Efficient Task Scheduling Approach Relevant to the Hardware/Software Co-Design of Embedded System

Barbara Koroušić Seljak

Computer Systems Department, "Jožef Stefan" Institute, Ljubljana, Slovenia

Task scheduling is the primary multitasking activity controlled by the real-time executive. As hardware/software co-design of embedded systems has been enabled by advances in computer technologies, reprogrammable hardware can be used to implement a co-processor to perform most of the kernel functions, including task scheduling. In this kind of system design, more complex scheduling approaches can be applied. In this paper, a complex scheduling approach, which takes into account advantages of evolutionary computation (i.e., neurocomputing and genetic search and optimization) is presented. First, we present a model based on the Hopfield-Tank neural network [11]. Then, we introduce modifications of the method based on the network model to improve the quality of the solutions. Finally, we propose a mixed approach of this evolutionary computation method and an extension of the Earliest Deadline First approach [3] for scheduling both types of periodic and aperiodic tasks. We also discuss simulation results that demonstrate performance that could be obtained by using this approach.

Keywords: task scheduling, embedded systems, hardware/software co-design, neural network, genetic algorithm.

1. Introduction

Embedded systems (ESs) have been around for several decades already. There are many ESs of different categories with or without strict real-time requirements. Generally, an ES may be described as a computing system that operates in concert with the real physical system, and is an integral part of the equipment. ESs are traditionally classified into *control systems*, where something has to be controlled or regulated (e.g., an assembly line), and *communication systems* (e.g., a telephone switching station). However, there are other classifications of ESs:

- *soft real-time* (e.g., an office automation system) vs. *hard real-time systems* (e.g., a fly-by-wire system), and
- *slow* (e.g., a patient-monitoring system) vs. *fast systems* (e.g., a radar system).

The world in which an ES is placed is almost always truly **asynchronous** and **concurrent**: events requiring response from the computer may occur at any time and simultaneously. Embedded computers must deliver their results correctly and within certain time bounds — *deadlines*. The term *hard real-time* refers to ESs that may malfunction if any deadline is missed. Soft real-time systems can operate satisfactorily even if some deadlines are missed [14]. Slow ESs respond at a rate commensurate with human reaction time, while fast ESs are now being built with response times below 1 microsecond.

As technology has advanced over the years, ESs have been growing in size, complexity and speed. Let us mention three application areas for ESs with very different system requirements:

- *Consumer products*, where low price and high performance are required (e.g., mobile phones, cars and video recorders).
- *Safety-related systems*, where product reliability and safety are of major concern (e.g.,

A previous short version of the paper appeared in *Proceedings of the IEEE International Symposium on Industrial Electronics*, Bled, Slovenia, July 12–16, 1999.

aircraft and medical equipment). In these systems, it has to be possible to formally verify their performance.

- Complex *large-scale communication and control systems*, where time-to-market is very important (e.g., telephone switching stations and oil platforms).

What characterizes all different types of ESs is that they are typically *dedicated systems*. When developing such systems using traditional design methodologies, the hardware and software components are developed separately and their final integration is based on an *ad hoc* method. Such an approach increases time-to-market and cost, and very often decreases quality and reliability of the product. Advances in some key enabling technologies, like system-level specification and simulation environments, formal design and verification methods, and high-level synthesis and framework technology, have enabled the ***hardware/software co-design*** of ESs [2].

2. An Embedded Operating System

The major part of hardware/software co-design is the partitioning. This is not just partitioning between hardware and software. It takes also reprogrammable hardware into account, letting designers analyse the additional tradeoffs that this kind of design implies. Configurable hardware has advantages and disadvantages: it enables some flexibility and the use of more complex scheduling algorithms, and increases the cost of debugging hardware, respectively. Therefore, many developers of ESs choose to run their applications on top of either an *embedded operating system* or *real-time executive*.

As applications of ESs get more complex, algorithms have to be broken down into tasks. The central function of an embedded operating system is to remove the burden of task handling from the code writer. Detailed knowledge of interrupts, timers, analog-to-digital converters, etc., are no longer needed. Besides, an application software that uses facilities provided by an embedded operating system is easier to be analysed.

An embedded operating system is smaller and simpler than the mini/mainframe type [5]. Its middle part is the *real-time executive* (RTEX). In embedded computers, each application task is written separately, calling on other system activities (resources) via the RTEX. The RTEX itself controls all multitasking activities:

- scheduling and dispatching (deciding when and why tasks should be run),
- mutual exclusion (policing the use of resources shared between tasks),
- synchronization and data transfer (making tasks possible to communicate with each other).

