Platform-aware Bandwidth-oriented Energy Management Algorithm for Real-Time Embedded Systems

Mauro Marinoni, Mario Bambagini, Francesco Prosperi, Francesco Esposito,
Gianluca Franchino, Luca Santinelli, Giorgio Buttazzo
{*name.surname*}*@sssup.it*
Scuola Superiore S. Anna Pisa, Italy

***Abstract*—A crucial objective in battery operated embedded systems is to work under the minimal power consumption that provides a desired level of performance. Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) are typical techniques used on processors and devices to reduce the power consumption through speed variations and power switching, respectively. The effectiveness of DVFS and DPM methods needs to be considered in the development of a power management policy for systems that consist of DVFS-enabled or DPM-enabled components.**

This paper explores how to efficiently reduce the power consumption of real-time applications with constrained resources, like energy, CPU and transmission bandwidth. A combined DVFS-DPM approach with a reduced complexity is proposed to make use of on-line strategies for embedded systems.

I. Introduction

Embedded systems cover a wide spectrum of application domains, such as consumer electronics, biomedical systems, surveillance, industrial automation, automotive, and avionics systems. In particular, the technology evolution of sensor and networking devices paved the way for plenty of new applications involving distributed computing systems, many of them deployed in wireless environments and exploiting the mobility and the ubiquity of components. In most cases, devices are battery operated, making energy-aware algorithms of paramount importance to prolong the system lifetime.

In each node of the system, at the processor level, two main mechanisms can be exploited to save energy: the Dynamic Voltage and Frequency Scaling (DVFS) and the Dynamic Power Management (DPM).

For DVFS processors, a higher supply voltage generally leads to both a higher execution speed/frequency and to a higher power consumption. On the other hand, DPM techniques are used to switch the processor off during long idle intervals, hence they tend to postpone tasks execution as long as possible still preserving the schedulability of the task set. At the network level, the energy consumption due to communication is usually managed by DPM techniques, although other mechanisms have been proposed in the literature, as the Dynamic Modulation Scaling (DMS) [1].

The research leading to these results has received funding from the European Community's ArtistDesign Network of Excellence under grant agreement no. 214373.

In micrometer CMOS technology, the dynamic power dissipation due to switching activities prevails against the static power dissipation caused by the leakage current. However, in most modern processors developed with sub-micron technology, the static power is comparable or even greater than the dynamic power [2], [3]. When the dynamic power is dominant, DVFS techniques are used to execute an application at the minimum processor speed that guarantees meeting real-time constraints. Conversely, when static power is dominant, there exists a critical processor speed below which the energy wasted is greater than that consumed at the critical speed [4]. For this reason, some authors recently proposed energy-aware algorithms that combine DVFS and DPM techniques to improve energy saving [5], [6].

In distributed systems there is the need of taking into account processor and network bandwidth to guarantee performance requirements. In particular, in wireless distributed embedded systems, energy consumption and quality of service represent two crucial design objectives. Messages have to be transmitted within a deadline to guarantee the desired quality [7], [8], and the transmission itself represents an energy cost to be minimized. Although a lot of research has been done to reduce power consumption while guaranteeing real-time requirements, most papers focus either on task scheduling or network communication. However, a co-scheduling of task and messages would allow exploring more degrees of freedom and could lead to higher energy saving.

Finally, an effective approach has to be platform independent and easily portable to new hardware just by changing a small set of parameters, such as the energy consumption of the CPU in different working modes.

Contributions: This work addresses the challenging problem of co-scheduling both tasks and messages in a distributed system with the objective of saving energy and meeting real-time constraints on the local nodes composing the system. An algorithm that minimizes the energy consumption of the system nodes is presented. The proposed approach takes into account the communication bandwidth reserved for each node and combines DVFS and DPM techniques to achieve higher energy saving.

This work improves the approach previously proposed by Santinelli et al. [9] in several directions. First of all, tasks are assumed to be sporadic, rather than peridoc. Second, the DVFS model is more realistic, since the CPU frequency is assumed to

vary within a set of discrete values, rather than in a continuous range. Third, the algorithm is completely redesigned to consider the effects of the execution platform, while containing the overall computational complexity. Finally, the experimental section includes new test cases and simulation results.

Organization of the paper: Section II presents the system model, in terms of resources, power dissipation, computational and communication workloads. Section III summarizes the theoretical results used to analyze the task set schedulability. Section IV illustrates the proposed approach, and Section V presents the scheduling algorithm implementing the method. Section VI reports the simulation results carried out to evaluate the performance of the approach, and finally, Section VII ends the paper with the concluding remarks.

A. Related work

A lot of research papers presenting DVFS algorithms for energy-aware real-time task scheduling have been published in the past years, such as [10]–[12]. Conversely, some other papers focused on DPM techniques, see for instance [13], [14] and the related works therein. DVFS scheduling algorithms, e.g., [15]–[17], tend to execute events as slow as possible, preserving timing constraints: by trading processor speed with energy consumption. DPM algorithms are used to switch the processor off during long idle intervals [2], [13], [18].

Some papers proposed the use of DVFS techniques at the CPU level and DPM approaches at the network level, but, to the best of our knowledge, not many papers combined DVFS and DPM techniques for task scheduling under communication constraints.

In [19], the authors addressed energy saving issues for a system composed by a DVFS capable CPU and a network interface supporting the DPM. They proposed two DVFS based algorithms: one, Limited Look-Ahead EDF (LLE), favors energy saving at the CPU level, whereas the other, Timeout Aware Scheduler (TAS), favors energy saving at the network level. LLE tries to minimize the average power wasted by all tasks using a modified version of the LaEDF algorithm [20]. Instead, TAS tries to maximize the sleep time of the network card by gathering the packet transmissions into bursts, exploiting LaEDF.

