An off-line multiprocessor real-time scheduling algorithm to reduce static energy consumption

Vincent Legout, Mathieu Jan
CEA, LIST
Embedded Real Time Systems Laboratory
F-91191 Gif-sur-Yvette, France
{vincent.legout,mathieu.jan}@cea.fr

Laurent Pautet
Institut Telecom
TELECOM ParisTech
LTCI - UMR 5141 Paris, France
laurent.pautet@telecom-paristech.fr

***Abstract*—Energy consumption of highly reliable real-time embedded systems is a significant concern. Static energy consumption tends to become more important than dynamic energy consumption. This paper aims to propose a new off-line scheduling algorithm to put as much as possible processors in low-power states instead of idling. In these states, energy consumption is reduced, enhancing the battery life-time of mission critical systems. However, no instruction can be executed and a transition delay is required to come back to the active state. Activating deeper low-power states requires to produce larger idle periods. As the processor usage is constant for a given task set, this objective implies reducing the number of idle periods. Our proposal is to modelize the processors idle time as an additional task. Then we formalize the problem as a linear equation system with the objective of reducing the number of preemptions (or executions) of this additional task. Simulations show that our algorithm is more energy efficient than existing algorithms.**

I. Introduction

Embedded systems tend to have a limited power supply usually provided by batteries. Therefore minimizing the power consumption is an important concern to increase the autonomy of the embedded electronics. For example, power consumption is a major concern in the design of Unmanned Aerial Vehicles, which must also perform real-time processing. Fulfilling both constraints in a reliable way is still challenging. In such real-time systems, many devices can consume power. This work concentrates on the consumption of processors.

The energy consumption can be divided into two categories: dynamic consumption and static consumption, the former being caused by the activity of the processor while the later (mainly due to leakage currents) cannot change when the processor is active, no matter the activity. Note that thermal considerations on energy consumption are out of the scope of this work. Solutions to save energy exist at both hardware and software level. DPM (*Dynamic Power Management*) and DVFS (*Dynamic Voltage and Frequency Scaling*) are the two main software solutions. DPM aims to reduce the static consumption by putting system components in low-power states where energy consumption is reduced but no instruction can be executed. A transition delay is required to get the system back to the active state. DVFS deals with dynamic consumption and tries to execute tasks at lower frequencies (and therefore lower supply voltage).

Dynamic consumption used to be more important than static consumption. Therefore, most of the research works were dedicated to design approaches based on DVFS. Existing algorithms rarely try to reduce the static consumption. They mainly use DPM only when DVFS is no longer efficient (e.g. [10], [6]). However, static consumption becomes more important than dynamic consumption, one reason being the higher density of chips or the smaller supply voltage [11]. We compared the DPM solution introduced in this paper and a DVFS solution using a specific processor. The results showed that DPM can already be more efficient than DVFS, especially when the number of available frequencies is limited. The efficient use of DPM is therefore becoming an important issue, in particular in real-time embedded systems. However, only few publications address the management of static consumption on multiprocessor systems. And the existing solutions are not efficient enough, they produce shorter idle periods than possible.

This paper explores the problem of maximizing the use of low-power states on symmetric multiprocessor embedded real-time systems in order to save energy when scheduling tasks. On hard real-time systems, deadlines must be respected, thus low-power states should be used with care in order for processors to be ready when the system needs them. Transition delays required to come back from a low-power state to the running state must be taken into account.

The contribution of this paper is to propose an off-line scheduling algorithm that reduces static consumption. While guaranteeing the schedulability of the task set, the objective of the algorithm is to increase the duration of the idle periods by merging them when possible. The processor usage being constant for a given task set, the time processors are expected to be idle remains identical whatever the scheduler used. Therefore the objective can also be expressed as the minimization of the number of idle periods. Contrary to current algorithms, it does not use priority assignment to schedule tasks. It formalizes the problem using linear programming such that the constraint (satisfying deadlines) and the objective (generating larger idle periods) of the scheduling algorithm can be met.

The remainder of this paper is as follows. Section II describes the main solutions found in the literature, then section III defines the processor and task models used in

this paper while section IV details our algorithm. Sections V presents the experimental results and section VI concludes.

