Feedback Scheduling of Power-Aware Soft Real-Time Tasks

Alberto Soria-Lopez, Pedro Mejia-Alvarez, Julio Cornejo

CINVESTAV-IPN. Sección de Computación, Departamento de Control Automático
Av. I.P.N. 2508, Zacatenco. México, DF. 07300
\{pmejia, jcornejo\} @cs.cinvestav.mx, soria@ctrl.cinvestav.mx

Abstract

In this paper we propose an energy-based feedback control scheduling framework for power-aware soft real-time tasks executing in dynamic environments, where real-time parameters are not known a priori. We propose the use of a controller based on an Energy Savings Ratio, which allows higher energy savings when higher miss deadlines are allowed to occur.

The scheduler uses the energy feedback to calculate the amount of workload to be adjusted and provides the input for the variable voltage optimization algorithm (VVA). The VVA algorithm is a greedy algorithm that adjusts the workload to optimize power consumption by computing a near optimal solution for the tasks’s voltage/speed selection problem.

Extensive set of tests are executed to simulate the performance of our energy feedback scheduling power-aware architecture under overload and underload conditions. In this simulations we used RMS and EDF as the scheduling policies. Simulation results show that the proposed architecture is capable of handling real-time tasks with unknown arrivals and execution times, and derive a system in which power savings are maximized.

1 Introduction

Energy management is becoming the limiting factor for the functionality of real-time embedded and portable devices because advances in battery technology are progressing slowly whereas computation and communication demands of this devices are increasing rapidly. Energy minimization is critically important for devices such as laptop computers, PCS telephones, PDA’s and other mobile and embedded computing systems simply because it leads to extended battery lifetime. However, the need for power-efficient designs is not solely associated with portable computing systems. Power dissipation has become a design constraint in virtually every type of computing system including desktop computers, network routers and switches, set-top entertainment systems and the most performance hungry computer servers. Several applications of embedded and portable real-time systems contain hard and soft real-time constraints and a strict limit on available battery power. A complicating factor for these applications is that tasks or service requests are often unpredictable, and current scheduling techniques are not adequate for power aware real-time computing.

Scheduling policies such as EDF or RMS [16] are capable of handling task sets with sophisticated timing parameters such as, aperiodic task arrivals, precedence constraints, shared resources, or execution on distributed environments. However, it is known that the execution of these scheduling policies is performed using an open loop [32]. The term open loop refers to the fact that once a task set is scheduled, their parameters (i.e., task’s execution time, period or deadlines) can not be adjusted in response to workload variations [32]. Open loop scheduling algorithms provide efficient performance guarantees in predictable environments, where the workload does not experience changes and can be precisely modeled. However, those real-time systems achieve poor performance in unpredictable environments, where the workload experience frequent changes. Another important issue is that open loop schedulers are often designed based on worst-case behavior of real-time tasks. This is an
appropriate solution for real-time systems with statically known time properties, but not for battery operated real-time systems where workloads are variable and the system have various degrees of unpredictability. The results obtained from this over-provisioning is a system with low performance and low utilization.

In this work, we introduce a feedback scheduling architecture for power-aware real-time tasks. The main parts of this architecture are an energy-based feedback scheduler and a power-aware optimization algorithm. The feedback scheduler attempts to keep the CPU utilization at high level, achieving high energy savings, and distribute the computing resources among real-time task to maximize control performance. The energy savings is used as the control variable in the system configuration.

The feedback scheduler, based on a P (proportional) controller, calculates the amount of workload to be adjusted and provides the input for the variable voltage optimization algorithm (VVA) [22]. The VVA algorithm is a greedy algorithm that computes a near optimal solution for the voltage (speed) selection problem. The VVA algorithm adjust the workload by selecting a set of speeds for execution of each task in the system. The process of selecting the execution speeds while minimizing the energy consumption in the system is the main goal of the VVA algorithm. The feedback scheduling architecture accepts workloads that exhibit a large variability in their workload and execute on a processor capable of handling several (discrete) speeds of execution.

The rest of this paper is organized as follows. In Section 2 related models and related work is reviewed. In Section 3, the system and energy models used are defined. In Section 4, the power-aware scheduling is formulated as an optimization problem. In Section 5 the feedback scheduling power-aware architecture is described and in section 6, the control task model is defined. In Section 7 simulation results are presented to show the performance of our framework. Finally, in Section 8 concluding remarks are presented.

2 Related Work on Variable Voltage Scheduling

Broadly speaking, there are two methods to reduce power consumption of processors through OS-directed energy management techniques. The first is to bring a processor into a power-down mode, where only certain parts of the computer system such as the clock generation and the timer circuits are kept running when the processor is in idle state. Most power-down modes have a trade-off between the amount of power savings and the latency overhead incurred during mode changes. For an application that cannot tolerate latency, as those in real-time systems, the applicability of power-down modes is limited. The second method is to dynamically change the speed of a processor by varying the clock frequency along with the supply voltage. Power Reduction via variable voltage scheduling (VVS) can be classified as static and dynamic techniques. Static techniques, such as static scheduling, compilation for low power [23] and synthesis of systems-on-a-chip [12], are applied at design time. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving dynamically arriving real-time tasks, light workloads or the system is idle.

Static (or off-line) scheduling methods to reduce power consumption in real-time systems were proposed in [35, 10]. These approaches address task sets with a single period and aperiodic tasks. Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of off-line periodic requests are proposed in [11]. Non-preemptive power-aware scheduling is investigated in [10]. Recent work on VVS includes the exploitation of idle intervals in the context of the Rate Monotonic and Earliest Deadline First (EDF) scheduling frameworks [29, 14, 6, 17]. Following the same VVS technique, other work [24, 26] consider the fact that some real-time tasks not always execute their worst-case execution times, and have the ability to dynamically reclaim unused computation time to obtain additional energy savings.