Poellabauer et al. [21] proposed an integrated resource management algorithm that considers both CPU and a bandwidth reservation protocol for the network interface, in wireless real-time systems. The aim of the proposed method is to guarantee task and message deadlines while reducing the power consumption. The resource management system is composed by two parts: a Task and Speed Scheduler (TSS) and a Packet Scheduler (PS). TTS is in charge of producing task scheduling and DVFS selection. Instead, PS is in charge of producing packets queuing and delivering to the network interface.

Kumar et al. [22] presented two slack allocation algorithms for energy saving based on both DVFS and DMS techniques. The authors consider a single-hop wireless real-time embedded system, where each task node is composed by precedence constrained message-passing subtasks. Furthermore, sub-tasks and messages are considered non-preemptable. Energy consumptions for both computation and communication are analyzed by a new metric, called normalized energy gain. The authors proposed two algorithms: the Gain based Static Scheduling (GSS) and the Distributed Slack Propagation (DSP). While the former is used off line and computes the slack considering the worst-case execution for each scheduled entity (sub-task or message), the latter is used on-line to exploit the additional slack available when tasks execute for less than their predicted worst-case computations.

II. System Models

We consider a distributed real-time embedded system composed by autonomous nodes interconnected through a shared media (e.g., wireless communication). A generic node executes a set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of sporadic tasks. Each task $\tau_i = (C_i, T_i, D_i)$ is characterized by a worst-case number of machine cycles C_i, a minimum inter-arrival time T_i, and a relative deadline $D_i \leq T_i$. At a generic instant t, the residual number of machine cycles of task τ_i is denoted as $c_i(t)$. The worst-case execution time is given by C_i/f, where f is the CPU frequency used by the system. A task τ_i generates an infinite sequence of jobs $\tau_{i,j}$, each having activation time $a_{i,j}$ and absolute deadline $d_{i,j} = a_{i,j} + D_i$. Tasks are scheduled by Earliest Deadline First (EDF) [23].

Each task τ_i produces a message m_i characterized by a payload M_i and a deadline L_i relative to the task activation, such that $L_i > D_i$. The absolute deadline of the message produced by job $\tau_{i,j}$ is denoted by $l_{i,j} = a_{i,j} + L_i$.

The analysis we are proposing assumes a given bandwidth allocation specified according to a Time Division Multiple Access (TDMA) scheme, modeled as a set of disjointed slots $B = \{slot_1, \ldots, slot_r\}$, where each slot $slot_k$ is described by a start time b_k^s and an end time b_k^e. Such slots are externally assigned by a network coordinator that guarantees that messages are transmitted/received within their deadlines. However, the design of the coordinator is out of the scope of this work.

To simplify the power management algorithm, task scheduling is decoupled from message communication by making the following assumptions:

1) The messages generated by a task are moved into a shared communication buffer and transferred to the transceiver internal buffer whenever the bandwidth is available. The transfer time is considered to be negligible.
2) To allow messages transfer between buffers, the processor must be active during communication slots, whereas CPU activity is not required outside such intervals.
3) The bandwidth slots allocated by the external network coordinator are such that any scheduling algorithm that guarantees the timing constraints of the task set also meets the messages deadlines.

A graphic explanation about all the defined parameters is reported in Figure 1 for both the computational and the communication components.

A. Power Model

Each node consists of a CPU (processing element) and a Transceiver (transmitting and receiving element). The CPU can be in one of the following states.

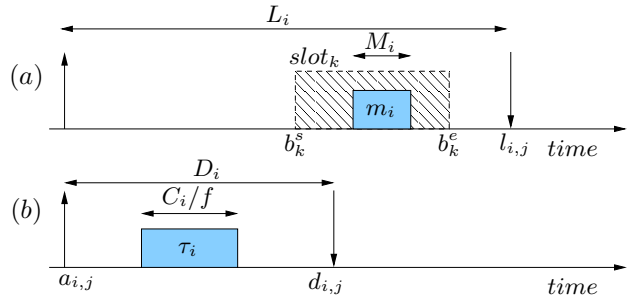

Fig. 1. Timing parameters for a task (b) and its generated message (a).

- *Active*. In this state, a device performs its job, executing tasks or handling messages. The power consumed in this state at frequency f_i is denoted as P_{a_i};
- *Standby*. In this state, the device does not provide any service, but consumes a small amount of power P_s to be ready to become active within a short period of time;
- *Sleep*. In this state, the device is completely turned off and consumes the least amount of power P_σ; however, it takes more time to switch to the active state.

For a processor that supports DVFS management, the power consumed in active mode depends on the frequency at which the processor executes. Such a frequency, measured in cycles per seconds (CPS), is assumed to vary in a set of F possible elements $[f_1 = f_{min}, f_2, \ldots, f_F = f_{max}]$, In particular, the processor power consumption in active mode is described according to the model proposed by Martin et al. [24]:

$$P_{a_i} = a_3 f_i^3 + a_2 f_i^2 + a_1 f_i + a_0, \tag{1}$$

which considers all components that affect power consumption, such as voltage, frequency, non-linearities and leakage effects.

Switching from two operating modes takes a different amount of time and consumes a different amount of energy, depending on the specific modes. In particular, the following notation is used throughout the paper: t_{xy} and E_{xy} denote the time and the energy required for a transition from state x to state y, respectively. For example, $t_{a\sigma}$ and $E_{a\sigma}$ represent the time and the energy required for the active-sleep transition. For all devices we have that $P_\sigma < P_s < \min_i P_{a_i}$ and $t_{sa} < t_{\sigma a}$. The switching overhead between the standby and the active mode is considered to be negligible, compared to the other switches, which is the same assumption made by other authors [25], [26].

In this work, we assume the usage of a standalone transceiver with minimal functionality, with only two states: ON and OFF. As the transceiver implements only the physical layer of the communication stack, the computational unit has to manage the remaining layers. This choice forces the device to be OFF if the CPU is in $Standby$ or $Sleep$ states. As our analysis needs only these two fundamental working states, it is applicable to all existing communication devices. In the following, the transceiver power consumption in the ON and OFF states are denoted as P_{on} and P_{off}, respectively.

