Proc. of the 10th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA 2005), Catania, Italy, Sept. 2005.

Adaptive DVS Management through Elastic Scheduling [*]

Mauro Marinoni and Giorgio Buttazzo
University of Pavia, Italy
Email: {mauro.marinoni, giorgio.buttazzo}@unipv.it

Abstract

Dynamic voltage scaling (DVS) is a technique used in modern microprocessors operated by battery to set voltage and frequency levels at proper values that meet performance requirements while minimizing energy consumption. Most of the present work on DVS management, however, is based on simplistic assumptions about the hardware characteristics that limit the real applicability of the proposed algorithms. Typical simplifying assumptions consider continuous processor speed, negligible overhead during voltage switching, task execution time linear with frequency, and tasks with equal power consumption. In this work, we enhance the task model to consider some of the real CPU characteristics, and integrate energy-aware algorithms with elastic scheduling to improve control performance of embedded systems running on architectures offering a limited number of operating modes. Implementation issues and experimental results for the proposed algorithm are also discussed.

1 Introduction

Most of modern embedded systems are operated by batteries, hence reducing energy consumption is one of the major issues to be solved. When such systems have also real-time requirements, then the issue of reducing energy consumption must be considered in conjunction with the one of meeting timing constraints. The problem of minimizing energy consumption while guaranteeing real-time constraints has been widely considered in the real-time literature [1, 2, 10], however most of the achieved results assume that the processor can continuously adapt its voltage and frequency to vary its power consumption.

Unfortunately, commercial processors only provide a limited number of operating modes, each characterized by a given voltage, clock frequency, and power consumption. Adapting a continuous model to a discrete DVS system clearly causes a waste of computational resource, because, in order to guarantee the feasibility of the task set, the processor speed must be set to the nearest level greater than the optimal one.

Recently, some authors proposed solutions for processors having discrete speed levels. For example, Mejia-Alvarez et al. [9] proposed an approach where a different frequency is assigned to each task; however, their processor model is simpler than the one used in this paper and the assignment problem is NP-hard, thus it can be solved on line only by a heuristic algorithm. More recently, Bini et al. [3] presented a method for approximating any speed level with two given discrete values, which are properly switched as a pulse width modulation signal to obtain its average value. Schedulability analysis to guarantee the feasibility of real-time task sets running under this mode was also presented.

In this paper, we present a novel DVS management algorithm that integrates energy-aware with elastic scheduling to cope with processors with a limited number of operating modes. To avoid wasting processing time due to speed quantization, we consider a more flexible task model, in which tasks can operate within a given range of periods, with different performance. Whenever the selected (discrete) speed level leaves some free processor bandwidth, elastic scheduling is invoked to reduce task periods to fully utilize the processor and increase the control performance. The algorithm allows the application to select between energy-oriented, performance-oriented, and user-defined strategies.

To better consider the effects of the hardware architecture on task execution times, we enhance the execution time model by splitting the code in two parts: one that varies with speed and one that is speed independent.

The proposed algorithm has been implemented in the Shark real-time operating system as a new scheduling module, and experimental results have been derived on an Athlon64 3000+ processor.

The rest of the paper is organized as follows. Section 2 presents the models used to describe the execution time and the energy consumption of a task. Section 3 describes

[*]This work has been partially supported by the Italian Ministry of University Research under contract 2003094275 (COFIN03) and contract 2004095094 (COFIN04).

the integrated DVS-elastic algorithm. Section 4 describes some experimental results, and Section 5 states our conclusions and future work.

2 Models

This section presents the models used throughout the paper for task execution times and power consumption. In addition, the elastic model is briefly recalled for the sake of completeness. To simplify the comparison between processors with different frequency range $[f_{min}, f_{max}]$, all the quantities of interest (power, computation times, etc.) will be expressed as a function of speed, defined as the normalized frequency $s = f/f_{max}$. Hence, the validity range for the normalized speed is $[s_{min}, s_{max}]$, where $s_{min} = f_{min}/f_{max}$ and $s_{max} = 1$. As for the voltage, the rule adopted in the algorithm is to select the minimum voltage level compatible with the frequency represented by the resulting speed. This approach is in line with the CPUFreq driver used in Linux, which leads to a simple and fast implementation.