Most of the above research work on VVS assumes that all tasks have identical power functions. Using an alternate assumption, efficient power-aware scheduling solutions are provided where each real-time tasks have different power consumption characteristics [5, 9]. A recent survey of dynamic VVS methods, considering continuous voltage has shown the performance of several algorithms [30].

Although systems which are able to operate on an almost continuous voltage spectrum are rapidly be-
coming a reality thanks to advances in power-supply electronics [7], it is a fact nowadays that most of the microprocessors that support dynamic voltage scaling use *discrete* voltage levels. Some examples of processors that support discrete voltage scaling are: (a) the Crusoe processor [34] which is able to dynamically adjust clock frequency from 200 to 700 MHz and from 1.1 V to 1.6 V, in 33 MHz steps; (b) the ARM7D processor [13] which can run at 33MHz and 5V as well as at 20MHz and 3.3V; (c) the Intel StrongARM SA1100 processor, which supports 11 clock speeds: 59-221 MHz in 14.7 MHz Steps [13]; and (d) the line of the Intel XScale architecture, the PXA250 supporting 3 speeds (200, 300 and 400MHz).

In the realm of control techniques used in the scheduling of real-time systems, the related work is divided in the following categories including real-time scheduling [8]: integrated control and multimedia (quality-of service), control and power-aware scheduling, and integrated control and computing applications.

In [28] sampling period selection for a set of control tasks is considered. The desired performance of a task is given as a function of its sampling frequency, and an optimization problem is solved to find the set of optimal task periods. The idea of using feedback scheduling in real-time systems has been proposed in [18], with the introduction of their FC-EDF (Feedback controlled EDF scheduling algorithm). A proportional controller regulates the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their CPU consumption. Several versions of each tasks are defined (which provide different quality of services), and the problem to solve is to maximize the quality of service provided by the set of tasks. The work presented in [18] has derived many other research work applying control theory in several case studies involving computing applications. The case studies are, Internet web servers [2], proxy cache relative hit ratio [19], microprocessor thermal management and real-time distributed systems [31]. In [15] a QoS framework is proposed for controlling the applications requests for system resources using the amount of allocated resources for feedback. It is shown that a *PID* controller can be used to bound the resource usage in a stable and fair way. In [1] tasks models suitable for multimedia applications are defined. A PI controller is used to adjust a portion of the CPU bandwidth. Control Theoretical approaches for QoS adaptation are also presented in [3].

The work presented in [20] describes a control theoretical dynamic frequency scaling for real-time multimedia workloads. This paper describes a formal feedback-control algorithm for dynamic voltage/frequency scaling in a portable multimedia system. Their aim is to save energy while maintaining a desired playback rate, in a MPEG portable multimedia playback system. The work of [27] describes a power-aware video decoding system for multimedia applications. This paper presents a feedback controller for video decoding that regulates the voltage for individual frames.

One of the main motivations of our paper, is that there is very few research work related to power-aware real-time scheduling using control theory techniques. The work in [25, 33] introduces a feedback controlled discrete VVS scheduling algorithm for periodic hard real-time tasks under the EDF scheduling policy. A PID controller is used to control incrementally the systems behavior to achieve its targets, while preserving the hard-real time requirements. Under this framework tasks worst-case execution times are known and their VVS algorithm is capable of dynamically reclaiming unused computation times of the tasks and idle times in the schedule to obtain additional energy savings.

To our knowledge, no previous work have considered the scheduling of power-aware unpredictable real-time workloads where a given number of deadline misses are allowed.

### 3 Task Model

In our framework we consider periodic preemptive and soft real-time tasks running on one processor. Tasks are independent and have no precedence constraints. The arrival time $b_i$ of task $\tau_i$ is unknown. The *life-time* of each task $\tau_i$ consists of a fixed number of instances $r_i$. After the execution of $r_i$ instances, the task leaves the system. We denote by $C_i$ the worst-case number of CPU processor cycles required by $\tau_i$. Under a constant speed $V_i$ (given in cycles per second), the worst-case execution time of $\tau_i$ is $\hat{c}_i = \frac{C_i}{V_i}$.

---

1 We assume that some tasks that leave the system may return at a later time.
The period of task $\tau_i$ is $P_i$ which is equal to the relative deadline ($D_i$) of the task. We assume that the tasks characteristics (e.g., $C_i$, $P_i$ and $D_i$) are known at arrival time of the tasks. For $C_i$, only its worst-case value its known at arrival time, but its value varies at each instance. The real (measured) value of $\hat{c}_i$ will be considered as $a_i$, considering $0 \leq a_i \leq \hat{c}_i$.

A schedule of periodic tasks is feasible (no task misses its deadline), if each task is assigned at least $C_i$ processor cycles before its deadline, at every instance. In our model, an specific number of deadlines misses may be allowed to occur. The utilization of a task is the amount of processor load that the task demands for execution: $U_i = \frac{C_i}{P_i}$ (or $\frac{C_i}{V_i P_i}$). We will use the RMS and EDF scheduling policies [16].

We assume that the CPU speed of any task can be changed at discrete levels between a minimum speed $V_{\min}$ (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed $V_{\max}$. $V_{ij}$ denotes the speed of execution of an instance of task $\tau_i$ when executes at speed $j$, where $V_{\min} \leq V_{ij} \leq V_{\max}$. The utilization of task $\tau_i$ when executes at speed $j$ is denoted by $U_{ij} = \frac{C_i}{V_{ij} P_i}$.