To perform these activities the executive calls on the procedures, which are facilities provided by the inner part of the real-time operating system, the *kernel*. The outer part of an embedded operating system consists of the *application program interfaces* and *real world interfaces*. The latter handle the system hardware that is driven by standard software routines, which typically includes AD/DA controllers, keyboard controllers, programmable timers, configurable I/O ports, serial communication devices, and the like.

The primary multitasking activity controlled by the RTEX is **scheduling**. However, it is an *NP-complete combinatorial optimization* (CO) *problem* [14]. It means that the problem of deciding when and why tasks should be run cannot be solved by an exact algorithm (such as linear integer programming [15]) because the number of schedules grows very fast with the number of application tasks.

This paper deals with *heuristic scheduling concepts* based on neurocomputing (i.e., the *Hopfield-Tank neural network (H-T NN)* [11]) and genetic search and optimisation. Although for scheduling algorithms applying such concepts there is no guarantee that for each problem case a solution found is the best, polynomial bounds on their computation times can be given.

3. Preliminaries

Let us make the main requirements and constraints of modern ESs:

1. Application tasks are of **periodic** and **aperiodic types**, and are interrelated by **precedence** and **mutual exclusion relations**. In general, they are **pre-emptive**, but under some circumstances for safety and security reasons, a task may temporarily become non-preemptive. This mutual exclusion requirement is modelled by using *critical sections*. Under certain constraints critical sections are allowed to be pre-emptive. Moreover, tasks are **not equally important** for the system.
2. A configurable hardware **kernel co-processor** is used to remove the burden of supporting the multitasking operations from the target processor. Such an approach is required to keep unpredictable time overheads on the target processor to an absolute minimum. A time overhead happens when the multitasking activities are performed entirely by software that is executing by the target processor. Time taken to execute this software is lost to the application tasks. However, in fast, complex or hard real-time ESs, this is not suitable. **Figure 1** shows an architecture suggested by Cooling and Fox [6] and proceeds from the dual-processor approach proposed by Halang and Colnarič [9, 4]. Here, low-level scheduling and dispatching are implemented in co-processor configurable hardware, while other kernel functions are implemented in co-processor software.
3. The scheduling concepts apply to **single target processor systems** only.

A brief description of the scheduling problem: Given a set of periodic and aperiodic tasks that are interrelated by precedence and mutual exclusion relations, a ***feasible** schedule* must be found. A schedule is feasible when tasks are ordered so that all their deadlines are met, with the minimum number of task pre-emptions and shortest possible time spent waiting for lower priority tasks to finish execution in critical sections.

In **Section 4**, we review the Hopfield-Tank neural network, and describe the approach used to solve the scheduling problem. We continue with the discussion regarding several of the H-T NN modifications, including the genetic search and optimization [8]. In **Section 5**, we propose a complex scheduling algorithm that is a mixed approach of the modified H-T NN and a specialized heuristic scheduling policy, an extended version of the *Earliest Deadline First* (EDF) [3]. Finally, in **Section 6**, we summarize and conclude the paper.

4. The Hopfield-Tank Neural Network

The H-T NN is a *Hopfield neural network* [10], which is a biologically inspired mathematical tool that has extensively been used to solve difficult CO problems, like the Travelling Salesman Person problem [7]. The great advantage of this network is that it generates solutions without necessity of training iterations. However, it has also a disadvantage that it does not always move from an initial state to the nearest stable

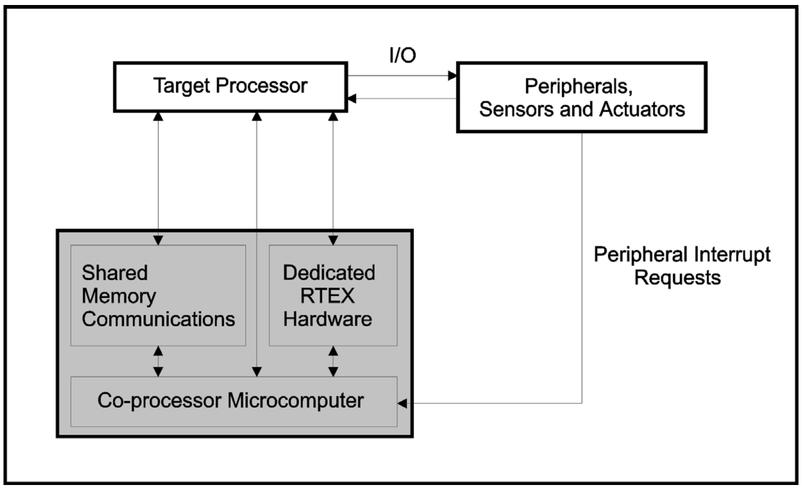
I/O
Target Processor
Peripherals, Sensors and Actuators
Shared Memory Communications
Dedicated RTEX Hardware
Co-processor Microcomputer
Peripheral Interrupt Requests

