Vers une implémentation matérielle d'un réseau de neurones pour le service d'ordonnancement de tâches au sein d'un SoC

Daniel Chillet, Sébastien Pillement, Olivier Sentieys

ENSSAT - Université de Rennes 1 - IRISA - Equipe R2D2
BP 80518 - 6 Rue de Kerampont - 22305 LANNION - FRANCE
Daniel.Chillet@enssat.fr, Sebastien.Pillement@enssat.fr, Olivier.Sentieys@enssat.fr

Résumé – Ce papier présente nos travaux de modélisation du problème d'ordonnancement de tâches pour des architectures hétérogènes à partir de réseaux de neurones. Les modélisations classiques, basées sur le modèle de Hopfield, nécessitent un grand nombre de neurones et posent des difficultés de convergence. Pour résoudre ces deux problèmes, nous proposons l'insertion de neurones inhibiteurs afin de limiter le nombre global de neurones nécessaires pour la modélisation du problème mais aussi pour assurer la convergence systématique du réseau vers une solution correcte. Dans l'objectif d'une implémentation efficace de ce service au sein des SoC, la maîtrise de ces deux points est un élément important pour limiter le coût matériel et assurer une rapidité de convergence.

Abstract – This paper presents our work on extending artificial neural networks use for real-time task scheduling to heterogeneous System-on-Chip architectures. The classical neural network modelizations need a high number of neurons and a lot of re-initialisations to ensure the convergence. To define efficient implementation of scheduling service for SoC architecture, we propose to include inhibitor neurons in the network. These neurons ensure rapid convergence and limit the number of needed neurons to modelized the scheduling problem, these two points are important to limit the hardware cost.

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). Un SoC intègre en général un cœur de processeur généraliste, des blocs spécifiques (IP) pour des traitements critiques, un ou plusieurs cœurs de processeurs de traitement du signal et de plus en plus fréquemment un élément reconfigurable. Sur la base de cette organisation générale, ces architectures offrent désormais des capacités d'intégration de systèmes complexes permettant de supporter la flexibilité applicative nécessaire. 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. L'ordonnancement consiste alors à définir les instants durant lesquels les tâches s'exécutent ainsi que les différentes cibles d'exécution. 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, parmi ceux là, les travaux de Cardeira 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. 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 ré-initialisations importants pour s'assurer de la convergence du réseau vers un état représentant une solution correcte.

Dans cet article, nous proposons une structure de réseaux de neurones différentes qui permet, d'une part une limitation importante du nombre global de neurones, et d'autre part qui assure 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 à la modélisation du problème d'ordonnancement. La section 3 présente notre solution qui s'appuie sur la mise en place de neurones inhibiteurs. La section 4 illustre notre proposition à partir d'un ensemble d'expérimentations pour lesquelles nous faisons varier le nombre de tâches. Finalement, la section 5, conclue ce papier et présente les perspectives de nos travaux.

2 État de l'art & modélisations classiques

Plusieurs propositions d'ordonnancement pour architectures multi-processeurs ont été développées et sont présentées dans la littérature [1, 15]. Des algorithmes proposant des solutions optimales ont été proposées [17], de même que des solutions de partitionnement pour machines multi-

processeurs [2, 3], mais il s'agit principalement de solutions pour des machines homogènes.

En règle générale, ces solutions sont difficilement applicables au contexte des architectures des SoC à cause 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 certains nombres de travaux vers l'étude de l'implantation de services sous forme matérielle [16, 13, 14]. Parmi les nombreuses solutions qui ont été proposées pour tenter de résoudre cette problématique, l'une d'elle repose sur les réseaux de neurones [7, 8]. Le modèle sous-jacent est basé sur la proposition de Hopfield [12] dont les auteurs de [18] proposent une règle de construction permettant de définir les poids de connexions entre neurones ainsi que les poids des entrées. Dans [6], les auteurs proposent une modélisation par réseaux de neurones du problème d'ordonnancement pour architectures homogènes. Cette proposition a été étendue vers les architectures hétérogènes par [4, 5]. Les principales limitations de ces dernières propositions concernent, une nouvelle fois, 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.

Les modélisations classiques du problème d'ordonnancement se basent sur la mise en place d'une structure où un neurone correspond à un cycle d'ordonnancement pour une tâche particulière [6]. La structure générale du réseau repose sur le modèle de Hopfield [12]. L'ordonnancement d'une tâche à un cycle donné est alors défini par l'activation d'un neurone. La définition du réseau de neurones passe traditionnellement par trois principales étapes qui sont :

- la modélisation du problème de telle sorte que les états des neurones définissent une solution possible ;
- la définition d'une 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.

