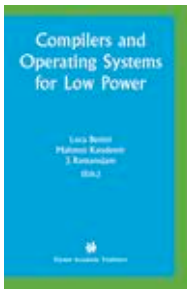

Compilers and Operating Systems for Low Power
Benini, Luca; Kandemir, Mahmut; Ramanujam, J. (Eds.)
Illustrated ed., 2003, 246 p., Hardcover
ISBN: 978-1-4020-7573-5

Ships within 2-3 weeks.

113,00 €

Print version
Recommend to others

All books by these editors
Benini, Luca
Kandemir, Mahmut
Ramanujam, J.

Related subjects
Computer Communications & Networks
Electronics & Electrical Engineering
General Computer Science
Programming, SWE & Operating Systems

About this book | Table of contents

Table of contents

Chapter #

A Modified Dual Priority Scheduling in Hard Real Time Systems to Improve Energy Saving.

M.Angels Moncusí*, Alex Arenas* and Jesus Labarta'
**Dpt d'Enginyeria Informàtica i Matemàtiques-Universitat Rovira i Virgili. 'Dpt d'Arquitectura de Computadors-Universitat Politècnica de Catalunya*

Abstract: We present a modification of the Dual Priority scheduling algorithm, for hard real-time systems, that takes advantage of its performance to efficiently improve energy saving. The approach exploit the priority scheme to lengthen the runtime of tasks reducing the speed of the processor and the voltage supply, thereby saving energy by spreading run cycles up to the maximal time constraints allowed. We show by simulation that our approach improves the energy saving obtained with a pre-emptive Fixed Priority scheduling.

Key words: energy-aware, on-line scheduling, hard real-time, dual-priority.

1. INTRODUCTION

The design of portable digital systems has a major drawback in the constraint of low power consumption [1] from the operability and lifelong of the systems point of view. A lot of efforts have been made during the last decade to minimize this problem, but the high performance of modern micro-processors and micro-controllers jointly with the increasing functionality of them obtained via software still requires improvements in the power-efficiency context.

In the use of scheduling strategies to save energy there exist two main approaches to reduce power consumption of processors, these approaches are speed reduction of the processor and power-down. The first approach consist in to turn low the clock frequency along with the supply of voltage whenever the system does not require its maximum performance. The second approach simply turns off the power when there are not tasks to execute in prevision, apart from the minimal amount of energy required by the idle processor state (clock generation and timer circuits). Both approaches are well suited for energy saving but their applicability should be accurately designed to obtain reliable operability, especially in hard real-time systems [2,3].

Recently, Shin and Choi [4] have proposed a power-efficient version of the Fixed Priority scheduling for hard real-time system that deal with the two approaches presented before. The main idea in their study is the use of a pre-emptive Fixed Priority scheduling (Rate Monotonic scheduling RMS [5] or Deadline Monotonic scheduling DMS) to organize the tasks according with the pre-emptive priority scheduler into a run queue that is used to exploit both, execution time variation and idle time intervals, to save energy by reducing speed and voltage or power down. The process ensures that all tasks meet their deadlines. However, the strategy of Shin and Choi [4] can only reduce the speed of the processor when there is only one task in the run queue, or bring the processor to power-down mode when there is an idle interval, otherwise the processor works at the maximum speed.

In this paper we present an improvement of the strategy followed by Shin and Choi [4] by using a modification of the Dual Priority scheduling, first proposed by Davis and Wellings [6]. We harness the ability of the Dual Priority to execute periodic tasks as late as possible to save energy.

The Dual Priority scheme was designed to execute aperiodic tasks without deadlines as soon as possible while preserving the deadline constraints of the periodic tasks. The algorithm is implemented as a three queue structure. The upper run queue, the aperiodic run queue and the lower run queue. Whenever a periodic task is ready to be executed enters the lower run queue, eventually this task can be pre-empted by an aperiodic task, and finally, if the task can not be delayed more because otherwise its deadline could be compromised, the task promotes to the upper run queue where its execution is prioritized.

This scenario is interesting even when no aperiodic tasks are present, as in our case of study, in this particular case the algorithm needs only two queues. The energy-reduction is obtained mainly by means of speed and voltage reduction and sometimes using power-down. Our approach consist in to run the tasks at the lowest speed that makes possible that the active task and the rest of tasks meet their timing constraints, without imposing the constraint of Shin and Choi [4] of only one task in the run queue to save energy, and power-down the processor when there is an idle interval.

This approach is especially interesting because the quadratic dependency of the power dissipation, in CMOS circuits, in the voltage supply [1]. The power dissipation satisfies approximately the formula

$$P \cong p_t C_L V_{dd}^2 f_{clk}$$

where p_t is the probability of switching in power transition, C_L is the loading capacitance, V_{dd} the voltage supply and f_{clk} the clock frequency. That means that it is always energetically favourable to perform slowly and at low voltage than quickly at high voltage.

The basic idea of the modified algorithm we present is to organize the run tasks in two levels of priorities. In the highest level there are the periodic tasks that their execution can no longer be delayed by tasks from the lower priority level otherwise they can miss their deadlines. The second level is occupied by those periodic tasks whose execution time can still be delayed without compromising the meeting of their deadlines. In its turn, each of the two levels is hierarchically organized according to any static priority assignment. To obtain an extra save in power another slight modification is introduced, the lower run queue is sorted by the promotional times instead of by fixed priorities. This approach is simple enough to be implemented in most of the kernels, in comparison with Shin and Choi [4], we only use an extra run queue and a promotion time for each periodic task in the system. Then, the amount of extra complexity introduced by this new algorithm is minimal.