Table I summarizes all the allowed modes with their total power consumption.

	Radio On	Radio Off
CPU Sleep	Not allowed	$P_\sigma + P_{off}$
CPU Standby	Not allowed	$P_s + P_{off}$
CPU On	$P_{a_i} + P_{on}$	$P_{a_i} + P_{off}$

TABLE I
ALLOWED POWER MODES.

III. SCHEDULABILITY ANALYSIS

This paper addresses the problem of guaranteeing real-time constraints of a sporadic task set minimizing the energy consumption of a computational node composed by a CPU and a transceiver.

Schedulability analysis of the task set is performed using the demand bound function dbf [27] to describe the application computational requirements, and the supply bound function sbf [28] to characterize the service provided by the processor. The next section briefly recalls the results used for the analysis.

A. Fundamental results

In the case of EDF, Baruah [29] showed that the dbf of a sporadic task set, in a generic interval, can be computed as

$$dbf(t_1, t_2) = \sum_{i \in \Gamma} \left(\left\lfloor \frac{t_2 + T_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1}{T_i} \right\rceil \right) C_i. \tag{2}$$

In the processing model considered in this paper, the processor can run at a frequency f whenever the processor is in active mode, while it is steady if the processor is in standby or sleep mode. Hence, the sbf linearly increases with slope f when the CPU is active, and remains constant in standby and sleep states. For a given power state, the sbf in an interval $[t_1, t_2)$ is computed as

$$sbf(t_1, t_2, f) = (t_2 - t_1) f. \tag{3}$$

The real-time constraints of a scheduling component are met if and only if, in any interval of time, the resource demand of the component never exceeds the resource supply curve. That is, if and only if

$$\forall t_1, t_2 \in \Re^+, \quad t_2 > t_1, \quad dbf(t_1, t_2) \leq sbf(t_1, t_2, f). \tag{4}$$

Ripoll [30] gave an upper bound L_a on the time interval in which Equation (4) must be checked, under the assumption that $D_i \leq T_i$ for each task τ_i. This work extends such an analysis taking into account the system frequency f as

$$L_a(f) = \quad max \left\{ D_1, \ldots, D_n, \frac{\sum_i (T_i - D_i) U_i}{\frac{f}{f_{max}} - U} \right\}.$$

Spuri [31] also defined the *Busy Period* (BP) as the longest interval of time where the processor is never idle, computed assuming synchronous and offset-free activations of tasks. This work extends the concept of BP considering a system working at frequency f. Hence, $BP(f)$ can be used as another upper bound for the schedulability test. Therefore, if $U < \frac{f}{f_{max}}$, to guarantee the feasibility it is sufficient to check Equation (4) just at the task deadlines in $[t_1, t_1 + L^*(f))$, where $L^*(f)$ is defined as

$$L^*(f) = min\{L_a(f), BP(f)\}. \tag{5}$$

To speed up the schedulability test, the QPA algorithm proposed by Zhang and Burns [32] could be used in our approach.

IV. Proposed Approach

The proposed approach mixes at run-time DVFS and DPM techniques to reduce energy consumption while meeting all task deadlines, if there exists a feasible schedule. The combination of DVFS and DPM is done by forcing a CPU sleep interval followed by an active interval executed at a fixed frequency. Such a frequency is selected to minimize the energy (per unit of computation) between the current and the next invocation of the analysis.

The j-th instance of the analysis is performed either at the end of an active interval or at the end of a communication slot. The former instant represents the beginning of an idle period that can be prolonged further by the analysis, while the latter is selected to exploit the slack, if any, collected during the forced activity inside the slot.

The following terminology is used to identify particular timing instants.

- t denotes the current time.
- $next\_act(t)$ denotes the next activation time after t.
- t_{a_j} denotes the time at which the j-th instance of the analysis is invoked. If there are pending jobs at time t, t_{a_j} is set at the current time t, otherwise t_{a_j} is postponed at $next\_act(t)$:

$$t_{a_j} = \begin{cases} next\_act(t), & \text{if no pending jobs at } t; \\ t, & \text{otherwise.} \end{cases} \quad (6)$$

  The index j referring to a particular instance will be omitted whenever not necessary.
- t_{w_i} denotes the latest time after t_{a_j} at which the processor can return active with frequency f_i and still guarantee the schedulability of the task set.
- t_{idle_i} denotes the first idle time after t_{w_i} assuming the processor is executing at frequency f_i.
- t_{e_i} denotes the effective time at which the processor can become idle considering the activity constraint inside bandwidth slots. Hence, if t_{idle_i} falls before b_k^s, t_{e_i} is set at t_{idle_i}; otherwise t_{e_i} is forced to occur at the end of the bandwidth slot, that is, $t_{e_i} = b_k^e$.

When the analysis is invoked at time t_a, the following actions are performed:

1) For each frequency f_i, the analysis derives the longest inactive interval δ_i exploitable in sleep state from t_a, such that the task set is still feasible when the CPU is turned active at $t_a + \delta_i$. A negative value of δ_i implies that the task set can not be schedulable at that frequency. δ_i is determined as the minimum among the inactive intervals computed for each deadline, that is

$$\delta_i(t) = \min_{d_j \in [t, t+L^*(f_i))} \left\{ d_j - \frac{dbf(t, d_j)}{f_i} - t \right\}. \quad (7)$$

