15th European Signal Processing Conference (EUSIPCO 2007), Poznan, Poland, September 3-7, 2007, copyright by EURASIP

HARDWARE TASK SCHEDULING FOR HETEROGENEOUS SOC ARCHITECTURES

Imène Benkermi, Daniel Chillet, Sébastien Pillement and Olivier Sentieys

ENSSAT – Rennes I University,
INRIA/IRISA - R2D2 Research Team
BP 80518 - 6 Rue de Kerampont 22305 LANNION - FRANCE
first.lastname@enssat.fr

ABSTRACT

This paper presents our work on extending artificial neural networks use for real-time task scheduling to heterogeneous System-on-Chip architectures. The Hopfield model is the neural network model considered in this study. We introduce new constructing rules to design neural network so that architecture heterogeneity can be considered. We show that these new rules ensure the network stabilization on states that take into account the architecture heterogeneity while meeting the imposed task constraints.

1. INTRODUCTION & RELATED WORKS

Many scheduling algorithms have been developed for real-time systems. Most of these algorithms take into consideration a very specific and homogeneous set of constraints. They take into account periodic, sporadic or aperiodic tasks, which may or may not allow preemption, on mono or multiprocessor architectures, but rarely combine them. In addition, optimality in finding solutions for these problems is even more difficult to obtain.

Several propositions addressing multiprocessor scheduling have been developed [1, 14, 11]. The PFair algorithm presented in these papers proposed an optimal solution for periodic tasks on homogenous multiprocessor. 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 a 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 System-on-Chip (SoC) architectures are always built with heterogeneous execution blocks. Furthermore, we can note also that defining on-line Pfair scheduling is still a difficult problem. In [11], the authors propose an approximate solution to reduce global complexity and to design hardware implementations.

On the other hand, the increasing complexity of nowadays applications, such as, signal and image processing algorithms, and the need to accelerate their execution, have led designers to investigate new hardware architectures. These SoCs generally integrate heterogeneous processing units, among other hardware components, on a same chip. They operate in parallel and heterogeneity is due to the fact that they can have different processing powers and execution times for the same code portion. This new kind of architectures requires adapted tools and mechanisms to take into account their specificities. Hence, some studies, especially on task scheduling, have been proposed [8, 4, 12].

Many solutions based on off-line scheduling algorithms have been proposed. They are often complex, and are not appropriate to real-time systems [7]; they are generally time costly and do not consider application dynamic behavior. In this context, approximated methods have been developed to solve this problem, such as genetic algorithms, simulated annealing or Artificial Neural Networks (ANN).

In this paper, we propose an on-line scheduling algorithm for heterogeneous multiprocessor architectures. We propose to extend Cardeira and Mammeri's work [5] on the ANNs use, particularly the Hopfield model [9], for on-line real-time task scheduling. These authors utilize the results obtained by Tagliarini and *al.* [15] who used the ANN theorical basis for optimization problems introduced by Cohen and Grossberg [15]. The main contribution of Tagliarini and *al.* was the introduction of a design rule that facilitates the construction of time evolution equations describing network behavior. This rule specifies the connection weights and external inputs that will enforce constraints expressed as equalities or inequalities.

Hence, the efficiency of applying ANNs for on-line task scheduling is demonstrated in [5]. The concerned software/hardware architectures can have different cumulated constraints, such as timing constraints (periods, ready times, deadlines, etc.), preemption and non-preemption constraints, precedence constraints, on mono or multiprocessor architectures. The ability to take into account numerous and different constraints is made possible by the additive character of the Hopfield model. The results of the application of this approach for real-time scheduling have shown their efficiency [5, 6], since it is a progressive approach and it offers interesting convergence speed which makes it suitable for on-line utilization.

As the work in [5] only considers homogeneous multiprocessor, the extension to heterogeneous multiprocessor architectures is investigated in this study. The paper is organized as follows. First, we present the Hopfield model and the main results obtained for this model, namely, Tagliarini and *al.*'s construction rule. Then, the application of this model to real-time task scheduling is shown. Finally, we explain how these results can be extended to take into account processor heterogeneity on the considered architecture.

2. HOPFIELD'S NEURAL NETWORK MODEL

Many studies on ANNs utilization have been investigated since Cohen and Grossberg [15] have proposed ANN models. The main characteristics to ensure the convergence toward a valid solution are the following ones: i) the neuron connection weight matrix is symmetric, ii) its diagonal elements are all equal to zero, iii) and all the elements of the matrix are non-positive.