The paper is organized as follows, in the next section we describe the basics of the Dual Priority scheduling. Section 3 is devoted to the modification of the algorithm to reduce energy consumption. Finally, in section 4 we present the experimental results and the comparison with Power Low Fixed Priority scheduling, and in section 5 we draw the conclusions.

2. DUAL PRIORITY SCHEDULING

We assume that the framework of the hard real-time system we are going to deal with is made up of periodic tasks[1]. These tasks — numbered $1 \le i \le n$ — are specified by their periods, worst case execution times and deadlines (T_i, C_i and D_i respectively).

The system is organized as concurrent tasks ruled by a pre-emptive priority-based scheduler whose details are described below. The computation times for context switching and for the scheduler are assumed to be negligible, this enable us to perform the analysis straightforward without danger of loosing generality. The extent to which these assumptions are realistic is discussed in the analysis of the algorithm given in [6], and it turns

1 The results are not exclusive for periodic tasks. We have considered only periodic tasks as a matter of simplicity.

out to be practical if the switch is subsumed to the worst case execution times of the different tasks.

The mechanics of the Dual Priority Scheduling algorithm is the following: Let us consider that the tasks have some initial priorities assigned according to a fixed priority criterion in such a way that two different periodic tasks have never the same priority. This initial priorities are altered by the scheduler according to the following scheme, first, two levels of priorities are organized, the highest level, or upper run queue (URQ) is for tasks that can no longer be delayed by less priority tasks otherwise they could miss their deadlines. The second level, or lower run queue (LRQ) is occupied by those periodic tasks whose execution time can still be delayed without compromising the meeting of their deadlines.

The scheduling algorithm is driven by the activation times of the tasks and the promotion instants from the LRQ to the URQ, whenever one of this time signals appears, in the following way, if:

a) The signal is the activation time (ta_{ik}) for the k^{th} instance of the periodic task i. In this case for all tasks with activation times less or equal to the current time t, the relative promotion time instant of task i (tp_i) is pre-computed as $tp_i = D_i - R_i$ (R_i corresponds to the worst case response time of task i [8]), this value can be computed off line and provides the maximum time a task can be delayed so that it can still meet its deadline. Those tasks with $tp_i=0$ are promoted to the URQ, and the rest are queued in the LRQ. After that, we compute the absolute promotion time instant for the k^{th} activation of task i in the LRQ as $tp_{ik} = ta_{ik} + L_i$, and a timer is activated to this value.
b) The signal is a promotion time instant (tp_{ik}) for the k^{th} instance of the periodic task i. In this case, all tasks in the LRQ with $tp_{ik} \leq tc$ (current time) are moved to the URQ. Now, tp_{ik} corresponds to $tp_{ik} = D_{ik} - R_i$, where D_{ik} is the absolute deadline for the k^{th} activation of task i ($t0_i + kT_i + D_i$), where $t0_i$ is the first instant arrive time.

Finally, the next executing task is selected by picking the highest priority task from the highest non empty priority levels (i.e. URQ or LRQ, in this order). It executes until its termination or a pre-emption of a higher priority task.

The on-line scheduling solution that Dual Priority Scheduling algorithm presented is operative in the vast majority of kernels and computationally efficient [6,7]. This algorithm was conceived to schedule tasks with hard deadlines in a hard real-time environment containing periodic, an aperiodic tasks coexisting. The goal of the Dual Priority Scheduling algorithm is to give good response time to aperiodic tasks delaying as much as possible the periodic tasks without compromising their deadline. In this hard real time scenario there appears spare time due to tasks not consuming all its worst case execution time. The Dual Priority algorithm can use this spare time to execute aperiodic tasks sooner, giving them a good response time. Our goal is to take advantage of this performance from the energy saving point of view, the scheduling algorithm can be modified to extract the maximum time extension allowed by the real-time system, and this lengthen of time execution will be accompanied of a speed and voltage supply reduction, and finally energy reduction, as we explain in the next section.

3. POWER-LOW MODIFIED DUAL PRIORITY SCHEDULING

We have modified the Dual Priority Scheduling algorithm to help power saving in a hard real-time system. The original Dual Priority guarantees to meet the periodic temporal constraints, then our modification only needs to care about when and how to reduce energy by slowing speed and voltage jointly (we are assuming a linear relation between speed and voltage supply decreasing).

We have ordered the URQ by the static priority of the tasks and the LRQ by their absolute promotion time L_{ik}. The decision to order the LRQ this way responds to the fact that the task with the lowest L_{ik} will promote earlier and then it will execute earlier than the others. If the URQ is empty, the first task of the LRQ will begin to execute as low as possible until its promotion or a pre-emption of another task. Figure 1 shows the pseudo code for the PLMDP (Power Low Modified Dual Priority Scheduling) The algorithm work as follows:

a) If both queues URQ and LRQ are empty, then the power-down mode is activated until the arrival of the next task instance ta_{ik} (lines L1-L4 of Figure 1).
b) If the queue URQ is empty but there are tasks in LRQ then the k^{th} activation of the task i with the highest priority in the LRQ (that is ordered in terms of absolute promotion time tp_{ik}) is activated (line L6 of Figure 1). Before the execution of this tasks the algorithm needs to fix the ratio of processor speed according with the maximum spreading in time that is allowed to execute this task i. The speed ratio is calculated following the heuristics proposed by Shin and Choi [4] that is built on the assumption that the delay is negligible. The safeness of the system under these conditions is proved on theorem 1 of the cited work. In this case, we calculate the time the current active task promotes, tp_{ik}, as the ta_{ik} plus the deadline D_i, and minus R_i (worst case response time of task i), see line L7 of Figure 1. After that, we determine the time we dispose before

