Balancing Energy vs. Performance in Processors with Discrete Voltage/Frequency Modes *

Mauro Marinoni
University of Pavia
Email: mauro.marinoni@unipv.it

Giorgio Buttazzo
Scuola Superiore S. Anna
Email: giorgio.buttazzo@sssup.it

Abstract

Applying classical dynamic voltage scaling (DVS) techniques to real-time systems running on processors with discrete voltage/frequency modes causes a waste of computational resources. In fact, whenever the ideal speed level computed by the DVS algorithm is not available in the system, to guarantee the feasibility of the task set, the processor speed must be set to the nearest level greater than the optimal one, thus underutilizing the system. Whenever the task set allows a certain degree of flexibility in specifying timing constraints, rate adaptation techniques can be adopted to balance performance (which is a function of task rates) vs. energy consumption (which is a function of the processor speed).

In this paper, we propose a new method that combines discrete DVS management with elastic scheduling to fully exploit the available computational resources. Depending on the application requirements, the algorithm can be set to improve performance or reduce energy consumption, so enhancing the flexibility of the system. A reclaiming mechanism is also used to take advantage of early completions.

1 Introduction

In battery powered real-time systems, reducing energy consumption through Dynamic Voltage Scaling (DVS) techniques may create overload conditions that can jeopardize the schedulability of the task set. Hence, the issue of reducing energy consumption must be considered in conjunction with the one of meeting timing constraints. Moreover, in current processors, the voltage level cannot be varied continuously, but only a limited number of voltage/frequency operating modes are usually available, causing the processor to run at a speed selectable within a discrete range. In this conditions, the speed selected by the power manager will likely be different than the ideal one that could minimize some cost function, thus either timing constraints are not met or energy is not minimized.

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

The problem of minimizing energy consumption while guaranteeing real-time constraints has been widely addressed in the real-time literature. However, most of the achieved results were derived under simplified system models, where the processor, for example, can change its voltage and frequency within a continuous range. 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 with a single speed level, the processor must be set to the nearest level greater than the optimal one [17].

Other authors focused on energy aware scheduling for specific task models. The most investigated task model is the periodic one [2, 3, 10, 23], but energy-aware algorithms have also been proposed and analyzed for aperiodic tasks [21], sporadic tasks [18] and mixed task sets [22].

Recently, some authors proposed solutions for processors having discrete speed levels. Some works deal with the problem splitting the task and running each part at a different frequency [9, 23, 12]. Mejia-Alvarez et al. [16] proposed an approach where each task is assigned a different frequency; however, the problem is NP-hard, thus it can be solved on line only by a heuristic algorithm. More recently, Bini et al. [5] 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.

In order to guarantee task timing constraints, most of the algorithms for hard real-time systems perform the analysis assuming that each task executes for its worst case execution times (WCET). This is usually a strong conservative hypothesis which may cause a waste of computational resources. To exploit the additional slack coming from early completions, some authors [4, 10] proposed to mix an off-line approach based on WCETs with an on-line reclaiming method.

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 [13], in which tasks can operate within a given range of periods, with different performance. The algorithm allows the application to select energy-oriented,

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

performance-oriented, and user-defined strategies.

An on-line reclaiming mechanism is integrated in the algorithm to exploit the unused computation time resulting from early completions of the jobs. To better consider the effects of the hardware architecture on task execution times, we use an enhanced execution time model [19, 13] that splits the code in two parts: one that varies with speed and one that is speed independent. This enables a more precise representation of the application code, allowing the user to distinguish, for example, between code for pure computations and code accessing peripheral devices.

The proposed algorithm has been implemented in the Shark [8] 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 introduces the models used to describe the execution time and the energy consumption of a task; Section 3 describes the integrated DVS-elastic algorithm; Section 5 describes some experimental results; and Section 6 states our conclusions and future work.

2 Models

This section introduces the models adopted in this work to represent task execution times and power consumption. Moreover, the elastic model is also 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 $s_{max} = 1$.