Hopfield's model is one of the models considered by their study. Hopfield proved the existence of a Lyapunov function, called *energy function* [9], and derived from Cohen and Grossberg's results, so that the ANN will evolve to a stable state without increasing this function which has the following form:

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

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.

The idea of Hopfield is that, since an ANN will tend to minimize the energy function while evolving to stable states, one can construct a network that tends to minimize this function. This will be done by associating the optimization problem variables to the function variables, i.e. putting the "cost function" of the problem into an equivalent form to equation 1.

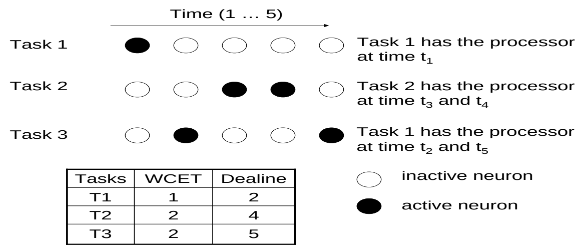
The different steps needed to solve optimization problems using Hopfield ANN are the following. First find an ANN topology in which neural outputs are interpreted as a solution to the problem. Then, put the "cost function" of the problem into a form equivalent to equation 1, in which the minima correspond to the best solutions for the problem, and finally, calculate the connection weights and external inputs. The ANN corresponding to the optimization problem is now found. One just has to let the ANN evolve to a stable state corresponding to the minima of the function. The ANN evolution is made by computing the activation function for each neuron, $I_i + T_{ij}x_j$. If its value is less than 0, the neuron is set inactive, elsewhere it is set to an active state.

3. APPLICATION TO REAL-TIME TASK SCHEDULING

The use of Hopfield ANNs for solving on-line real-time task scheduling problem has been first proposed in [5]. By making a good choice of the Hopfield ANN topology, and by calculating the energy function corresponding to the model constraints imposed, it is shown how this ANN can evolve to a stable state representing a solution (if it does exist) to the task scheduling while meeting the imposed real-time constraints. Most of approximate methods for scheduling problem solving considers a set of restrictive constraints. However, ANNs offer the possibility to deal with scheduling problems with heterogeneous constraints. A progressive and systematic method allows them to modelize problem and to built specific ANN. This later offers as well remarkable convergence speed to stable state, which makes it very attractive for online use. The following representation scheme for the network has been adopted (see figure 1):

- Neurons are arranged in a matrix form, where lines correspond to tasks and columns correspond to time units. The scheduling length is generally supposed equal to the least common multiple of the task periods.
- An active neuron will indicate that the corresponding task is being executed at the corresponding time unit.

After this scheme construction, the energy function corresponding to the problem must be calculated which will allow to obtain the different connection weights and external inputs of the neurons. This function is the sum of the different energy functions corresponding to the different constraints to be taken into consideration.


Time (1 ... 5)
Task 1
Task 2
Task 3
Task 1 has the processor at time t1
Task 2 has the processor at time t3 and t4
Task 1 has the processor at time t2 and t5
inactive neuron
active neuron

Tasks	WCET	Dealine
T1	1	2
T2	2	4
T3	2	5

Figure 1: Hopfield ANN evolution result for an example of preemptive tasks using one processor at 100%.

To solve the scheduling problem, Tagliarini and *al.*,[15], introduce the *k-out-of-N* rule. This rule allows the construction of a network with N neurons for which the evolution leads to a stable state with "exactly k active neurons among N" (an active neuron is a neuron with an output equal to one). The following cost function

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

is minimal when the active neuron sum is equal to k, and is positive in the other cases. After applying some mathematical transformations, a function equivalent to equation 1 is of the form:

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

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

with $\overline{\delta_{i,j}}$ is the inverse of Kronecker function and is equal to zero if $i = j$, and one otherwise.

The ANN corresponding to these results respects all the convergence criteria, and can then be used.

Since the Hopfield model is additive, the *k-out-of-N* rule can be applied to several sets of neurons at a time in order to take into account different types of optimization constraints. The new weights and inputs imposed by each constraint can be directly added to those already existing (since the model is additive). The ANN will evolve to a state satisfying all the constraints simultaneously (if such a state does exist). In this case, the energy function is minimal (equals zero). It is different from zero if the ANN is blocked on a local minimum of this function, or if there is no solution for this problem.