Fig. 1. Suggested Architecture.

state. In order to improve the solution quality, we propose a stochastic approach of embedding the principles of genetic search and optimisation into the H-T NN to escape from local minima.

Let us briefly describe Hopfield networks. They belong to the class of *recurrent* neural networks. In a continuous version of the Hopfield network, which is an alteration of the primary discrete Hopfield network and origination of the H-T NN, the input of neuron i and its state are denoted by u_i and V_i, respectively. V_i has the range [0,1], and is bounded by the asymptotes of the *sigmoid function* $g(u_i)$. In most cases, this function meets the Cohen-Grossberg requirements for stability [18]. The basic idea is that a Hopfield network is started at some initial state, and converges to the one of a finite number of stable states, which is closest to the initial state. This corresponds to minimizing an energy function that formulates the optimization problem as a constrained problem. A typical choice of the energy function is

$$E = -\frac{1}{2}\sum_{\substack{i,j\\ i\neq j}} \omega_{i,j} V_i V_j + \sum_i I_i V_i \qquad (1)$$

which is a *Liapunov function* [4] for the system. Parameter $\omega_{i,j}$ determines (weights) the effect of the connections from neuron i to j, while I_i denotes the bias current of neuron i.

One of the principal advantages of Hopfield networks is the rapid computation power and speed which can be obtained through hardware implementation, and this consideration is even more valuable for ESs [17].

4.1. Formulation of the Scheduling Problem

We applied the H-T NN to the scheduling problem of *periodic tasks*, which is a subproblem of task scheduling. Periodic tasks have an important feature that their invocations repeat in the same pattern through consecutive *major time frames* (MTFs). Possibly in most applications of embedded computers, the MTF is identified by a preliminary harmonisation of task frequencies (periods). This feature allows us to concentrate on scheduling tasks within one MTF.

Let us define a set of n invocations of n_p preemptive and partially ordered periodic tasks, where $n = \sum_{i=1}^{n_p} F/p_i$, F is the length of the MTF and p_i is the period of task P_i. Given an initial ordering of the set, a feasible uniprocessor schedule must be found.

The problem solution may be represented as a $(n_p + 1) \times F$ permutation matrix. Each of the first n_p matrix rows corresponds to a particular periodic task, while each column corresponds to a particular position in the schedule. The last matrix row is an auxiliary one, and it corresponds to the processor state (idle/active).

Example 4.1. *An embedded computer may be required to manage a sample loop at 50Hz, while performing signal processing at 20Hz, and displaying results at 10Hz. Details of the application tasks are given in* **Table 1**.

Processor execution time should be organized as a series of time slots. Each one has the same duration, in this case 10ms. Tasks may be allocated to specific time slots, resulting in the scheduling of **Figure 2**. The point at which the allocation repeats defines the duration of the MTF, here being 100ms.

In **Table 2**, a translation of the diagram of **Figure 2** into a permutation matrix is given:

$$\begin{array}{r|cccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
P_1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
P_2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
P_3 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\text{Processor} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\end{array}$$

Table 2. Permutation Matrix Representing a Feasible Schedule.

Periodic Task	Period (ms)	Computation Time (ms)	Deadline (ms)
P_1	20	10	20
P_2	50	10	40
P_3	100	20	80

Table 1. Example Task Set.