If more voltage levels can be used for a given frequency, the rule adopted in this work is to select the minimum voltage level compatible with the frequency selected by the algorithm.

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

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

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} 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. [14, 15] derived the following relation to describe the power consumption as a function of the speed:

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

The K_3 term is a 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, each denoted by: $\tau_i(C_i, T_{i_{min}}, T_{i_{max}}, E_i)$. In the following, T_i denotes the actual period of task τ_i, which is constrained to be in the range $[T_{i_{min}}, T_{i_{max}}]$. Moreover, $U_{i_{max}} = C_i/T_{i_{min}}$ and $U_{i_{min}} = C_i/T_{i_{max}}$ denote the maximum and minimum utilization of τ_i, whereas $U_{max} = \sum_{i=1}^{n} U_{i_{max}}$ and $U_{min} = \sum_{i=1}^{n} U_{i_{min}}$ denote the maximum and minimum utilization of the task set.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

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 [11]. Hence, if $U_{max} \leq 1$, 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 [6] for details) that a solution can always be found if $U_{min} \leq U_d$.

As shown in [6], if Γ_f is the set of tasks with 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:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_{max}} - (U_{v_{max}} - U_d + U_f)\frac{E_i}{E_v} \quad (3)$$

where

$$U_{v_{max}} = \sum_{\tau_i \in \Gamma_v} U_{i_{max}}; \quad U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}}; \quad E_v = \sum_{\tau_i \in \Gamma_v} E_i.$$

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 (3) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if $U_{min} \leq U_d$, the iterative process ends when each value computed by equation (3) is greater than or equal to its corresponding minimum $U_{i_{min}}$.

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 consumption in systems with discrete operating modes. In the following, we assume the task set is feasible when the processor runs at the maximum speed and all tasks execute at their maximum period, that is,

$$U_{min}(s_{max}) \leq U_d \quad (4)$$

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

- **Performance oriented**: 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 1. The power manager is invoked every time a new task enters/leaves the system or a new speed is selected by the application.

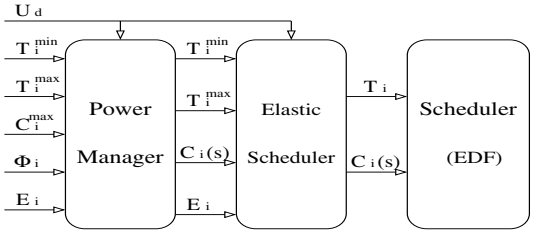

Figure 1. Block diagram of the algorithm.

3.1 Computing the frequency bounds

The algorithm starts computing a subrange of speeds $[s_e^*, s_p^*]$ in which the schedulability of the task set is guaranteed and there is no energy waste when the tasks run at their minimum periods.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

In the energy-oriented strategy, assuming a single speed is used for the whole application, the minimum theoretical speed s_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:

$$\begin{aligned} U(s) &= \sum_{i=1}^{n} \frac{C_i(s)}{T_i} \\ &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_i} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_i} \\ &= \frac{U_D}{s} + U_F \end{aligned} \quad (5)$$

where U_D is the processor utilization due to the frequency-dependent code, estimated at the maximum speed, whereas U_F is the one that is frequency independent. The minimum utilization (U_{min}) computed with the maximum periods can also be expressed as a function of speed. 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.

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,

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

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

$$s_e = \min_k \{s_k \mid s_k \geq s_e^*\}; \quad (6)$$

$$s_p = \max_k \{s_k \mid s_k \leq s_p^*\}. \quad (7)$$

Once the speeds s_e and s_p are computed and task set schedulability is guaranteed in the worst-case situation, there can be different 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, 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 $U_{max}(s_p) \leq U_d$, 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 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 computed using the speed level set by the power manager or selected by the user.

3.3 Power consumption and elastic coefficients

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 $E_i \propto P_i(s)C_i(s)$ where $P_i(s)$ is the power consumed by task τ_i, as defined in equation (2).