Il a été démontré que sous certaines conditions, l'évolution du réseau ainsi construit va naturellement conduire à la minimisation de ladite fonction d'énergie [10, 11]. On dit alors que le réseau converge vers un état stable, et cet état est à énergie minimale (égale à 0). Pour construire le réseau, [18] propose de s'appuyer sur une règle de construction du réseau appelée règle `k-de-N`. Cette règle permet de s'assurer que `k` neurones parmi `N` seront actifs lorsque le réseau aura convergé. La modélisation du problème complet d'ordonnancement passe par l'application de plusieurs règles `k-de-N` sur le réseau de neurones, or l'additivité, inhérente au modèle de Hopfield, des différentes règles provoque l'apparition de minima locaux. Ces minima locaux sont à l'origine des nombreuses séquences de ré-initilisations du réseau puisqu'ils provoquent la stabilisation du réseau vers des états qui ne sont pas des solutions correctes.

Dans [5], l'auteur montre comment l'ajout d'une règle `(0-ou-k)-de-N` permet de couvrir le problème de l'ordonnancement de tâches pour des architectures hétérogènes. La structure proposée est présentée sur la figure 1.

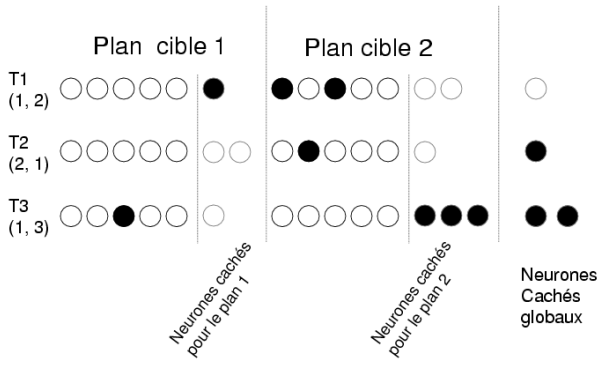

FIG. 1 – Modélisation du problème de l'ordonnancement pour des architectures hétérogènes [5]

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.

3 Notre proposition

L'originalité de notre proposition repose sur la réduction du nombre de neurones ainsi que sur l'absence de réinitialisation du réseau pour assurer sa convergence vers une solution correcte. L'idée générale de notre proposition consiste à s'assurer que si une tâche est ordonnancée une fois sur une cible d'exécution alors elle ne peut pas l'être sur une autre cible pour la même période. Pour parvenir à cela, nous proposons la mise en place de neurones inhibiteurs qui "captent" l'instanciation de la tâche sur une cible d'exécution, et qui bloquent alors l'instanciation sur une autre cible. La modélisation globale peut alors être représentée par le schéma de la figure 2.

Sur cette figure, seuls les neurones d'une tâche (tâche T_i) sont représentés. Ces neurones représentent les cycles d'ordonnancement possibles de ladite tâche. Le `Plan cible 1` correspond alors aux cycles `1, 2, ..., 5`. Les neurones notés $Nh_{i,p}$ sont les neurones inhibiteurs des différents plans d'exécution. Le neurone inhibiteur $Nh_{i,p}$ est connecté à l'ensemble des neurones du plan p de telle façon qu'il devienne actif lorsque le nombre de neurones du plan p a atteint la valeur souhaitée (charge de la tâche T_i sur le plan p : $C_{i,p}$). Une fois l'activation d'un neurone inhibiteur obtenue, la connectique avec tous les autres neurones de tous les autres plans est définie telle que lesdits neurones vont converger vers un état inactif.

L'évolution du réseau se comporte alors de la façon sui-

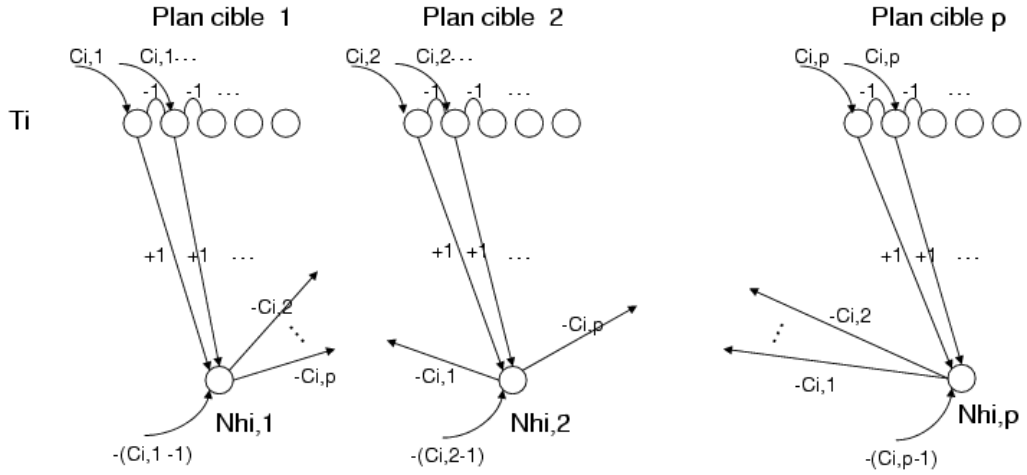

FIG. 2 – Modélisation de la connectique avec neurones inhibiteurs. (Pour des raisons de lisibilité, les neurones inhibiteurs ont été décalés vers le bas).

