tâche est ordonnancée une fois sur une cible d'exécution alors elle ne peut pas être à nouveau ordonnancée sur une autre cible pour la même période. En termes de règle de construction, il s'agit d'étendre la règle de $k-de-N$ vers une nouvelle règle appelée règle $(0-ou-k)-de-N$. Cette règle permet de préciser que le nombre de neurones actifs pour une ligne i et un plan de cible d'exécution p est soit égal à zéro, soit égal à $C_{i,p}$ (avec $C_{i,p}$ la charge de la tâche i sur le plan d'exécution p). Pour cela, le réseau de neurones est étendu et des neurones dits *cachés* sont ajoutés pour assurer la convergence. Sur la figure 4, les neurones cachés sont ceux placés dans les zones grisées et sont dépendants des charges de calculs des tâches sur chacune des cibles d'exécution.

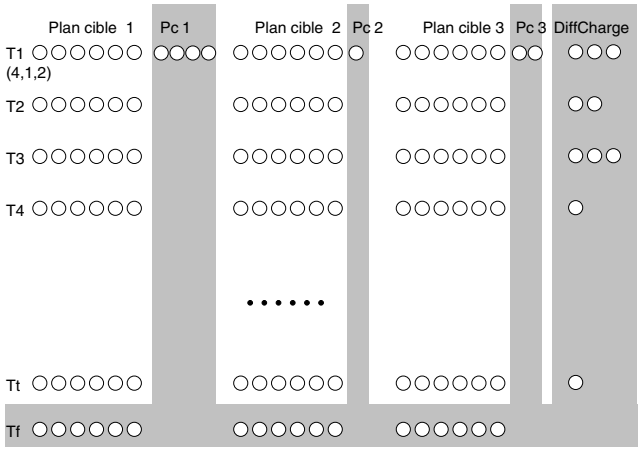


Figure 4. Réseau de neurones modélisant le problème d'ordonnancement pour architectures hétérogènes. La figure illustre la modélisation d'un système composé de t tâches. Chaque tâche dispose d'une charge d'exécution spécifique pour chaque cible d'exécution, par exemple la tâche $T1$ a une charge de 4 sur la cible d'exécution 1, de 1 sur la cible d'exécution 2 et de 2 sur la cible d'exécution 3. Les neurones placés dans les zones grisées nommées Pc_i permettent d'assurer la convergence des règles $(0-ou-k)-de-N$ appliquées sur le plan i. Les neurones placés dans la zone grisée nommée $DiffCharge$ permettent de modéliser l'instanciation de la tâche sur un seul des plans.

Sur cette figure, un certain nombre de neurones *cachés* sont nécessaires pour assurer une convergence correcte. Ce nombre de neurones cachés évolue linéairement en fonction du nombre de tâches mais dépend aussi des caractéristiques des tâches. Pour des applications complexes, le nombre global de neurones pour la modélisation du problème est alors un point critique pour assurer une convergence rapide mais aussi pour envisager une implémentation matérielle peu coûteuse.

Les principales limitations de cette nouvelle proposition concernent le nombre important de neurones pour la modélisation du

problème ainsi que la nécessité de ré-initialiser un grand nombre de fois le réseau pour s'assurer de sa convergence vers une solution correcte. On retrouve ici le problème de l'additivité des règles, qui est d'ailleurs plus important puisque le nombre de règles utilisées est accru. En effet, on peut noter que plus le nombre de règles « empilées » les unes sur les autres est important, plus le nombre de minima locaux augmente et moins la probabilité de converger vers une solution correcte est grande. Ce point est particulièrement délicat en ce qui concerne le contrôle du réseau, puisqu'il faut être en mesure de détecter la stabilisation du réseau, de rejeter une solution non valide, d'effectuer un apport d'énergie, et de relancer une convergence du réseau.