The power consumption of the task $\tau_i$ is denoted by $g_i(V)$, assumed to be a strictly increasing convex function, specifically a polynomial of at least second degree[7]. $g_i(V) \approx (K f V^2)$, where $K$ is the output capacitance, and $f$ is the frequency of the clock (its exact form depends on the technology used)[12].

If the task $\tau_i$ occupies the processor during the time interval $[t_1, t_2]$, then the energy consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(V(t)) dt$.

For each task $\tau_i$ in the system we define a set of speeds of execution which will be called class $N_i$. The size of $N_i$ depends on the number of discrete voltages that the processor supports. The energy consumption ratio of task $\tau_i$, when executing at speed $j \in N_i$ is computed by

$$E_{ij} = \frac{e_{ij}}{e_{i1}} \quad (1)$$

where $e_{ij}$ denotes the energy consumption of task $\tau_i$ executing at speed $j$, and $e_{i1}$ is the energy consumed by task $\tau_i$ executing at its maximum speed.

The energy savings ratio of task $\tau_i$, when executing at speed $j \in N_i$, is computed by

$$E_{sij} = (1 - \frac{e_{ij}}{e_{i1}}) \times 100 \% \quad (2)$$

We assume that all items $j \in N_i$ follow a non-decreasing order. Each task $\tau_i$ in the system accrues an accumulated energy consumption $e_i(k)$ upon executing a number of instances during the interval of time between $[(k-1)W, kW]$, where $k$ is defined as the loop sampling time and $W$ is defined as the loop sampling period. $e_i(k)$ denotes the amount of energy consumption accrued by all the tasks in the system during $[(k-1)W, kW]$, that is,

$$e_i(k) = \sum_{j=1}^{n} e_{ij}(k) \quad (3)$$

The maximum energy consumption of task $\tau_i$ during sampling time $k$, denoted as $e_{i \max}(k)$, is achieved when the task executes at its maximum speed, $j = 1$.

Each task $\tau_i$ in the system accrues an accumulated energy savings ratio $E_{s_i}(k) = (1 - \frac{e_i(k)}{e_{i \max}(k)}) \times 100$, during $[(k-1)W, kW]$. $E_i(k)$ denotes the amount of energy savings ratio accrued by all the tasks in the system during sampling time $k$, that is,

$$E_i(k) = \left(1 - \frac{e_i(k)}{e_{i \max}(k)}\right) \times 100 \quad (4)$$

where $e_{i \max}(k) = \sum_{j=1}^{n} e_{ij \max}(k)$ denotes the maximum energy savings accrued by all tasks in the system during sampling time $k$. Note that all instances of task $\tau_i$ will execute at the same speed $j$ during the sampling interval.

Each task $\tau_i$ executing at speed $j$ will have a utilization:

$$U_{eij} = \frac{\hat{c}_{ij}}{P_i} \quad (5)$$

where $\hat{c}_{ij} = C_i/V_{ij}$ denotes the worst-case execution time of task $\tau_i$ when executing at speed $j$. The worst-case execution time of task $\tau_i$ is computed by considering the worst-case number of CPU cycles of the task, $C_i$. The worst-case utilization of the task set is denoted by $U_e = \sum_{i=1}^{n} U_{eij}$. The worst-case utilization for the sampling time $k$ is given by

$$U_{e}(k) = \sum_{i=1}^{n} \frac{\hat{c}_{ij}}{P_i} \quad (6)$$
The measured utilization $U_m(k)$ of all tasks over sampling time $k$ is computed by,

$$U_m(k) = \frac{\sum_{i=1}^{n} A_{ij}(k)}{W} \quad (7)$$

$A_{ij}(k)$ denotes the real (measured) execution time of task $\tau_i$ over the sampling period, while executing at speed $j$. Note that $A_{ij}(k)$ is the sum of execution times of all instances of task $\tau_i$ over the sampling period. The number of instances of task $\tau_i$ in the sampling period is $M = [\frac{W_i}{I_i}]$. For different instances, the real execution time of task $\tau_i$, $A_{ij}(k)$ may vary.

The miss deadline ratio $Mr(k)$ of all tasks over sampling time $k$ is computed by,

$$Mr(k) = \frac{\text{Number of miss deadlines in sampling time } k}{\text{Total number of instances in sampling time } k} \quad (8)$$

In our implementation, we make the following additional assumptions:

- **the scheduler will be capable of adjusting the workload** (by changing the speeds of each task) at task arrivals, task departures and at the end of each sampling period $W$.

- **the time overhead associated with voltage switching is negligible.** According to [34] the time overhead associated with voltage switching in the Transmeta Crusoe microprocessor is less than 20 microseconds per step. The worst-case scenario of a full swing from 1.1 V to 1.6 V takes 280 microseconds. In case that there is an imposing bound on the rate of voltage change, it is assumed that the voltage change overhead can be incorporated in the worst-case workload of each task.

- **different tasks have different power consumptions.** This assumption is based in the real-life fact that the power dissipation is dependent on the nature of the running software of each task in the system. This assumption is justified taking into consideration the following examples: some tasks use more of the memory system (in addition to the cache), some tasks use the floating point unit more than others, and some ship the tasks to specialized processors (e.g., DSPs, micro-controllers, or FPGAs).

The following example is defined to clarify the framework used by our task model.