II. Related Work

One of the first approaches to use DPM on hard real-time systems was proposed by Lee et al. [12] for tasks with dynamic or static priority on a uniprocessor system. Their goal was to extend the duration of the idle periods of the processor. When the processor is idle, it delays task executions even when tasks are ready and start executing tasks just in time to prevent a deadline miss. Jejurikar et al. [10] and Chen and Kuo [8] improved the solution by increasing the duration of the procrastination period (i.e. the time spent delaying a task execution). Zhu et al. [20] proposed another improvement when tasks do not use all their worst case execution time.

However, these last three solutions all use DPM on top on DVFS. Indeed, their main common objective is to decrease the dynamic consumption which is modelized as a function of the running frequency such as $P_{dyn} \approx f^{\alpha}$ (with α between 2 and 3) ([19], [2], [7]). Thus P_{dyn} being a convex and increasing function of the processor frequency, there is a critical frequency f_{crit} such that running at a frequency lower than f_{crit} is not efficient compared to DPM. Therefore many algorithms using both DPM and DVFS, like [10] or [8], only use DPM when DVFS schedules tasks at $f < f_{crit}$.

On multiprocessor systems, when energy consumption is not a concern, no optimal scheduling algorithm exists for partitioned scheduling (i.e. where tasks are bounded to a specific processor) and only few exist for global scheduling (i.e. where migration is allowed). These optimal global algorithms are mainly based on fair scheduling like PFair [3]. However, partitioned scheduling is more widespread than global scheduling because it is equivalent to uniprocessor scheduling when tasks have been assigned to processors.

Energy-aware scheduling algorithms mainly use partitioned scheduling. For periodic task systems, Chen et al. [6] first partition tasks and then use power-aware uniprocessor scheduling algorithms to decrease both static and dynamic consumptions. Seo et al. [17] and Haung et al. [9] also use partitioned scheduling. Their algorithms first partition tasks off-line, then allow task migration on-line. Indeed, tasks can finish earlier than expected and the idea is to migrate them to create larger idle periods. However, these algorithms are on-line and they do not have a global overview of the duration of all idle periods on one hyper-period.

Bhatti et al. [4] used global scheduling with DPM. The goal of their algorithm, AsDPM, is to use as few processors as possible, letting the others asleep. When one task or more are ready to be scheduled, AsDPM always keeps one processor busy and activates the other ones only when required. However, the algorithm is not optimal and can activate more processors than needed because it cannot anticipate all future job executions.

All these algorithms use DVFS or DPM and base their scheduling decisions on task priority assignment. It activates the higher priority task as soon as it becomes ready. This approach can be suitable for DVFS but it prevents the creation of large idle periods and thus an effective use of DPM. Among the other approaches that does not use priority assignment, one was proposed by Lemerre et al. [13]. They use linear programming to compute off-line a valid schedule. The advantage of this approach is to be able to add scheduling objectives in the linear equation system. For example Megel et al. [14] used it with the objective of decreasing the number of preemptions for optimal multiprocessor global scheduling.

III. Model

This section introduces the notations used in the remainder of this paper.

A. Processors

The system has m identical processors (with $m > 1$). Global scheduling is adopted, i.e. tasks and jobs can migrate from one processor to another. We make the optimistic assumption that task preemptions and migrations have no effect on energy consumption. This assumption shall be relaxed in future works.

To use DPM, we assume processors have at least one low-power state. In a low-power state, a processor cannot execute any instruction and its energy consumption is reduced. When the system decides to wake up a processor, the processor takes a certain amount of time to come back to the nominal state and it cannot execute any instruction while waking up. We assume that the consumption used while waking up is equal to the consumption in the nominal state.

When the system has to choose a low-power state, it must be aware of the duration of the idle interval which must be greater than the transition delay required to wake up. In the literature the minimal length of an idle period to use a low-power is called the *Break-Even time* (BET).

B. Tasks

We consider a set Γ of n independent, preemptible and periodic tasks. Each task τ releases jobs periodically every period T and has a worst case execution time C. Tasks have implicit deadlines, i.e. deadlines are equal to periods. The task set hyper-period is named H and is the least common multiple of all periods of tasks in Γ. Task utilization u is the ratio $\frac{C}{T}$ and the task set global utilization is the sum of all utilizations: $U = \sum_{i=0}^{n-1} u_i$. The job set J_Γ contains all jobs of Γ scheduled during the hyper-period H.