some other task will promote to the URQ to calculate the maximum speed reduction allowed. Here we describe the different scenarios we can find:

```
L1  if empty(URQ) then
L2    if empty(LRQ) then
L3      Set timer to (next $ta_j$ - wake up delay)
L4      Enter power-down mode
L5    else
L6      Active task = LRQ.head    -- active task=$Task_i$
L7      $tp_i = ta_i + D_i - R_i$
L8      if $ta_k < tp_i$ and $tp_k < tp_i$ then
L9        $\text{Speed} = \frac{1}{ta_k - tc}$              -- minimum speed
L10     else
L11       if $tp_i < tp_j$ and $P_i < P_j$ then     -- $j \in hp(i)$
L12         $\text{Speed} = \frac{\min(tp_j - tp_i, \text{remainig}(C_i))}{\min(tp_j, td_i) - tc}$
L13       else $tp_i < tp_m < tp_j$ and $P_m < P_i < P_j$
L14         $\text{Speed} = \frac{\min(tp_j - tp_i, \text{remaining}(C_i))}{\min(\max(tp_m, tp_i + \text{remainig}(C_i)), td_i) - tc}$
L15       endif
L16     endif
L17     Execute active task
L18   endif
L19 else
L20   Active Task = URQ.head;
L21   if URQ.head.next = NIL then
L22     $\text{Speed} = \frac{\min(tp_k - tc, \text{remainig}(C_i))}{\min(tp_k, td_i) - tc}$
L23   else
L24     Speed = 1.0                             -- maximum speed
L25   endif
L26   Execute active task
L27 endif
```

Figure 1. Pseudo code for the Power Low Modified Dual Priority Scheduling

If there exists some task j, not yet arrived, with a promotion time (tpj) shorter than the promotion time of the active task (task i), then, as soon as this task j arrives, it will pre-empt the active task. Before the arrival time of this new task j we have an interval time to execute the active task with a speed reduction (see L8-L9 of figure 1). In this case the speed of the processor should be the minimum possible speed.

On the other hand, if the next promotion time will be the promotion time of the active task, then there is no reason to restrict the speed reduction until the following promotion time (tp_k) of any task k.

If the priority of task k in the URQ is higher than the priority of active task then we can execute the active task i, until this time tp_k, reducing speed (line L11 of Figure 1). To assign the corresponding speed, in this case, we calculate the amount of work that the task should execute (Γ_k) to do not compromise the temporal constraints. Γ_k will be the minimum time of the difference between the promotion time of task k and the promotion time of task i and the remaining execution time of C_i. We also calculate the time we have to execute this work Γ_k, i.e. the difference between the minimum time of the promotion time of task k and the deadline of task i and the current time. (line L12 of Figure 1).

If the priority of task k is lower than the priority of active task (L13) we should look for the promotion time of a higher priority task of active task, because task k will never pre-empt task i. (see Figure 2 for a graphic explanation). Γ_k is calculated as explain before, and to calculate the time the task dispose to do the work Γ_k we have to calculate the difference between the minimum time between the maximum between tpk and the tp_i plus the remaining Ci, and the deadline of task i, and the current time tc (L14). (see Figure 2 for a graphic explanation of the different possibilities).

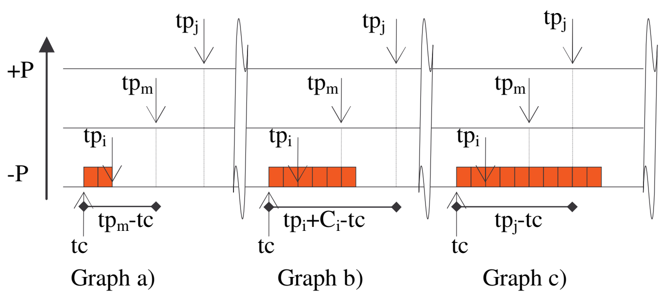

Figure 2. Maximum extension time in three different situations

c) If the URQ has only one task to execute, then this is the active task (line L20 of Figure 1) and the processor speed is calculated as the quotient between the minimum time of the next promotion time and the remaining

C_i and the total time available to execute this tasks, that now is the minimum between the next promotion time and the current task deadline (see L21-L22 of Figure 1)

d) If there are more than one task in the URQ, the first task is executed at the maximum speed allowed by the processor (see L24 of Figure 1).

At practice it is obvious that only certain discrete values of the frequency of the clock, and then speed, are available, in this case the selection is always a frequency equal or larger than the calculated one to ensure time constraints.

In this algorithm, the speed is calculated in the basis that all tasks consume its WCET, but at practice, the tasks execute only part of this WCET nevertheless it is impossible for the scheduler to know a priory the fraction of WCET they will used. This implies that, the speed calculated is the minimal that guarantee the theoretical time constraints. The difference between the theoretical time constraints and the time consumed at practice could be normally used for the next executing task, to reduce its speed.