**Example 1.** Suppose that we have 3 tasks, $\tau_1 = (C_1 = 800, P_1 = 2000)$, $\tau_2 = (C_2 = 400, P_2 = 2000)$, and $\tau_3 = (C_3 = 200, P_3 = 2000)$ with four instances, each executing during the sampling interval $[(k-1)W, kW]$. Note that in this case, $W = 8000$. Assume that the tasks execute following a Rate Monotonic scheduling policy where task $\tau_1$ has the highest priority and $\tau_3$ the lowest priority. The workload described was computed assuming a maximum speed of 1.0 for each task, obtaining a total load of $\frac{800+400+200}{2000} = 70\%$. In this example a power consumption function $g_i(V) = kV^2$ is considered, where $k=1$. The energy consumption is computed as $E = \sum_i g_i(V)I_i$, where $I_i = \frac{C_i}{V}$ is the interval of time that some task is executing.

Suppose that, if executed with maximum speed (speed = 1.0), for one sampling interval, the tasks would have the following energy consumption, under a worst-case workload, $e_1^{max}(k) = 3200$, $e_2^{max}(k) = 1600$, $e_3^{max}(k) = 800$ and $e^{max}(k) = 5600$.

Suppose that tasks execute with a varying workload (lower than their worst-case) between instances. That is, the number of processor cycles for all instances of each task is,

$C_{11} = 800, C_{12} = 600, C_{13} = 500, C_{14} = 800; C_{21} = 200, C_{22} = 200, C_{23} = 150, C_{24} = 400; C_{31} = 200, C_{32} = 50, C_{33} = 150, C_{34} = 100$

Also, suppose that the power-aware scheduler decides to execute these tasks with speed $V = 0.7$.

The actual energy consumption of different instances (i.e., $e_{ij} = (C_{ij}/V) \ast (V)^2$) in this case, are as follows.

For $\tau_1: 560, 420, 350, 560$.
For $\tau_2: 140, 140, 105, 280$.
For $\tau_3: 140, 35, 105, 70$.

In this case, since $\sum_{j=1}^{3} A_{ij} = 4150/0.7 \approx 5928.57$, $U_m(k) = 5928.57/8000 \approx 0.74$. no tasks misses its deadline.

$e_1(k) = 1890$, $e_2(k) = 665$, $e_3(k) = 350$, where $e_i(k) = (\sum_j C_{ij}/0.7) \ast (0.7)^2$.

$e(k) = e_1(k) + e_2(k) + e_3(k) \approx 2905$.

$E_s(k) = (1 - e(k)/e^{max}(k)) \ast 100 \approx (1 - 2905/5600) \ast 100 \approx 51.87\%$.

Suppose now that the power-aware scheduler decides to execute these tasks with speed $V = 0.5$. 

In this case, the energy consumption of different instances, are as follows.
For $\tau_1$: 400, 300, 250, 400.
For $\tau_2$: 100, 100, 75, 400.
For $\tau_3$: 100, 25, 75, 50.
$\sum_{i=1}^{3} A_{ij} = 4150/0.5 = 8300$.
$e_1(k) = 1350$, $e_2(k) = 475$, $e_3(k) = 250$, where $e_i(k) = (\sum_j C_{ij}/0.5) \times (0.5)^2$.
$e(k) = e_1(k) + e_2(k) + e_3(k) \approx 2075$.
$E_s(k) = (1 - e(k)/e^{max}(k)) \times 100 = (1 - 2075/5600) \times 100 \approx 62.94\%$.
Note that in this case, since $U_m(k) = 8300/8000 \approx 1.0375$, instances $\tau_{2.4}$, $\tau_{3.1}$ and $\tau_{3.4}$ miss their deadlines, therefore, $Mr(k) = 3/12 = 25\%$.

4 Formulation of the Problem

The problem to be solved in our power-aware real-time framework is to efficiently select voltages/speeds of execution in an unpredictable environment, where a percentage of deadline misses are allowed, such that the the energy savings are maximized.

In a dynamically varying power-aware real-time system, simply committing to the maximum CPU speed and (predictively) changing modes (shutting down the processor or other devices) during idle intervals, despite of its simplicity, yields sub-optimal results with respect to power consumption, in view of the convex power/speed relation. Additionally, since real-time systems are often unpredictable (advance knowledge of the timing characteristics of tasks is not always available), optimal solutions (with zero deadline misses) are sometimes difficult to attain. Furthermore, in critical mission-oriented applications, some tasks may allow some deadline misses with the purpose of saving additional energy and thus extending the life of the battery.

To be able to adjust the workload and react to a dynamically varying real-time environment the FPAS scheduler must control and monitor, as frequent as possible, different parameters of the workload.

This problem is formulated as the following power-aware optimization problem. At each task arrival or departure and the end of each sampling period $W$, the feedback scheduler must find a set of speeds $x$, for the execution of each task such that the energy savings of the system are maximized.

That is,

$maximize \quad Z = \sum_{i=1}^{n} \sum_{j \in N_i} S_{ij} x_{ij} \quad (9)$

$subject\ to \quad U_e(k + 1) \leq U_e(k) + \Delta U(k) \quad (10)$

$x_{ij} = \begin{cases} 1 & \text{if speed } V_{ij} (j \in N_i) \text{ is selected for task } \tau_i \\ 0 & \text{otherwise} \end{cases}$

Condition 10 (workload limit) indicate that the worst-case utilization $U_e(k + 1)$ in the next sampling period $(k + 1)$, must be less or equal than the utilization in the actual sampling period $U_e(k) + \Delta U(k)$. If $\Delta U(k)$ (the adjustment on the workload, provided by the controller) is a positive number, it will indicate that additional workload can be accepted in the system, and hence execution speed of the tasks should be decreased. In the other hand, if $\Delta U(k)$ is a negative number, it will indicate that the workload must be reduced, and hence the execution speed of the tasks should be increased. In both cases, our Feedback scheduling power-aware architecture will try to minimize energy consumption.

