Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

A Neural Network Model for Real-Time Scheduling on Heterogeneous SoC Architectures

Daniel Chillet, Sebastien Pillement and Olivier Sentieys

Abstract— **With increasing embedded application complexity, designers have proposed to introduce new hardware architectures based on heterogeneous processing units on a single chip. For these architectures, the scheduling service of a realtime operating system must be able to assign tasks on different execution resources. This paper presents a model of artificial neural networks used for real-time task scheduling to heterogeneous system-on-chip architectures. Our proposition is an adaptation of the Hopfield model and the main objective concerns the minimization of the neuron number to facilitate future hardware implementation of this service. In fact, to ensure rapid convergence and low complexity, this number must be dramatically reduced. So, we propose new constructing rules to design smaller neural network and we show, through simulations, that network stabilization is obtained without re-initialisation of the network.**

I. Introduction

A high number of scheduling algorithms have been developed for constraint satisfaction in real-time systems. The majority of these algorithms consider very specific and homogeneous set of characteristics (such as periodic, sporadic tasks, preemptive tasks, homogeneous multiprocessor architectures, etc). Optimal solutions are difficult to find, and the problem becomes NP-hard when all these constraints must be satisfied. To solve this type of problem, approximate methods are classically used, such as genetic algorithms, simulated annealing and Artificial Neural Networks (ANNs).

On the other hand, embedded applications are usually implemented on complex System-on-Chip (SoC) which are built around heterogeneous processing units. Classically, the system is built around a general-purpose processor which runs an Operating System (OS). The other resources, such as Digital Signal Processor core(s), intellectual property block(s) or dynamically reconfigurable accelerator(s), have to be controlled by the OS. In particular, task instantiation on execution resources is realized by using the scheduling OS service. As each task can be defined for several targets, this service must decide, on-line (i.e. at run time), on which resource the task should be instantiated.

Neural networks have demonstrated their efficiency in optimizing problems that take into account several constraints. They converge in a reasonable time (i.e. in a few cycles) if the number of neurons and connections between neurons can be limited as much as possible. 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.

Daniel Chillet is associate professor at ENSSAT, Rennes I University; Sebabtien Pillement is associate professor at IUT, Rennes I University; Olivier Sentieys is professor at ENSSAT, Rennes I University; INRIA/IRISA - UMR 6074 - R2D2 Research Team - BP 80518 - 6 Rue de Kerampont 22305 LANNION - FRANCE - first.lastname@irisa.fr

In this paper, we propose an on-line scheduling based on a neural network for heterogeneous system-on-chip (SoC) architectures with a limited number of neurons. In section II, we present the state of art of the scheduling solutions based on neural networks. Our proposition is detailed in section III. Results are presented in section IV. Finally, we present our conclusions and discuss our future researches to define the hardware implementation of the scheduling service.

II. Related works

Many scheduling algorithms have been developed for constraint satisfaction in real-time systems. The majority of these algorithms take into consideration very specific and homogeneous constraints. They take into account periodic, sporadic or aperiodic tasks, which may or may not allow preemption, on mono or multiprocessor architectures, etc., but rarely combine them. In the context of SoC architecture, specific task scheduling services have been proposed to take into account parallel and heterogeneity characteristics [11], [4], [14]. These service implementations are often complex, and are not always appropriate to real-time systems [9]; they are generally time costly and do not consider the dynamic behavior of the application. Propositions addressing multiprocessor scheduling have been developed [1], [15], [13]. The PFair algorithm is proposed in these papers which focus on an optimal solution for periodic tasks on homogeneous multiprocessors. In fact, a particular solution has been declined to ensure partitioning on multiprocessor systems [2], [3]. The main idea of the PFair algorithm is to assign each task on a processor, with an uniform and fixed execution rate by breaking tasks into quantum-length subtasks. Preemptions and migrations are completely free with this scheduling algorithm, and this can increase execution time due to first-level cache misses. Another major limitation of this solution concerns the targeted multiprocessor which must be homogeneous, and it is now admitted that SoC architectures are always built with heterogeneous execution blocks. Furthermore, we can also note that defining on-line Pfair scheduling is still a difficult problem. In [13], the authors propose an approximate solution to reduce global complexity and to design hardware implementations.