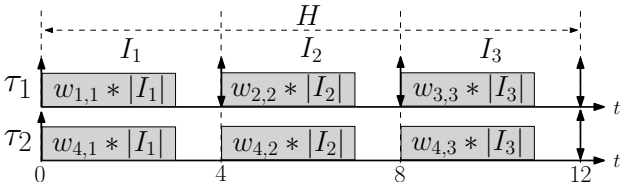
The situation where global utilization is equal to the number of processors is trivial, all processors are always active. More generally, a task set with $U \in \mathbb{N}$ is schedulable such as U processors are always active while the others are always sleeping, therefore generating no transition delay. Thus we assume global utilization U is such as:

$$m - 1 < U < m \tag{1}$$

And if $m - x - 1 < U < m - x$, set $m = m - x$ and let the x last processors always asleep ($x \in \mathbb{N}^*, x < m$).

We adopt the representation from [13] and [14] where the hyper-period is divided in intervals, an interval being delimited

Fig. 1. An example of 2 tasks with periods of 4 and 12 respectively



by two task releases. I is the set of intervals and $|I_k|$ is the duration of the k^{th} interval. A job can be present on several intervals, and we note $w_{j,k}$ the weight of job j on interval k. The weight of a job on an interval is defined as the fraction of processor required to execute job j on interval k. J_k is the subset of J_Γ that contains all active jobs in interval k. E_j is the set of intervals on which job j can run. It must contain at least one interval. The example task set in figure 1 has two tasks and four jobs (jobs 1 to 3 from τ_1 and job 4 from τ_2). In this example, E_4 is $\{1, 2, 3\}$ and J_1 is $\{1, 4\}$.

Verifying the schedulability of the task set can be expressed as a linear problem from which the weights of all jobs on every interval can be obtained using a linear solver. For the task set to be schedulable, the following equations must hold [13]:

$$\forall k, \sum_{j \in J_k} w_{j,k} \leq m \tag{2}$$

$$\forall k, \forall j, 0 \leq w_{j,k} \leq 1 \tag{3}$$

$$\forall j, \sum_{k \in E_j} w_{j,k} \times |I_k| = j.c \tag{4}$$

Where $j.c$ is the worst case execution time of job j. The first inequality means that the utilization on an interval cannot exceed the number of processors. The second inequality forbids the duration of a job on an interval to be negative or to exceed the length of the interval. And the third equation guarantees that all jobs are completely executed.

Solving the linear program given by equations (2), (3) and (4) gives the weight of all jobs on every interval thus a complete and valid schedule. This schedule however does not optimize the energy consumption. Our goal is to add constraints and objectives to generate large idle periods to put the processor in low-power states. This is the contribution of this work in the next section.

IV. Algorithm

This section introduces the detailed algorithm used to obtain the weights of jobs in a hyper-period such that it generates large idle periods. The name of the algorithm is LPDPM, as in Linear Programming DPM.

A. Motivation

We assume that processors can switch to a low-power state on every idle period (i.e. $BET = 0$) and that tasks always use their worst case execution time. Thus, regardless of the scheduler used, the processors utilization is constant over a hyper-period. Thus to evaluate the DPM power efficiency of a scheduling algorithm, the number of idle periods generated can be used, the less the better as: 1) the number of wake up transitions from a low-power state (and therefore of the associated penalties) is reduced and 2) the duration of idle periods increases allowing the use of deeper low-power states. Depending on the length of the idle period, the most efficient low-power state fulfilling the transition delay can be selected.

Fig. 2. Global-EDF schedule

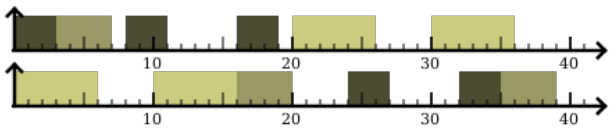


Figure 2 pictures the schedule on a two processors system of a task set composed of three tasks (with WCET and period of (3,8), (6, 10) and (4,16)) with Global-EDF. This example demonstrates that classical schedulers like Global-EDF are not suitable for DPM. They generate more idle periods than necessary.