Since $U_e(k)$ denotes a worst-case utilization, the measured utilization $U_m(k)$ will always be $U_m(k) \leq U_e(k)$. However, $U_m(k) \leq 100\%$ but $U_e(k)$ may be greater than $100\%$. Note that if $U_e(k) > 100\%$ it will not necessarily imply that the task set is suffering an overload, because $U_e(k)$ denotes a worst-case value.

Note that in our formulation, the energy savings are not linked with the Miss Ratio. However, the higher value of $Mr(k)$ will allow higher energy savings.

The feedback scheduler is able to adjust the workload only at new task arrivals and departures and at the end of each sampling period. In any case, because of the high variability of the real-time workload it is possible that some overload occur, during the last sampling period, and that some tasks may miss their deadlines, before the workload is adjusted to meet our optimality criteria.
5 Feedback Scheduling Power-Aware Architecture

In order to meet our optimality criteria, the feedback scheduler adjusts the load of the system by controlling the speed of execution of the tasks. The architecture proposed to solve the problem is illustrated in Figure 1, and described in detail in the following subsections.

5.1 Task Arrivals and Execution

The inputs to our system are real-time tasks that arrive in the system at unknown times. The feedback scheduler contains an acceptance mechanism which decides whether or not the arriving tasks can be accepted in the system. The acceptance mechanism is based on the following condition,

If all tasks in the system are already executing at its maximum speed levels, and the measured utilization \( U_m(k) \) is at 100%, then no new tasks will be allowed to enter in the system. Otherwise, the arriving tasks will be accepted in the system.

Due to the high variability of the real-time workload, \( U_m(k) \) will be subject to frequent changes.

Once a task is accepted, it will be sent to the ready task queue of the scheduler and the variable voltage algorithm (VVA) will execute to compute a new set of speeds for each task in the system. While the VVA algorithm is computing the set of speeds (by solving the optimization problem), it will consider the most recently used workload limit (condition 10 of the optimization problem), \( U_e(k) \).

Finally, the dispatcher will select for execution the tasks with highest priority according to a pre-defined scheduling policy (RMS or EDF[16]).

Once the dispatcher selects the task \( \tau_a \) for execution, it will execute at the speed \( V_{aj} \) selected by the VVA Algorithm. While task \( \tau_a \) is executing it will change its speed only until the next sampling period. When a task leaves the system, VVA will execute to re-calculate the speed of the remaining tasks in the system.

5.2 Feedback Control Loop

The feedback scheduler uses a closed control loop[4, 18]. The control law is evaluated at the end of each sampling period. The control law includes a monitor for energy savings ratio, a Proportional controller and a feedback real-time scheduler which includes the VVA algorithm, and a dispatcher.

1. Energy Monitor. The energy monitor verifies the energy savings ratio, \( E_s(k) \) (controlled variable). While tasks are executing, the feedback scheduler will record the values of the controlled variable at each sampling period \( k \).

2. Proportional Controller. The Proportional Controller [4] (PC) compares the control variables against a set point data (previously computed) and produces...
an error. This error denotes the difference between the set point data and the controlled variable. At each sampling period, the PC computes the amount of workload that is necessary to adjust in order to reduce the error as much as possible. The workload change computed by the PC in $Ue(k) = Ue(k + 1) = Ue(k) + \Delta U(k)$. The PC compensates for workload variations (using its control function), while keeping the controlled variable as close as possible to the set point value (performance reference value). The PC controller transfers the workload adjustment $\Delta U(k)$, to the variable voltage algorithm, for computing a new set of speeds for workload adjustment (performance reference value). The PC controller transfers the workload adjustment $\Delta U(k)$, to the variable voltage algorithm, for computing a new set of speeds for the tasks in the system. Hence, the output of the controller is $\Delta U(k)$.

The PC controller used in our feedback scheduling framework supports the following control variables:

- **Controlled Variable.** This is the variable controlled by the feedback scheduler to obtain the required performance of the system. The controlled variable is computed at the end of the sampling period. In our case we use the energy savings ratio $Es(k)$ as our controlled variable. This controlled variable is defined over the sampling period $[(k - 1)W, kW]$.

- **Performance Reference.** The performance reference denote the performance required for the system to function according to a set point data. The performance reference used in our framework is the target energy savings ratio, $Es_r$. For example, a value of $Es_r = 40\%$ will indicate that the PC must try to change (reduce or increase) the workload so as to reach the target value provided and to reduce the error to zero. The error in the energy savings ratio, $Es(k)$ is computed by $E_{error} = Es_r - Es(k)$, that is the difference between the target energy savings and the measured energy savings.

- **Manipulated Variable.** This is a system parameter that the controller can change dynamically. The change produced can affect the controlled variable. In our framework, the manipulated variable is the worst-case utilization $Ue(k)$. Usually $Es(k)$ increases when $Ue(k)$ increases. Note that $Ue(k)$ may be higher than 100%.

### 3. Variable Voltage Algorithm.
The dynamic voltage scaling algorithm (VVA) is capable of adjusting the speed of execution of each task in the system, according to the workload adjustment $\Delta U(k)$ provided by the PC controller. The VVA changes in consequence the worst-case utilization $Ue(k)$ on each sampling period. The change in $Ue(k)$ is performed by changing the speeds of execution of the tasks in the system.