2.1 Execution Time Model

Typically, task execution times are considered to be inversely proportional to the clock frequency and are modeled as $C_i(s) = C_{i_{max}}/s$, where $C_{i_{max}}$ is the task execution time at the maximum processor speed. Extensive experiments on real hardware, however, show that this assumption is not correct. A more accurate model is to split the execution time in two parts: one dependent on the CPU frequency, and one independent. While the former part is due to the code that works with the processor or with the hardware running at the CPU frequency, the latter part comes from the code that uses hardware devices that are not affected by frequency changes. For example, the video output operates at the frequency of the PCI bus, so its execution time does not change with the CPU speed.

Let $C_{i_{max}}$ be the execution time evaluated at the maximum processor speed, and let ϕ_i be the percentage of code which deals with the frequency-dependent hardware. Then, the task execution time can be modeled as

$$C_i(s) = \frac{\phi_i C_{i_{max}}}{s} + (1-\phi_i)C_{i_{max}}. \tag{1}$$

Unfortunately, classifying the code in the two parts described above is not easy, because the actual execution times depend on the architecture on which the task is running. For example, operations that rely on RAM memory are frequency-dependent if running on ARM processors and frequency-independent if running on x86 architectures.

The value of ϕ_i can be estimated experimentally by measuring the execution time of a task at the maximum frequency ($C_{i_{max}} = C_i(1)$) and at the minimum frequency ($C_{i_{min}} = C_i(s_{min})$). In fact, since by equation (1),

$$C_{i_{min}} = \frac{\phi_i C_{i_{max}}}{s_{min}} + (1-\phi_i)C_{i_{max}}$$

then ϕ_i can be computed as

$$\phi_i = \frac{C_{i_{min}} - C_{i_{max}}}{C_{i_{max}}} \frac{s_{min}}{1-s_{min}}. \tag{2}$$

Figure 1 shows the function $C(s)$ for a set of different values of ϕ. It can be seen that the simplified model that considers the execution time inversely proportional to the frequency (equivalent to the case $\phi = 1$) gets worse as the frequency decreases.

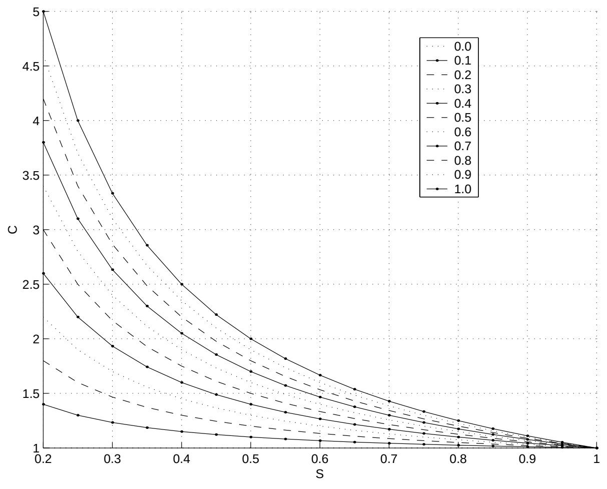

Figure 1. C(s) as a function of the ϕ parameter.

2.2 Energy Consumption Model

In CMOS integrated circuits, the dominant component of power consumption is the dynamic power dissipation due to switching, which is given by

$$P = C_{eff} V_{dd}^2 f$$

where C_{eff} is the effective capacity involved in switching, V_{dd}^2 is the supply voltage and f is the clock frequency. The value of the capacity C_{eff} depends on two factors: the load capacity C being charged/discharged and the activity weight α, which is a measure of the actual switching activity. Thus, $C_{eff} = \alpha * C$. Moreover, a voltage reduction causes an increase of the delays in the gates, according to the following formula:

$$D = k \frac{V_{dd}}{(V_{dd} - V_t)^2}$$

where k is a constant and v_t is the threshold voltage. Observing that the processor speed is directly proportional to the clock frequency f and inversely proportional to the gate delay, it turns out that the power consumption of a processor grows with the cube of its speed. The overall energy consumption of the system, however, also depends on other components of lower grade. Martin et al. [7, 8, 11] derived the following relation to describe the power consumption as a function of the speed:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0. \tag{3}$$

The K_3 term is the coefficient related to the consumption of those components that vary both voltage and frequency. The K_1 coefficient is related to the hardware components that can only vary the clock frequency, whereas K_0 represents the power consumed by the components that are not affected by the processor speed. Finally, the second order term (K_2) describes the non linearities of DC-DC regulators in the range of the output voltage.