Now we present an example to better appreciate the functioning of our algorithm (PLMDP) in comparison with the LPFPS. The benchmark used is the same presented by Shin and Choi [4], Table 1. In the Figures 3 and 4 we represent the execution of both algorithms (LPFPS and PLMDP), when all tasks consume the 100% of their WCET, and in the Figures 5 and 6 we represent the execution of the algorithms, when all tasks consume the 50 % of its WCET. In all these Figures, the vertical up arrows represent the arrival of the task to the system, the vertical down arrows represent the promotion time of the task, and finally the horizontal arrows stands for the time the task could lengthen its execution time. Each box represent five time units (although our minimal calculation unit corresponds to 1 time unit), and each line corresponds to task T1, T2 and T3 respectively. The shaded circles represent idle time in the system, observe that in Figures 3 and 4 there is no idle time, that is, the tasks use all possible time.

Task	T	D	WCET	R	D-R	P
T1	50	50	10	10	40	1
T2	80	80	20	30	50	2
T3	100	100	40	80	20	3

Table 1. Benchmark task set used by Shin and Choi[4]

In Figure 3 we have represented the behaviour of the LPFPS. The LPFPS algorithm is driven by the Fixed Pre-emptive Priority Scheduling. This algorithm consist on executing tasks as low as possible while satisfying time constraints. LPFS reduce clock speed along with voltage supply only when there is a unique task ready to be executed, otherwise the scheduling does not guarantee the time constraints of the rest of the system tasks. It also powers down the processor when there are not ready tasks (see Shin and Choi [4] for an extended explanation).

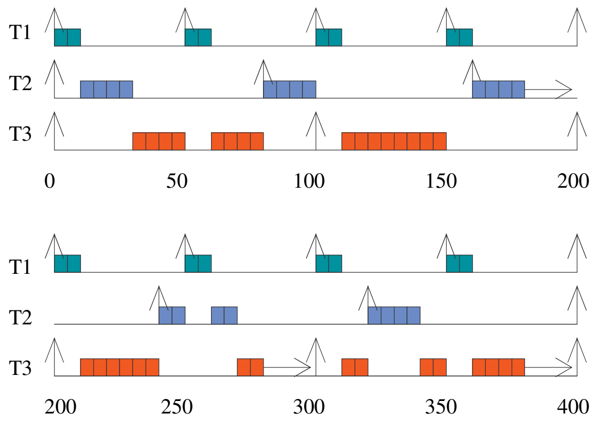

Figure 3. Execution time in LPFPS when all tasks use 100% WCET.

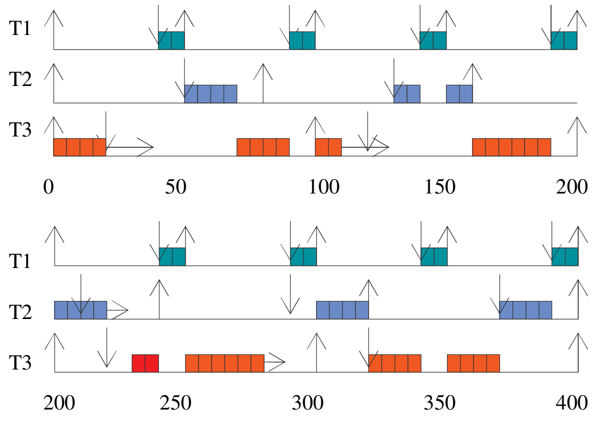

Figure 4. Execution time in PLMDP when all tasks use 100% WCET.

Now, let focus our attention in Figure 4, that represents the behaviour of our algorithm, it is as follows: At t=0, all three tasks arrive to the system and then they are placed at the LRQ sorted by its promotion time (tp_{T3}=20, tp_{T1}=40, tp_{T2}=50), the first task to promote according with our scheme will be T3, then it is activated. Its promotion time arrives at t=20, and we can execute this task until t=40 (promotion time instant of T1) without any problem. Executing T3 as late as possible implies that the execution time of T3 should start at its promotion time (t=20) and it would be pre-empted at t=40, that means that it has 40 time units to execute 20 time units, we can then reduce the speed and the power supply. At t=40, T1 is promoted and pre-empt T3 because T1 has a higher priority. T1 is now the active task, and has to execute at maximum speed because T3 is in the URQ. T1 executes 10 units time and finishes by its deadline, at time t=50. At that moment, T2 promotes and as it is the higher task in the URQ, it executes at the maximum speed until it finishes, at T=90. At this time T3 is the unique task in the URQ so it can be executed until the next promotion time, T=90. T3 executes 40 units time it can execute at low speed, but as it remain 40 units at maximum to finish its WCET, it has to execute at maximum speed. The algorithm continues with this behaviour until time 200. At time t=200 we have, again, all task in the LRQ, but now the first promotion time corresponds to T2 (tp_{T2}=210). T2 is the active task. In order to know how much time this task could execute its remaining time, we should look for the maximum value between tp_{T2}=210 plus the remaining time (20 units time) and tp_{T3}=220 so T2 continues until t=230, and T2 finishes. After that, T3 is the active task, it is alone in the URQ but its remaining time is 40 units and at t=250 there will be a promotion time of T1, so T3 has to execute at maximum speed during 20 units time. The algorithm continues with this behaviour during all its hyper-period. After that the tasks will repeat the same pattern.