3. Notre proposition

L'originalité de notre proposition repose sur la réduction du nombre de neurones ainsi que sur l'absence de ré-initialisation du réseau pour assurer sa convergence vers une solution correcte. L'idée générale qui a été exploitée dans les travaux précédents (une tâche ordonnancée une fois sur une cible d'exécution ne peut pas être ordonnancée sur une autre cible d'exécution pour la même période) est ici exploitée d'une façon différente au travers de la mise en place de neurones inhibiteurs. Le rôle de ces neurones consiste à « capter » l'instanciation d'une tâche sur une cible d'exécution, et « d'inhiber » alors l'instanciation de cette même tâche sur une autre cible d'exécution [21, 22].

Le principe de fonctionnement d'un neurone inhibiteur s'appuie sur une connectique particulière d'un neurone spécifique (le neurone inhibiteur) avec les neurones à inhiber. La figure 5 représente l'ensemble des neurones de la tâche i pour la ressource d'exécution j. Pour faciliter la compréhension, nous groupons les N neurones modélisant l'ordonnancement de la tâche i sur la ressource j dans l'ensemble $\varepsilon_{i,j} = \{n_{i,j,1}, n_{i,j,2}, \ldots n_{i,j,N}\}$. L'application est composée de t tâches et l'architecture contient P ressources d'exécution. La règle $k-de-N$ est appliquée sur l'ensemble de neurones $\varepsilon_{i,j}$, avec $k = C_{i,j}$, et $C_{i,j}$ la charge de calcul de la tâche i sur la ressource j. En connectant les neurones $n_{i,j,k}$ au neurone inhibiteur $nh_{i,j}$ avec un poids égal à 1, alors dès que l'ensemble $\varepsilon_{i,j}$ dispose de $C_{i,j}$ neurones actifs le neurone inhibiteur associé $nh_{i,j}$ peut devenir actif. En effet, en fixant l'entrée du neurone inhibiteur à $Ih_{i,j} = -C_{i,j} + 1$ alors son état $xh_{i,j}$ est donné par :

$$xh_{i,j} = Ih_{i,j} + \sum_{k=1}^{N} x_{i,j,k} \cdot T_{(n_{i,j,k};nh_{i,j})} \qquad (7)$$

avec $xh_{i,j}$ l'état du neurone inhibiteur $nh_{i,j}$, $T_{(n_{i,j,k};nh_{i,j})}$ le poids de la connexion entre le neurone $n_{i,j,k}$ et le neurone inhibiteur $nh_{i,j}$. Les poids de connexion $T_{(n_{i,j,k};nh_{i,j})}$ étant tous égaux à 1 alors l'état $xh_{i,j}$ du neurone inhibiteur $nh_{i,j}$ est donné par :

$$xh_{i,j} = -C_{i,j} + 1 + \sum_{k=1}^{N} x_{i,j,k} \qquad (8)$$

L'application de la règle $k-de-N$ sur l'ensemble de neurones $\varepsilon_{i,j}$ (avec $k = C_{i,j}$) conduit l'état $xh_{i,j}$ du neurone inhibiteur $nh_{i,j}$ à la valeur 1 lorsque $C_{i,j}$ neurones sont actifs dans l'ensemble $\varepsilon_{i,j}$, c'est-à-dire lorsque la tâche i est entièrement ordonnancée sur la ressource d'exécution j, donc que $\sum_{k=1}^{N} x_{i,j,k} = C_{i,j}$ L'activation d'un neurone $nh_{i,j}$ doit alors inhiber l'activation des neurones des autres ressources de calcul. Cela est assuré en fixant les poids $T_{(nh_{i,j};n_{i,l,k})}$ de connexion du neurone inhibiteur $nh_{i,j}$ avec les autres neurones de la tâche i à une valeur négative suffisamment grande. Sur la figure 5, si le neurone inhibiteur $nh_{i,j}$ a été activé par l'ordonnancement de la tâche i sur la ressource j, alors les neurones de la ressource i ne peuvent plus être activés pour les autres ressources l dès lors que le poids $T_{(nh_{i,j};n_{i,l,k})}$ est fortement négatif. Pour assurer ce fonctionnement, il faut fixer la valeur de $T_{(nh_{i,j};n_{i,l,k})}$ à une valeur telle que $T_{(nh_{i,j};n_{i,l,k})} \leq -I_{i,l} \ \ \forall l \neq j$.