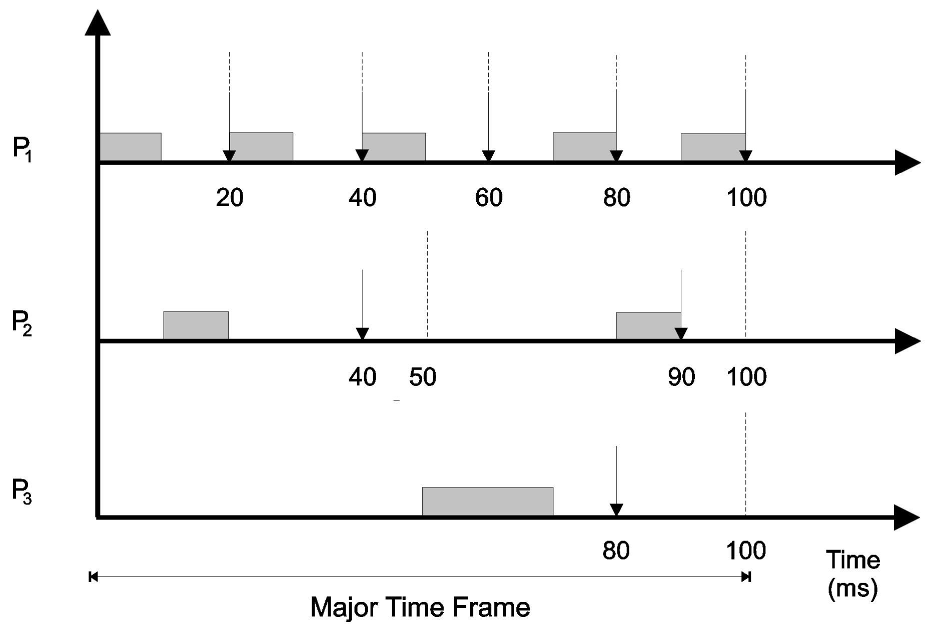

Fig. 2. Illustration of a Feasible Schedule.

The matrix elements of the first n_p (three) rows and F (ten) columns with the value 1 define the active state of a corresponding task. Similarly, the elements from the first n_p rows and F columns with the value 0 define the ready-to-run or suspended state of a corresponding task. The elements of the last matrix row define the processor state, which is idle if the element value is 1, and active otherwise. Time periods that correspond to the matrix elements of the last row with the positive value could be used to schedule aperiodic tasks. However, in this simple example, aperiodic tasks are not considered. But in modern ESs, the multitasking design has to forsee situations when aperiodic tasks arrive (are put into the ready-to-run state). Therefore, a schedule of periodic tasks has to be formed so that processor loading is spread evenly over the MTF to give the best performance in the presence of aperiodic tasks.

4.2. The H-T NN Model

In the H-T NN model for the scheduling problem, each entry (P_i, k) in the permutation matrix (where P_i denotes the task/row and k the position/column) is represented by neuron state $V_{i,k}$. While matrix values are allowed to vary continuously, the final result is interpreted by replacing each entry with either 1 or 0, depending on whether the neuron state value is high or low. **Figure 3** shows the structure of the H-T NN of n neurons, where n corresponds to $(n_p + 1, F)$.

The scheduling problem has to be encoded so that its *cost* and *penalty functions* correspond to the ***network energy function***. It is required that the network favours stable states that correspond to the problem permutation matrices representing feasible schedules. Thus, in a state representing a feasible schedule,

(a) at most one element in each matrix column, and

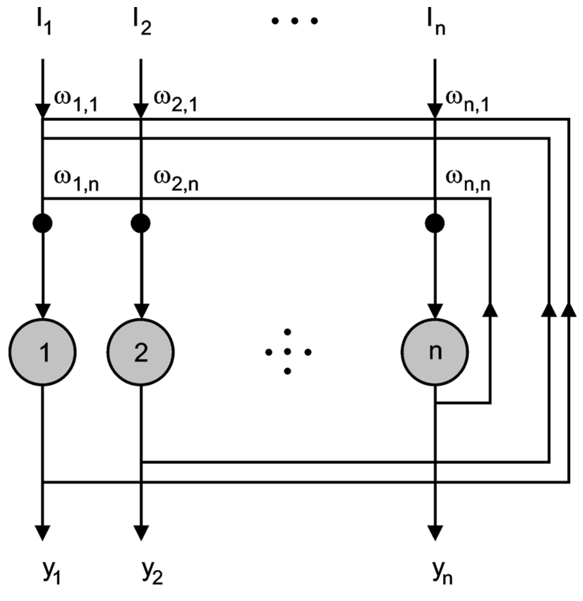

Fig. 3. H-T NN Model for the Scheduling Problem.