Figure 1 shows the result of the *k-out-of-N* rule to the scheduling problem of three preemptive tasks $T_1(C_1 = 1, D_1 = 2)$, $T_2(C_2 = 2, D_2 = 4)$ and $T_3(C_3 = 2, D_3 = 5)$ on a monoprocessor architecture, where C_i represents the worst case execution time of task T_i and D_i its deadline. First, the *k-out-of-N* rule is applied for each task (for each line of the neuron matrix). For example, for task T_1, the rule is applied to the first two neurons (since $D_1 = 2$) of the first line with

$k = 1$ (since $C_1 = 1$). Finally, to ensure processor use at each time unit in a mutually exclusive manner, the *k-out-of-N* rule is applied to each column with $k = 1$.

The simulation results of the technique application for different task configurations are given in [5]. These results show that the ANN-based scheduling algorithm can be re-executed several times (since the network can reach a stable state in few milliseconds) when the function is different of zero, to determine if it is or not a local minima.

4. APPLYING NEURAL NETWORKS TO TASK SCHEDULING IN HETEROGENEOUS MULTIPROCESSOR ARCHITECTURES

The on-line scheduling problem of a set of real-time tasks having deadlines on an architecture with m processors (with $m \geq 2$), is a complex problem for which optimal algorithms does not exist [13]; it is an NP-Complete problem. In addition, well known scheduling algorithms, such as EDF (Earliest Deadline First) and RM (Rate Monotonic) [10], used for task sets executing on a monoprocessor architecture, cannot be directly applied to the multiprocessor case. Thus, optimal EDF algorithm in a monoprocessor environment, is not optimal any more in a multiprocessor environment [7].

In the following sections, we propose to extend the ANN-based method to take into account hardware architectures composed of heterogeneous processors.

4.1 Considered model and hypotheses

The ANN topology proposed in [5] is modified in this paper by extending the ANN with as many plans as processor types in the system (note that one processor type may contain multiple homogeneous processors). Thus, each neuron plan will represent one type of processors, where the lines represent the tasks and the columns represent the scheduling units of time. All neurons are obviously totally interconnected in order to be able to use Hopfield's results.

Tasks are specified by temporal constraints, essentially, worst case execution times of each task on each type of processor (or plan) it will eventually execute. For example, a task T_i can be represented by $T_i(R_i, C_{i0}, C_{i1}, ..., C_{im}, D_i, P_i)$, where R_i is T_i's ready time, C_{ij} the WCET of T_i on processor type j (it is equal to zero if the task cannot be executed on this processor), D_i is T_i's deadline and P_i its period.

Furthermore, the tasks are supposed to be independent in order to facilitate the comprehension of this work; the model can be easily extended later for precedence constraints since corresponding results already exist [6]. In addition, communication costs are taken into account within worst case execution times.

For the scheduler, we assume that it runs on the processor where the unique operating system manages all the architecture resources, while having a global view of the system state accessible at any time. Thus, this scheduler will deliver a global scheduling for tasks arriving to the system and which have to be dispatched among the different processors, while meeting their constraints.

In the considered multiprocessor environment, a real-time task scheduling algorithm is valid if all the deadlines of the tasks distributed among the processors are met. In addition, the condition that a processor can execute only one task at a time, and that a task is treated by only one type of processor at a time (i.e. migration between types of processors is not allowed) are imposed in the considered model. Migration between processors of the same type are however allowed.

The task scheduling sequence on the m processors that enables all the tasks to meet their respective constraints, must be computed now.

4.2 The approach

To simplify the presentation of the approach, we suppose that each task T_i occurs once, that $D_i = P_i$ and $R_i = 0$. The extension to a model with more general parameters can easily be made. The constraint model imposed supposes that, during each task period, this task must be executed on only one type of processor. First, for all neurons of the period on the different plans, the application of *k-out-of-N* rule is to be applied. Where k is equal to the maximum of task WCETs on the different plans, and N is the set of these neurons, extended with a number of slack neurons corresponding to the difference between the maximum and the minimum (other than zero) among the task WCETs on the different processors. The application of this rule ensures that there will be at least one task execution during its execution period. Slack neurons are introduced to "absorb" active states if the number of already neuron active states on the chosen processor is sufficient, i.e. corresponds to the execution time of the task on this processor.

Let us take the example of 3 tasks represented in figure 2. These tasks are executed on an architecture with one processor by plan (1pbp).