B. Approach

During a hyper-period, one or more idle periods are generated and one or more processors switch to a low-power state. The objective is to have as few idle periods as possible. To address this problem, we choose to modelize the idle time with an additional task τ'. τ' is a periodic task with a period equals to H and a utilization equals to $m - U$. τ' thus only has one job in a hyper-period. Note that this operation is feasible only because $m - 1 < U < m$ as the utilization of τ' must be less than 1. The new task set has a global utilization of m and is therefore schedulable.

Getting as few idle periods as possible is now equivalent to decreasing the number of preemptions of τ' inside the hyper-period. This objective should now be added to the linear equation system.

However, it should be noted that τ' does not represent the actual idle time when tasks are executed. Indeed, tasks usually do not use all their worst case execution time. Thus, at run-time, processors can be idle while not executing τ' and multiple processors can be idle simultaneously. Modelizing the expected idle time with τ' is just a way to help generating a schedule with guaranteed idle periods in the worst case scenario.

C. Objectives

As stated in the last section, the hyper-period is divided into multiple intervals. An interval in which the weight of τ' is 1 is called an idle interval. And an interval in which the weight of τ' is 0 is called a busy interval.

In order to generate as few preemptions as possible, an interval should be either an idle or a busy interval and similar intervals (busy or idle) should be consecutive. Say differently, the weight of τ' on every interval should be either 1 or 0, and similar intervals where the weight of τ' is identical should

be consecutive. Intervals where the weight of τ' is strictly between 0 and 1 are the less attractive because they include a preemption. To simplify, the weight of τ' on interval k is written w_k. The objectives can be summarized as:

- w_k should be either 1 or 0
- Intervals where $w_k = 1$ should be consecutive
- Intervals where $w_k = 0$ should be consecutive

However, it may be unavoidable to generate non-busy/idle intervals, that is where the weight of τ' would be neither 1 nor 0.

D. Formalization

Objectives should now be written as linear equations in order to use them in our linear equation system. As defined in the last paragraph, the first objective is to obtain as many intervals as possible where the weight of τ' is one, that is idle intervals. In terms of linear programming, expressing this constraint requires adding a new variable to express the non-available floor function. Let f_k be a binary variable such that:

$$f_k = \begin{cases} 1 & \text{if } w_k = 1 \\ 0 & \text{else} \end{cases} \tag{5}$$

This equation can be linearized as:

$$w_k + f_k \geq 1 \tag{6}$$

Such that minimizing f_k forces w_k to be equal to 1. The objective is therefore to minimize the sum of all f_k in order to obtain a maximum number of idle intervals, that is with w_k equal to 1.

The second objective is to obtain periods where w_k is zero, that is busy intervals. Like f_k, let e_k be a binary variable such that:

$$w_k - e_k \leq 0 \tag{7}$$

And having for objective to minimize the sum of all e_k will increase the number of busy intervals.

The next goal is to make idle or busy intervals consecutive. Let fc_k and ec_k be two binary variables such that:

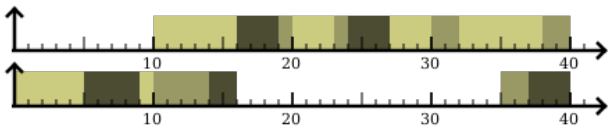
$$fc_k = \begin{cases} 1 & \text{if } f_k = 1 \text{ and } f_{k+1} = 0 \\ 0 & \text{else} \end{cases} \tag{8}$$

$$ec_k = \begin{cases} 1 & \text{if } e_k = 1 \text{ and } e_{k+1} = 0 \\ 0 & \text{else} \end{cases} \tag{9}$$

Those two equations can be modelized as:

$$\begin{cases} f_k - f_{k+1} - fc_k \leq 0 \\ -f_k + fc_k \leq 0 \\ f_{k+1} + fc_k \leq 1 \\ -fc_k \leq 0 \end{cases} \tag{10}$$

Fig. 3. LPDPM schedule



$$\begin{cases} e_k - e_{k+1} - ec_k \leq 0 \\ -e_k + ec_k \leq 0 \\ e_{k+1} + ec_k \leq 1 \\ -ec_k \leq 0 \end{cases} \tag{11}$$

Minimizing the sum of all fc_k and the sum of all ec_k is then going to make idle or busy intervals consecutives. Therefore, the final objective is (subject to equation (2), (3), (4), (6), (7), (10) and (11)):

$$\text{Minimize} \sum_k f_k + e_k + fc_k + ec_k \tag{12}$$

E. Scheduling inside an interval

Resolving the linear system gives a weight for each task on every interval such that the number of idle periods is minimized. It outputs a solution that satisfies the linear system which however may not be the optimal solution. To schedule tasks inside intervals, EDZL [18] or IZL [14] can be used.