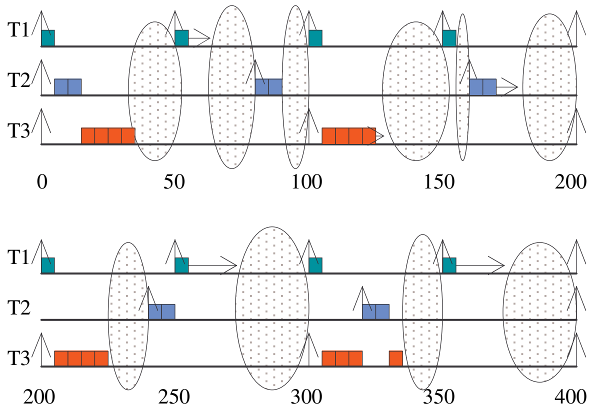

Figure 5. Execution time in LPFPS when all tasks use 50% WCET.

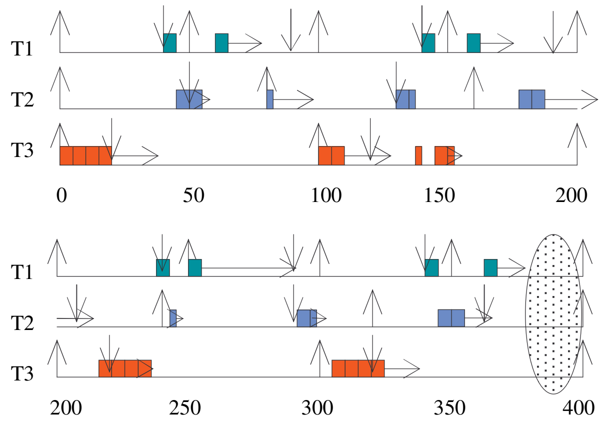

Figure 6. Execution time in PLMDP when all tasks use 50% WCET.

In this way the algorithm uses the exceeding time to work at slow processor speed and low voltage. In the Figures 5 and 6 we represent the execution of both algorithms, in a different situation, when all tasks consume the 50% of its WCET.

In Figure 5 we have represented the behaviour of the LPFPS (Shin and Choi [4]), and in Figure 6 we represent the behaviour of our algorithm. Although the main behaviour of the algorithm is identical to the behaviour described before, this new situation provokes more idle time of the processor that should be used in energy saving. At time 0, all three tasks arrive to the system, but now, task T3 finishes at time 40 because it only executes the 50 % of its WCET. Task T1 is now the active task in the URQ, it executes 5 units time at maximum speed because its WCET is 10 and its deadline is 50, but this task finishes at time 45 because it now executes only 5 units. After that, there is only task T2 in the LRQ that promotes at time 50 to the URQ. At time 50 it will arrive task T1, it enters the LRQ and promotes to the URQ at time 90. Then we can reduce the clock speed expecting to finish at its deadline at time 80, the minimum between the deadline of the active task (t=80) and the promotion time of a higher priority task (t=90). As task T2 executes only a half of its WCET, it finishes at time 63 and the processor continues with task T1 that now can reduce speed again, expecting to finish by its deadline that is in this case the minimum between the promotion time of task T2 and the deadline task T1. After that, task T1 executes at low speed and finishes by time 80. At time 80 it arrives task T2 to the LRQ and it stands alone until time 100 when task T1 and task T3 arrive. The promotion time of task T3 occurs at time 120 while the promotion time of task T2 occurs at time 130, so that in the LRQ task T3 will have the highest priority because the promotion time is earlier. For that reason task T2 should execute at the lowest possible speed only until task T3 arrives and pre-empt task T2.

To summarize the comparison, in Figures 3 and 4, when tasks consume all its WCET, nor LPFPS nor PLMDP have idle interval times. In this particular case there is not big differences between the performance of both algorithms. The only difference is that the energy saving occurs at different times but globally the total amount is the same. On the other hand when tasks consumes its 50 % of WCET, Figures 5 and 6, PLMDP has only 20 units of idle time, while LPFPS has 167 free units time, this effect translates in our algorithm in an energy saving of around 300 % with respect to LPFPS.

In general, real time systems behave in a mixed situation with a few tasks consuming 100% of its WCET and the rest consuming fractions of its WCET, then our algorithm shows to improve the energy saving obtained with a fixed priority scheduling algorithm.

4. EXPERIMENTAL RESULTS

To check the capabilities of the PLMDP approach, we have simulated several task sets (synthetic and real) and compared the total energy results per hyper-period obtained in front of the Low Power Fixed Priority Scheduling (LPFPS) proposed by Shin and Choi [4]. For completeness, we have plot the performance of both schemes in the example task set explained before and represented by Table 1, the results of this comparison are exposed in Figure 7. In the experiment we vary the percentage of consumption of the worst case execution time (WCET) of tasks to better analyse the performance in different situations.

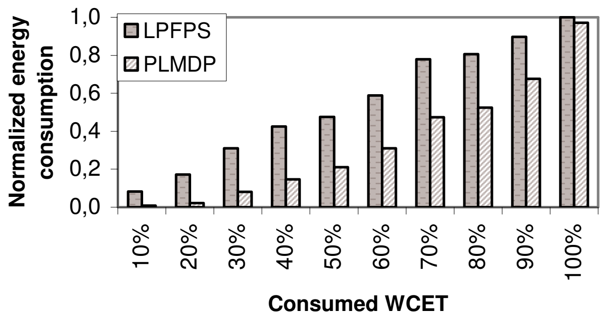

Figure 7. Comparison of both algorithms in the task set proposed by Shin and Choi [4]

In this example the average improvement calculated as the ratio between the energy consumption of LPFPS and the energy consumption of PLMDP, is 1,62 i.e. we save the 38,27% of the average energy consumed by LPFPS. Note that, even when the 100% of the WCET is consumed (see Figures 3 and 4), the total consumption energy is improved by our algorithm. This difference in energy is due to the use of the idle time to reduce the processor speed in different instants during the hyper-period.