2.3 Elastic Task Model

In our framework, each task is considered as flexible as a spring, whose utilization can be modified by changing its period within a specified range. More specifically, each task is characterized by four parameters: a worst-case computation time C_i, which depends on the processor speed according to equation (1), a minimum period $T_{i_{min}}$ (considered as a nominal period), a maximum period $T_{i_{max}}$, and an elastic coefficient E_i. The elastic coefficient specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration: the greater E_i, the more elastic the task. Hence, we consider a set of n elastic tasks, where each task is indicated by:

$$\tau_i(C_i, T_{i_{min}}, T_{i_{max}}, E_i).$$

In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_{min}}, T_{i_{max}}]$. Moreover, we define $U_{i_{min}} = C_i/T_{i_{min}}$ as the nominal utilization of task τ_i, $U_{max} = \sum_{i=1}^{n} C_i/T_{i_{min}}$ and $U_{min} = \sum_{i=1}^{n} C_i/T_{i_{max}}$.

Note that both U_{max} and U_{min} depend on the processor speed, hence any load variation due to a speed change is always subject to an *elastic* guarantee and is accepted only if there exists a feasible schedule in which all the periods are within their range. In our framework, tasks are scheduled by the Earliest Deadline First algorithm [6]. Hence, if $U_{max} \leq U_d$, all tasks can be created at the minimum period $T_{i_{min}}$, otherwise the elastic algorithm is used to adapt the tasks' periods to T_i such that $\sum \frac{C_i}{T_i} = U_d \leq 1$, where U_d is some desired utilization factor. It can easily been shown (see [4] for details) that a solution can always be found if $U_{min} \leq U_d$.

As shown in [4], if Γ_f is the set of tasks that reached their maximum period (i.e., minimum utilization) and Γ_v is the set of tasks whose utilization can still be compressed, then to achieve a desired utilization $U_d < U_{max}$ each task has to be compressed up to the following utilization:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_{max}} - (U_{v_{max}} - U_d + U_f)\frac{E_i}{E_v} \tag{4}$$

where

$$U_{v_{max}} = \sum_{\tau_i \in \Gamma_v} U_{i_{max}} \tag{5}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}} \tag{6}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i. \tag{7}$$

If there exist tasks for which $U_i < U_{i_{min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{max}}$ (so that $U_i = U_{i_{min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and equation (4) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if the desired utilization U_d is greater than or equal to the minimum possible utilization $U_{min} = \sum_{i=1}^{n} \frac{C_i}{T_{i_{max}}}$, the iterative process ends when each value computed by equation (4) is greater than or equal to its corresponding minimum $U_{i_{min}}$.

All tasks' utilizations that have been compressed to cope with an overload situation can return toward their nominal values when the overload is over.

2.4 Overall task model

To integrate the execution time model with the elastic one, each task will be denoted as follows:

$$\tau_i(C_{i_{max}}, \phi_i, T_{i_{min}}, T_{i_{max}}, E_i)$$

where the meaning of the parameters has been explained in the previous sections.

3 Algorithm Description

The algorithm proposed in this paper combines DVS management with elastic scheduling to enhance performance or reduce energy consumptions in systems with discrete operating modes. In the following, we assume that $U_{min}(s_{max}) \leq U_d$ (where $U_d \leq 1$), otherwise no feasible solution can be found and the task set is rejected by the feasibility test. The U_d parameter allows the user to tune the effective load on the processor according to the actual overhead introduced by the kernel, which can be measured off line. A value $U_d = 1$ should never be used, since other internal kernel activities (e.g., the interrupt handlers for the network or other peripheral devices) could create critical transient overload conditions.