2) To ensure that the CPU is active during the assigned bandwidth slots, the wake up time t_{w_i} is set equal to the minimum between $t_a + \delta_i$ and the beginning of the next slot b_k^s

$$t_{w_i} = min\{t_a + \delta_i(t_a), b_k^s\}. \quad (8)$$

3) For each frequency f_i, the analysis also computes the next idle time t_{idle_i} from t_{w_i} assuming worst-case executions. In particular, t_{idle_i} is computed as the minimum value satisfying the following recurrent relation:

$$t_{idle_i}^{s+1}(t_a) = \sum_{\tau_j \text{ active}} \frac{c_j(t_a)}{f_i} + \sum_{j \in \Gamma} \left(\left\lfloor \frac{t_{idle_i}^s}{T_j} \right\rfloor - \left\lfloor \frac{t_a}{T_j} \right\rfloor \right) \frac{C_j}{f_i}. \quad (9)$$

initialized with value $t_{idle_i}^0(t_a) = t_a + \sum_{\tau_j \text{ active}} \frac{c_j(t_a)}{f_i}$. The analysis then computes the effective idle time t_{e_i} taking into account the bandwidth constraint.

$$t_{e_i} = \begin{cases} t_{idle_i}, & t_{idle_i} < b_k^s; \\ b_k^e, & \text{otherwise.} \end{cases}$$

4) Under a frequency f_i, the energy consumption E_i in the interval $[t_a, t_{e_i}]$ is computed as the sum of the energy spent in sleep mode in $[t_a, t_{w_i}]$ and in active mode in $[t_{w_i}, t_{e_i}]$, that is

$$E_i(t_a, t_{w_i}, t_{e_i}) = (t_{w_i} - t_a)P_\sigma + (t_{e_i} - t_{w_i})P_{a_i}. \quad (10)$$

Since each frequency f_i causes a different amount of computation in the interval $[t_a, t_{e_i}]$, denoted as $W_i(t_a, t_{e_i})$, the normalized parameter *Energy Per Cycle* (EPC_i) is introduced, representing the energy cost per instruction cycle. It is computed as

$$EPC_i(t) = \frac{E_i(t_a, t_{w_i}, t_{e_i})}{W_i(t_a, t_{e_i})}. \quad (11)$$

A detailed analysis about the computation of W_i is carried out in Appendix A.

5) Among the possible frequencies that guarantee feasibility, the approach selects f^* featuring the minimum EPC_i. t_w^* and t_e^* denote the wake up time and the effective idle time resulting from the selected frequency f^*, respectively.
6) If the interval $[t, t_w^*)$ is shorter than $t_{a\sigma} + t_{\sigma a}$, it is not possible to adopt the sleep state to wake up within t_w^*, so the standby state is chosen; otherwise, the sleep state is selected.
7) The instant of the next occurrence of the analysis $t_{a_{j+1}}$ is set equal to t_e^*; however, if the next idle time is advanced due to early completions, the analysis is triggered as soon as the idle occurs and $t_{a_{j+1}}$ is updated accordingly.

Figure 2 illustrates an example that clarifies the steps of the proposed approach. In the example, the CPU supports three different frequencies sorted in ascending order: f_1, f_2 and f_3.

In the example, frequency f_1 leads to an unfeasible schedule, since δ_1 is negative, whereas f_2 and f_3 produce feasible solutions, since both δ_2 and δ_3 are positive.

Notice that, when considering frequency f_2, $t_a + \delta_2$ falls before b_k^s, hence $t_{w_2} = t_a + \delta_2$, whereas for f_3, t_{w_3} is set equal to the beginning of the slot b_k^s, as $t_a + \delta_3 \geq b_k^s$.

For both frequencies f_2 and f_3, t_{e_2} and t_{e_3} takes the value of b_k^e, as both t_{idle_2} and t_{idle_3} occur after b_k^s.

To choose between the two feasible frequencies (f_2 and f_3), the normalized energy consumption is computed. Such a value is intrinsically derived from the platform power model.

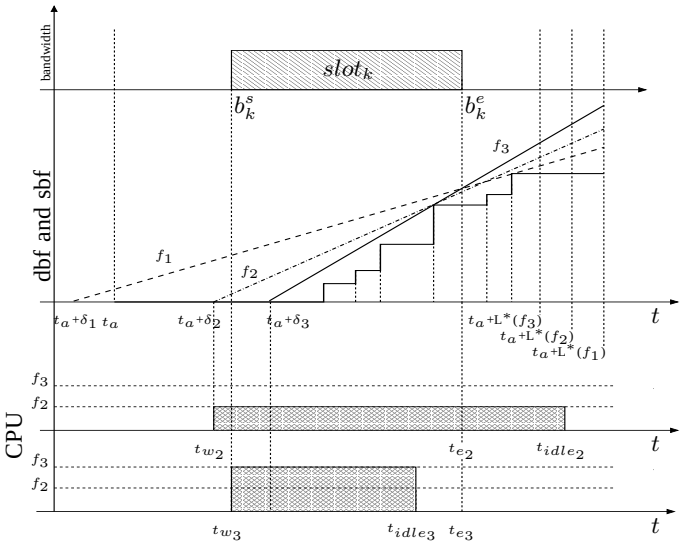

Fig. 2. Example of the analysis' behaviour.

V. Algorithm

This section illustrates the *Discrete Energy Aware Scheduling* (*DEAS*) algorithm that implements the approach introduced in Section IV.

Algorithm 1 Discrete Energy Aware Scheduling - DEAS

procedure (t)
1: **Input**: $t : \forall k, t \notin [b_k^s, b_k^e)$
2: compute t_a according to Equation (6);
3: **for all** f_i **do**
4: compute $\delta_i(t_a)$ as in Equation (7);
5: **if** $\delta_i(t_a) < 0$ **then**
6: set f_i not feasible and **continue**;
7: **end if**
8: compute t_{w_i} according to Equation (8);
9: compute t_{e_i}, W_i as shown in Appendix A;
10: compute EPC_i according to Equation (11);
11: **end for**
12: compute f^* feasible that minimizes EPC_i;
13: set wake up time at t_w^*;
14: set CPU frequency to f^*;
15: **if** $t_w^* - t \geq t_{a\sigma} + t_{\sigma a}$ **then**
16: put the processor in sleep state;
17: **else**
18: put the processor in standby state;
19: **end if**

A. Complexity

Equation (6), executed at line 2, has the complexity of an extraction from an ordered list of task activation times; that is, $O(1)$. On the other hand, the insertion complexity is $O(log_2(n))$, where n is the number of tasks. Given n and the maximum number of deadlines a single task can produce in the analysis interval, p, the maximum number of analysis points of the dbf is np. The upper bound of p is computed as the number of occurrences of the task with the shortest period in the analysis interval: $\left\lceil \frac{max_i L^*(f_i)}{min_i\{T_i\}} \right\rceil$. Supposing to arrange the active deadlines in a sorted list, with complexity $O(log_2(n))$ (as the active deadlines are always n) to keep the ordering, the computation of dbf, executed every time the algorithm is invoked, has a total complexity of $O(log_2(n)np)$.