3.4 Online reclamation of unused bandwidth

Real-time tasks are characterized by worst-case computation times, but most jobs usually run for much less time. Our DVS algorithm takes advantage of such a saving for a further reduction of the processor speed.

When a job is released, determining its actual computation requirements is very hard (if not impossible), so the conservative assumption of the worst-case value must be used. When the job completes, it is possible to compute the saved time by accounting the actual execution. Instead of wasting this time leaving the CPU idle, the processor can be slowed down to further reduce energy consumption. To do so, each time a job is released or completed, the computation of the working speed is performed using the actual status of the task set. For this purpose 3 new variables are added to the task model:

- e_i is the execution time actually consumed by the current job of task τ_i;

IEEE COMPUTER SOCIETY

- c_{i_D} is the worst-case remaining computation time due to the frequency-dependent part of the task at the maximum speed;
- c_{i_F} is the worst-case remaining computation time related to the frequency-independent code.

Note that, under the Shark kernel, these variables can be easily handled using the Job Execution Time (JET) monitor. Moreover, at any time, $C_i(s) = e_i + c_{i_F} + \frac{c_{i_D}}{s}$. These variables are updated upon the occurrence of the following events: job release, job completion, and job preemption. When a job is released, e_i is reset to zero, while c_{i_D} and c_{i_F} are restored to their worst-case values ($\phi_i C_{i_{max}}$ and $(1-\phi_i)C_{i_{max}}$, respectively). If the job activation triggers a speed switch, e_i is increased by δ, which is the overhead to switch the speed. When a job completes, the two remaining computation times (c_{i_D} and c_{i_F}) are set to 0, while e_i is increased of δ. At a context switch, the variables are updated to account for the amount of used computation (σ). In order to maintain the worst-case approach, the algorithm first subtracts σ from c_{i_F} until its exhaustion, then starts decreasing c_{i_D} with a rate that depends on the speed. The speed-dependent code is the last to be subtracted because, in general, the amount of frequency-dependent code could vary from different jobs.

At job termination or activation, the reclaiming algorithm recomputes the working speed. Tasks periods are left unchanged to save overhead and avoid extra jitter in task activations. If all tasks terminated the current job and the CPU has enough time to switch down the frequency (the next activation is at least ahead of δ) the chosen speed is s_{min}, otherwise the minimum speed that guarantees the actual status of the task set is computed. The actual task utilization can be computed as:

$$U_i(s) = \frac{1}{T_i}(e_i + c_{i_F} + \frac{c_{i_D}}{s}).$$

So, the total utilization of the task set is expressed as:

$$\begin{aligned} U(s) &= \sum_{i=1}^{n} U_i(s) = \sum_{i=1}^{n} \frac{1}{T_i}(e_i + c_{i_F} + \frac{c_{i_D}}{s}) \\ &= \sum_{i=1}^{n} \frac{e_i}{T_i} + \sum_{i=1}^{n} \frac{c_{i_F}}{T_i} + \frac{1}{s}\sum_{i=1}^{n} \frac{c_{i_D}}{T_i}. \end{aligned}$$

Imposing $U(s) \leq U_d$, the ideal dynamic speed resulting from the reclaiming algorithm results to be

$$s^*_{dyn} \geq \frac{\sum_{i=1}^{n} \frac{c_{i_D}}{T_i}}{U_d - \sum_{i=1}^{n} \frac{e_i}{T_i} - \sum_{i=1}^{n} \frac{c_{i_F}}{T_i}}. \quad (8)$$

The chosen discrete frequency is the minimum one that satisfies the inequality, then

$$s_{dyn} = \min_k \left\{ s_k \mid s_k \geq s^*_{dyn} \right\}. \quad (9)$$

If the reclaiming algorithm was triggered by the completion of a job τ_j and the proposed speed s_{dyn} is equal to the actual one, then the frequency switch time δ is subtracted from e_j.