At the application level, the user can choose among three high level strategies:

- **Energy saving**: energy consumption is minimized by selecting the lowest processor speed s_e that guarantees schedulability with the maximum periods; then, if $U_{min}(s_e) < U_d$, periods are reduced by the elastic algorithm to reach the desired utilization U_d, thus improving the control performance.

- **High performance**: control performance is maximized by selecting the lowest processor speed s_p that provides full performance, that is, that guarantees schedulability with the minimum periods; if $s_p > s_{max}$, that is, if $U_{max}(s_{max}) > U_d$, then s_p is set to s_{max} and task periods are enlarged by the elastic algorithm to reach feasibility with the desired utilization U_d.

- **User mode**: this mode allows the user to manually select a speed level s_u included in the range [s_e, s_p] defined by the two previous modes. If $U_{max}(s_u) > U_d$, periods are enlarged by the elastic algorithm to reach feasibility with the desired utilization U_d.

The algorithm consists of three hierarchical levels. At the top level, the power manager performs the acceptance test and computes the working frequency according to the selected strategy. At the medium level, the elastic scheduler computes the task periods and passes the task set to the system scheduler at the bottom level (EDF in the specific case). We can see each level as a function that converts the input task model into a new one accepted at the lower level. The hierarchical structure of the algorithm is illustrated in Figure 2.

The power manager is invoked every time a new task enters/leaves the system or a new speed is selected by the application. Task parameters used by this model are: task execution time at maximum frequency ($C_{i_{max}}$), the percentage of frequency-dependent code (ϕ_i), parameters of the task power model (K_3, K_2, K_1, K_0) and bounds of the task periods ($T_{i_{min}}, T_{i_{max}}$).

3.1 Computing the frequency bounds

Most commercial processors do not allow a continuous variation of voltage and frequency, but only provide a limited number of operating modes, each characterized by specific values for supply voltage, frequency, and power consumption.

Assuming that a single speed has to be used for the whole application, if the selected strategy is energy

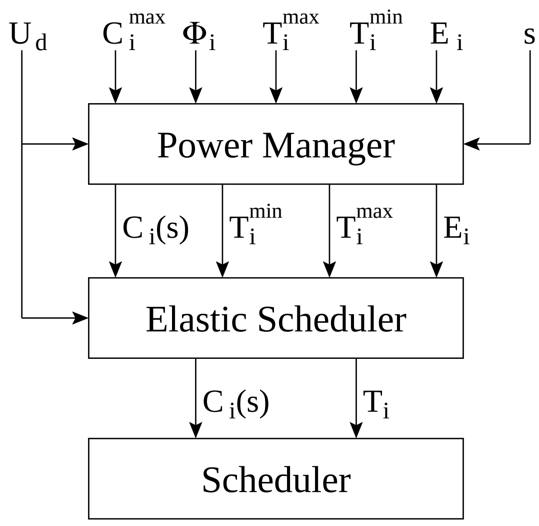

Figure 2. Block diagram of model flow.

oriented, then the minimum theoretical speed s_e^* (i.e., in a continuous range) is computed as the speed that minimizes energy consumption while guaranteeing the schedulability of the task set.

Considering the computation time model expressed in equation (1), the total processor utilization can also be expressed as a function of the processor speed:

$$\begin{aligned} U(s) &= \sum_{i=1}^{n} \frac{C_i(s)}{T_i} \\ &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_i} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_i} \\ &= \frac{U_D}{s} + U_F \qquad (8) \end{aligned}$$

where U_D is the processor utilization due to the frequency-dependent code, estimated at the maximum frequency, whereas U_F is the one that is frequency independent.

The minimum utilization computed with the maximum periods can also be expressed as a function of speed:

$$\begin{aligned} U_{min}(s) &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_{i_{max}}} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{max}}} \\ &= \frac{U_{D_{min}}}{s} + U_{F_{min}}. \qquad (9) \end{aligned}$$

Imposing $U_{min}(s) = U_d$ (desired utilization), the related speed is given by

$$s_e^* = \frac{\sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{T_{i_{max}}}}{U_d - \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{max}}}} = \frac{U_{D_{min}}}{U_d - U_{F_{min}}}.$$

If s_e^* is out of the range [0,1], the task set is not feasible and it is rejected by the guarantee test.