(b) exactly $(c_i + B_i)F/P_i$ elements in the matrix row corresponding to task P_i must be of value 1, and all the others of value 0. The parameters c_i and B_i denote the worst-case computation time and blocking time of task P_i, respectively. Task P_i is blocked when it is waiting for lower priority tasks to finish execution in critical sections.

For example, the terms of the energy function, which satisfy issue **(a)**

$$\left(1 - \sum_{i=1}^{n_p+1} V_{i,k}\right)^2 + \sum_{i=1}^{n_p+1} V_{i,k}(1 - V_{i,k}) \quad (2)$$

give a minimum when exactly one of the network neurons that correspond to k-th ($k = \{1, 2, \ldots, F\}$) matrix column have an output of 1 (i.e., there is one entry of 1 in each matrix column). The requirement that no more than one element in each matrix column can be of value 1 requires an addition of F slack neurons in the network. These correspond to the elements of the last matrix row that represents the processor state (idle/active) in each time slot of the MTF.

In a similar way we satisfy issue **(b)**, as well as other problem constraints, like partial ordering of tasks related by precedence relations, critical sections, infrequent pre-emptions of running tasks, and even spread of processor loading [14].

Once the energy function is formed, it can be used to derive ***strength of connections*** between pairs of neurons ($\omega_{(i,k),(j,l)}$), as well as ***values of external inputs*** to the neurons ($I_{i,k}$). For example, if we expand equation (2) and ignore the constant, we can rewrite the **(a)** part of the energy function as

$$-\frac{1}{2} \sum_{i=1}^{n_p+1} \sum_{\substack{j=1 \\ j \neq i}}^{n_p+1} \omega_{(i,k)(j,k)} V_{i,k} V_{j,k} - \sum_{i=1}^{n_p+1} I_{i,k} V_{i,k} \quad (3)$$

which is identical to the typical form of the energy function of the H-T NN (see equation (1)), provided we make the following definitions:

$$\omega_{(i,k),(j,k)} = \begin{cases} -2, & i \neq j \\ 0, & \text{otherwise,} \end{cases} \quad (4)$$
$$I_{i,k} = 1.$$

When these parameters are defined for all the problem constraints, the H-T NN ***updating rules***:

$$u_{i,k}^{(t+1)} = u_{i,k}^{(t)} + \Delta t\Big(\sum_{j=1}^{n_p+1} \sum_{l=1}^{F} \omega_{(i,k)(j,l)} V_{j,l} + I_{i,k} - u_{i,k}^{(t)}\Big), \quad (5)$$
$$V_{i,k} = g(u_{i,k}) = \frac{1 + th(\lambda u_{i,k})}{2}$$

and ***initial conditions***:

$$u_{init} = -\frac{ath\left(2\frac{1}{n_p+1} - 1\right)}{\lambda} \quad (6)$$

can be generated. In order to prevent the network being trapped in an unstable equilibrium in which the voltage of each neuron is equal, a certain amount of noise must be added to this initial value of equation (6). A random value distributed linearly on $\pm 0.1 u_{init}$ is added to the mean value to give the initial voltage on each neuron.

4.3. Evaluation

We simulated this H-T NN model by using *Mathematica* [19] as a prototyping tool for experimenting with the neural network paradigm. The simulation results showed that the constraints of the task scheduling problem are competing with each other. Problem solutions were local minima and not feasible schedules. **Figure 4** shows a distribution of solutions generated by the H-T NN for the problem case of **Example 1**. The largest number of simulation runs gave locally optimized solutions, and only few solutions were either very close to or very far away from the global optimum.

4.4. Method Modifications and Extensions

In order to improve the quality of the solutions, we have made some modifications and extensions of the H-T NN.

First, we defined ***weighting parameters*** for each term in the network energy function using an expensive trial-and-error approach. We could state that good values for constraint weighting parameters exist in very narrow and difficult-to-find regions of the parameter space.

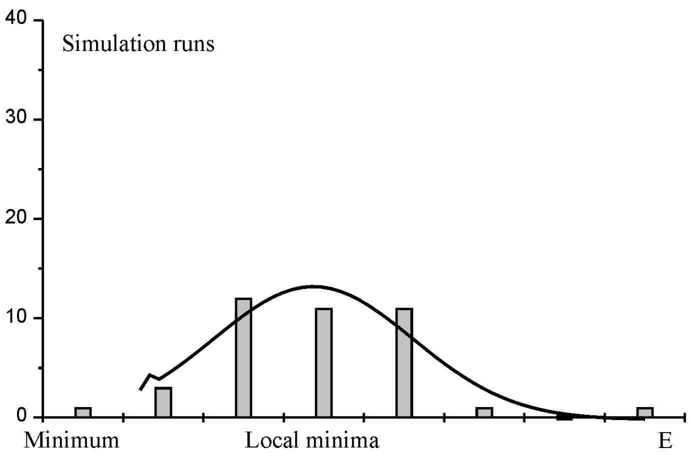

Fig. 4. Distribution of the Scheduling Problem Solutions.