4 Quality of Control

In control applications the performance of a periodic control task is a function of the activation period. Increasing the task activation period leads to a performance degradation, which is typically measured through a Performance Index $J(T)$ [7, 20]. Often, instead of using the performance index, many algorithms use the difference $\Delta J(T)$ between the index and the value of the performance index $J*$ of the optimal control. Many control systems belong to a class in which the function expressing the degradation is monotonically decreasing, convex and can be approximated as $\Delta J(T_i) = \alpha_i e^{-\frac{\beta_i}{T_i}}$ where the magnitude α_i and the decay rate β_i characterize the single task. The evaluation of the whole task set is computed as

$$\Delta J = \sum_{i=1}^{n} w_i \Delta J(T_i) = \sum_{i=1}^{n} w_i \alpha_i e^{-\frac{\beta_i}{T_i}}$$

where w_i are weights used to characterize the relative importance of the tasks.

To have a common scale for all task sets, the Quality of Control index used in this paper is expressed as $QoC = \frac{\Delta J_{nom}}{\Delta J}$ where ΔJ_{nom} is the value of the index calculated when tasks run at their nominal periods. A value of 1 means that all tasks are running with nominal periods.

5 Experimental Results

To validate the proposed approach, the elastic-DVS algorithm has been implemented in the S.Ha.R.K. kernel [8] as an external scheduling module. The experiments were performed on an AMD Athlon64 3000+ with PowerNow! technology, whose clock accepts four frequencies: 1000, 1800, 2000 and 2200 MHz. Hence, the available normalized speeds are 0.4545, 0.8181, 0.9090, and 1, respectively. This section describes four experiments, each showing a different characteristic of the proposed algorithm. The first experiment shows the advantages of using the elastic task model when working on a processor with discrete clock frequencies. The second experiment is used to test how the user-selected speed s_u affects the power consumption and the quality of control. The third experiment shows how the average power can be reduced by making elastic coefficients proportional to individual task power consumption. Finally, the last experiment illustrates the advantage of using the reclaiming mechanism.

All the tasks used in the experiments have the following characteristics:

- T_{min} was generated as a random variable uniformly distributed in the range $[1ms, 100ms]$;
- T_{max} was generated as the product of T_{min} and a random number with Gaussian distribution, mean 4 and standard deviation 2;

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

- C_{max} was computed to give each task the same fraction of the total utilization when running at the full speed with nominal period: $U_{max}(s_{max}) = U_{tot}/n$;
- ϕ was generated as a random value with Gaussian distribution, mean 0.5 and standard deviation 0.4;
- All the elastic coefficients were set equal to 1.

Finally, the desired utilization was set to $U_d = 0.9$ to take overheads into account, and all the weights in the quality of control index were set to 1 to simplify the comparison in the experimental results.

5.1 Effects of the elastic task model

This experiment is aimed at evaluating the advantages of using the elastic task model with respect to an approach with fixed task periods. In a first simulation, the energy-oriented approach is applied on a set of elastic tasks with periods in $[T_{min}, T_{max}]$ and compared with the case in which all task periods are fixed and equal to T_{max}.

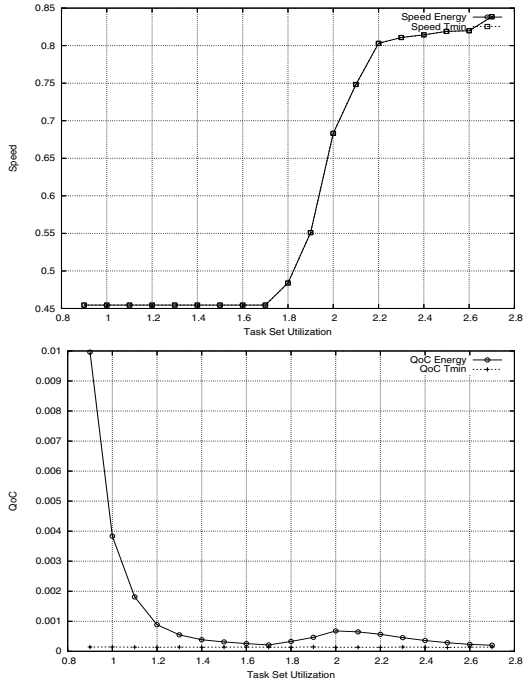

Figure 2. Comparison of the energy-oriented approach on tasks with elastic and fixed (maximum) periods.