In the performance-oriented strategy, if $U_{max}(s_{max}) > U_d$, the speed s_p^* that guarantees the best performance is clearly s_{max}. Otherwise, the best theoretical speed s_p^* to achieve full performance is computed as the minimum speed that guarantees schedulability with the nominal periods.

Considering that U_{max} can be expressed as

$$\begin{aligned} U_{max}(s) &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{s T_{i_{min}}} + \sum_{i=1}^{n} \frac{(1-\phi_i) C_{i_{max}}}{T_{i_{min}}} \\ &= \frac{U_{D_{max}}}{s} + U_{F_{max}} \qquad (10) \end{aligned}$$

imposing $U_{max}(s) = U_d$, the best theoretical speed s_p^* is given by

$$s_p^* = \frac{\sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{T_{i_{min}}}}{U_d - \sum_{i=1}^{n} \frac{(1-\phi_i) C_{i_{max}}}{T_{i_{min}}}} = \frac{U_{D_{max}}}{U_d - U_{F_{max}}}.$$

Hence, in general,

$$s_p^* = \begin{cases} \frac{U_{D_{max}}}{U_d - U_{F_{max}}} & \text{if } U_{max}(s_{max}) \le U_d \\ s_{max} & \text{otherwise} \end{cases}$$

3.2 Frequency selection and period adjustment

Due to the discrete range of frequencies, it may not be possible to set the CPU speed at s_e^* or s_p^*. Hence, we set

$$s_e = \min_k \{s_k \mid s_k \ge s_e^*\}; \qquad (11)$$

$$s_p = \min_k \{s_k \mid s_k \ge s_p^*\}. \qquad (12)$$

Once the speeds s_e and s_p are computed and task set schedulability is guaranteed in the worst case situation, there can be some possible strategies to select the operating frequency as a function of the high level approach.

- If the objective is to minimize energy consumption, the actual speed is set to s_e. If $s_e > s_e^*$, then $U_{min}(s_e) < U_d$. Hence, to fully utilize the processor, task periods are reduced through elastic scheduling to bring the task set utilization at the desired level U_d.

- If the objective is to improve performance, the actual speed is set to s_p. Note that, if $U_{max}(s_p) \le U_d$, all tasks can run at their nominal period and the elastic algorithm is not used, otherwise task periods are expanded to reach the desired utilization U_d.

- Finally, if the user decides to select a specific speed $s_u \in [s_e, s_p]$ (among the available levels), then the elastic method is invoked to reach the desired utilization U_d.

It is worth observing that, in the energy-oriented strategy, the elastic mechanism is always used to reduce periods to bring the processor utilization up to U_d, so improving the control performance whenever possible. Such an improvement is larger when the number of available speeds is small. Clearly, the values of computation times used in the elastic method are estimated using the speed level computed by the power manager or selected by the user.

Another advantage of using the elastic approach in this context is that, if tasks have different power consumption, elastic coefficients can be set to reduce the energy of the tasks with higher power consumption. In fact, since the energy consumed by a task in a given interval is proportional to the number of jobs executed in that interval, elastic coefficients can be assigned so that tasks with higher power will be more compressed, that is are subject to a larger period variation to decrease their energy consumption. To obtain this result, the elastic coefficient E_i can be set as

$$E_i \propto P_i(s) C_i(s). \qquad (13)$$

where $P_i(s)$ is the power consumed by task τ_i, as defined in equation (3).

4 Experimental Results

The proposed algorithm has been implemented as a scheduling module of the Shark real-time operating system [5]. In this section we present two sets of experiments: the first one is aimed at verifying the execution time model introduced in Section 2, while the second one presents some results related to the proposed algorithm as a function of the workload. Experiments are performed on an AMD Athlon64 3000+, whose clock can be set at four frequencies: 1000, 1800, 2000 and 2200 MHz, corresponding to the following normalized speeds: 0.4545, 0.8181, 0.9090, 1. At the present stage, we were not able to measure the actual system consumption, so we could not present experimental values for the parameters of the energy consumption model (see section 2.2) or some experimental results on the actual energy saving. This issue will be addressed in the future.