Some solutions have also been proposed through the use of neural networks. For example, in [5], [6], the authors have proposed the ANNs for on-line real-time scheduling. Their solution extends the results obtained in [16] where the theoretical basis for ANN design for optimization problems

1-4244-1380-X/07/$25.00 ©2007 IEEE

defined in [8], [10]. By using a Hopfield model [12], they ensure the existence of a Lyapunov function, called *energy function*. This model ensures that the network evolution converges towards a stable state for which the optimization constraints are respected. This function is defined as:

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

where T_{ij} is the connection weight between neurons i and j, x_i is the state of neuron i and I_i is the external input of neuron i.

Based on this model, a design rule that facilitates the neural network construction can be defined using equality or inequality constraints. The *k-out-of-N* rule is a major result in ANN for optimization. It has been introduced in [16] and reused in multiple works. For example, in [7], the authors define a fuzzy state for the neuron and propose a scheduling problem formulation for a homogeneous multiprocessor with no process migration and constrained times. The network convergence is a complex problem, the complexity is $O(n^3 \times c^3)$, with n the number of tasks and c the number of the machines.

The *k-out-of-N* 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 evolution of the network can start from random states. The energy function is defined as:

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

This function is minimal when the active neuron sum is equal to k, and is positive in the other cases. Equation 2 can be written as follows:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N} T_{ij}\, x_i \cdot x_j - \sum_{i=1}^{N} I_i\, x_i \qquad (3)$$

$$\text{with} \quad \begin{cases} T_{ij} = -2\,\overline{\delta_{i,j}} & \forall i,j \\ I_i = 2k-1 & \forall i \end{cases}$$

$\overline{\delta_{i,j}}$ is the inverse Kronecker function and is equal to zero if $i = j$, or one in the other cases.

Cardeira and Mammeri [5], [6] demonstrate the efficiency of applying ANNs for on-line task scheduling. The ability to take into account numerous and different constraints is made possible by the additive character of the Hopfield model. The results for this real-time scheduling solution exhibit interesting convergence speed which make ANN suitable for on-line utilization. But this technique has two major drawbacks. The first is the important number of necessary slack neurons needed to model the problem. The number of slack neurons is cycle dependant, i.e. when the schedule time increases, the number of slack neurons increases too. The second problem 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 minima must be detected and the network needs to be re-initialized.

In this paper, the number of slack neurons is drastically reduced and independent of the period. Moreover no re-initialization is necessary due to the fact that there is no local minimum in the energy function. The direct consequence is the simplification of network control.

III. Scheduling problem modelling through neural network

Our proposition is guided by two main objectives: to reduce the number of neurons and to guarantee network convergence without re-initialization.

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 matrix form, with the size $N_T \times N_C$, 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 matrices organized in layers are required to model the different execution resources. New slack neurons are then necessary to manage the exclusive execution of each task on resources. Figure 1 presents an example of network with p resource layers. For each couple (task i, resource j), $C_{i,j}$ new slack neurons must be added. 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. This number is equal to $\sum_{i=1}^{NT}\sum_{j=1}^{p} C_{i,j} + p \cdot NC$.

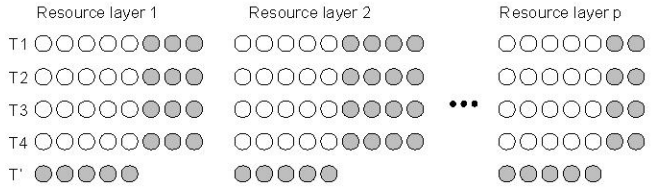

Fig. 1. Classical structure used to model the scheduling problem with ANN. Grey circles represent slack neurons

In order to reduce the number of required neurons, we propose two modifications of the neural network structure. These two modifications correspond to an adaptation of the Hopfield model.

Firstly, a specific neuron is placed for each task and each type of execution resource, this neuron is called *inhibitor neuron*. The main idea consists in creating a mutual exclusion between the possible task instantiations on execution resources. This mutual exclusion is ensured by the presence of one inhibitor neuron $IN_{i,j}$ by task i and by execution resource j. In figure 2, an example of the scheduling problem is presented with one task T_i and R possible resources. A