Figure 2 shows the results in term of selected speed (upper graph) and quality of control (lower graph) when the total utilization of the task set at full speed ($U_{max}(s_{max})$) varies from 0.8 to 2.8 with a step of 0.1. For each step, the value on the y-axis represents the mean obtained on 100 task sets of 50 tasks each. We can see that, although the two algorithms select the same speed, the quality of control of the performance-oriented strategy is always higher. As expected, for high workloads, elastic tasks tend to run with larger periods, hence the QoC decrease towards the one of the fixed task set. On the other hand, when the maximum utilization of the task set is less than one, the QoC achievable by the elastic approach is significantly higher than that obtained by the fixed tasks.

In a second simulation, the performance-oriented strategy is applied on a set of elastic tasks with periods in $[T_{min}, T_{max}]$ and compared with the case in which all task periods are fixed and equal to T_{min}. As in the previous experiment, results are derived in terms of speed and quality of control and are illustrated in Figure 3. Since the performance-oriented approach also considers energy issues, the quality of control index is less than 1, where the value of 1 corresponds by definition to the performance index of the tasks running with the minimum periods. When the two algorithms select the same speed, the quality reduction in the elastic task set (with respect to the fixed set) is relatively small. On the other hand, a higher decrease in the QoC is compensated by a larger speed reduction (i.e., energy saving). Also note that, when the system is overloaded, the task set with fixed periods may not be feasible, whereas the elastic application can always be executed with degraded performance.

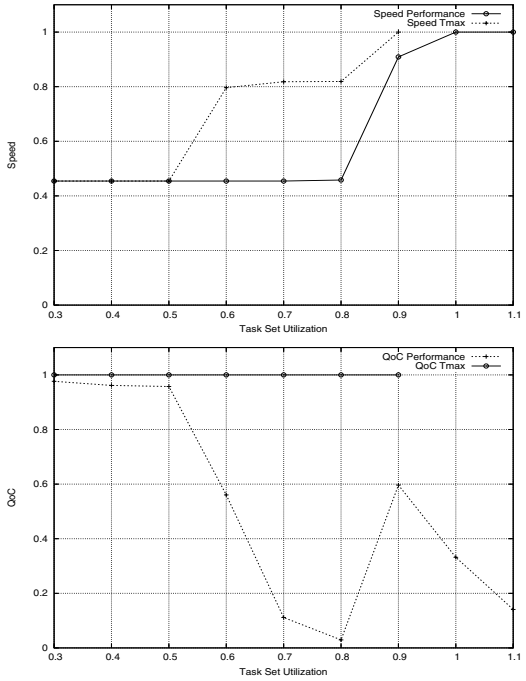

Figure 3. Comparison of the performance-oriented approach on tasks with elastic and fixed (minimum) periods.

Note that the QoC index decreases as the task set utilization grows because periods are enlarged. The spike at $U = 0.9$ is due to a speed switch from 0.4545 to 0.8181. Such a speed increase creates a slack, which is used by the algorithm to reduce periods, so increasing the QoC.

5.2 Impact of the user-defined strategy

Sometimes, the energy-oriented strategy could be too penalizing in term of performance, and the performance-

IEEE COMPUTER SOCIETY

oriented one could be too energy consuming. In this cases, the processor speed can be manually selected by the user at a level suitable for the application aims. This experiment shows the effect of the user-defined strategy on the task set behavior. The experiment has been performed on a task set with maximum utilization of 0.9 and 1.1. For each utilization, 100 task sets with a random number of tasks in the range [20,100] have been generated. For every task set, the quality of service has been computed at each speed between s_e and s_p. The results are reported in Figure 4.