As stated in the optimization problem, the objective of the VVA algorithm is to adjust the $Ue(k)$ workload to $Ue(k + 1) = Ue(k) + \Delta U(k)$. The process of selecting speeds for execution while maximizing the energy savings of the system requires the exploration of a large number of combinations, which is too time consuming to be computed on-line. In order to solve our optimization problem, we propose to use our previously developed Variable Voltage Scheduling Algorithm (VVA) [22] which solves the power-aware optimization procedure in a low computation time. This VVA allows the feedback scheduler to handle power-aware real-time tasks with low cost while maximizing the energy savings of the system. The objective of the VVA is to solve the power-aware optimization problem described by Equations 9 and 10. This optimization problem is formulated as a multiple-choice knapsack problem (MCKP) with binary variables [21]. Its solution is based on the optimization procedure of the PORTS Scheduling Server [22].

The optimization procedure used to solve our problem consists of three parts:

1. A *reduction algorithm*, which converts the original MCKP to a standard KP.
2. An *approximation algorithm* (e.g. Enhanced Greedy Algorithm) capable of finding an approximate solution to the reduced KP, and
3. A *restoration algorithm*, which re-constructs the solution of the MCKP from the solution of the standard KP.

The optimization procedure, proposed in [22] is based on the reduction of the MCKP to the equivalent KP using the convex hull concept [21].
6 Control Task Model

In this section, we will formulate the mathematical model [4] of the power-aware architecture shown in Figure 1. The controlled system shown in the dotted rectangle of Figure 1 includes a variable voltage algorithm (VVA) and a real-time scheduler.

Despite the difficulty to model and control non-linear systems, such as the power-aware real-time system proposed in this paper, for design purposes, the system will be considered linear.

The mathematical model proposed is shown in Figure 2. Our goal in the modeling is to find the transfer function (in the $z$ domain) used by the system to transform the input (change in requested utilization $\Delta U(k)$) to its corresponding output (energy savings ratio, $Es(k)$).

Starting from the control input $\Delta U(k)$, the total utilization $Ue(k)$ represent the integration of the control input. $Ue(k)$ is computed by,

$$Ue(k + 1) = Ue(k) + \Delta U(k)$$ (11)

Since task execution times are unknown and time variant, the energy savings ratio $Es(k)$ computed will be affected by function $G_S(k)$. This function represents the workload variation and is difficult to model for the following reasons,

- **unpredictability on the execution time of the tasks**: tasks may have different execution paths, because of the structure of the tasks code (e.g. if, else, case, for, while, etc).
- **unpredictability caused by voltage/speed changes on the tasks**: it is not known a priori the voltages computed by the VVA algorithm for the execution of the tasks. This causes an additional unpredictability on the execution time of the tasks.
- **workload variation**: tasks arrivals are unknown and execution time between instances may vary. This may cause variations on the estimated utilization $Ue(k)$.

Given that $G_{Es}(k)$ is time-variant, we can use the worst-case value of $G_{Es} = \max\{G_{Es}(k)\}$. $G_{Es}$ is defined as the Gain obtained for the worst-case utilization ratio. The value of $G_{Es}(k) = 1.1$ was obtained from experimental data.

The transfer function from $\Delta U(k)$ to $Es(k)$ is given by,

$$\frac{G_S}{z - 1}$$ (12)

6.1 Energy Savings Control Algorithm

At each sampling period, the controller computes the control signal $\Delta U(k)$ based on the Energy Savings Error, $E_{error}(k) = Es_{r} - Es(k)$.

In this paper we use a proportional controller that is capable of achieving zero steady state error given the system model that includes an integrator. The closed loop transfer function using the controller with gain $K_p$ is,

$$\frac{K_pG_S}{z - (1 - K_pG_S)}$$ (13)

The controller gain $K_p$ allows to place the only pole $1 - K_pG_S$ for this transfer function. In the z-plane, the stability boundary is the unit circle $|z| = 1$. The system is stable when all poles are located inside the unit circle and unstable when the pole is located outside [18]. We choose a value of $K_p = 0.9091$ that allows a zero overshoot and guarantees the stability of the control loop.

7 Simulation Experiments

The following simulation experiments were designed to test the performance of our Feedback Power-Aware Scheduling Scheduler (FPAS). The feedback scheduler will emulate the execution of a variable speed processor.

The goals in this simulation experiments are:
1. to measure the quality of the results over a large set of dynamic tasks that arrive at the system at arbitrary instants of time.

2. to measure the performance of our FPAS algorithm under overload and underload conditions.

3. to measure and compare the performance and run-time of our Feedback Power-Aware Scheduler against an Open Loop Control.

A software simulator was developed on a Pentium IV (3 GHz) PC, under the Linux Red Hat 8.0 Operating System. This simulator allows modifications on the control law, scheduling policies, control parameters or task generation methods, and is comprised of the following four modules,

- **Tasks Generation**: randomly generates tasks with their associated timing parameters ($C_i$, $P_i$, $D_i$, and tasks arrival times $b_i$).

- **Feedback Real-Time Scheduler**: Simulates task executions following a pre-defined scheduling policy. Also, is capable of monitoring the controlled variables, $E_s(k)$, $U_m(k)$ and $M_r(k)$ for the P controller.

- **P Controller**: Calculates the required utilization change $\Delta U(k)$. The control parameters and controller gains for the FPAS controller are computed according to the values described in Table 1. In Section 6.1 we defined how the control parameters for the FPAS are computed. These controller parameters were computed based on [18].