The computation of the δ_i, at line 4, involves a complexity $O(np)$. The computation performed at line 9 has a complexity of $O(nq)$, where q is defined as the maximum number of activations a task can generate in $[t_{a_j}, t_{a_j} + max_i L^*(f_i) + max_k\{b_k^e - b_k^s\}]$. The reason is that the algorithm analyzes all the activations, computing the actual workload and any idle gap, as shown in Appendix A. The q upper bound is computed as $\left\lceil \frac{max_i L^*(f_i) + max_k\{b_k^e - b_k^s\}}{min_i\{T_i\}} \right\rceil$. The computation of the EPC_i has complexity $O(1)$. Hence, the for loop, executed at line 3, has a complexity of $O(nF(p+q))$, where F is the total number of available frequencies.

Globally, the proposed algorithm has a complexity of $O((log_2(n) + F)qn)$, being $q \geq p$.

B. Example

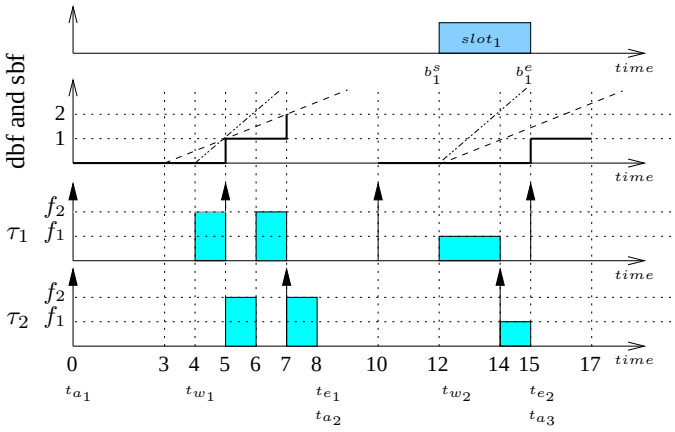

Fig. 3. DEAS example.

The behavior of the DEAS algorithm is now illustrated using the example in Figure 3. The assigned bandwidth is composed by one slot in the interval $[b_1^s, b_1^e]$ equal to [12,15]. The CPU allows 2 frequencies equal to $f_1 = 5$ and $f_2 = 10$, and schedules a task set of 2 synchronous implicit periodic tasks with periods $T_1 = D_1 = 5$ and $T_2 = D_2 = 7$, and worst-case execution cycles $C_1 = 10$ e $C_2 = 10$. For the task set under analysis, we have $L^*(f_1) = 4$ and $L^*(f_2) = 2$. The result of the off-line computation for L_{max} is 7. The power consumptions in the active state are $P_{a_1} = 3$ and $P_{a_2} = 6$, while $P_\sigma = 1$ and $t_{a\sigma} + t_{\sigma a}$ is considered negligible for the sake of simplicity.

The algorithm has its first invocation at $t_{a_1} = 0$ because two jobs are already pending. Both frequencies guarantee the task set feasibility with wake up times $t_{w_1} = 3$ and $t_{w_2} = 4$, respectively.

Executing at frequency f_1, the first idle time t_{idle_1} occurs at $t = 13$ because, from time $t_{w_1} = 3$, the busy period consists of three instances of τ_1 and two instances of τ_2, for a total execution of 10 units of time. Due to the bandwidth activity constraint, t_{e_1} is set to 15. Instead, running at frequency f_2, the next idle time t_{idle_2}, from time $t_{w_2} = 4$, occurs at time $t = 8$ and, since it falls before b_1^s, we have $t_{e_2} = 8$. Once the interval $[t_{w_i}, t_{e_i}]$ is determined, the algorithm computes $EPC_1 = 0.71$ and $EPC_2 = 0.70$, and therefore sets $f^* = f_2$.

Using f_2, the second invocation of the algorithm occurs at $t = 8$, causing the postponement of t_{a_2} to $t = 10$. Running at frequency f_2, the CPU can wake up at 13, but, for the activity constraint, t_{w_2} is set to $b_1^s = 12$. Consequently, $t_{idle_2} = 13$ and $t_{e_2} = b_1^e = 15$. Instead, using frequency f_1, the algorithm obtains $t_{w_1} = 12$, $t_{idle_1} = 18$, and $t_{e_1} = b_1^e = 15$.

In such a scenario, the energy consumptions are $EPC_1 = 0.73$ and $ECP_2 = 1$, and therefore the chosen frequency is $f^* = f_1$.

VI. Experimental results

This section presents a set of experimental results that show the effectiveness of our approach with respect to other classical solutions. The results are obtained by simulation using a synthetic workload under three power consumption profiles derived from real platforms according to the power consumption model described in Section II-A.

