A Cellular Automaton Based Approach for Real Time Embedded Systems Scheduling Problem Resolution

Fateh Boutekkouk

IResearch Laboratory on Computer Science's Complex Systems ReLa(CS)^2,
Oum El Bouaghi University, BP 358, Algeria
fateh_boutekkouk@yahoo.fr

Abstract. Real Time Embedded Systems are becoming ubiquitous. Since these systems have autonomous batteries, their design must minimize power consumption in order to extend batteries life time. On the other hand, Cellular Automaton (CA) appears a good choice to simulate the future behavior of complex dynamic and parallel systems. Due to some intrinsic characteristics such as neighborhood and local transitions, CA can exhibit some complex behaviors. In this work, we apply CA to model the well known problem of Real time scheduling and eventually to optimize the power consumption of a Real time multi-cores embedded system with periodic tasks. CA algorithm is focused on the so-called technique: Dynamic Voltage Scaling (DVS). The proposed CA is a 2D grid.

Keywords: Real time embedded systems, Real time scheduling, Cellular automata, DVS, Power consumption.

1 Introduction

Real Time Embedded Systems (RTES) are reactive systems. They interact with the external world via sensors and actuators and must provide correct results but without missing a specified deadline. RTES may be hard, soft or firm. RTES are generally subjected to a variety of constraints beyond temporal ones such as surface, weight, power consumption and cost. RTES include a logical part (or application) which is composed of a set of dependent and/or independent tasks, periodic and/or aperiodic tasks. Each task is characterized by a set of parameters such as period, arrival time, execution time which may be expressed as WCET (Worst Case Execution Time), ACET (Average Case Execution Time) or BCET (Best Case Execution Time), priority, etc. The hardware part is composed of a set of embedded processors which are connected by a communication support such as buses or an embedded network. Each processor is also characterized by a set of parameters like clock frequency, local memory size, power consumption, programmability level, etc. Since RTES have autonomous batteries, their design must minimize power consumption in order to extend batteries life time. Thus power consumption management and reduction become a challenge. To deal with this problem, researchers have proposed a set of power estimation and reduction techniques at many levels of abstraction and at both software and hardware stages.

R. Silhavy et al. (eds.), *Artificial Intelligence Perspectives and Applications,*
Advances in Intelligent Systems and Computing 347, DOI: 10.1007/978-3-319-18476-0_2

Among these techniques, Dynamic Voltage Scaling or DVS is becoming more and more interesting for power reduction [2, 3, 6, 11]. The idea behind DVS is to enable a processor to change its voltage level dynamically (during execution) thus the processor operates on a set of modes and the transition from a level to another level consumes a time and a power (we call this the overhead due to transition between modes). A high voltage implies a high clock frequency. Voltage reduction leads to a considerable reduction in power consumption but on the prize of clock frequency reduction that means a longer execution time. In a real time context, a longer execution time can lead to a deadline missing. Finding the right voltage level is a serious problem. To solve this dilemma, we will assume that each embedded processor have three functioning modes: The high frequency mode with the highest power consumption level, the middle frequency mode with an acceptable power consumption level and finally the low frequency mode with the lowest power consumption level. Using some temporal information during tasks execution, we can choose the right modes that lead to optimal results. For instance by measuring the distance between the task deadline and its end time or measuring the processor usage ratio, we can find a good distribution of tasks on processors with appropriate frequencies levels. In this work, we investigate the idea of using cellular automaton (CA) to model and simulate real time tasks scheduling and allocation problems in multi-cores embedded systems.