Unfortunately, as explained previously, the solver can generate intervals where the weight of τ' is neither 1 nor 0 (non-idle/busy intervals). Those intervals where $0 < w_k < 1$ are the only intervals where the on-line scheduler should be careful not to generate additional idle periods if possible. In particular, an additional idle period should be merged with a previous idle period to save transition delays. To solve this problem, we choose to schedule τ' at the beginning or at the end of the interval to stick the execution of τ' in the current interval to the one in the previous or next interval.

Moreover, to share the load between processors, the processor executing τ' can be changed when τ' is preempted. It does not increase the number of idle periods and could be an interesting way, in a future work, to handle thermal impact of our solution on the energy consumption.

Figure 3 pictures the schedule of the task set from subsection IV-B with LPDPM. Where Global-EDF was generating 9 short idle periods, LPDPM creates 2 much larger idle periods.

V. Evaluation

In order to compare our solution with existing algorithms, we conducted a simulation-based experimental study using the energy information of the STM32L boards, which are based on the ARM Cortex-M3 processor [1]. It has four low-power states described in table I. Using a simulator, we generated random task sets and scheduled them on two hyper-periods with several schedulers. This simulation was conducted with 4 processors and each task set has 10 tasks. Task utilizations are computed randomly between 0.01 and 0.99 with a uniform

Fig. 4. Mean number of idle periods

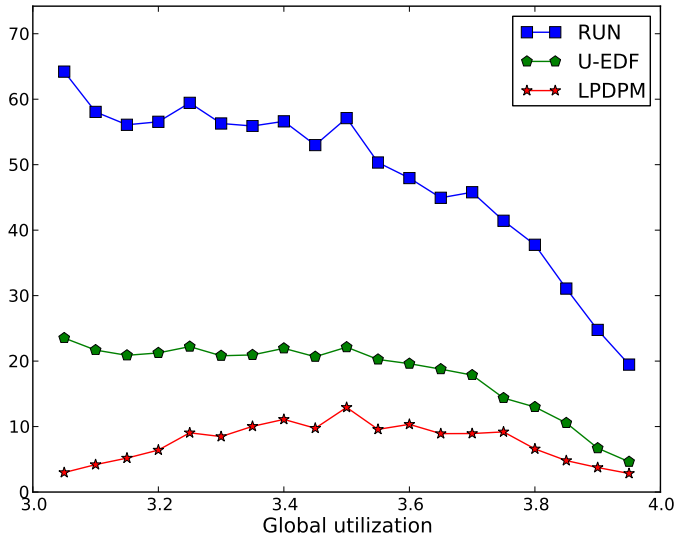
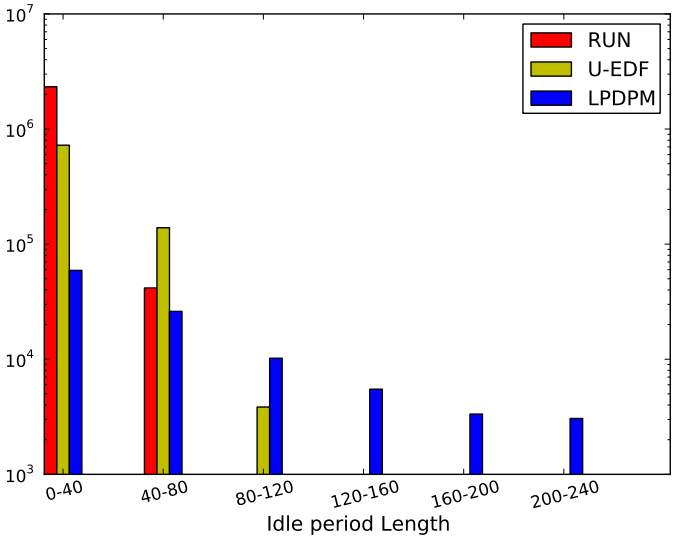


Fig. 5. Distribution of idle periods lengths



distribution using the well-known UUniFast algorithm from Bini et Buttazzo [5]. The period of each task is also chosen randomly between 10 and 100.