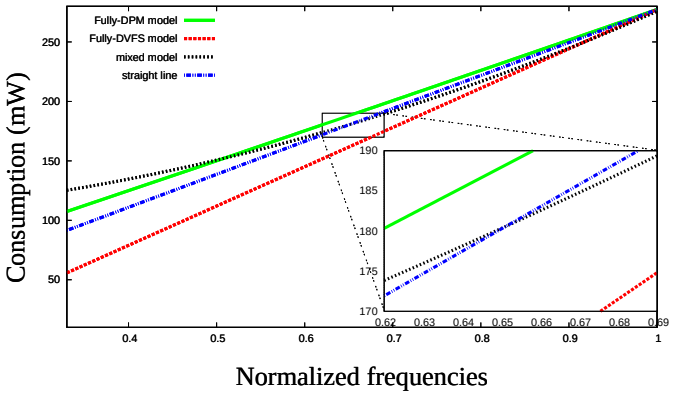

Fig. 4. Speed-Power characterization for different platforms.

To test the behavior of our algorithm, three CPU consumption profiles (*power models*), shown in Figure 4, are introduced. Each profile is described by the power consumption coefficients $[a_3, a_2, a_1, a_0]$:

- *Fully-DPM* $[0.0, 5.6, 246.12, 25.93]$;
- *Fully-DVFS* $[0.0, 0.0, 330.62, -53.32]$;
- *Mixed* $[0.0, 150.55, 24.5, 100.78]$.

The fully-DPM model has been extracted from the Microchip dsPIC [1] datasheet, interpolating the typical consumptions. The other two models have been synthetically derived from the first one to achieve different but comparable behaviors, at the same time, with respect to the original. Fully-DPM and fully-DVFS represent opposite cases: in a fully-DVFS scenario, halving the speed (doubling the execution time) always implies a reduction of the energy consumption, while in fully-DPM cases, the consumption increases. A model is defined as DPM or DVFS according to its position with respect to the straight line. Such a line represents a theoretical situation in which slowing down has the same energy consumption of executing at a different speed. The mixed model has a threshold frequency f_{th} at speed 0.65 meaning that its behavior is DVFS-like above f_{th} and DPM-like below.

The frequency range of the CPU used in the simulation is $[12.5, 40]$ MHz. The sleep state consumption P_σ is 1.49 mW and the wake up time takes about 20 ms. The standby state has a higher consumption P_s of 9.9 mW, but a shorter wake up time within 8 cycles. As for the fully-DPM model consumptions, such values were extracted from the dsPIC datasheet. All the simulations have been executed using a set of 8 evenly distributed frequencies.

For comparison purposes, four scheduling policies have been implemented in the simulator:

- *EDF* with no energy considerations, where the processor is assumed always active at the maximum frequency, even during idle intervals.
- *pureDVFS* on top of EDF, where the CPU runs with the minimal speed, computed off-line, that guarantees feasibility according to the task set. The actual speed is the lowest frequency greater than the minimal one.
- *pureDPM*, where, as soon as there is an idle time and no assigned bandwidth, the task execution is postponed as much as possible and then scheduled by EDF at the maximum speed.
- *DEAS*, the algorithm introduced in this paper.

An execution scenario is characterized by the tuple (U, n_t, B, n_B), where U denotes the utilization of the task set, n_t the number of tasks, B the communication bandwidth (expressed as a percentage of the hyperperiod), and n_B the number of chunks in which the bandwidth is split. All the slots are generated with the same length, whereas slot positions are randomly generated with a uniform distribution.

Given the total utilization factor U, individual task utilizations are generated according to a uniform distribution [33].

Payload and message deadlines are generated to meet the hypothesis on messages guarantee. The computed values are not described here because they have no effect on the task scheduling algorithm, as described in Section II.

Trying to find a trade-off between the simulation accuracy and the simulation time (it increases exponentially with the number of tasks), each result was computed as the average consumption of 30 executions. To simplify comparisons, the results are normalized against the value obtained applying the EDF policy to the same tuple (U, n_t, B, n_B).

In the first experiment, the energy consumption is evaluated as a function of the utilization U and the number of tasks n_t. All the three algorithms have been tested with Bandwidth $B = 0.3$, $n_B = 5$ chunks and three different utilization factors. The results show that both U and n_t do not affect energy consumption significantly, therefore the graph is not reported.

The next experiments evaluate the energy consumption, under different power models, as a function of the utilization factor U with $n_t = 7$, $B = 0.3$ and $n_B = 10$.

Results show that DEAS always outperforms the other algorithms for all power models and for any utilization.

As shown in Figure 5, due to the activity constraint posed by the bandwidth slots, DEAS outperforms pureDPM even in fully-DPM power models. Instead, Figure 6 shows that in fully-DVFS power models such a constraint has no effect on pureDVFS: keeping the system active represents the default behavior and, with respect to the analyzed power model, the best solution. For this reason, DEAS and pureDVFS have similar performances.

Under a fully-DPM and in a mixed context, Figure 5 and Figure 7 shows that pureDVFS acts better than pureDPM

[1] DSPIC33FJ256MC710 microcontroller

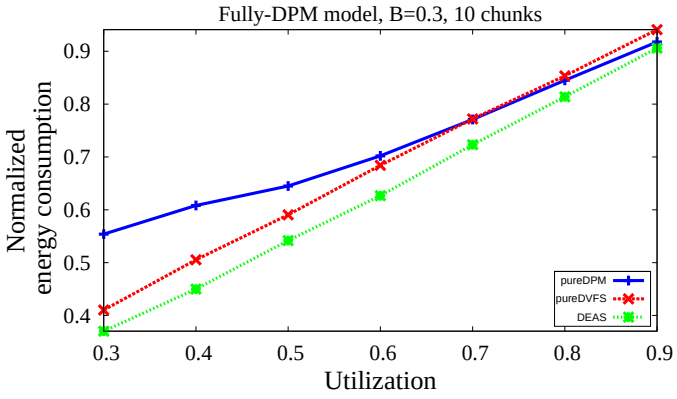

Fig. 5. Analysis of consumptions with a fully-DPM power model.