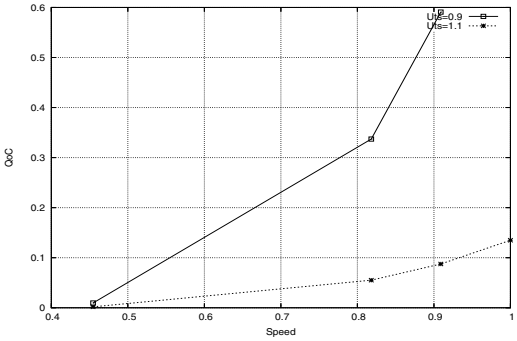

Figure 4. QoC for the user defined strategy.

The experiment shows that a trade-off between performance and energy consumption is achievable by acting on the processor speed. If the selected value is within the computed bounds s_e and s_p, the feasibility of the schedule is always guaranteed. It is interesting to observe that, in applications where the task set is static (e.g., in OSEK compliant systems [1]) the speed selection could be done automatically by a tool according to user-defined parameters, such as the minimum allowed QoC, the maximum available mean power, and so on.

5.3 Power consumption and elastic coefficients

As mentioned in Section 3.3, assigning the elastic coefficients to express job energy consumption allows to privilege tasks tasks with lower power demand. This experiment is aimed at showing how the elastic coefficients may affect the mean power consumption. To do so, the power-related assignment is compared with the one in which all elastic coefficients have the same value.

The power model is the one expressed in Equation (2), where, for the sake of simplicity, K_2, K_1 and K_0 are set to 0, and only K_3 is managed. The maximum utilization of the task set at the maximum speed ($U_{max}(s_{max})$) is 1.1 and 100 task sets (of 4 tasks) are generated for each configuration. From a configuration to the next one, the value of K_3 is increased by a value equal to the task index ($K_3^i = K_3^i + i$). For each task set, the power consumed by tasks with fixed elastic coefficients and with power-related ones is computed as

$$P = K_3 s^3 \frac{C(s)}{T}.$$

Then, the mean is computed among all task sets for the same configuration and approach. For each coefficient configuration, the comparison between the two assignments is expressed as the ratio between the power-related approach and the fixed-coefficients approach.

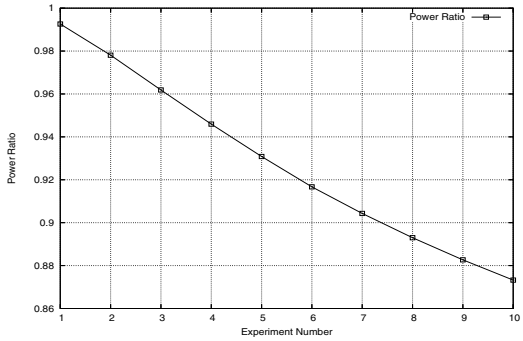

Figure 5. Power consumption ratio.

Figure 5shows that the power-related assignment of the elastic coefficients can produce a significant power saving in the application. In this example, the reduction starts from 0.8%, when the ratio between the maximum and minimum is 1.2727, to 12.7%, when the ratio grows to 4. Also note that implementing this features does not increase the runtime overhead of the acceptance test.

5.4 The online reclaiming mechanism

This experiment shows the effect of the reclaiming mechanism on the power consumption. The energy-oriented strategy was used on a task set with $U_{max}(s_{max}) = 2.0$. To generate early completions, job execution times were set equal to rC_{max}, where r was varied from 0.2 to 1. For each value of r, 100 task sets were generated and the off-line and the dynamic approaches were compared in terms of speed and power consumption. For the dynamic algorithm, speed and power consumption were computed as means over the hyperperiod. Results are reported in Figure 6, which shows the ratio between the dynamic and the off-line approach as a function of r.