- **Open Loop Control**: It simulates task executions following a pre-defined scheduling policy. In the Open Loop (OL) controller, tasks are executed at a fixed speed for the complete duration of each simulation, and its resulting $E_s(k)$, $U_m(k)$ and $M_r(k)$ are measured.

- **Variable Voltage Algorithm**: Computes the execution speeds for each tasks. The FPAS Scheduler uses a near-optimal approximation algorithm (VVA) [22] described in Section 5.2.

The results obtained from our simulations are shown in Figures 3 and 4. Each plot in the graphs represents the average of 1000 simulation runs for the FPAS and Open Loop controllers. The total simulation time was 200 seconds. In this simulations we assume a set of speeds of executions as in the Crusoe TM5400 Processor[34], with normalized values in the interval [0.1]. The normalized values of $V_{ij}$ simulated are: { 0.285, 0.333, 0.380, 0.428, 0.476, 0.523, 0.571, 0.619, 0.666, 0.714, 0.761, 0.809, 0.857, 0.904, 0.952, 1.0 }. Each time a task executes for an interval of time $I = 0.5$ seconds (sampling period), its energy consumption [14, 29] is computed by $E_i = I \cdot V_{ij}^2$.

The tasks timing parameters were computed as follows,

- Tasks time arrivals $b_i$ were computed with an initial set of 5 tasks with $b_i = 0$, followed by 3 sets of 5 additional tasks with task arrivals of $b_i = 20, 40, 60$ seconds respectively. The initial utilization for the total number of tasks $n = 20$ tasks is considered $U_{ini} = \sum U_{i=1}^{20} = 95\%$.

- **Estimated (worst-case) execution time** $\hat{c}_{ij}$: this parameter is computed as $\hat{c}_{ij} = \frac{C_i}{V_{ij}}$, where $\hat{c}_{ij}$ is the execution time of task $\tau_i$ and is generated following a uniform distribution in the interval [0.3,0.8] ms.

- **Actual execution time** $A_{ij}(k)$: this parameter is computed following a uniform distribution with values in the interval $[(\hat{c}_{ij} - 0.2\hat{c}_{ij}), (\hat{c}_{ij} + 0.2\hat{c}_{ij})]$.

- **Period** $P_i$: the period is computed as follows. $u_i = U_{ini}/n$, $\hat{P}_i = \hat{c}_{ij}/u_i$, $P_i = [\hat{P}_i, \hat{P}_i + 35ms]$.

The initial value of the speed of all tasks on each simulation is $j = 8$, $V_{08} = 0.619$. The performance of the Feedback Schedulers is measured using the following metrics:

<table>
<thead>
<tr>
<th></th>
<th>$E_s$</th>
<th>$U_m$</th>
<th>$M_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$ (RMS)</td>
<td>0.9091</td>
<td>0.185</td>
<td>0.35</td>
</tr>
<tr>
<td>$K_p$ (EDF)</td>
<td>0.9091</td>
<td>0.185</td>
<td>0.129</td>
</tr>
</tbody>
</table>

Table 1. Controller Parameters
• **Percentage of Energy Savings, %ES**: The percentage of energy savings for our Feedback Power-Aware Scheduler (FPAS) and for the Open Loop Control, for all tasks is computed by, 
\[ %ES = \sum_{k} Es(k) \times 100. \]

• **Miss Deadline Ratio, Mr**: this metric is computed by the ratio of the sum of missed deadlines on all tasks, MD, and the total number of deadlines of all tasks, TMD.

Our main interest in the simulations is to compare the energy savings as a function of the miss ratio for the FPAS and the Open Loop control. Figures 3 and 4 show the results using the RM and EDF scheduling policies respectively on underload and overload conditions. Table 2 show the standard deviation for all the points from these figures.

From these Figures is important to note that, in our simulations the target references for each type of controller were changed incrementally to reflect the variations on energy savings as a function of the miss ratio.

Target energy savings $Es_r$ for RMS and EDF for the FPAS controller are the following: 0, 0.065, 0.13, 0.195, 0.26, 0.325, 0.39, 0.455, 0.52, 0.585, 0.65, 0.715, 0.78, 0.845, 0.91, 0.975. For the open loop control, the different energy savings were obtained by changing the speeds for all tasks from $V_{ij} = 0.285$ to $V_{ij} = 1.0$.

From Figures 3 and 4 we note that the $Es$ control yield higher (or equal) energy savings than those obtained from the Open Loop Control. That is, for a given Miss Ratio the energy savings obtained by the $Es$ control is higher (or equal) than the energy savings from the open loop control. The highest difference in energy savings of the FPAS controller (compared with that of the Open loop control) are obtained when miss ratio is less than 20 %.

In order to obtain a degree of confidence for our simulation results we computed the standard deviation for each plot in the graphs. In Table 2 we show the standard deviation for all points from the graphs.

With the above results it can be concluded that the FPAS Scheduler improves the performance of the Open Loop control when applied to a power-aware scheduling environment.

### 7.1 Examples of Control

In this section we provide an example of each of the controllers used in previous experiments.

As discussed before, Figures 3 and 4 show the energy savings obtained as a function of the Miss Ratio under RMS and EDF respectively. These results denote the average of 1000 simulations. In this section, we are interested in showing the behavior of the controls for one specific simulation executing under RMS. Figures 5 and 6 show simulation examples using the RMS scheduling policy for each type of control (energy savings $Es$ and open loop control $Ol$). The average miss ratios for these examples are 34.24% and 38.21% for FPAS and open loop control respectively.
From Figure 5 it can be noted that the FPAS control adjusts the tasks speeds to the lowest possible speed; this situation is achieved by the EGA algorithm, which tries to maximize energy savings assigning the lowest possible speed, based on $U_e(k)$. This behavior explains the higher energy savings for the FPAS control (Figures 3 and 4).