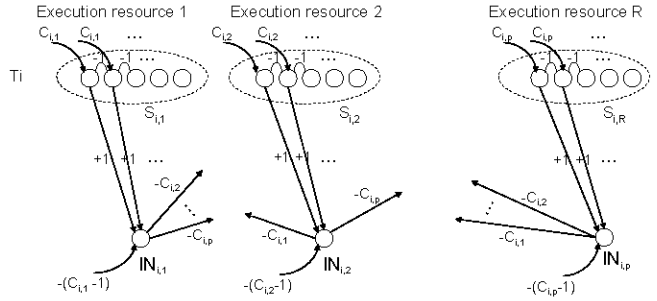

Fig. 2. Scheduling problem modelled with the use of inhibitor neurons, $C_{i,j}$ is the worst case execution time of task T_i on resource j

set of N_C neurons is called $S_{i,j}$ ($N_C = 5$ in the case of figure 2) and represents the possible scheduling cycles of task i on the resource j. For each resource j, the worst case execution time (WCET) of task i is defined as $C_{i,j}$. The set of neurons $S_{i,j}$ is configured (definition of inputs and weights) to converge towards $C_{i,j}$ active neurons among N_C. When one set $S_{i,j}$ has $C_{i,j}$ active neurons, the state of inhibitor neuron $IN_{i,j}$ can be changed, and set to the value 1. So, all the neurons included in the other sets will be forced to the inactive state (this is due to the great negative value $-C_{i,k}$ of weights which cancel the neuron inputs of all the other sets $S_{i,k}$ with $k \neq j$).

The main characteristic of this neuron 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.

Secondly, we propose to remove the slack neurons which model the possible inactivity of the processor (idle cycles). In [5], [6], the authors add one or several lines of slack neurons to ensure the network convergence when applying a *k-out-of-N* rule on each vertical line of neurons. The number of added lines in each layer is equal to the number of identical processors in this layer (for example in figure 8.a, two resources can execute the tasks, so two lines of slack neurons are added, line T'_1 and T'_2). In this case, the convergence can not be always obtained, re-initializations are often necessary to extract the network from a local minimum of the energy function.

To delete these lines, we propose the application of a k_1-*out-of-*N_1 classical rule on the horizontal sets of neurons and a *at-most-*k_2*-among-*N_2 rule on the vertical sets of neurons. For example, if N_T tasks must be scheduled on p identical resources (i.e. p processors in the same layer) during the N_C cycles, N_C *at-most-p-among-*N_T rules must be applied on each vertical set of N_T neurons. To ensure the convergence towards a valid solution, the problem concerns the application of two rules on two sets of neurons with one common neuron between the two sets. Figure 3 shows an example of two sets of neurons, the first is composed of three horizontal neurons $n1, n2, n3$, the second is composed of three vertical neurons $n1, n4, n5$ (i.e. neuron $n1$ is the common neuron of the two sets). The classical additivity for several rules says that if a k_1-*out-of-3* rule is applied on the first set and a *at-most-*k_2*-among-of-3* rule is applied on the second set, then the inputs and weights are defined as follows:

- inputs are equal to $I_1 = (2 \cdot k_1 - 1) + (2 \cdot k_2 - 1)$, $I_2 = I_3 = (2 \cdot k_1 - 1)$ and $I_4 = I_5 = (2 \cdot k_2 - 1)$
- weights are equal to $w_{i,j} = -2 \qquad \forall\, i, j = \{1 \dots 5\}$

To ensure the *at-most-*k_2*-among-of-3* rule, k_2 slack neurons can be added with a specific weight connection with others neurons. In the figure 3, the slack neurons $(sn1, sn2)$ are represented.

To remove the slack neurons while ensuring convergence, we firstly need to simplify the *k-out-of-N* rule by a simple re-definition of input and weight values. Rather than use the energy function given in equation 2, we propose to rewrite energy as $E = (\frac{1}{\sqrt{2}}k - \frac{1}{\sqrt{2}}\sum_{i=1}^{N} x_i)^2$. This modification does not change the convergence point of the energy function. This expression can be transformed and written as:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}(-1)\; x_i \cdot x_j - \sum_{i=1}^{N}(k - \frac{1}{2})\; x_i \qquad (4)$$

With a simple re-definition of the threshold value of the sigmoid function, which is classically fixed at value $\frac{1}{2}$, it is possible to simplify the input value of each node. We define a sigmoid function with a threshold value fixed at 0, it corresponds to a shift of the sigmoid function. In this case, the input and weight values are equal to:

$$\begin{aligned} T_{ij} &= -1 \cdot \overline{\delta_{i,j}} \qquad \forall i, j \\ I_i &= k \qquad \forall i \end{aligned} \qquad (5)$$