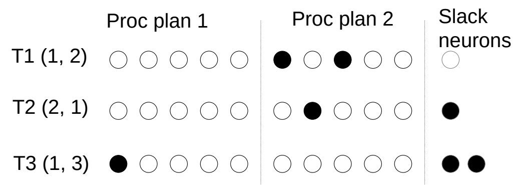


Figure 2: Execution example of 3 tasks on 2 different processor types. Slack neurons "absorb" undesirable active states.

The network converges towards a stable state respecting the applied rule. However, in order to make sure that this execution will be on only one processor (migration is not allowed), all active neurons for a task period must be enforced to be on the same processor. To do this, the introduction of a new design rule is necessary. We call it *(0-or-k)-out-of-N* rule, and we apply it for the period of each task T_i on processor type j with $k = C_{ij}$.

4.3 The new *(0-or-k)-out-of-N* design rule

In order to ensure that only 0 or k neurons will be active among N neurons, we first extend the N neurons with k slack neurons, as presented in figure 4

The $N + k$ neurons must respect the following conditions:

$$\sum_{i=1}^{N} x_i \cdot \sum_{i=N+1}^{N+k} x_i = 0 \qquad \text{and} \qquad (\sum_{i=1}^{N+k} x_i - k)^2 = 0$$

The first condition ensures exclusion of active states between "original" neurons and slack ones. The second con-

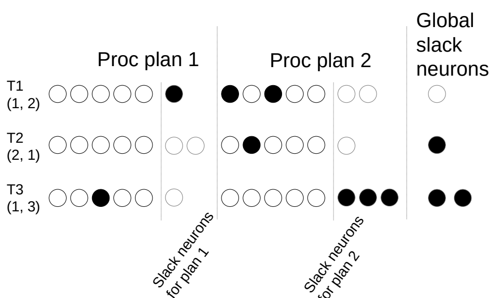


Figure 3: Network extension for *(0-or-k)-out-of-N* design rule

dition ensures that k neurons will be active among the $N+k$ neurons. So the following cost function

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

is minimal when the active neuron sum is equal to 0 or k, and is positive in the other cases. After applying some mathematical transformations on this function in order to put it in the form of the energy function 1, we obtain

$$\begin{aligned} E &= -\sum_{i=1}^{N+k} (-1) \cdot x_i^2 - \tfrac{1}{2} \sum_{i=1}^{N+k} \sum_{\substack{j=1 \\ j \neq i}}^{N+k} (-1) \cdot x_i \cdot x_j \\ &- \sum_{i=1}^{N+k} (2k - \tfrac{1}{2}) \cdot x_i - \tfrac{1}{2} \sum_{i=1}^{N+k} \sum_{\substack{j=1 \\ j \neq i}}^{N+k} (-1) \cdot x_i \cdot x_j \\ &- \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=N+1}^{N+k} (-2) \cdot x_i \cdot x_j \end{aligned} \tag{5}$$

Hence, the following neuron parameters are obtained for the three terms which respect the convergence criteria:

$$\begin{aligned} T_{ij} &= \begin{cases} 0 & \text{if } i = j \\ -4 & \text{if } i = 1 \ldots N \text{ and } j = N+1 \ldots N+k \\ -4 & \text{if } i = N+1 \ldots N+k \text{ and } j = 1 \ldots N \\ -2 & \text{else} \end{cases} \\ I_i &= 2k - \frac{3}{2} \qquad \forall i \end{aligned} \tag{6}$$

These results are added to those presented above (applied in figure 2). The next step will be to take into account mutual exclusion in each processor.

4.4 Mutual exclusion

It is important to ensure that a processor is allocated to the tasks that request it, in a mutually exclusive manner. To do this, the *k-out-of-N* rule application to each column representing a scheduling instant, with $k = n$ (number of processors of the considered type) and N the task number by column extended with n neurons is applied. The n extended neurons allow to "absorb" the unused cycles. Unused cycles appear when all tasks have been scheduled in other cycles. To manage this extension, we create n virtual tasks. Hence, this step will ensure that we will converge to "at most n active neurons by column at a given time".

Furthermore, some convergence problems can appear when the network is composed of more than two processor plans. To limit these problems, we must ensure that if a task is allocated on a processor, it can't be allocated in another processor. It corresponds to a unique allocation rule. The formulation of this rule is given as

$$\sum_{i \in plan_1} x_i \cdot \sum_{i \in plan_2} x_i \ldots \sum_{i \in plan_p} x_i = 0.$$

Developing this function until obtaining a form equivalent to Cohen and Grossberg's energy function, leads to the following parameters :

$$\begin{aligned} T_{ij} &= \begin{cases} 0 & \text{if } i \in plan_p \, ; and \; j \in plan_q \; with \; p = q \\ -1 & \text{if } i \in plan_p \; and \; j \in plan_q \; with \; p \neq q \end{cases} \\ \mathrm{I}_i &= 0 \qquad \forall i \end{aligned} \tag{7}$$

This last rule must be added to the rule given in expression 6 to built a correct neuron network.