CA [8] is a computation model used to model and simulate behaviors of complex dynamic and parallel systems. We can see a CA as a grid of cells together form a neighborhood. Each cell has a state and can change its state according to its neighborhood state. This is called a local transition rule. By applying local rules on cells synchronously or asynchronously, the CA changes its global state and some complex emerging behaviors can be produced. This is a primary inherent characteristic of CA. Informally, our proposed CA is a two dimensional grid where each cell represents the information of the task allocation on a processor with a certain frequency mode. Thus a cell state is a triplet (task, processor, frequency mode). Of course when more that one task is allocated to the same processor, a scheduling policy (example Rate Monotonic, Deadline Monotonic, etc.) must be defined. In order to simulate the CA, we have to define some local transition rules. Such transition rules can for instance, change the allocation, the priority or the frequency mode of a task. By applying these rules continually, we can observe some emerging behaviors or some good configurations with minimal power consumption. The rest of paper is organized as follows: section two is devoted to some pertinent works. The formal definition of our proposed CA is presented in section three. In section four, we discuss our implementation and some results before the conclusion.

2 Related Work

Literature on employing CA to solve multiprocessors scheduling problem is rich. However, we can state that:

1. Most of these works cope with classical multiprocessors systems but not Real time embedded systems [4, 7, 10, 12].

2. Most of existing works do not take into consideration power consumption.
3. Some works target only bi-processors systems. Thus the corresponding CA is supposed binary [9].
4. Some work, hybrid CA with some optimization evolutionary heuristics such as genetic algorithms or ant colony to discover the best local rules [1, 5]. We think that this technique may consume a huge amount of time to find good solutions.

In this work, we try to formulate the Real time scheduling problem in embedded multi-cores systems including DVS strategy using a simple CA characterized by graphical capabilities and defined with some local rules to reduce the power consumption under temporal constraints (deadlines) and balancing between the usage ratios of processors.

3 Formal Definition

Formally a cellular automaton is defined as CA = (D, S, N, £, F) where:

D is the grid dimension = 2. Each column of the grid represents a task and each line represents a processor number to which the task is allocated.

S is the CA cells states. Each state is defined as a triplet (task name, processor number, frequency mode). N is the neighborhood of a cell, it is defined as follows:

In dynamic priority based scheduling, the neighborhood N of a task T is equal to the set of tasks allocated to the same processor as T and it includes T itself. Since tasks priorities change over time, all tasks allocated to the same processor have an impact on their states. For this reason, N includes all tasks allocated to the same processor.

In static priority based scheduling, the neighborhood N of a task T is equal to the set of tasks allocated to the same processor as T having higher priorities than T and it includes T itself. Since tasks priorities are pre-known, only higher priorities tasks have impact on lower priorities tasks allocated to the same processor. For this reason, N includes higher priorities tasks. **£** is the local transition function. **F** is the global transition function of CA.

£ : NOM x PROC x MODE --> NOM x PROC x MODE
NOM designate tasks domain (string).
PROC designate processors domain (natural).
MODE designate frequency mode (enumerate = (low, middle, high)).

We defined three transition functions as follows:

The first function is applied on tasks missing their deadlines in case of dynamic priority based scheduling EDF (Earliest Deadline first). The role of this function is to change the processor and/or frequency mode of a task. Normally, when applying this function, we expect the task will respect its deadline in next periods.

The second function is similar to the first one but it is applied in case of static priority based scheduling DM (Deadline Monotonic). The third function is applied on tasks respecting their deadlines in the case of dynamic and static priority based scheduling.

The role of this function is to change the processor and/or frequency mode of a task. Normally, when applying this function, we expect the power consumption will be reduced. The section bellow details the algorithms of the three transition functions.

```
Algorithm1
begin
 choose randomly a task T missing its deadline allocated to processor Pi
  if  (mode (T) = high) then
    if (exists a processor  Pj)  and  (its rate_usage  < 0.33)) then
              migrate T on Pj
              set mode(T) to 'low'
       else
          if (exists a processor Pj) and (its rate_usage < 0.66)) then
               migrate T on Pj
               set  mode(T)  to 'middle'
          else
            if (exists a processor Pj) and  (its rate_usage < 1 )) then
               keep T on Pi
               set mode(T) to 'high'
             else
                 if (mode(T) = middle) then
                    keep T on Pi
                    set mode(T) to 'high'
                  else
                    if (mode(T) = low) then
                       keep T on Pi
                       set mode (T) to 'middle'
                     end if
end
```

The idea behind algorithm 1 is to migrate a task missing its deadline on a new processor which is not very busy and according to its ratio usage, we fix the right frequency mode so we keep energy consumption at lower levels if it is possible. If there is no chance to find such a processor, we keep the task on the same processor and set the frequency mode to lower levels.