With these new input and weight values, we can propose a simple additivity between k_1-*out-of-*N_1 and *at-most-*k_2*-among-*N_2 rules. The main idea consists in firstly applying k_1-*out-of-*N_1 on horizontal lines and k_2-*out-of-*N_2 on vertical lines and secondly modifying the weights of horizontal lines. So, the common neuron has its input fixed to the value $k_1 + k_2$ as shown in figure 4. The modification of the weight values of the horizontal rule (here the horizontal rule k_1-*out-of-3*) compensates the increase of input values on the common neuron. This compensation is done by decreasing the weight value between the horizontal neurons and the neuron which belong to the two rules. An example of additivity of two rules is given in figure 4. This construction corresponds to an energy function which can be expressed as:

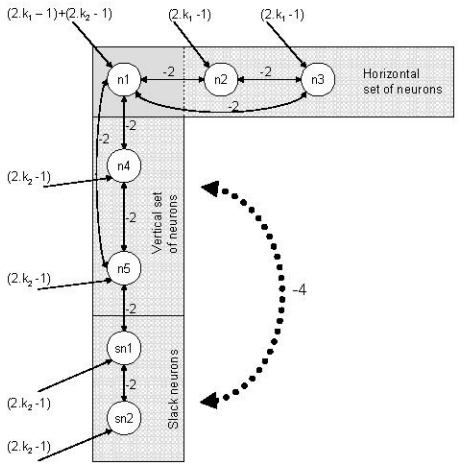

Fig. 3. Additivity of k_1-*out-of*-N_1 and *at-most*-k_2-*among*-N_2 rules with slack neurons (the dashed arrow indicates that all the neurons in the two sets must be connected with a weight value equals to -4)

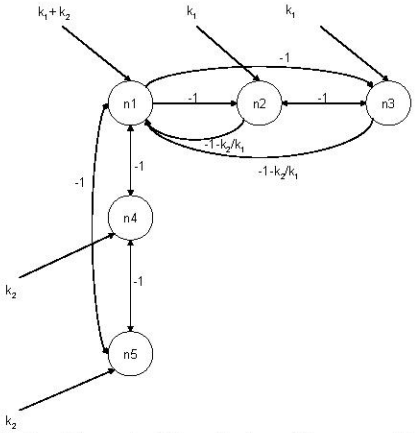

Fig. 4. Additivity of k_1-*out-of*-N_1 and *at-most*-k_2-*among*-N_2 rules without slack neurons

$$E = \left(k_1 - \sum_{i=1}^{3} x_i \right)^2 + \prod_{j=0}^{k_2} \left(j - \sum_{i \in \{1,4,5\}} x_i \right)^2 \quad (6)$$

The first term is null when k_1 neurons are active among the three horizontal neurons. The second term is null when 0 or 1 or 2 or ... or k_2 neurons are active among the three vertical neurons. So the energy function is null when k_1 neurons of N_1 are active and when at most k_2 neurons of N_2 are active.

A complete example is given on figure 5 which concerns three tasks (T_1, T_2 and T_3) to schedule on p homogeneous resources (on layer j) during five schedule cycles (with $p \leq 3$ on the figure). To simplify the figure, all connections between neurons are not represented. The input value of each neuron is increased by the value p which corresponds to the number of resources on the considered layer j. This figure does not show the inhibitor neurons which ensure the exclusive task instantiation on the different resource layers.

In combining our two propositions, we can model the task scheduling problem on a heterogeneous SoC architecture

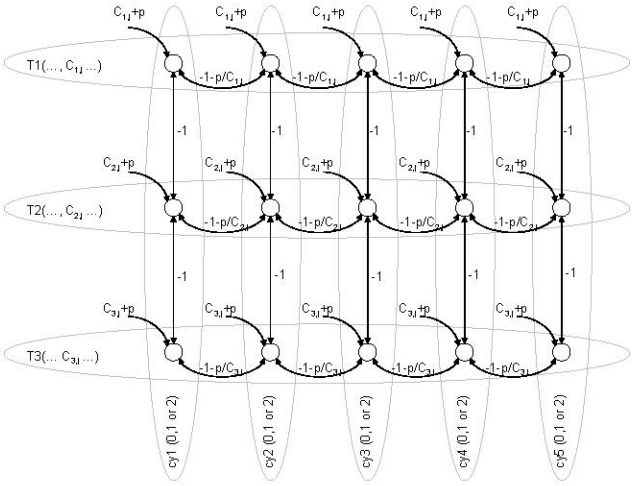

Fig. 5. Our proposition for rule additivity between *k-out-of-N* and *at-most-k-among-of-N* rules

with an important decrease of slack neurons and without any re-initializations of the neural network. In the next section, we compare these structures to other previous works.