When $r = 1$, the ratio is almost 1 because tasks run for their worst-case execution time, as supposed in the off-line algorithm. The real value of the ratio is less than 1 because a sporadic idle times allow the reclaiming algorithm to set the speed to s_{min}. The ratio decreases with r and reaches

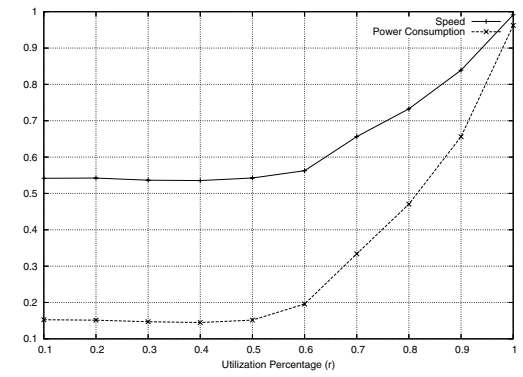

Figure 6. Speed and power ratio.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

a stable level for values of r smaller than 0.6. This is explained with the limitation imposed by the value of the minimum discrete speed (0.4545), which does not allow the algorithm to produce the desired reduction.

6 Conclusions

In this paper we presented an integrated approach that combines DVS techniques with elastic scheduling to balance control performance with energy consumption in embedded systems running on architectures with a limited number of operating modes. An enhanced task execution time model was used to consider some real architecture characteristics, such as 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, both for maximizing performance and minimizing energy consumption. Simulation experiments also illustrated the effectiveness of a reclaiming mechanism that takes advantage of early completions to perform a further reduction of the processor speed.

As a future development, we plan to use the proposed approach on a different platform which allows to obtain direct measurements of the real power consumptions, so that a full set of tests can be carried out to precisely compare our method with other related work proposed in the literature.

We also plan to apply the proposed approach to wireless mobile networks, for prolonging battery lifetime of a team of mobile robots that need to operate under stringent performance constraints.

References

[1] *OSEK/VDX standard*. URL: http://www.osek-vdx.org.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proc. of 13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of 22th IEEE Real-Time Systems Symposium*, London, England, December 2001.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, May 2004.

[5] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *Proc. 17th IEEE Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[6] 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.

[7] G. Buttazzo, M. Velasco, P. Marti, and G. Fohler. Managing quality-of-control performance under overload conditions. In *Proc. 16th IEEE Euromicro Conference on Real-Time System*, Catania, Italy, July 2004.

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

[9] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage. In *Proc. International Symp. on Low Power Electronics and Design*, pages 197–202, Monterey, California, USA, August 1998.

[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.*, pages 219–228, San Jose, California, September 2002.

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

[12] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron. Procrastinating voltage scheduling with discrete frequency sets. In *Proc. of Design Automation and Test in Europe (DATE)*, Munich, Germany, March 2006.

[13] M. Marinoni and G. Buttazzo. Adaptive dvs management through elastic scheduling. In *Proc. 10th IEEE Conf. on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005.

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

[15] 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.

[16] 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.

[17] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. 18th ACM Symp. on Operating System Principles*, pages 89–102, Banff, Canada, October 2001.

[18] A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proc. 24th IEEE Real-time Systems Symposium*, pages 52–62, Cancun, Mexico, December 2003.

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

[20] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Proc. 17th IEEE Real-Time Systems Symposium*, pages 13–21, Washington DC, USA, December 1996.

[21] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadrin. Power-aware qos management in web servers. In *Proc. 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[22] D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In *Proc. Conf. on Asia South Pacific design automation: electronic design and solution fair*, pages 653–658, Yokohama, Japan, January 2004.

[23] Y. Zhu and F. Mueller. Feedback edf scheduling of real-time tasks exploiting dynamic voltage scaling. *Real-Time Systems Journal*, 3(1):33–63, December 2005.