The idea of algorithm 2 is similar to the first one with a bit difference in neighborhood.

Algorithm 3 tries to reduce the power consumption by measuring the distance between the task deadline and its end execution time (Tend_exe) and according to the task frequency mode and processor ratio usage, we decide whether we migrate or keep the task on the same processor.

Algorithm2

```
begin
choose randomly a task allocated to processor Pi whose neighborhood includes at least
a task T missing its deadline
  if (mode(T)= high) then
     if (exists a processor Pj) and (its rate_usage < 0.33)) then
              migrate T on Pj
             set mode(T) to 'low'
        else
           if (exists a processor Pj) and (its rate_usage < 0.66)) then
                migrate T on Pj
                set mode(T) to 'middle'
            else
              if (exists a processor Pj) and (rate_ usage <1))   then
                 keep T on Pi
                 set mode(T) to 'high'
               else
                  if (mode(T) = middle) then
                     keep T on Pi
                     set mode (T) to 'high'
                   else
                     if (mode (T) = low) then
                       keep T on Pi
                       set mode (T) to 'middle'
                     end if
   end
```

Algorithm3

```
begin
choose randomly a task T respecting its deadline allocated to processor Pi
if (mode(T) = high  and  (deadline−Tend_exe)/deadline > 0.5) then
    keep T on Pi
    set  mode(T) to 'middle'
    else
      if (mode (T)= high  and (deadline−Tend_exe)/deadline < 0.5 ) then
          if (exists a processor Pj) and (its rate_usage < 0.33) then
               migrate T on Pj
               set mode (T) to 'middle'
           else
              if (mode(T)=middle  and (deadline−Tend_exe)/deadline > 0.5 ) then
                 keep T on Pi
                 set mode (T) to 'low'
               else
                  if (mode (T) = low) then
                     keep T on Pi
                  end if
   end
```

4 Motivational Example

In order to test our approach, we will consider an example of a system including 20 periodic tasks with P (period) = 30 cycles and 5 processors. All tasks information are indicated in table 1. We note that arrival dates of tasks are generated randomly between [0, αi] where αi is given (i is the number of the period). Initially tasks allocation is done randomly. Each processor is characterized by a color and three frequency modes that are: High frequency mode or H (frequency = 1, powerPerCycle = 2 watt).

Middle frequency mode or M (frequency = 1.5, powerPerCycle = 1.5 watt).

Low frequency mode or L (frequency = 2, powerPerCycle = 1 watt).

The actual WCET of a task, noted AWCET = WCET * frequency.

The power consumed by a task = AWCET * powerPerCycle. The overhead due to transition between modes is given in table 2. All tasks allocated to the same processor will be colored by the processor color. Tasks missing their deadline will be colored by black. Simulation time = 15 periods.

Table 1. Tasks parameters

Task Id.	Processor	WCET	Deadline	Mode	Arrival
Task0	P0	3	5	H	0
Task1	P4	3	6	M	2
Task2	P2	3	4	M	2
Task3	P1	5	9	H	0
Task4	P4	5	8	M	3
Task5	P3	4	5	M	1
Task6	P3	3	3	H	0
Task7	P4	4	5	M	1
Task8	P3	2	5	M	2
Task9	P3	3	5	M	2
Task10	P1	4	6	H	1
Task11	P4	2	4	L	0
Task12	P0	2	3	M	2
Task13	P3	3	3	H	1
Task14	P3	2	4	L	1
Task15	P0	5	7	L	2
Task16	P0	2	2	L	0
Task17	P3	3	6	H	1
Task18	P4	4	8	H	0
Task19	P3	1	3	L	0

Table 2. Transitions overheads

Transition	Transition Time (cycles)	Transition Power (watt)
H to M	1	1
H to L	1	1
M to H	1	1
M to L	1	1
L to H	1	1

Figures from 1 to 5 show respectively the CA state in the first period, CA progression in an arbitrary period, processors usage ratios, processors power consumption, number of tasks missing their deadlines for processor P0 using EDF. For the sake of space, we do not show results of other processors.