IV. Results and comparisons

In this section, we present some comparisons of our proposition with the previous solutions proposed in [6]. We show that our proposal is more efficient (i.e. convergence speedup) and does not need network re-initialisations.

A. Comparisons with previous works

The first example concerns the schedule of three tasks T_1, T_2 and T_3 on one execution resource. To solve this problem, the previous proposition needed to add one neuron line to model the processor idle cycles. In figure 6.a, the initial state of the network is presented. Figure 6.b presents one possible network final state. In their paper [6], the authors indicate that convergence is obtained after more than 600 fired neurons.

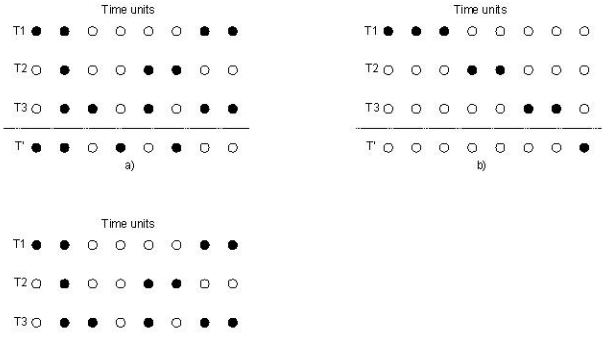

Fig. 6. Example of state representation of neural network (task WCET are equal to $3, 2, 2$ for respectively T_1, T_2, T_3) a) Initial state with one slack neuron line; b) Stable state and valid solution; c) Initial state without slack neurons

Figure 6.c presents our model for the same scheduling problem. As we can see, the number of neurons is lower

than the classical solution. Moreover, for all simulations from random initial state, our network always converges to a valid solution. This convergence is then obtained with an average number of fired neurons equal to 79.

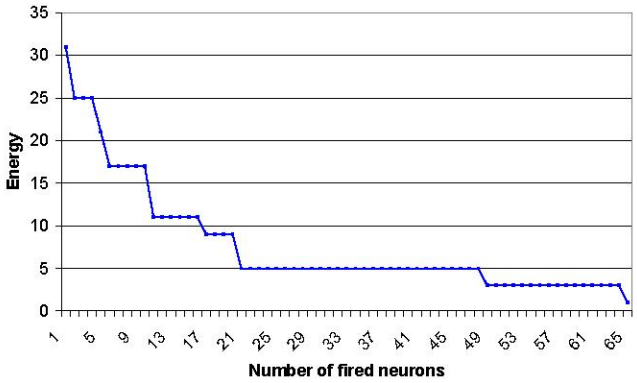

Fig. 7. Example of energy function evolution with our network model

Figure 7 shows the evolution of the energy function for one simulation. This simulation converges to a valid solution in 65 fired neurons. Contrary to the classical energy function evolution, we can notice that this function decreases monotonously with the number of fired neurons.

Figure 8.a presents another example with two execution resources which is modelled by the classical network. On figure 8.b, we can see the limitation of required neurons to model the same problem with our proposition. The classical solution converges to a valid solution with more than 300 fired neurons. Our proposition limits the number of fired neurons to approximatively 70.

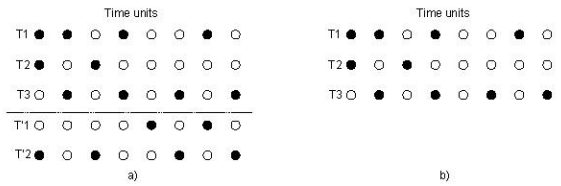

Fig. 8. a) Initial state of neural network; b) Initial state of our neural network

The number of slack neurons removed is directly proportional to the hyperperiod which can be important. This reduction favorably impacts the hard implementation.

B. Results on a System-on-Chip architecture

The actual SoC architectures are composed of several heterogeneous resources (between five and ten resources) and execute applications which are composed of ten to twenty tasks. We consider that several tasks are defined for different resources while the others are defined for one specific resource.