To test our algorithm, we have also performed several experiments using 100 different synthetic task sets for each experiment. All tasks sets are formed by 10 schedulable periodic tasks, and for each task, we vary from 10% to 100% of the WCET consumption. In all the experiments, we check how harmonicity could affect the results, using harmonic task set and non-harmonic task set, and we also check how workload could change the results varying both the workload of the system, and the tasks workload. To summarize we have made three groups of experiments:

a) Varying the load of the system between 50% to 90%. The maximum task workload was fixed to 20%. The periods range from 100 to 1000 time units for the non-harmonic task sets and from 1024 to 131072 for the harmonic task sets (Figure 8-11)
b) Varying the ratio between the maximum task period (Tmax) and the minimum task period (Tmin) from 0,1 to 0,00001. The periods range from Tmin to Tmax. The workload of the system is fixed to 80 % and the maximum task workload was fixed to 20%. (Figure 12)
c) Varying the maximum of task workload between 10% and 40%. The workload of the system was fixed to 80% and the periods are range from 100 to 1000 units of time for the non-harmonic task sets and from 1024 to 131072 for the harmonic task sets. (Figure 13)

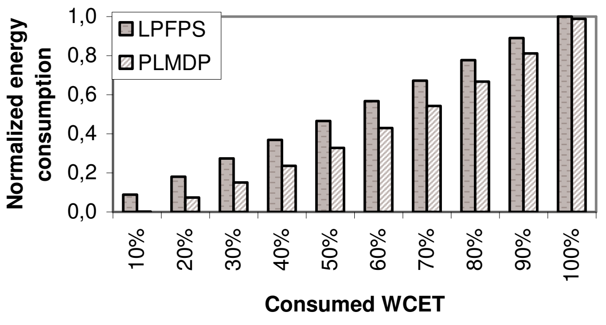

Figure 8. Comparison of both algorithms when the workload of the system is 80%.

In Figure 8 we can see the influence of the usage of different percentage of WCET in the efficiency of energy consumption. When all the tasks consume all its WCET, the improvement of our algorithm is not representative, but as tasks consume lower percentages of WCET our algorithm improvement is very important. The normalized mean deviation of the energy consumption for the LPFPS is 0,014, being the maximum normalized deviation 0,018 and the minimum normalized deviation 0,004. This implies that the accuracy of our results is within the 2,57% of error. And for the PLMDP, the normalized mean deviation of the energy consumption is 0,026, being the maximum normalized deviation 0,040 and the minimum normalized deviation is 0,002. In that case the accuracy of the results is within the 6,13%. The average improvement of our algorithm in this case is 1,25 times the energy efficiency obtained by LPFPS .

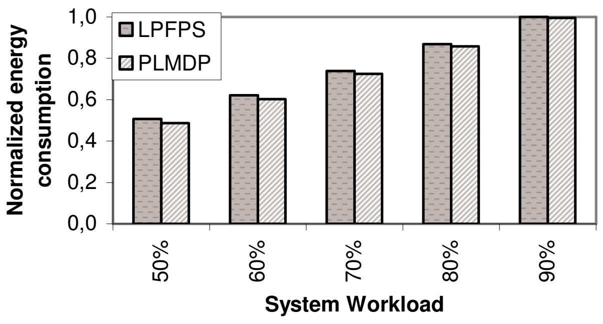

Figure 9. System workload variation when all tasks consume the 100% of WCET.

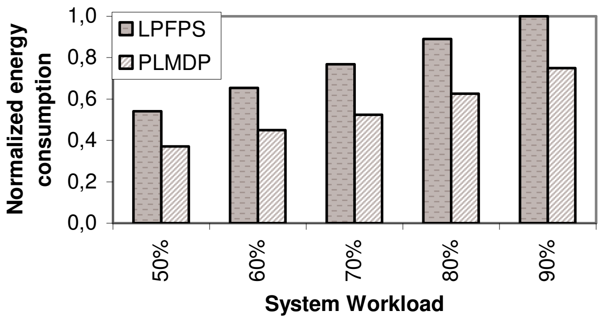

Figure 10. System workload variation when all tasks consume the 50% of WCET

In Figures 9 and 10, we can see the influence of the system workload variation when all tasks consume its 100 % of WCET and its 50 % of WCET respectively. In the former, the average improvement of our algorithm is 1.02 times the energy efficiency obtained by LPFPS, while the improvement increases as the percentage of WCET consumption decreases, being 1.42 in the case of 50% of WCET consumption. In the extreme case of 10% of WCET consumption the improvement achieves the ratio value of 34.98. The reason of this increment is because our algorithm can adapt its behaviour to the real load of the system, executing almost always at reduced speed. The conclusion is that the system workload affects to both algorithms being the interval of differences in the energy consumption [0,01 - 0,02] when tasks consume 100%, [0,17 - 0,26] when tasks consumption is 50% of its WCET. These differences in energy consumption between both algorithms are practically constant when varying the workload of the system.