En associant un neurone inhibiteur à chaque ressource d'exécution, il est alors possible de s'assurer de la convergence du réseau vers une solution n'ayant qu'un seul ensemble $\varepsilon_{i,j}$ de neurones actifs. La modélisation globale peut alors être représentée par le schéma de la figure 6. Dans ce schéma, les neurones placés dans les zones grisées (NH_j) sont les neurones inhibiteurs de la ressource d'exécution j. Par rapport à la solution définie dans [20] (voir figure 4), le nombre de neurones cachés est plus faible et surtout ne dépend pas de la charge des tâches sur la cible d'exécution mais est fixe et égal à 1 par tâche et par cible d'exécution. La connectique globale de la structure est donnée sur la figure 7.

Contrairement aux modélisations proposées précédemment, cette dernière proposition est basée sur une connectique non symétrique. Cette non symétrie concerne toutes les connexions qui mettent en jeu les neurones inhibiteurs. En effet, un neurone inhibiteur $nh_{i,j}$ dépend de l'état des neurones correspondant à la tâche i et au plan j. Par contre, il n'existe pas de connexion du neurone $nh_{i,j}$ vers les neurones de la tâche i de la ressource d'exécution j, ce qui se traduit par $T_{(nh_{i,j};n_{i,j,k})} = 0$, alors que $T_{(n_{i,j,k};nh_{i,j})} = 1$. Cette non symétrie des connexions est importante puisqu'elle remet en cause les conditions de stabilité et de convergence définies par Hopfield (notamment la symétrie de la matrice de connexions). Toutefois, comme nous l'expliquons ci-dessous, la convergence du réseau est bien assurée. Celle-ci peut être simplement expliquée de la façon suivante.

1. À l'état initial, le système est dans un état tel que tous les neurones inhibiteurs $nh_{i,j}$ sont inactifs ;

2. À partir de cet état, chaque ensemble $\varepsilon_{i,j}$ de neurones (correspondant à une ressource d'exécution) va avoir tendance à converger vers un état tel que $C_{i,j}$ neurones deviennent actifs (règle $k-de-N$ sur les neurones du plan j, avec $k = C_{i,j}$) ;

3. À partir du moment où l'un des ensembles $\varepsilon_{i,j}$ dispose de ses $C_{i,j}$ neurones actifs, si le neurone inhibiteur $nh_{i,j}$ est évalué,

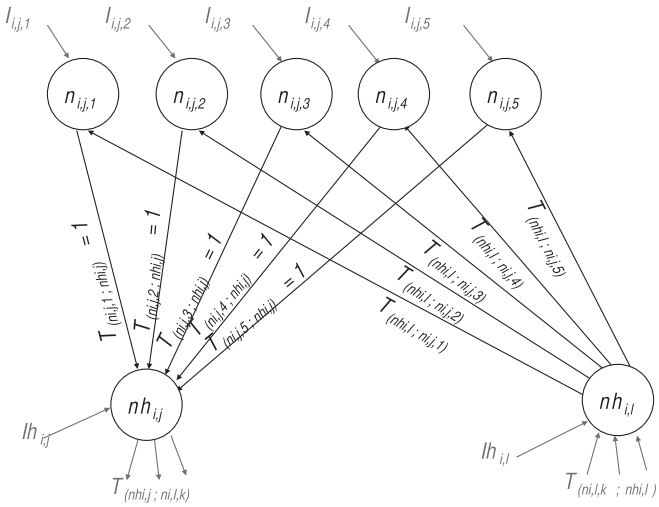


Figure 5. Principe de mise en œuvre d'un neurone inhibiteur. Un neurone inhibiteur est associé à chaque cible d'exécution. Son rôle consiste à capter l'instanciation d'une tâche sur la cible et à inhiber l'instanciation sur une autre cible.