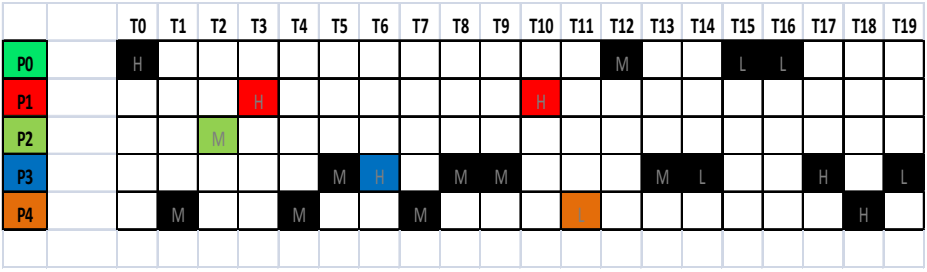

Fig. 1. CA state in the first period using EDF

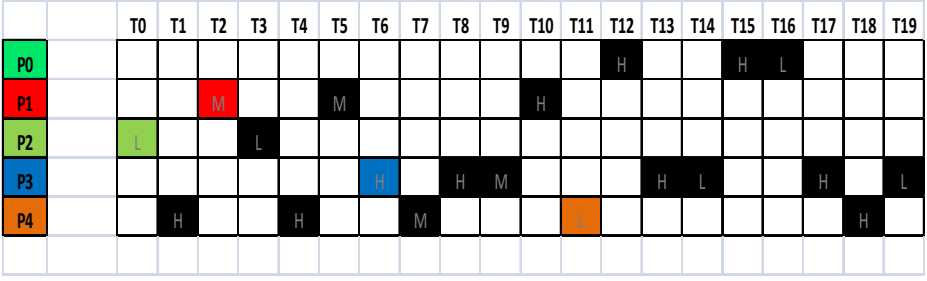

Fig. 2. CA progression

Processor P0 shows a decreasing in its usage ratio, power consumption, and the number of tasks beyond their deadlines over early periods, however, when the CA progresses, these parameters become more stable. Figure 6 shows the overall system power consumption evolution over simulation periods. We can remark that during first simulation periods, the consumed power was in its low level because frequencies modes were set to 'low' that means a big number of tasks missed their deadlines. When the CA progresses, the consumed power increases to its high level which corresponds to high frequencies modes. At this stage, all the tasks respect their deadlines but the consumed power is maximal. After that, we observe a certain alternation between low and high levels before

power falling. This alternation is due to the fact that CA applies local rules to compromise between power consumption and tasks deadlines respect.