The next step was an application of ***the Increasing Gain Hopfield Network*** model [16]. The idea of this model is that the *gain* (λ, see equation (5)) is increasing exponentially during the state evolution.

Finally, we tried to improve the quality of network results by ***genetic search and optimisation***:

- First, we used the elitism approach [12] and selected 25 best above-average H-T NN solutions of the scheduling problem to generate an *initial population*. These solutions were encoded using binary strings of length of $(n_p+1)F$. String positions corresponded to task locations in the schedule, and their values of 1 or 0 to the running or suspended state, respectively. In other words, j-th position with value of 1 implied task i to start running at time $(k-1)\Delta t$, where $j=(i-1)F+k$, $k \in \{1,2,\ldots,F\}$ and Δt denoted the basic time slot. Similarly, j-th position with value of 0 implied task i to be suspended at time $(k-1)\Delta t$.
- Then, a *cost function* needed to evaluate fitness of population members was selected. We decided to choose the same form as for the H-T NN energy function:

$$E = -\frac{1}{2}\sum_{\substack{i,j\\ i\neq j}} \omega_{i,j} V_i V_j + \sum_i I_i V_i \qquad (7)$$

  where $\omega_{i,j}$ is the connection weight between string positions i and j, V_i and V_j are binary values of i and j respectively, and I_i is the bias value i. We obtained the connection weights and bias values by translating the terms of the problem constraints into the form of the H-T NN energy function.
- Finally, we applied multi-point crossover and mutation *genetic operators* to recombine information from two high-ranked parent strings (chromosomes) drawn from the current population. The crossover operator produced two offsprings by exchanging one or more substrings between the parents. The mutation operator altered values at the randomly selected string positions. Mutation was applied at far lower rate than crossover. Crossover may not always work the way we want to. By occasionaly subjecting each of the genes in a chromosome to a small probability of mutation, the results of a bad crossover can be reversed.

The simulation results of the H-T NN applied to the task scheduling problem [14] showed that, with respect to all modifications and extensions, this heuristic method is still ***unpredictable***. The *rejection rate* of 20% at the *processor utilisation* of 90% (i.e., the probability that a readied-to-run task will fail to meet its deadline) is favourable when compared with other specialized scheduling heuristics [13]. In Figure 5, it can be seen that the rejection rate of the Earliest Deadline approach, which has proved to be the most efficient specialized scheduling heuristic, is up to 10% at the processor utilisation of 20%. But there is a problem that by using the H-T NN approach in real-time, we cannot guarantee that at least the most important application tasks will meet their deadlines. However, the method could be very efficient for scheduling problems without competing constraints.

5. Mixed Approach

In order to guarantee the best possible performance, we suggested a mixed approach for scheduling *periodic* and *aperiodic tasks* in embedded and real-time systems.

The H-T NN could be used for time-driven scheduling of periodic tasks within the MTF. Schedules produced by the H-T NN carry information about the processor activity/idleness. The elements of the last row in the permutation matrix with the value of 1 denote time slots in which the processor is idle.

When a given aperiodic task is readied, a current schedule of periodic tasks and already readied-to-run aperiodic tasks must be ***dynamically upgraded***. We use an extension of the EDF policy [14] for dynamic upgrading of task schedules. This policy extension considers realistic factors of tasking in real-time and ESs. We extended the EDF policy with an *acceptance test*, which is calculated each time a new aperiodic task is readied-to-run:

- The testing algorithm reuses the permutation matrix generated by the H-T NN as a *slack* search domain. A slack is an amount of time that can be used to execute a given task without causing any other more important task to miss its deadline.
- The algorithm takes into account the problem of the search space limitation caused by the MTF bound. Besides, it considers variations in task computation times (instead of fixed worst-case computation times).