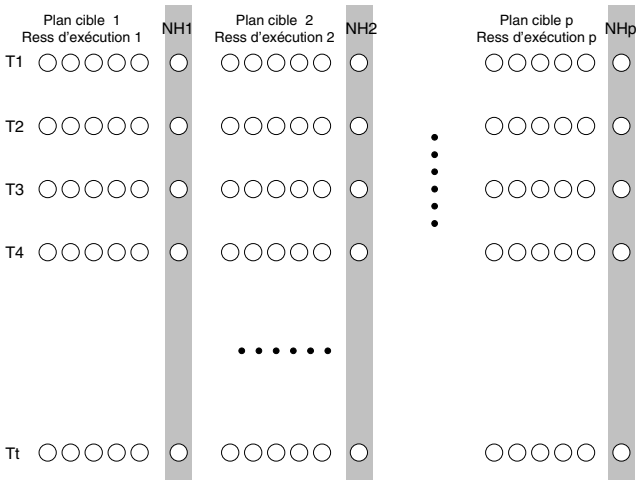


Figure 6. Modélisation du problème d'ordonnancement avec neurones inhibiteurs. Les neurones inhibiteurs sont placés dans les zones grisées. Cette modélisation nécessite un neurone inhibiteur par tâche et par ressource de calcul.

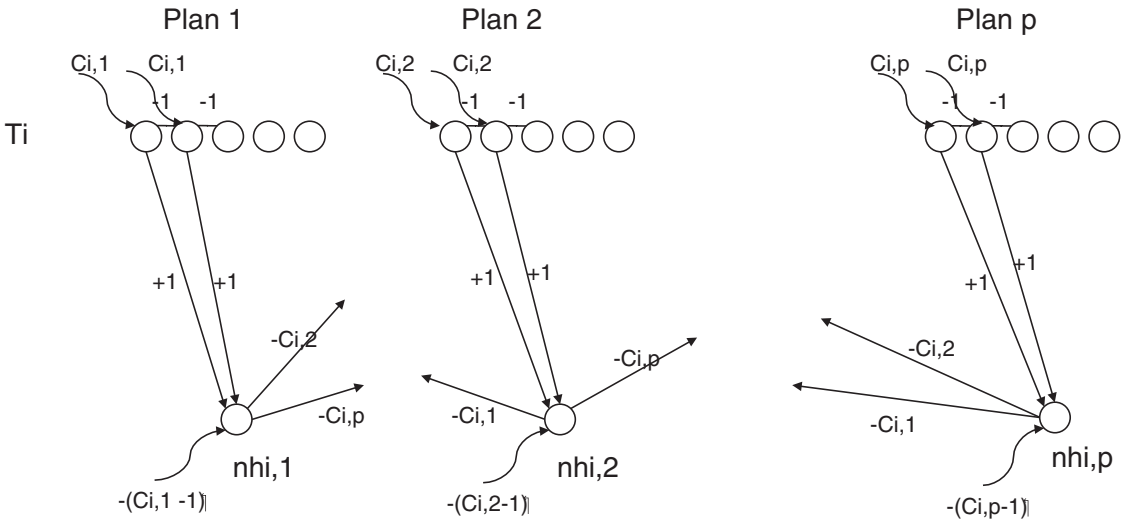


Figure 7. Modélisation de la connectique du problème d'ordonnancement avec neurones inhibiteurs. (Pour des raisons de lisibilité, les neurones inhibiteurs ont été décalés vers le bas).

alors il deviendra actif, c'est d'ailleurs la seule combinaison qui peut amener à l'activation de l'un des neurones inhibiteurs ;
4. Lorsqu'un seul neurone inhibiteur est actif, le poids de ses connexions avec tous les autres neurones des autres ressources d'exécution annule le poids de l'entrée de ces neurones ;
5. Dans ce cas, lorsque les neurones des autres plans d'exécution seront évalués, ils resteront/passeront dans un état inactif ;
6. Finalement, le système va converger vers un état tel que $C_{i,j}$ neurones de l'ensemble $\varepsilon_{i,j}$, dont le neurone inhibiteur $nh_{i,j}$ est actif, seront dans l'état actif, et les neurones de tous les autres plans seront dans l'état inactif.