2000 task sets were generated for each global utilization. Then, each task set was scheduled by the following schedulers: RUN [16], U-EDF [15] and LPDPM. RUN and U-EDF are two optimal multiprocessor scheduling algorithms aiming to reduce the number of preemptions and migrations. The implementation of LPDPM uses IBM ILOG CPLEX to solve the linear problem. We limited the solving time to 60 seconds and we rejected the task set when a solution was not found.

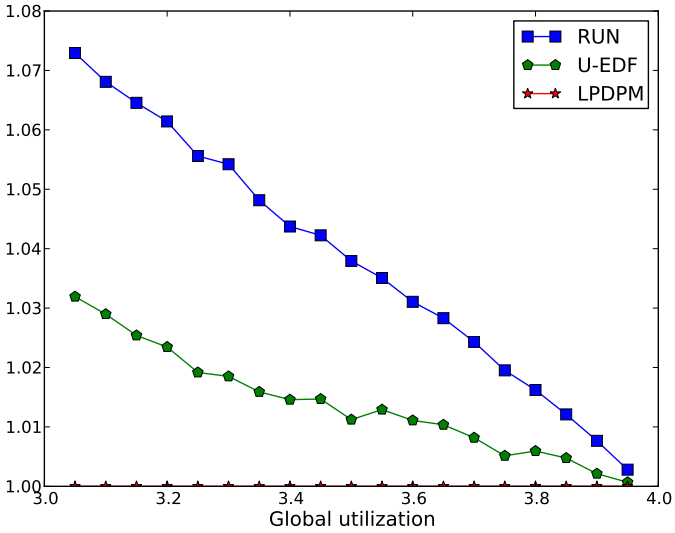
TABLE I
STM32L LOW-POWER STATES

Mode	Current consumption	Transition delay
Run	7.8 mA	
Sleep	2.3 mA	0.1
Low power run	25 μA	0.4
Stop	3.1 μA	0.8
Standby	1.55 μA	5

The mean number of idle periods is plotted on figure 4 for each global utilization. Figure 5 gives the repartition of the idle periods lengths for each scheduler. Those figures illustrate the fact that LPDPM generates less and larger idle periods. For example, all idle periods created by the other algorithms have a length less then 120 while LPDPM can generate idle periods with a length up to 240. Note that the ordinate is plotted on a logarithmic scale in figure 5. Figure 6 gives the relative consumption of other schedulers, the consumption of *LPDPM* being always one. The consumption is computed based on the values from table I. LPDPM is always more energy efficient and the difference is of course reduced when the global utilization increases.

Finally, figure 7 pictures the number of preemptions for each scheduler. Even if RUN and U-EDF are two algorithms

Fig. 6. Relative consumption (LPDPM = 1)



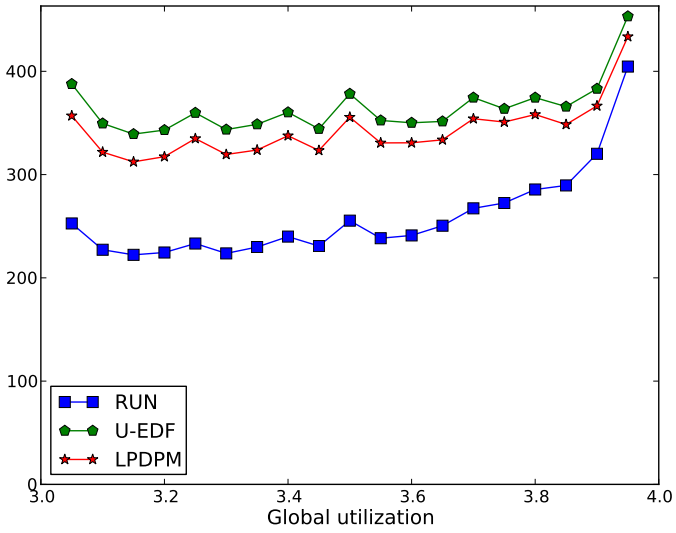
specifically designed to reduce the number of preemptions, LPDPM behaves as well as U-EDF and only less than 1.5 worse than RUN. This makes LPDPM viable. Improving the linear problem to decrease the number of preemptions of the regular tasks is left for future works.