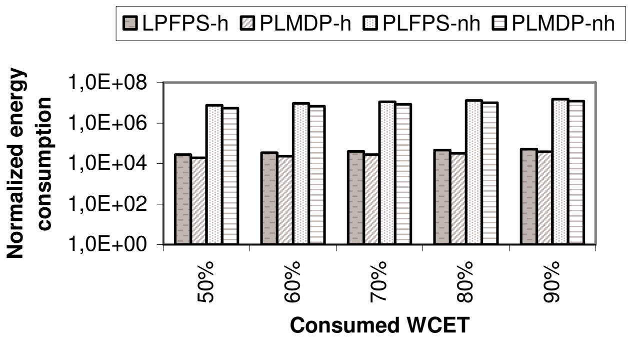

Figure 11. System workload and harmonicity of the tasks periods variation

In Figure 11, note that the normalized energy consumption is represented in logarithmic scale, we can compare the improvement of energy consumption of our algorithm (PLMDP) in front of the LPFPS, when all task consume its 50% of the WCET, in two different situations: when the task periods are harmonic (LPFPS-h, PLMDP-h) and when the task periods are non-harmonic. In the former, the mean improvement achieved by our algorithm is 1,42 and in the latter is 1,29. The accuracy of our results are within a 6,15% in the case of LPFPS with harmonic task, 18,5% in the case of PLMDP with harmonic task and for the non-harmonic tasks is 5,17% and 9,85% in the case of LPFPS and PLMDP respectively. In general we see that the differences in the accuracy of the results due to the statistics is lower for LPFPS than for PLMD because while the LPFPS reduces speed when there is a unique task ready to be executed, the behaviour of PLMD is more complex and dependent of the particular task set.

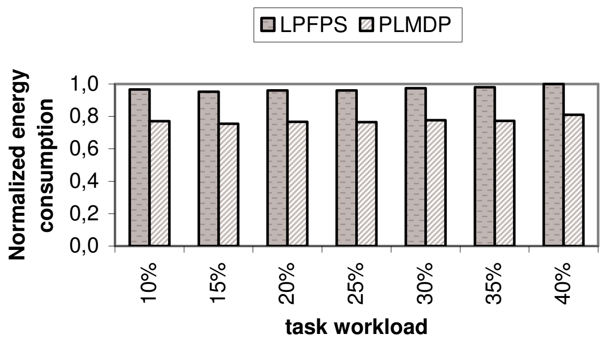

Figure 12. Maximum task workload variation. Non-harmonics periods

We checked the performance of the system when the maximum task workload varies (Figure 12). In this situation, we can not see significant differences in energy consumption independently if the task set is harmonic or not. Finally, the performance is evaluated when the ratio of periods enlarges. The results represented in figure 13 show that the variations in energy consumption.

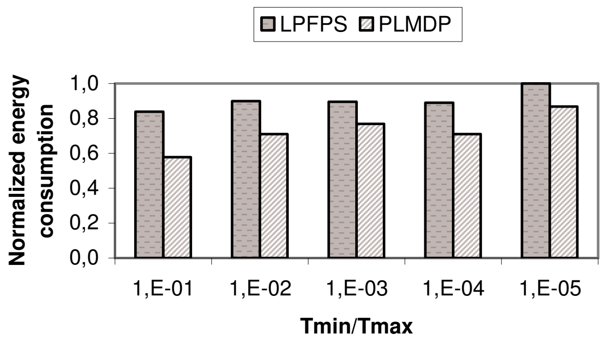

Figure 13. Tmin/Tmax variation

To conclude the present analysis, we have also collected some real time applications: the Avionics task set [9], an Inertial Navigation System (INS) [10], and a Computerized Numerical Control Machine (CNC) [11].

The two first sets represent critical mission applications and the last one is an automatic control for specific machinery. The characteristics of the tasks sets are the followings (Table 2 to 4):

Task	T	D	WCET
T1	100	100	5,1
T2	20000	20000	300
T3	2500	2500	200
T4	2500	2500	500
T5	4000	4000	100
T6	5000	5000	300
T7	5000	5000	500
T8	5900	5900	800
T9	8000	8000	900
T10	8000	8000	200
T11	10000	10000	500
T12	20000	20000	300
T13	20000	200000	100
T14	20000	20000	100
T15	20000	20000	300
T16	100000	100000	100
T17	100000	100000	100

Table 2. Avionics benchmark task set[9]

Task	T	D	WCET
T1	250	250	118
T2	4000	4000	428
T3	62500	62500	1028
T4	100000	100000	2028
T5	100000	100000	10028
T6	125000	125000	2500

Table 3. INS benchmark task set[10]

Task	T	D	WCET
T1	2400	2400	35
T2	2400	2400	40
T3	4800	4800	180
T4	4800	4800	720
T5	2400	2400	165
T6	2400	2400	165
T7	9600	9600	570
T8	7800	7800	570

Table 4. CNC benchmark task set [11]

The results of energy consumption for each application are pictured in Figures 14 to 16. The average factor of improvement of our algorithm in front of LPFPS is 1,18 times for the avionics data set, 1,21 times for the INS task set and 2.09 times for the CNC data set.