4. Résultats

Afin de valider notre proposition, nous avons développé un simulateur de réseaux de neurones. Le point d'entrée de notre simulateur est une spécification des caractéristiques des tâches de l'application (voir tableau 1.b).
Nous avons appliqué la méthode de construction présentée dans [20] sur une architecture composée de deux types de ressources, chaque type disposant de deux cibles d'exécution (le modèle de l'architecture de test est présenté à la figure 8). Les résultats de simulations sur un ensemble de tâches indépendantes variant de 2 à 7 sont donnés dans le tableau 2.

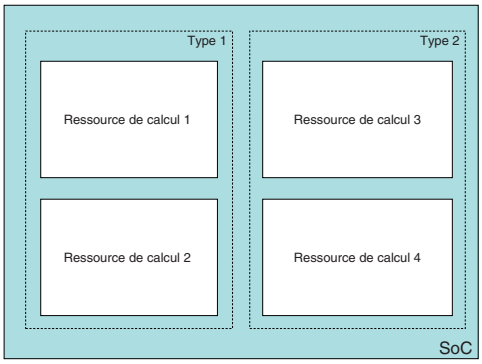


Figure 8. Modèle de l'architecture de test utilisée pour illustrer notre proposition.

Ces résultats montrent clairement que le nombre neurones pour modéliser le problème est important (ligne « surcoût »). De même, ce tableau montre que ce modèle nécessite un nombre de ré-initialisations du réseau qui rend son utilisation quasiment impossible dans le contexte des SoC où, rappelons le, nous cherchons à calculer l'ordonnancement en cours d'exécution de l'application (en ligne).
En utilisant notre structure de réseau de neurones, nous obtenons des résultats d'ordonnancement tels que ceux présentés à la figure 9. Globalement, notre proposition permet de réduire le coût du réseau. En effet, les résultats présentés dans le tableau 3 montrent que le nombre de neurones pour la modélisation du problème reste faible et surtout varie linéairement par rapport au nombre de tâches (pour un nombre de cycles d'ordonnancement donné).
Notons que dans les propositions précédentes, notamment dans [1], beaucoup de neurones *cachés* sont ajoutés à la modélisation, ce qui conduit, entre autres, à un ralentissement de la convergence. Ici, le surcoût engendré par les neurones inhibiteurs est de l'ordre de 10 % ce qui reste maîtrisé. Dans [21], nous avons comparé l'efficacité de la convergence des solutions classiques avec notre proposition et nous avons montré que

Tableau 1. a) Caractéristiques d'exécution des tâches pour les deux types de ressources de calcul ; b) Point d'entrée de notre outil de simulation de réseaux de neurones.

	Charges	
Tâches	$C_{i,1}$	$C_{i,2}$
T_1	1	2
T_2	2	1
T_3	4	2
T_4	3	5
T_5	4	6
T_6	3	2
T_7	2	3

(a)

```
Taches T1 T2 T3 T4 T5 T6 T7
Plans P1 P2
NbProc 2 2
NbCycles 10
ChargesParPlan T1 1 2
ChargesParPlan T2 2 1
ChargesParPlan T3 4 2
ChargesParPlan T4 3 5
ChargesParPlan T5 4 6
ChargesParPlan T6 3 2
ChargesParPlan T7 2 3
```

(b)

Tableau 2. Résultats de convergence d'un réseau de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de 2 plans d'exécution ayant chacun de 2 ressources d'exécution et pour un ordonnancement à réaliser sur 10 cycles. Le modèle de réseau de neurones est celui développé dans [20]. Notons que pour chaque ré-initialisation, une dizaine d'évaluations de neurones sont nécessaires pour stabiliser le réseau.

Nombre de tâches		2	3	4	5	6	7
Nb neurones	(N_1)	88	116	146	178	204	230
Nb utile neurones	(N_2)	40	60	80	100	120	140
Surcoût en neurones cachés	(N_1/N_2)	2,2	1,93	1,82	1,78	1,7	1,64
Moyenne ré-initiatlisations	(R_1)	27	167	750	4556	9842	14289

Tableau 3. a) Résultats de convergence d'un réseau de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de deux ressources d'exécution. Le modèle utilisé pour ces résultats s'appuie sur la mise en place des neurones inhibiteurs ; b) Facteurs de réduction du nombre de neurones et du nombre de ré-initialisations obtenus par notre proposition par rapport à la proposition développée dans [20]. Pour obtenir le facteur de réduction du nombre d'évaluations, il faut tenir compte du fait que le modèle défini dans [20] nécessite une dizaine d'évaluations de neurones pour chaque ré-initialisation.