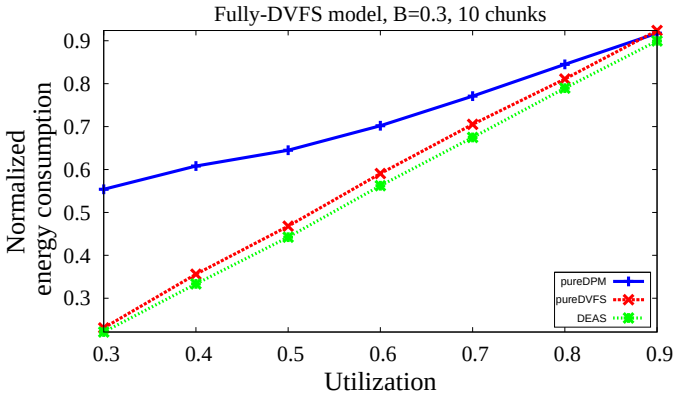

Fig. 6. Analysis of consumptions with a fully-DVFS power model.

for low U values, because the CPU can not be switched off inside bandwidth slots. Instead, for higher utilization values, the consumptions are similar. Note that all the graphs show that DEAS is always able to select the right balance between DVFS and DPM depending on the specific characteristics of the architecture.

VII. Conclusions

This paper addressed the problem of reducing the energy consumption in distributed embedded systems with time and communication constraints. The proposed solution exploited both DPM and DVFS techniques to reduce the energy consumption within each node, balancing them depending on the specific architecture characteristics, actual workload, and bandwidth allocation.

The method has been developed under realistic assumptions, such as discrete frequency levels, mode switch overhead, and communication constraints. Experimental results showed that the combined DPM/DVFS approach dominates each individual technique (pureDPM and pureDVFS) for all power models and any task set utilization.

To simplify the analysis, in this paper the communication bandwidth was assumed to be statically allocated according to a TDMA scheme. As a future work, we plan to extend the method to schedule both tasks and messages to further reduce the energy consumption.

Appendix

This section presents the procedure $compute\_te\_W$, formally defined in Algorithm 2, that computes the effective

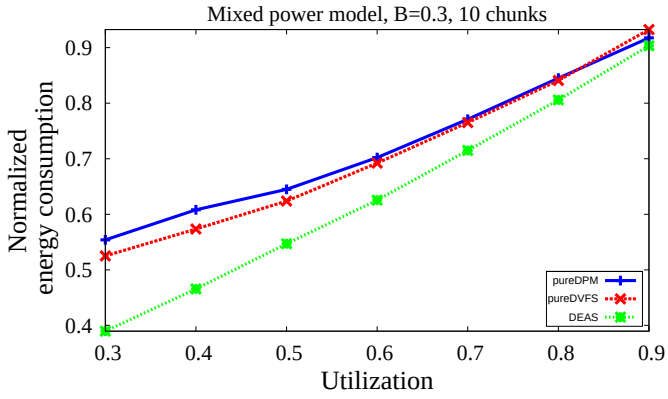

Fig. 7. Analysis of consumptions with a Mixed power model.

idle time t_{e_i} and the effective workload W_i, concepts already introduced in Section IV. The algorithm, based on the current workload, computes next idle times t_{idle_i} till t_{e_i} is found, taking into account bandwidth constraints. However, note that the procedure output is composed by t_{e_i} and W_i only. To reduce the DEAS algorithm complexity, such computations are integrated into a single routine.

Algorithm 2 Procedure to compute t_{e_i} and W_i

procedure $compute\_te\_W$
1: **input**: f_i, t_{w_i}
2: **output**: t_{e_i}, W_i
3: $W_i = 0$; $t_{start} = t_{w_i}$;
4: **loop**
5: $t^0_{idle_i} = t_{start} + \sum_{\tau_j \text{ active}} \frac{c_j(t_{start})}{f_i}$;
6: **do**
7: compute $t^s_{idle_i}$ according to Equation (9);
8: **if** $t^{s-1}_{idle_i} < b^e_k \leq t^s_{idle_i}$ **then**
9: $t_{e_i} = b^e_k$;
10: W_i += $(t_{e_i} - t_{start})f_i$;
11: **return**;
12: **end if**
13: **while** $t^{s-1}_{idle_i} \neq t^s_{idle_i}$;
14: W_i += $(t^s_{idle_i} - t_{start})f_i$;
15: **if** $t^s_{idle_i} \notin [b^s_k, b^e_k]$ **then**
16: $t_{e_i} = t^s_{idle_i}$;
17: **return**;
18: **else**
19: **if** $next\_act(t^s_{idle_i}) \geq b^e_k$ **then**
20: $t_{e_i} = b^e_k$;
21: **return**;
22: **end if**
23: $t_{start} = next\_act(t^s_{idle_i})$;
24: **end if**
25: **end loop**

Figure 8 shows the effective workload of three key scenarios. For each case, the effective workload is represented by the sum of the slashed areas. Such a value is expressed as number of machine cycles, so the working frequency must be considered.

First, to determine t_{idle_i} the iterative approach is initialized as described in Section IV. Whenever $t^s_{idle_i}$, at a generic step s, crosses the end of the current bandwidth slot $t^{s-1}_{idle_i} < b^e_k \leq t^s_{idle_i}$ (cases a and c), the procedure stops setting $t_{e_i} = b^e_k$ and accounting in W_i the workload between the beginning of the

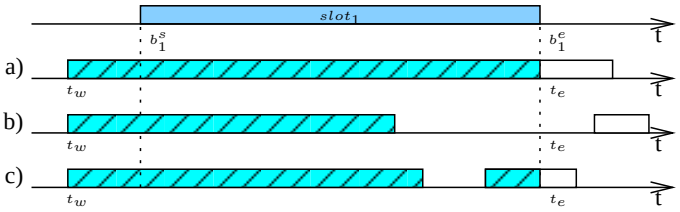

Fig. 8. Examples of effective workload.