In comparison with the EDF, the extension gives a good performance. We used *processor rejection rate* (see **Figure 5**) and *weighted guarantee ratio* (see **Figure 6**) as two measures of performance. The rejection rate is the probability that a ready-to-run task will fail to meet its deadline. The weighted guarantee ratio is the performance metric that reflects both the percentage of tasks which make their deadlines, and their relative worth to the system. It is expressed as $WGR = 100\frac{\sum_{i=1}^{n}(e^{i-1}|\{\tau_G\}|)}{\sum_{i=1}^{n}(e^{i-1}|\{\tau_A\}|)}$, where i denotes the task importance level (tasks may be differently important for the application), e^{i-1} corresponds to the default weight of task importance level i, $|\{\tau_G\}|$ is the total number of tasks at importance level i, that are guaranteed to meet their deadlines, and $|\{\tau_A\}|$ is the total number of all tasks at this level [1].

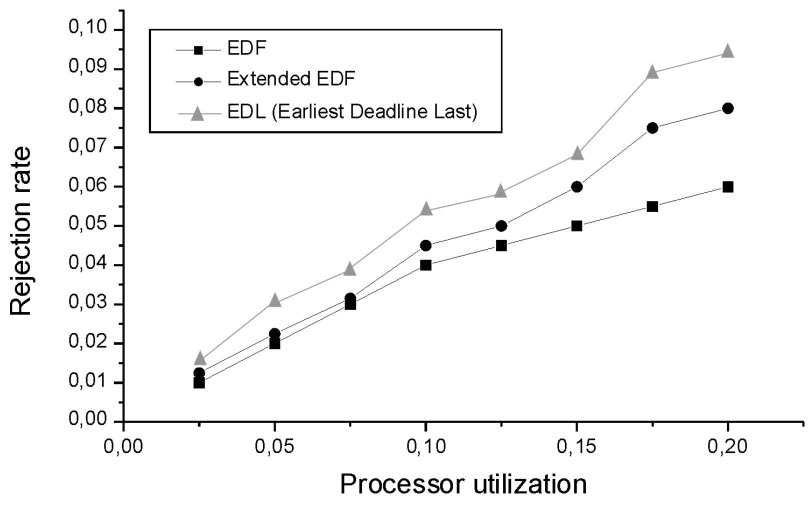

Fig. 5. Rejection Rate vs. Processor Utilisation.

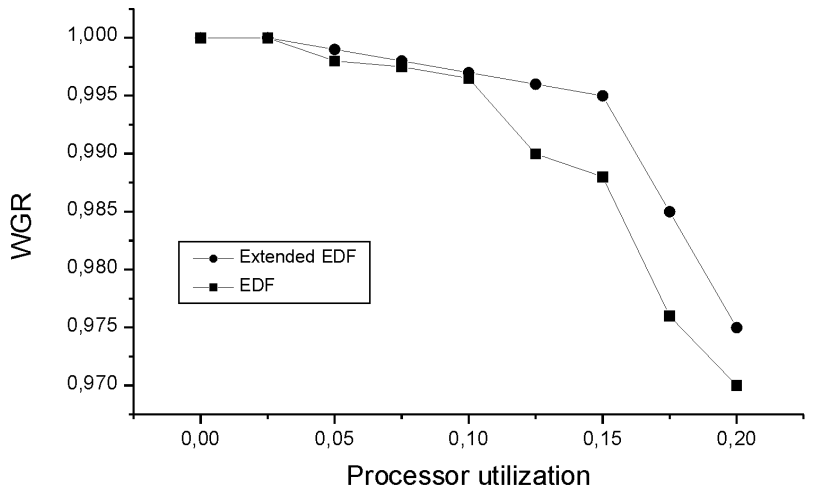

Fig. 6. Weighted Guarantee Ratio vs. Processor Utilisation.

The reader interested in EDF and its extensions is referred to [13, 14].

The mixed task scheduling approach could be used to solve the problem of jointly scheduling periodic and aperiodic application tasks in hybrid static/dynamic systems for the following reasons:

- It attempts to service both periodic and aperiodic types of interrelated tasks without causing their deadlines to be missed.
- Priority is given to the most important tasks.
- Our goal was to minimise the amount of calculation. The algorithm is polynomial in time complexity.
- Moreover, it offers remarkable scheduling predictability since the algorithm refers the timetable of periodic tasks given as a permutation matrix, which is built (and modified when needed) by the H- T NN before application run-time.