	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)

Nombre moyen d'évaluations de neurones						
Nombre de tâches	2	3	4	5	6	7
Nb neurones	84	126	168	210	252	294
Nb utile neurones	80	120	160	200	240	280
Moyenne évaluations	339	539	1025	1170	1583	1951
Surcoût en neurones cachés	5%)					

b)

Réduction nb neurones	37%	34%	34%	32,7%	30,8%	30%
Réduction nb évaluations	32%	70%	84,6%	98,4%	95,8%	96,8%

c)

TAB. 1 – a) Description des tâches ; b) Résultats de convergence des réseaux de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de deux ressources d'exécution ; c) Réductions du nombre de neurones ainsi que du nombre de ré-initialisations apportées par notre proposition.

vante :

- À l'état initial, le système est dans un état tel que tous les neurones inhibiteurs sont inactifs ;
- À partir de cet état, chaque plan p de neurones va avoir tendance à converger vers un état tel que $C_{i,p}$ neurones deviennent actifs : règle `k-de-N` sur les neurones du plan p ;
- À partir du moment où l'un des plans dispose de ses $C_{i,p}$ neurones actifs, si le neurone inhibiteur $Nh_{i,p}$ est évalué, alors il deviendra actif, c'est d'ailleurs la seule combinaison qui peut amener à l'activation de l'un des neurones inhibiteurs ;
- Lorsque un seul neurone inhibiteur est actif, le poids de sa connexion avec tous les autres neurones des autres plans annule le poids de l'entrée de ces neurones ;
- Dans ce cas, lorsque les neurones des autres plans seront évalués, ils resteront/passeront dans un état inactif ;
- Finalement, le système va converger vers un état tel que les neurones du plan, dont le neurone inhibiteur est actif, pourront être dans l'état actif, et les neurones de tous les autres plans seront dans l'état inactif.

4 Résultats

Afin de tester notre proposition sur des cas applicatifs complexes, 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. Les tableaux 1.a 1.b et 1.c donnent une illustration des résultats d'ordonnancement pour des applications dont le nombre de tâches est compris entre 2 et 7. L'architecture ciblée est ici composée de deux ressources d'exécution dont les charges sont données dans le tableau 1.a. Le tableau 1.b montre 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 [6], un grand nombre de neurones *cachés* sont ajoutés à la modélisation, ce qui conduit entres autres à un ralentissement de la convergence. Ici, le surcoût engendré par les neurones inhibiteurs est de l'ordre de 5% ce qui reste très maîtrisé. Dans [9], nous avons comparé l'efficacité de

la convergence des solutions classiques avec notre proposition et nous avons montré que cette dernière était plus efficace. Le tableau 1.c reprend ces résultats et comme on peut le voir, le nombre de neurones nécessaires pour modéliser le problème est assez fortement réduit, de l'ordre de 30%. La réduction du nombre d'évaluations des neurones est quant à elle beaucoup plus drastique, elle dépasse les 98% pour un système d'une complexité limitée. Notons d'ailleurs que cette réduction s'accroît lorsque la complexité de l'application augmente.

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. Nous avons montré que notre proposition permet une limitation importante du nombre de neurones tout en assurant une convergence systématique du réseau vers une solution correcte. Nous avons validé par simulation le bon fonctionnement de la structure du réseau de neurones proposée et nous sommes actuellement en phase de développement d'un générateur de réseaux de neurones matériels.

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 et des neurones qui reste raisonnable.

Références

[1] J. Anderson and A. Srinivasan. Pfair scheduling : Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, Cheju Island, South Korea, december 2000.

[2] 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.

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

[4] 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.

[5] Imène Benkermi. *Modèle et algorithme d'ordonnancement pour architectures reconfigurables dynamiquement.* PhD thesis, ENSSAT-Université de Rennes, 2007.

[6] 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.

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

[8] 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.

[9] 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.

[10] M.A. Cohen and S. Grossberg. Absolute stability of global pattern formation andparallel memory storage by competitive neural networks. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5) :815–826, Sept./Oct. 1983.

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

[12] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52 :141–52, 1985.

[13] Paul Kohout, Brinda Ganesh, and Bruce 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.

[14] 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.

[15] D. Liu and Y.H. Lee. Pfair scheduling of periodic tasks with allocation constraints on multiple processors. In *Proc. of the 18th International Parallel and Distributed Processing Symposium*, volume 03, page 119, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[16] Andrew Morton and Wayne M. Loucks. A hardware/software kernel for system on chip designs. In *Proceedings of the ACM Symposium on Applied Computing*, pages 869–875, 2004.

[17] 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.

[18] G. Tagliarini, J. Fury Christ, and W. E. Page. Optimization using neural networks. *IEEE Trans. Comput.*, 40(12) :1347–58, December 1991.