4.1 Validating the execution time model

Experiments were carried out with a group of five periodic tasks with different characteristics (i.e., ϕ parameter), and the execution time of each task was estimated for all available speeds. The body of each task is composed as follow:

- **Integer (τ_1)**: 7000000 operations on integer numbers;

- **Float (τ_2)**: 90000 floating point operations and trigonometric functions;
- **Text1 (τ_3)**: 700000 integer operations mixed with the output of 9000 characters in text mode without screen scrolling;
- **Text2 (τ_4)**: same as τ_3 but with 2100000 integer operations and only 3000 characters;
- **Graphics (τ_5)**: like τ_3 but with 1500000 integer operations and 150 characters printed in graphics mode.

Results are shown in Table 1, which reports the mean execution times (in milliseconds), each obtained on 10000 task activations.

	speed			
	0.4545	0.8181	0.9090	1.0000
τ_1: Integer	2.796	1.554	1.399	1.271
τ_2: Float	2.752	1.529	1.376	1.251
τ_3: Text1	2.309	2.158	2.097	2.078
τ_4: Text2	2.727	1.794	1.678	1.582
τ_5: Graphics	2.506	1.839	1.756	1.687

Table 1. Task execution times as a function of normalized speed.

Then, the ϕ_i value of each task was computed using Equation (2) and measured values were compared against the theoretical values given by equation (1). Table 2 reports the ϕ value for each task and the computation time error of the measured value with respect to the theoretical one. It is worth observing that, even though ϕ changes from 0.0926 to 1 the relative error is less than 2%.

Figure 3 shows the measured values of the execution times on the theoretical curves given by equation (1).

Notice that, using the simplified model of a fully frequency-dependent task ($C(s) = \frac{C_{i_{max}}}{s}$), the error would be much higher. For example, the theoretical execution time of task τ_3 (Text1) running at the lowest frequency would be 4572, while the real one is 2309 (98% faster).

4.2 Experiments on the algorithm

In this experiment we tested the behavior of the DVS-elastic algorithm as a function of the workload. The load was generated using the same set of tasks described in Section 4.1, and varied by scaling the execution times to increase the maximum utilization ($U_{max}(s_{max})$) from 0.45 to 1.8. All elastic coefficients have been set to 1 for simplicity, and the desired utilization was set to $U_d = 0.9$ to take overheads into account. Figure 4 illustrates the

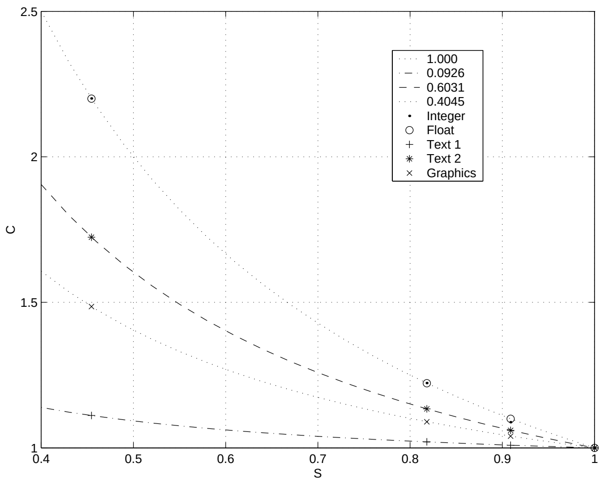

Figure 3. Comparison between actual execution times and theoretical values.

speed levels s_e^*, s_p^*, s_e and s_p computed by the algorithm as a function of load under the energy-aware and the performance-oriented mode, respectively.