This section shows the results obtained for a specific architecture which is composed of 5 different resources. It corresponds to: resource r_1 General Purpose Processor (GPP), resources r_2, r_3 and r_4 specific Intellectual Property blocks (IP) and resource r_5 a Dynamically Reconfigurable Accelerator. The complete application is composed of 10 tasks. Seven tasks have been described for the GPP resource (t_2, t_3, t_4, t_6, t_8, t_9, t_{10}). Three tasks have been defined for the three IP blocks (t_1, t_5, t_7). Finally, four tasks have been also described for DRA (t_2, t_3, t_6, t_8, t_9, t_{10}). This information is summarized in the matrix which define the WCET of each task on each resource (value ∞ indicates that the task cannot be executed on the corresponding resource).

$$C_{i,j} = \left|\begin{array}{cccccccccc|c} \infty & 2 & 2 & 4 & \infty & 4 & \infty & 4 & 4 & 2 & r_1 \\ \infty & \infty & \infty & \infty & \infty & \infty & 10 & \infty & \infty & \infty & r_2 \\ \infty & \infty & \infty & \infty & 5 & \infty & \infty & \infty & \infty & \infty & r_3 \\ 4 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & r_4 \\ \infty & 2 & 1 & \infty & \infty & 2 & \infty & 2 & 1 & 2 & r_5 \\ t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} & \end{array}\right| \quad (7)$$

Figure 9 presents results obtained on the application with the number of tasks varying from 1 to 10.

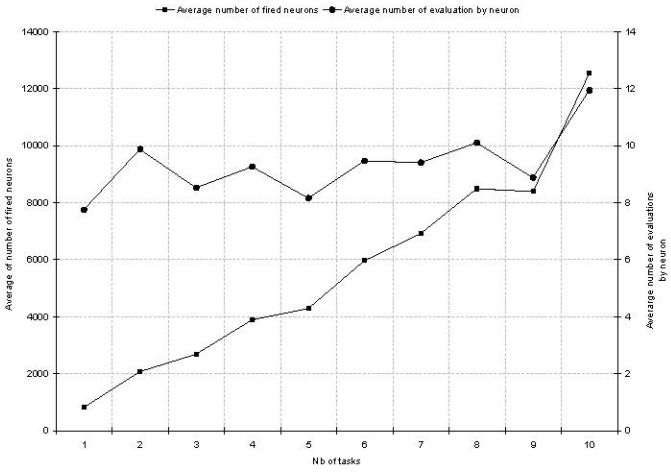

Fig. 9. Results for a SoC architecture running application; Ten tasks of the complete application are introduced step by step

This figure shows that the number of fired neurons seems to evolve linearly with the number of tasks, i.e. linearly with the neural network complexity. Another important result is the relatively constant number of evaluations of each neuron. In our example, each neuron is evaluated between 8 and 12 times.

V. Conclusion and Future Work

In this paper, a scheduling service modelled by neural network is presented. An adaptation of the Hopfield model is proposed to facilitate the hardware implementation of the scheduling service. 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 of network to ensure the convergence. To limit the number of neurons, we propose replacing the numerous slack neurons of classical solutions by several inhibitor neurons. Based on these inhibitor neurons, a new structure of a neural network

is presented. The main characteristic is its capacity to always ensure the convergence towards a valid solution.

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 which is the consequence of the classical additivity of *k-out-of-N* rules. We propose replacing 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.

We have shown the efficiency of our proposition compared to the previous works. These two contributions are important advances for our future works which consist in defining a hardware implementation of the scheduling service in the context of SoC architecture. Globally, the limitation of the network size ensures that the hardware implementation will be as simple as possible, i.e. as fast as possible. Furthermore, the absence of network re-initialization is also interesting because it will probably lead to a simple control of the neural network.

Our future researches will consist of defining a hardware implementation of the neuron, which is not too complex, and to define a complete network. The major problem for the hardware implementation concerns the large number of connections between neurons. By exploiting the connection symmetry between neurons, we think that several optimizations can be applied to limit hardware cost.

REFERENCES

[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] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. Technical Report 472, ULB, 2002.

[5] 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.

[6] 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.

[7] R.-M. Chen and Y.-M. Huang. Multiprocessor task assignment with fuzzy hopfield neural network clustering technique. *In Neural computing & applications*, 10(1):12–21, 2001.

[8] M. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE transactions on systems, man, and cybernetics*, 13, no 5:815–826, 1983.

[9] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems*. John Wiley & Sons, England, 2002.

[10] S. Grossberg. *Studies of mind and brain : neural principles of learning, perception, development, cognition and motor control*. D. Reidel publishing Company, 1988.

[11] 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.

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

[13] 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.

[14] J. Noguera and R. M. Badia. Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling. *ACM Trans. on Embedded Computing Systems*, 3(2):385–406, may 2004.

[15] 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.

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