Finally, each line of the network is built as representing as it is presented to the figure 4.

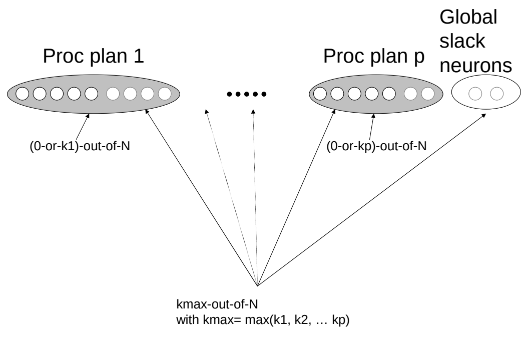


Figure 4: Global modelization of scheduling problem

5. APPLICATION EXAMPLE

Several simulations have been conducted to study the efficiency of the proposed solution. We took a set of tasks with execution times given in table 1.a. to be executed on two types of processors. We supposed first that each type contains one computing unit (one processor by plan: *1 pbp*), so that the solution must deliver a scheduling sequence with a maximum of one task by processor at a time. And then we add to each plan one processor (*2 pbp*).

On table 1, we can observed that the convergence is faster when the number of processors by plan increases. The number of possible solutions increases due to the large degree of freedom to schedule tasks.

For the first case (with one processor by plan), the simulation of the neuronal network produces solutions as presented in figure 5.

This figure shows the schedule of all the tasks is correct. Indeed, for each cycle, only one task is scheduled on each processor. Nevertheless, we can see that tasks are often preempted during execution. This problem must be solved by applying specific rule which ensure that active neurons for a task are cycle grouped (`succ-k-out-of-N` rule) [6].

	WCET		
Task	$C_{i,1}$	$C_{i,2}$	P_i
T_1	1	2	20
T_2	2	1	20
T_3	4	2	20
T_4	3	5	20
T_5	4	6	20
T_6	3	2	20
T_7	2	3	20

a)

Average init		
Task nb	1 pbp	2 pbp
2	1,73	1,83
3	2,98	2,98
4	5,80	5,20
5	16,91	6,76
6	31,20	11,90
7	72,43	13,98

b)

Table 1: (a) Task execution times on each processor type, (b) Average number of initializations needed for stabilization (pbp: processor by plan)

```
     Processor plan 1     Processor plan 2
T1 -------------------- -----------#-------#
T2 -------------------- ----#---------------
T3 -##-#-------#------- --------------------
T4 -------------------- --##---#--#--#------
T5 -------------------- #----##--#----##----
T6 #------#--------#--- --------------------
T7 -----------------#-# --------------------
```

$time(cycles) \longrightarrow 0\ to\ 20$ $time(cycles) \longrightarrow 0\ to\ 20$

Figure 5: Result for 7 tasks scheduled on a SoC composed of two types of processor (# indicates that the corresponding task is running at this cycle

6. CONCLUSION AND FUTURE WORK

In this paper, the extension of ANNs utilization for real-time task scheduling, to heterogeneous multiprocessor architectures, is proposed. New design rules, allowing the calculation of connection weights and external inputs for neurons adapted for the system model, is presented. With these obtained neuron parameters the ANN is able to converge to a stable state meeting the considered system model constraints, namely, the heterogeneity of processing units on which tasks can execute. The simulations conducted consider one period by scheduling length. They led to satisfying results for the ANN-based scheduling algorithm that show the efficiency of the new rule introduced in this paper. Extending these simulations to more general tasks, with multiple periods by scheduling length, and comparing these results to those of existing methods for heterogeneous systems is the aim of our next works.

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

[8] B. Hamidzadeh, D. Lilja, and Y. Atif. Dynamic scheduling techniques for heterogeneous computing systems. *Journal of Concurrency: Practice and Experience*, 7:633–652, October 1995.

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

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

[11] 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.

[12] 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.

[13] S. Sahni. Preemptive scheduling with due dates, operational research. *Operational Research*, 27:925–934, Sept.-Oct. 1979.

[14] 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.

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