Nombre de tâches		2	3	4	5	6	7
Nb neurones	(N_3)	44	66	88	110	132	154
Nb utile neurones		40	60	80	100	120	140
Moyenne évaluations	(E_2)	168	267	518	582	797	972
Surcoût en neurones cachés		1,1					

(a)

Facteur de réduction du nombre de neurones	($\frac{N_1}{N_3}$)	2	1,75	1,66	1,62	1,55	1,49
Facteur de réduction du nombre d'évaluations	($\frac{10\times R_1}{E_2}$)	1,6	6,25	14,5	78,3	123	147

(b)

cette dernière était plus efficace. Le tableau 3.b reprend ces résultats et comme on peut le voir, le nombre de neurones nécessaires pour modéliser le problème est assez fortement réduit, variant d'un facteur 1,5 à 2. Le nombre de neurones, même s'il est diminué fortement, reste un obstacle important en vue d'une intégration matérielle du service d'ordonnancement sur une architecture de type SoC. Si le coût matériel de l'implémentation est un enjeu important, le temps de convergence l'est tout autant. Sur ce point, notre solution apporte une amélioration beaucoup plus significative. En effet, la réduction du nombre d'évaluations des neurones dépasse les 95 % pour un système d'une complexité limitée. Notons d'ailleurs que cette réduction s'accroît lorsque la complexité de l'application augmente. Ce point est un élément important qui nous conforte dans l'idée que ce type de structure est particulièrement bien adaptée à une plate forme matérielle de type SoC.

5. Conclusion

Nous avons présenté une structure de réseaux de neurones pour le problème de l'ordonnancement de tâches pour des architectures hétérogènes de type SoC. Notre proposition s'appuie sur une connectique particulière entre neurones et la mise en place de neurones inhibiteurs. Nous avons validé notre proposition au travers de simulations et nous avons montré qu'un tel réseau permet de limiter le nombre de neurones nécessaires à la modélisation du problème. L'apport principal de notre proposition concerne la limitation très forte du nombre de cycles pour assurer la convergence du réseau et aussi par l'assurance d'une convergence systématique vers une solution correcte, c'est-à-dire supprimant les convergences vers des minima locaux de la fonction d'énergie. Ce dernier point est très important puisqu'il

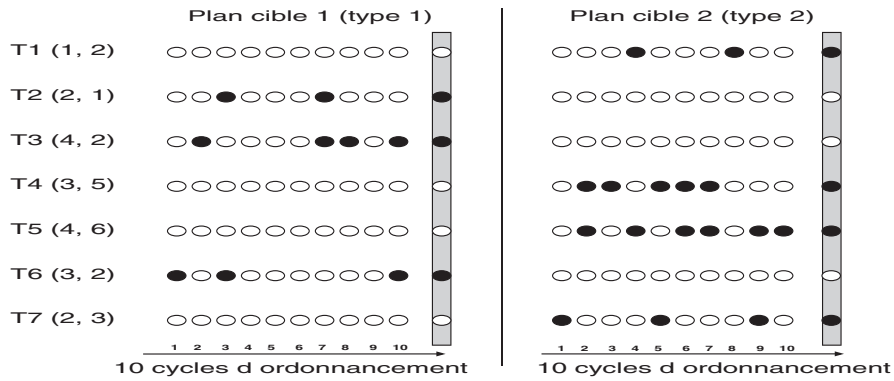


Figure 9. Exemple de convergence obtenue avec notre structure de réseau de neurones. Chaque tâche est instanciée une fois et une seule sur l'une des cibles d'exécution. La présence de deux cibles d'exécution par plan peut permettre l'ordonnancement simultané de deux à quatre tâches par cycle (c'est le cas par exemple du cycle 7 durant lequel les tâches 2, 3, 4 et 5 sont ordonnancées).

permet de se passer des nombreuses ré-initialisations du réseau de neurones qui sont nécessaires dans le cas du modèle de Hopfield.