VI. Conclusion

In this paper we introduced LPDPM, an off-line power-aware scheduling algorithm for multiprocessor real-time systems. It focuses only on static energy consumption. The algorithm tries to increase the duration of the idle periods such that deeper low-power states can be activated. It introduces an additional task accounting for the time where processors are expected to be idle. It then decreases the number of preemptions of this task.

Contrary to other DVFS or DPM algorithms, LPDPM uses an off-line approach. Another difference is that LPDPM does

Fig. 7. Mean number of preemptions



not consist in mapping task to priorities in order to schedule the system. It assigns weights to tasks on intervals where each interval is delimited by two task releases. Linear programming is used to get all the weights. With this approach, constraints and objectives can be formally expressed. The constraint is to meet deadlines. The objective is to decrease the number of preemptions of the idle task. Simulations show that LPDPM significantly decreases the number of idle periods and is therefore more energy efficient than existing algorithms.

The objective of LPDPM is to minimize the number of idle periods. And then, based on the length of the idle period, a low-power state can be selected. Thus the linear equation system does not depend on the processors used. However, the assumption that minimizing the number of idle periods is always more energy efficient can be false with some processors. Therefore, we plan to relax this assumption in future works.

At run-time, if the actual execution time of tasks is lower than their WCET, the duration of idle periods increases. It could potentially allow the use of deeper low-power states. Therefore, an on-line algorithm should be able to use the unused computation time to increase the length of existing idle periods, following the same idea as Seo et al. [17]. At present, LPDPM only works for periodic tasks with implicit deadlines. We plan to extend our approach to schedule sporadic tasks with constrained deadlines.

Energy consumption depends on the temperature. Using LPDPM, the idle processor used for each new idle period can be changed to share the load and decrease the temperature of processors. But future works should modelize thermal impacts on energy consumption such that LPDPM could generate thermal-aware schedules.

ACKNOWLEDGEMENT

The authors would like to thank Vincent David for his remarks in the early stages of this work.

REFERENCES

[1] *ST Microelectronics. STM32L151xx and STM32L152xx advanced ARM-based 32-bit MCUs. Reference Manual RM0038, 2011.*

[2] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of the 22nd IEEE Real-Time Systems Symp.*, pages 95–, 2001.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proc. of the 25th annual ACM symposium on Theory of computing*, pages 345–354, 1993.

[4] M. Bhatti, M. Farooq, C. Belleudy, and M. Auguin. Controlling energy profile of rt multiprocessor systems by anticipating workload at runtime. In *SYMPosium en Architectures nouvelles de machines*, 2009.

[5] E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *Proc. of the 16th Euromicro Conf. on Real-Time Systems*, pages 196–203, 2004.

[6] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 408–417, 2006.

[7] J.-J. Chen and C.-F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 28–38, 2007.

[8] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, pages 153–162, 2006.

[9] H. Huang, F. Xia, J. Wang, S. Lei, and G. Wu. Leakage-aware reallocation for periodic real-time tasks on multicore processors. In *Proceedings of the 2010 Fifth International Conference on Frontier of Computer Science and Technology*, pages 85–91, 2010.

[10] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, pages 275–280, 2004.

[11] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, pages 1–8, 2010.

[12] Y.-H. Lee, K. Reddy, and C. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proc. of the 15th Euromicro Conf. on Real-Time Systems*, pages 105 – 112, 2003.

[13] M. Lemerre, V. David, C. Aussaguès, and G. Vidal-Naquet. Equivalence between schedule representations: Theory and applications. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 237–247, 2008.

[14] T. Megel, R. Sirdey, and V. David. Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules. In *Proc. of the 31st IEEE Real-Time Systems Symp.*, pages 37–46, 2010.

[15] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *Proceedings of the 17th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 15 –24, 2011.

[16] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proc. of the IEEE 32nd Real-Time Systems Symp.*, pages 104–115, 2011.

[17] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1540–1552, 2008.

[18] H.-W. Wei, Y.-H. Chao, S.-S. Lin, K.-J. Lin, and W.-K. Shih. Current results on edzl scheduling for multiprocessor real-time systems. In *Proc. of the 13th IEEE Intl Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 120–130, 2007.

[19] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–, 1995.

[20] Y. Zhu and F. Mueller. Dvsleak: combining leakage reduction and voltage scaling in feedback edf scheduling. In *Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 31–40, 2007.