It can be observed that $\Delta U$ is adjusted to compensate for each of the three additional task sets that are loaded at $b_1 = 20$, 40 and 60 seconds respectively. After $t = 40$, when the $U_e$ is over 100% the $Es$ control will adjust the speeds continuously in order to maintain the target energy savings reference.

From Figure 6 it can be noted that, now the open loop control will maintain a constant energy savings throughout the entire simulation time. As there are no speed changes during the entire simulation, it is not possible to achieve superior energy savings than those of the optimal EGA algorithm.

Their corresponding energy savings are 75.208% and 72.643% for FPAS and open loop control respectively. The target references for the $Es$ control is 71% and the constant speed for the $Ol$ control is 0.571.

All Figures are divided in 4 graphs, where the x-axis denotes execution time in seconds. The upper left graph shows energy savings and speed average. The speed average is the average of the speeds of all tasks at a given time instant. The lower left graph shows the behavior of the control variable $\Delta U$. The upper right graph shows the behavior of $Um$ and $Ue$. The lower right graph shows the behavior of the deadline miss ratio variable.

Table 2. Energy Savings and Standard Deviations

<table>
<thead>
<tr>
<th>%Mr</th>
<th>Std. Dev</th>
<th>% Es</th>
<th>Std. Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>47.4370</td>
<td>0.3020</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>47.4500</td>
<td>0.3080</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>47.6870</td>
<td>0.2200</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>49.8690</td>
<td>0.2160</td>
</tr>
<tr>
<td>0.0010</td>
<td>0.0010</td>
<td>52.1820</td>
<td>0.1960</td>
</tr>
<tr>
<td>0.0070</td>
<td>0.0030</td>
<td>54.5720</td>
<td>0.1750</td>
</tr>
<tr>
<td>0.0220</td>
<td>0.0040</td>
<td>57.2850</td>
<td>0.1290</td>
</tr>
<tr>
<td>0.0430</td>
<td>0.0050</td>
<td>60.1970</td>
<td>0.1200</td>
</tr>
<tr>
<td>0.0660</td>
<td>0.0050</td>
<td>63.3300</td>
<td>0.0960</td>
</tr>
<tr>
<td>0.0900</td>
<td>0.0070</td>
<td>67.3140</td>
<td>0.0885</td>
</tr>
<tr>
<td>0.1230</td>
<td>0.0090</td>
<td>71.4050</td>
<td>0.0780</td>
</tr>
<tr>
<td>0.1670</td>
<td>0.0110</td>
<td>75.5320</td>
<td>0.0730</td>
</tr>
<tr>
<td>0.2180</td>
<td>0.0130</td>
<td>79.5780</td>
<td>0.0540</td>
</tr>
<tr>
<td>0.2870</td>
<td>0.0150</td>
<td>84.4100</td>
<td>0.0410</td>
</tr>
<tr>
<td>0.3800</td>
<td>0.0150</td>
<td>89.0200</td>
<td>0.0230</td>
</tr>
<tr>
<td>0.4460</td>
<td>0.0170</td>
<td>91.6130</td>
<td>0.0180</td>
</tr>
</tbody>
</table>

| EDF |
|-----|-----|-----|-----|
| 0 | 0 | 47.4220 | 0.2970 |
| 0 | 0 | 47.4290 | 0.3130 |
| 0 | 0 | 47.6950 | 0.2310 |
| 0 | 0 | 49.8760 | 0.2080 |
| 0 | 0 | 52.1640 | 0.2070 |
| 0.0010 | 0.0050 | 54.4970 | 0.1850 |
| 0.0410 | 0.0110 | 57.2830 | 0.1250 |
| 0.0980 | 0.0090 | 60.1990 | 0.1190 |
| 0.1510 | 0.0080 | 63.3110 | 0.0840 |
| 0.2010 | 0.0090 | 67.3040 | 0.0850 |
| 0.2670 | 0.0120 | 71.4180 | 0.0660 |
| 0.3570 | 0.0130 | 75.5660 | 0.0630 |
| 0.4420 | 0.0130 | 79.5600 | 0.0490 |
| 0.5370 | 0.0150 | 84.4190 | 0.0310 |
| 0.6710 | 0.0120 | 89.0140 | 0.0100 |
| 0.7440 | 0.0110 | 91.6070 | 0.0070 |
8 Conclusions

In this paper, we proposed a closed-loop feedback real-time scheduling architecture, where the workload is dynamically adjusted by a Feedback $P$ (proportional) controller. The feedback scheduling architecture accepts workloads that exhibit a large variability in their workload and execute on a processor capable of handling several (discrete) speeds of execution and under strict limits on available battery power.

The main parts of this architecture are an energy feedback scheduler and a power-aware optimization algorithm. The feedback energy scheduler attempts to keep the CPU utilization at high level, minimize deadline misses, maximize energy savings and distribute the computing resources among real-time task to maximize control performance. The energy savings are used as control variables in the system configuration. A mathematical model of the power-aware architecture was formulated to facilitate the tuning of the controlled system.

Extensive set of tests are executed to simulate the performance of our feedback scheduling power-aware architecture and to compare its performance agains an open loop controller. Simulation results show that the proposed architecture is capable of handling real-time tasks with unknown arrivals and execution times, and derive a system in which power savings are maximized.

References


[34] www.transmeta.com