À partir de ce modèle de réseaux de neurones, nous avons entamé des travaux de description VHDL afin d'évaluer le coût d'une implémentation matérielle d'une telle structure. À partir des caractéristiques de l'application, nous produisons une description VHDL synthétisable du réseau de neurones. Nos premiers résultats de synthèse sur circuits FPGA confirment que la convergence est atteinte en quelques cycles pour une complexité globale du réseau qui reste raisonnable. Si ces premiers résultats d'implémentation sont encourageants, l'implémentation efficace d'un ordonnanceur complet dans un SoC reste relativement difficile, compte tenu du nombre de neurones qui est encore important dans cette modélisation. Toutefois, moyennant une réduction plus forte de ce nombre de neurones, nous pensons que le service d'ordonnancement pourra faire l'objet d'une implémentation matérielle efficace dans les années à venir. En effet, le parallélisme implicite de ce type de structures se prête, *a priori*, très bien à une implémentation matérielle. De plus, le calcul élémentaire (évaluation d'un neurone) est un calcul simple dont le coût d'implémentation doit pouvoir être maîtrisé. Nous pensons que, dans le contexte des architectures de SoC, ce type de structures exhibe des caractéristiques intéressantes et devrait permettre de « calculer » l'ordonnancement en ligne de quelques dizaines de tâches.

Références

[1] C. CARDEIRA and Z. MAMMERI, Preemptive and non-preemptive real-time scheduling based on neural networks, In *Proc. of Distributed Computer Control Systems*, pages 67-72, Toulouse, France, September 1995.

[2] C. LIU and J. LAYLAND, Scheduling algorithms for multiprogramming in a hard real-time environment, *J. of the ACM*, 20(1):46-61, 1973.

[3] J. Y.-T. LEUNG and J. WHITEHEAD, On the complexity of fixed-priority scheduling of periodic real-time tasks, *Performance Evaluation 2*, 2:237-250, 1982.

[4] K. SCHILD and J. WÜRTZ, Off-line scheduling of a real-time system. In K. M. George, editor, *Proceedings of the 1998 ACM Symposium on Applied Computing, SAC98*, pages 29-38, Atlanta, Georgia, 1998. ACM Press.

[5] N. MEGOW and A.S. SCHULZ, On-line scheduling to minimize average completion time revisited, *Operations Research Letters*, 32(5) :485-490, 2004.

[6] X. LU, R. SITTERS, and L. STOUGIE, A class of on-line scheduling algorithms to minimize total completion time, *Oper. Res. Lett.*, 31(3) :232-236, 2003.

[7] M. YOUNG and L.C. SHU, Hybrid online/offline scheduling for hard real-time systems, Technical Report SERC-TR-100-P, 1991.

[8] A. SRINIVASAN, P. HOLMAN, J.H. ANDERSON, and S. BARUAH, The case for fair multiprocessor scheduling. In *Proc. of the 17th International Symposium on Parallel and Distributed Processing*, page 114, Washington, DC, USA, 2003.

[9] S. BARUAH and N. FISHER, The partitioned multiprocessor scheduling of sporadic task systems. In *Proc. of the 26th IEEE International Real-Time Systems Symposium*, pages 321-329, Washington, DC, USA, 2005.

[10] S. BARUAH and N. FISHER, The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems, *IEEE Trans. Comput.*, 55(7):918-923, 2006.

[11] A. MORTON and W.M. LOUCKS, A hardware/software kernel for system on chip designs. In *Proceedings of the ACM Symposium on Applied Computing*, pages 869-875, 2004.

[12] P. KOHOUT, B. GANESH, and B. JACOB, Hardware support for real-time operating systems. In *CODES+ISSS '03 : Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 45-51, New York, NY, USA, october 2003. ACM Press.