current Busy Period and the end of the bandwidth slot b_k^e. Note that, if the recurrent relation converges ($t_{idle_i}^{s-1} = t_{idle_i}^s$) outside the bandwidth slot, no slot occurs during the analyzed Busy Period, hence $t_{e_i} = t_{idle_i}^s$ and W_i is increased by $(t_{idle_i}^s - t_{w_i})f_i$. If $t_{idle_i}^s$ converges inside the bandwidth slot (cases b and c), W_i is incremented by $(t_{idle_i}^s - t_{w_i})f_i$. Then, the routine checks whether a new task activation occurs after the end of the current bandwidth slot, i.e. $next\_act(t_{idle_i}^s) \geq b_k^e$. In such a case (case b), the effective idle time t_{e_i} is set to b_k^e and W_i is increased by $(t_{idle_i}^s - t_{w_i})f_i$; otherwise (case c), the contribution of the next Busy Period must be taken into account considering $next\_act(t_{idle_i}^s)$ as a starting instant.

REFERENCES

[1] C. Schurgers, V. Raghunathan, and M. B. Srivastave, "Modulation scaling for real-time energy aware packet scheduling," in *Global Telecommunications Conferance (GLOBECOMM 01)*, San Antonio, Texas (USA), 2001, pp. 3653–3657.

[2] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. of the 41st ACM/IEEE Design Automation Conference (DAC)*, 2004, pp. 275–280.

[3] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, pp. 68–75, 2003.

[4] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems," in *ICCAD '07: Proc. of the 2009 International Conference on Computer-Aided Design*. New York, NY, USA: ACM, 2007, pp. 289–294.

[5] V. Devadas and H. Aydin, "On the interplay of dynamic volatage scaling and dynamic power management in real-time embedded applications," in *EMSOFT'08: Proc. of the 8th ACM Conference on Embedded Systems Software*. New York, NY, USA: ACM, 2008, pp. 99–108.

[6] B. Zhao and H. Aydin, "Minimizing expected energy consumption through optimal integration of dvs and dpm," in *ICCAD '09: Proc. of the 2009 International Conference on Computer-Aided Design*. New York, NY, USA: ACM, 2009, pp. 449–456.

[7] C. Nastasi, M. Marinoni, L. Santinelli, P. Pagano, G. Lipari, and G. Franchino, "Baccarat: a dynamic real-time bandwidth allocationpolicy for ieee 802.15.4," in *Proc. of IEEE Percom 2010, International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2010)*, Mannheim, Germany, 2010.

[8] A. Koubâa, M. Alves, E. Tovar, and A. Cunha, "An implicit gts allocation mechanism in ieee 802.15.4 for time-sensitive wireless sensor networks: theory and practice," *Real-Time Syst.*, vol. 39, no. 1-3, pp. 169–204, 2008.

[9] L. Santinelli, M. Marinoni, F. Prosperi, F. Esposito, G. Franchino, and G. Buttazzo, "Energy-aware packet and task co-scheduling for embedded systems," in *Proc. of the tenth ACM international conference on Embedded software*, ser. EMSOFT '10, 2010, pp. 279–288.

[10] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min, "Performance camparision of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symp.*, San Jose, California, September 2002, pp. 219–228.

[11] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: Frequency-aware static timing analysis," in *Proc. 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003, pp. 40–51.

[12] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, May 2004.

[13] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo, "Periodic power management schemes for real-time event streams," in *CDC'09: Proc. of the 48th IEEE Conference on Decision and Control*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 6224 – 6231.

[14] ——, "Adaptive dynamic power management for hard real-time systems," in *RTSS'09: Proc. of the 30th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2009.

[15] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, 1995, pp. 374–382.

[16] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. of the 39th ACM/IEEE Design Automation Conference (DAC)*, 2002, pp. 183–188.

[17] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, 2001, pp. 95–105.

[18] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems," in *International Conference on Computer-Aided Design (ICCAD)*, 2007, pp. 289–294.

[19] B. Mochocki, D. Rajan, X. S. Hu, C. Poellabauer, K. Otten, and T. Chantem, "Network-aware dynamic voltage and frequency scaling," in *RTAS '07: Proc. of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 215–224.

[20] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *SOSP '01: Proc. of the 18th ACM Symposium on Operating Systems Principles*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 89–102.

[21] J. Yi, C. Poellabauer, X. S. Hu, J. Simmer, and L. Zhang, "Energy-conscious co-scheduling of tasks and packets in wireless real-time environments," in *RTAS '09: Proc. of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 265–274.

[22] G. S. A. Kumar and G. Manimaran, "Energy-aware scheduling of real-time tasks in wireless networked embedded systems," in *RTSS '07: Proc. of the 28th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 15–24.

[23] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[24] T. Martin and D. Siewiorek, "Non-ideal battery and main memory effects on cpu speed-setting for low power," *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, pp. 29–34, 2001.

[25] J. Zhuo and C. Chakrabarti, "System-level energy-efficient dynamic task scheduling," in *Proc. of the 42nd ACM/IEEE Design Automation Conference(DAC)*, 2005, pp. 628–631.

[26] C.-Y. Yang, J.-J. Chen, C.-M. Hung, and T.-W. Kuo, "System-level energy-efficiency for real-time tasks," in *the 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, 2007, pp. 266–273.

[27] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *In Proc. of the 11th Real-Time Systems Symposium*. IEEE Computer Society Press, 1990, pp. 182–190.

[28] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–39, 2008.

[29] S. K. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Syst.*, vol. 24, no. 1, pp. 93–128, 2003.

[30] I. Ripoll, A. Crespo, and A. K. Mok, "Improvement in feasibility testing for real-time systems," in *IEEE Real-Time Systems Symposium*, 1996.

[31] M. Spuri, "Analysis of deadline scheduled real-time systems," INRIA Rocquencourt, Tech. Rep., 1996.

[32] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, pp. 1250–1258, 2009.

[33] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proc. of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, June 2004.