6. Conclusion

The mixed approach of evolutionary computation techniques and the Earliest Deadline First policy is a new concept, of our own devising. It has a sound theoretical basis. Unlike many other scheduling policies, it takes into account practical, realistic factors of tasking in embedded systems. Its performance was validated by simulation. We used *Mathematica* as a prototyping tool for experimenting with our task scheduling concept.

Acknowledgement

The author gratefully acknowledges the contributions of Professor Jim E. Cooling.

References

[1] S. R. Biyabani, J. A. Stankovic, The Integration of Deadline and Criticalness in Hard Real-Time Scheduling, presented at the *Proceedings of the real-time systems symposium*, (1988) Huntsville, Alabama.

[2] R. Camposano, J. Wilberg, *Design Automation for Embedded Systems*, Kluwer Academic Publishers, Boston, 1996.

[3] M. I. Chen, K. J. Lin, Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems, *Real-Time Systems — The International Journal of Time-Critical Computing Systems*, 2:4 (1990) pp. 325-345.

[4] M. Colnarič, W. A. Halang, *Predictability of Temporal Behaviour of Hard Real-Time Systems*, PhD. Thesis, University of Maribor, Slovenia, 1992.

[5] J. E. Cooling, *Real-Time Software Systems*, International Thompson Computer Press, 1997.

[6] A. M. Fox, J. E. Cooling, A Co-Processor Scheduler for Embedded Real-Time Systems, Presented at the *Proceedings of Mechatronics*, (1996) Guimaraes, Portugal, 1996.

[7] M. R. Garey, D. S. Johnson, *Computers and intractability a Guide to the Theory of NP-Completeness*, Mathematical series, W. H. Freeman and Company, San Francisco, 1979.

[8] D. E. GOLDBERG, *Genetic Algorithms in Search, Optimisation, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[9] W. A. HALANG, A. D. STOYENKO, *Constructing Predictable Real Time System*, Kluwer Academic Publishers, 1991.

[10] J. J. HOPFIELD, Neurons with graded response have collective computational properties like those of two-state neurons, *Biophysics*, 81 (1984), 3088-3092.

[11] J. J. HOPFIELD, D. W. TANK, 'Neural' Computation of Decisions in Optimization Problems, *Biological Cybernetics*, 52 (1985), 141-152.

[12] K. A. DE JONG, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph. D. Thesis, University of Michigan, 1975.

[13] B. KOROUŠIĆ-SELJAK, Task scheduling policies for real-time systems, *Microprocessors and Microsystems*, 18:9 (1994), 501-511.

[14] B. KOROUŠIĆ-SELJAK, *Methods and Tools for Development of a Real-Time Kernel*, Co-Processor Unit. PhD. Thesis, University of Ljubljana (Slovenia), 1997.

[15] E. LAWLER, *Combinatorial Optimization: Networks and Matroids Holt*, Reinhart and Winston, 1976.

[16] M. A. ODIJK, *Two models solving the increasing energy problem of the Continuous Hopfield Network model*, Technical report 93-87, Faculty of Technical Mathematic and Informatics, Delft University of Technology, Julianalaan 132, 2628 BL Delft (The Netherlands), 1993.

[17] K. SMITH, M. PALANISWAMI, M. KRISHNAMOORTHY, Neural Techniques for Combinatorial Optimization with Applications, *IEEE Transactions on Neural Network*, 9:6 (1998), 1301-1318.

[18] G. A. TAGLIARINI, J. F. CHRIST, E. W. PAGE, Optimisation Using Neural Networks, *IEEE Transactions on Computers*, 40:12 (1991), 1347-1358.

[19] S. WOLFRAM, *Mathematica, A System for Doing Mathematics by Computer*, Addison-Wesley, 1996.

Received: April, 2000
Revised: June, 2000
Accepted: July, 2000

Contact address:
Barbara Koroušić Seljak
Computer Systems Department
"Jožef Stefan" Institute
Jamova 39
1000 Ljubljana
Slovenia
phone: +386 1 4773-363
fax: +386 61 2519-385
e-mail: `barbara.korousic@ijs.si`

BARBARA KOROUŠIĆ SELJAK was born in Ljubljana, Slovenia. She received the B. Sc., M. Sc. and Ph. D. degrees in computer science and informatics from the University of Ljubljana in 1989, 1992 and 1997, respectively. Presently, she works as a research assistant in the Computer Systems Department at the "Jožef Stefan" Institute in Ljubljana. Her research interests include embedded systems, real-time operating systems, and biologically inspired computing.