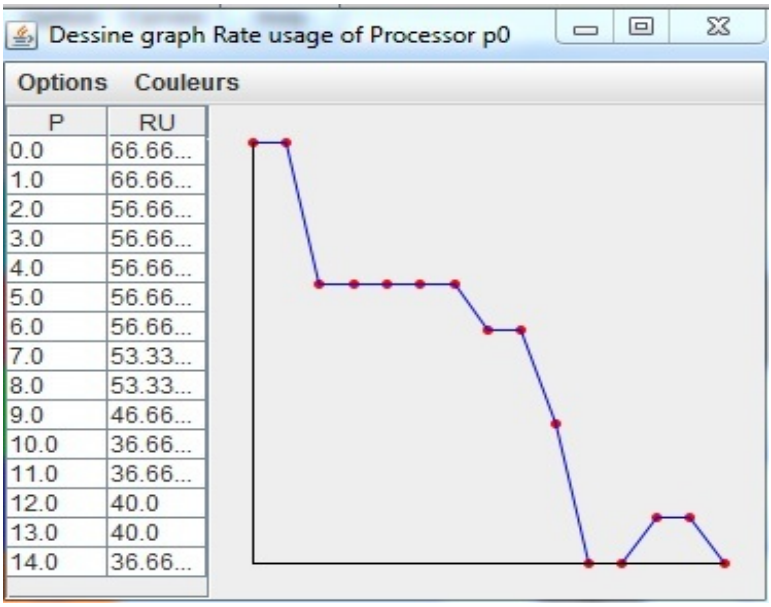

Fig. 3. Processor P0 usage ratios progression

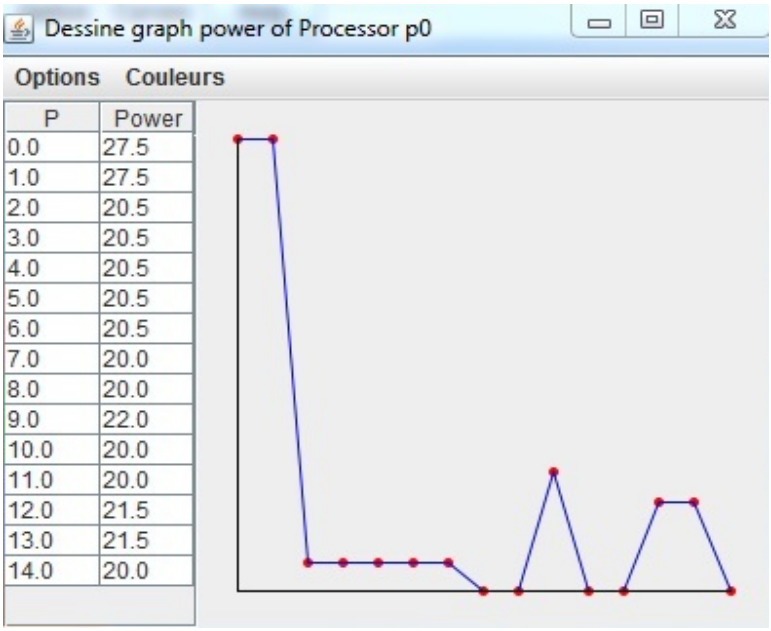

Fig. 4. Processor P0 power consumption progression

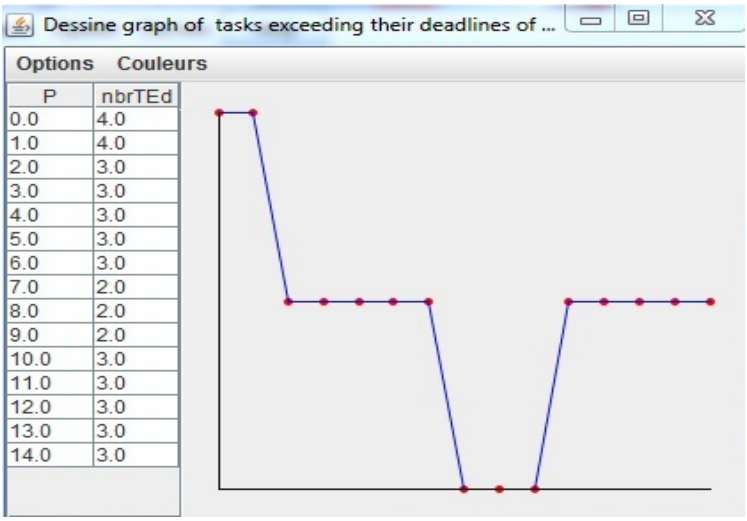

Fig. 5. Number of tasks missing their deadlines progression for P0 processor

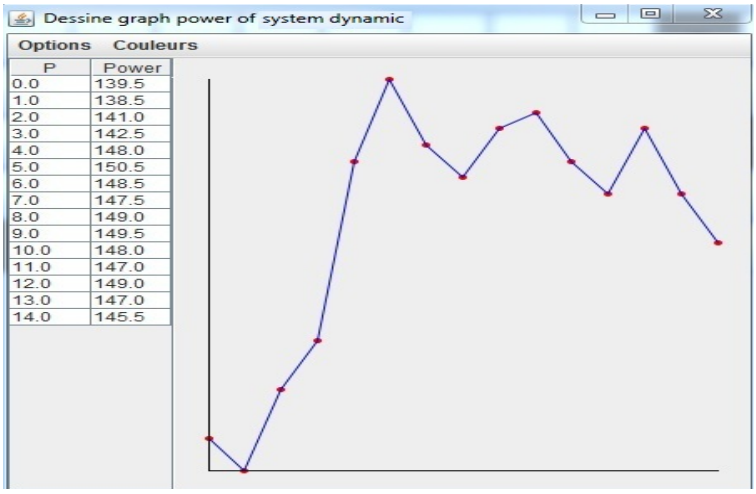

Fig. 6. System power consumption progression

5 Conclusion

In this paper, we presented a new approach to resolve the scheduling and allocation problems in real time embedded systems with periodic tasks using cellular automata. Our CA is 2D grid. We defined three local rules to change the state of cells. These rules try to optimize power consumption, balancing usage ratios of processors and minimize the number of tasks missing their deadlines. We adopted two well known real time scheduling algorithms that are DM and EDF. We tested our CA on a simple

example whose results show that DM is better than EDF. The main advantage of our CA resides in its simplicity, its dynamic neighborhood and the set of local rules to compromise between power consumption and deadlines respect. According to our first experimentations and with comparison to genetic algorithms where the research space is very large, we can state that our CA is able to find good solutions in a short time by applying the most appropriate local rule depending on dynamic neighborhood state. By applying these rules continuously, we can observe a big enhancement in the quality of solutions. However, and in order to confirm this conclusion, we have to test our CA on other examples with different parameters.

References

1. Agrawal, P., Rao, S.: Energy-Aware Scheduling of Distributed Systems Using Cellular Automata. In: 6th Annual IEEE International Systems Conference (IEEE SysCon 2012), Vancouver, BC, Canada (2012)
2. Albers, S., Antoniadis, A.: Race to idle: new algorithms for speed scaling with a sleep state. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 1266–1285. SIAM (2012)