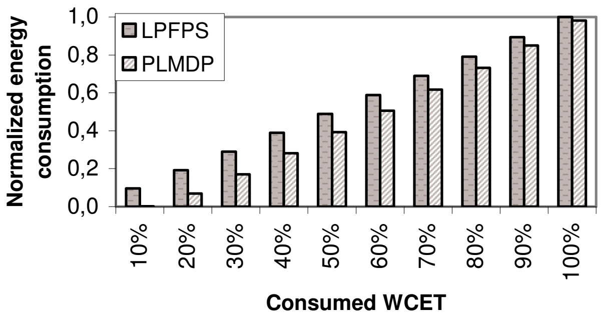

Figure 14. Comparison of both algorithms in the avionics task set[9]

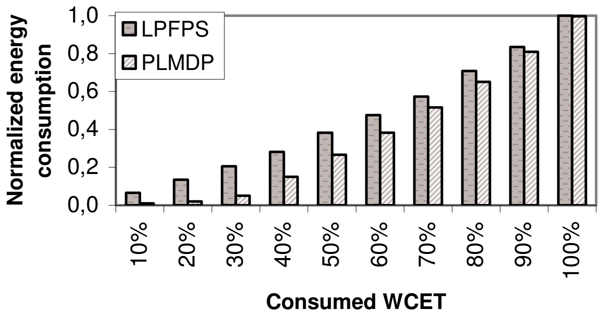

Figure 15. Comparison of both algorithms in the INS task set[10]

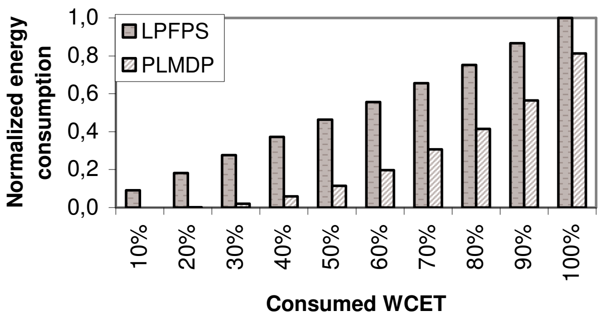

Figure 16. Comparison of both algorithms in the CNC task set [11]

If we pay attention to the specific behaviour of the individual benchmarks we observe that the relationship between periods and WCET's are responsible of the main differences between both approaches, i.e., for example in the avionics and INS task sets, there is a sub-set of tasks that have a very large period compared with the respective WCET, this fact implies that for a long time there is a unique task in the system, and then our algorithm behaves very similar to LPFPS.

On the other hand, we observe also that both algorithms behave similar when the WCET is exhausted, in three of the four case studies, that is so because in general, in this case, there is not any extra time to consume, and then no more energy could be saved using only a scheduling strategy. However, in the CNC task set there appears a particular configuration of tasks that have very large ratio between the periods and WCET, but still it is possible to take advantage of many short times with significant reduction of speed even when the tasks are using the whole WCET, while LPFPS, in this same situation, has usually a few large time intervals where the speed can be reduced (see Figure 16). In the opposite situation, i.e. when the tasks consume less than a 10% of the WCET, it is difficult to perceive the differences. Finally, when the utilization of the WCET is around its half the differences between both performances are more relevant.

All the experiments represent the results of the normalized average energy obtained, varying the consumed worst execution time from 10% to 100%. We run the simulation over one hyper-period (that is, the minimum common multiple of the task's period).

5. SUMMARY

We have presented a modification of the Dual Priority Scheduling to improve the Fixed Priority Scheduling power aware while maintaining the low complexity of the algorithmic. This approach has been shown to over-perform the mentioned LPFPS power saving by an average factor than range from 1,17 up to 2,09 depending on the real time application. The algorithm does not increase the complexity of the LPFPS and can be implemented in most of the kernels.

6. REFERENCES

[1] A.P. Chandrakasan, S. Sheng and R. W. Brodersen, “Low-power CMOS digital design”, *IEEE Journal of Solid-State circuits*, vol. 27, pp. 473-484, April 1992.
[2] D. Mosse, H. Aydin, B. Childers and R. Melhem, “Compiler-assisted power-aware scheduling for real-time applications” Workshop on Compilers and Operating systems for Low Power COLP 2000, Philadelphia, Pennsylvania, October 2000.
[3] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, “Determining optimal processor speeds for periodic real-time tasks with different power characteristics” 13th Euromicro Conference on Real-Time Systems, Delft, Netherlands, June 2001.
[4] Y. Shin and K. Choi, “Power conscious Fixed Priority scheduling in hard real-time systems” DAC 99, New Orleans, Louisiana, ACM 1-58113-7/99/06, 1999.
[5] C. L. Liu and J.W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment”, JAMC 20, pp. 46-61, 1973.
[6] R. Davis and A. Wellings, "Dual Priority scheduling", Proceeding IEEE Real Time Sistems Symposium, pp. 100-109, 1995.
[7] A. Burns and A.J. Wellings, “Dual Priority Assignment: A practical method for increasing processor utilization”, Proceedings of 5th Euromicro Workshop on Real-Time Systems, IEEE Computer soc. Press, pp. 48-55, 1993.
[8] M. Joseph and P. Pandya, "Finding response times in a real-time system", British Computer Society Computer Journal, 29(5): 390-395, Cambridge University Press, 1986.
[9] C. Locke, D. Vogel and T. Mesler, “Building a predictable avionics platform in Ada: a case study", Proceedings IEEE Real-Time Systems symposium, December 1991.
[10] A. Burns, K. Tindell and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers", IEEE Transactions on Software Engineering, 21, pp. 475-480, May 1995.
[11] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi and H. Shin, "Visual assessment of a real-time system design: a case study on a CNC controller", Proceedings IEEE Real-Time Systems symposium, December 1996.