[13] P. KUACHAROEN, M. SHALAN, and V. MOONEY III, A configurable hardware scheduler for real-time systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pages 96-101, Las Vegas, USA, June 2003.

[14] C. CARDEIRA and Z. MAMMERI, Neural network versus max-flow algorithms for multiprocessor real-time scheduling, *IEEE Proceedings of EURWRTS*, pages 175-180, 1996.

[15] C. CARDEIRA, M. SILVA, and Z. MAMMERI, Handling precedence constraints with neural network based real-time scheduling algorithms. In *Proc. of the 9th Euromicro Workshop on Real Time Systems*, pages 207-214, Toldeo, Spain, june 1997.

[16] J. J. HOPFIELD and D. W. TANK, Neural computation of decisions in optimization problems, *Biological Cybernetics*, 52:141-52, 1985.

[17] M.A. 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.

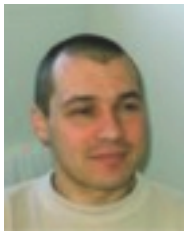
[18] S. GROSSBERG, *Studies of Mind and Brain : Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*, volume 70. Reidel Press Bonston, 1982.

[19] I. BENKERMI, S. PILLEMENT, and O. SENTIEYS, Application des réseaux de neurones à l'ordonnancement de tâches temps réel sur une architecture multiprocesseurs hétérogènes, *SympAAA'2003*, 14-17 Octobre 2003.

[20] I. BENKERMI, *Modèle et algorithme d'ordonnancement pour architectures reconfigurables dynamiquement*. PhD thesis, ENSSAT-Université de Rennes, 2007.

[21] D. CHILLET, S. PILLEMENT, and O. SENTIEYS, A neural network model for real-time scheduling on heterogeneous soc architectures. In *International Joint Conference on Neural Networks, IJCNN 2007*, Orlando, Floride, august, 12-17 2007.

[22] D. CHILLET, S. PILLEMENT, and O. SENTIEYS, Vers une implémentation matérielle d'un réseau de neurones pour le service d'ordonnancement des tâches au sein d'un soc. In *GRETSI 2007*, 2007.



Daniel **Chillet**

Daniel Chillet, docteur de l'université de Rennes 1 en 1997, est maître de conférences à l'ENSSAT de l'Université de Rennes 1. Ses activités de recherche portent sur deux domaines qui sont d'une part, la définition de méthodologies de conception des unités de mémorisation pour des systèmes sur puce et, d'autre part, la prise en compte des aspects reconfiguration dynamique au sein de ces mêmes systèmes. Dans ce double contexte, il s'intéresse notamment à l'impact de la reconfiguration sur les unités de mémorisation ainsi qu'à la gestion efficace de la dynamicité de la reconfiguration par le système d'exploitation.

Sébastien **Pillement**

Sébastien Pillement est docteur de l'Université de Montpellier II (1998) en systèmes informatiques, automatiques et micro-électronique. Il est actuellement maître de conférences à l'Université de Rennes I, antenne de Lannion. Il est rattaché à l'Institut de Recherche en Informatique et Systèmes Aléatoires pour les aspects recherche. Ses activités de recherche concernent les méthodologies de conception des systèmes complexes et plus particulièrement les systèmes reconfigurables. Travaillant sur l'ensemble des aspects architecturaux, il s'intéresse aux aspects gestion dynamique des architectures reconfigurables dynamiquement, à l'implémentation matérielle de nouvelles architecture et au support de communications flexible.

Olivier **Sentieys**

Olivier Sentieys, docteur de l'université de Rennes 1 en 1993, est professeur à l'ENSSAT (Lannion) depuis 2002. Ses travaux de recherche portent sur les méthodes de conception d'architectures et de circuits intégrés pour le traitement du signal et les télécommunications mobiles. En particulier, il aborde actuellement les thèmes du calcul reconfigurable (architectures et outils pour la compilation et la synthèse), des circuits à faible consommation, des architectures spécifiques et de l'arithmétique en virgule fixe. Il est responsable de l'équipe de recherche CAIRN de l'IRISA.