3. Aydin, H., Melhem, R., Mosse, D., Mejıa-Alvarez, P.: Power-aware scheduling for periodic real-time tasks. IEEE Transactions on Computers 53(5), 584–600 (2004)
4. Carneiro, M.G., Oliveira, M.B.: Synchronous cellular automata-based scheduler initialized by heuristic and modeled by a pseudo-linear neighborhood. Natural Computing Journal 12(3), 339–351 (2013)
5. Ghafarian, T., Deldari, H., Akbarzadeh-T., M.R.: Multiprocessor scheduling with evolving cellular automata based on ant colony optimization. In: 14th International CSI Computer Conference, CSICC 2009, pp. 431–436 (2009)
6. Ishihara, T., Yasuura, H.: Voltage scheduling problem for dynamically variable voltage processors. In: Proceedings of the International Symposium on Low Power Electronics and Design, Monterey, CA, pp. 197–202 (1998)
7. Laghari, M.S., Khuwaja, G.A.: Scheduling Techniques of Processor Scheduling in Cellular Automaton. In: International Conference on Intelligent Computational Systems (ICICS 2012), January 7-8, Dubai (2012)
8. Schiff, J.L.: Cellular Automata: A Discrete View of the World. Wileyinterscience (2007)
9. Seredynski, F.: Scheduling Tasks of a parallel program in two-Processor Systems with the use of Cellular Automata. Journal Future Generation Computer Systems Special Issue: Bio-inspired Solutions to Parallel Processing Problems 14(5-6), 351–364 (1998)
10. Seredynski, F., Zomaya, A.: Sequential and parallel cellular automata-based scheduling algorithms. IEEE Transactions on Parallel and Distributed Systems 13(10), 1009–1023 (2002)
11. Shin, Y., Choi, K., Sakurai, T.: Power optimization of real-time embedded systems on variable speed processors. In: Proceedings of the International Conference on Computer Aided Design, pp. 365–368 (November 2000)
12. Swiecicka, A., Seredynski, F., Zomaya, A.: Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. IEEE Transactions on Parallel and Distributed Systems 17(3), 253–262 (2006)