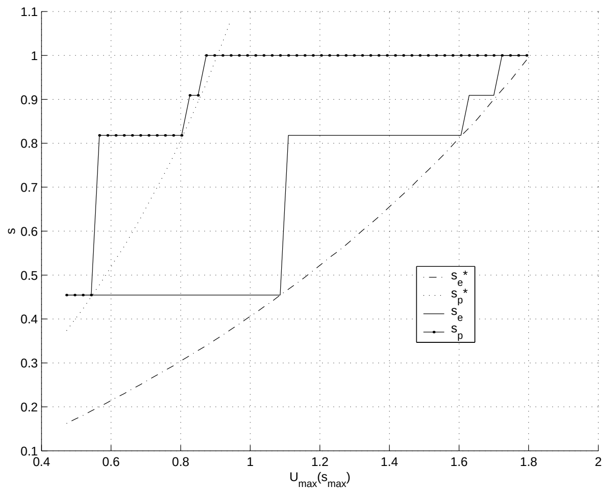

Figure 4. Speed bounds computed by the algorithm as a function of the load for the energy-aware and the performance-oriented mode.

As clear from the graphs, when the load is less than U_d, even the performance-oriented strategy is able to reduce energy consumption, by finding the minimum speed that can guarantee all the tasks at their minimum periods. On the other hand, the energy-aware strategy allows a reduction in terms of energy consumption up to an overload of about 70% (i.e., load = 1.7), when the maximum overload that the classical elastic algorithm can manage is 80% (i.e., load = 1.8), for the specific task set.

A significant improvement achieved with the integrated

	ϕ	speed						
		0.4545		0.8181		0.9090		1.000
τ_1: Integer	1.0000	2.797	0.04%	1.554	0	1.383	1.16%	1.271
τ_2: Float	1.0000	2.752	0	1.529	0	1.376	0	1.251
τ_3: Text1	0.0926	2.309	0	2.121	1.74%	2.097	0	2.078
τ_4: Text2	0.6031	2.727	0	1.794	0	1.677	0	1.582
τ_5: Graphics	0.4045	2.506	0	1.838	0.05%	1.755	0.06%	1.687

Table 2. ϕ values and estimation errors.

algorithm can be seen in Figure 4 for $U_{max}(s_{max}) = 1.1$. In fact, without elastic scheduling, the task set would not be feasible with all tasks running at their nominal periods (T_{min}), and the use of a pure energy-aware algorithm with discrete speeds and maximum periods would waste processor utilization, penalizing performance (since tasks would run at the lowest possible rate). In fact, for this value of $U_{max}(s_{max})$, the utilization factor with maximum periods is 0.62, which is a lot less than the desired value $U_d = 0.9$. Using the proposed approach, the desired utilization can be reached with a normalized speed $s = 0.8181$, and tasks can run with shorter periods computed by the elastic algorithm, so improving the application performance.

5 Conclusions

In this paper we presented an integrated approach that combines DVS techniques with elastic scheduling to improve control performance of embedded systems running on architectures with a limited number of operating modes. The task execution time model was enhanced to consider some real architecture characteristics, such the access to peripherals, whose execution is not scalable with the clock frequency.

Experimental results on an AMD Athlon64 3000+ with four operating modes showed the validity of the proposed execution model, and illustrated the advantage of the integrated approach when the objective is to maximize system performance or minimize energy consumption.

As a future work, we plan to integrate the proposed methodology with a reclaiming mechanism that will be able to take advantage of early completions to further reduce the processor speed. We also plan to apply our approach to prolong the battery lifetime of a team of mobile robot systems that need to achieve a common goal under stringent performance constraints.

References

[1] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, London, England, December 2001.

[3] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *IEEE Proceedings of the Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[4] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, March 2002.

[5] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time system development. In *Proc. 13th IEEE Euromicro Conf. on Real-Time System*, Delft, The Netherlands, June 2001.

[6] C. Liu and J. Layland. Scheduling algorithms for multi-programming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, January 1973.

[7] T. Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, 1999.

[8] T. Martin and D. Siewiorek. Non-ideal battery and main memory effects on cpu speed-setting for low power. *IEEE Transactions on VLSI Systems*, 9(1):29–34, 2001.

[9] P. Mejia Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306, May 2004.

[10] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mossé. *Power Aware Computing*, chapter Power Management Points in Power-Aware Real-Time Systems. R. Graybill and R. Melhem, Plenum/Kluwer Publishers, 2002.

[11] J. Wang, B. Ravindran, and T. Martin. A power aware best-effort real-time task scheduling algorithm. In *Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems, IEEE International Symposium on Object-oriented Real-time Distributed Computing*, May 2003.