Using Imprecise Computation Techniques for Power Management in Real-Time Embedded Systems

Geovani Ricardo Wiedenhoft and Antônio Augusto Fröhlich

Abstract Embedded systems present severe limitations in terms of processing and memory capabilities and are often powered by batteries, making energy an important resource to be managed. This work explores energy as a parameter for Quality of Service (QoS) of embedded systems. The goal is to guarantee the battery lifetime specified by the application and yet preserve the deadlines of essential (hard real-time) tasks. We propose equations to check at project-time if a given set of tasks are schedulable. At execution-time, a preemptive scheduler for imprecise tasks based on the EDF algorithm prevents the optional subtasks execution when ever there is the possibility of deadline loss or battery exhaustion. A prototype was developed in EPOS using power management mechanisms provided by the system.

1 Introduction

Embedded systems are computational platforms dedicated to execute an usually known set of tasks with specific objectives. Typically, these systems present severe limitations in terms of processing and memory capabilities. Some of them, due to the mobile nature of their applications, are also powered by batteries with a limited supply of energy. Considering all these limitations, it is important for the mobile embedded system to be able to manage energy consumption without compromising system's performance.

Embedded systems hardware can rely on several mechanisms to manage energy consumption. Among them, are techniques of DVS (Dynamic Voltage Scaling) and resources hibernation. Some works in the literature explore the integration of these techniques with approaches that guarantee quality of service (QoS). Most of these

Geovani Ricardo Wiedenhoft · Antônio Augusto Fröhlich
Laboratory for Software and Hardware Integration, Federal University of Santa Catarina, PO Box 476 – 88049-900 – Florianópolis, SC, Brazil
e-mail: grw,guto@lisha.ufsc.br

approaches, however, only seek to minimize energy consumption with the main focus on traditional QoS metrics for processing, memory and communication. In a previous work [10], we argue that it is not enough just ensure traditional QoS metrics if, by doing so, the system runs out of battery and is unable to complete its computations.

We consider energy as a QoS parameter to meet the battery lifetime specified by the system developer, thus using QoS in terms of energy. In this work, the goal is not only to reduce energy consumption, but to improve the application utility in a system with limited energy charge, ensuring the battery lifetime and the deadlines of hard real-time tasks. The proposed approach expects the developer to define the period that the embedded system must be operational. By monitoring battery lifetime, the scheduler is able to select the tasks that will be executed or it can decrease QoS levels in order to reduce energy consumption and enhance system lifetime.

To achieve the proposed goal, the QoS control of applications was inspired by imprecise computation [5]. Imprecise computation divides tasks into two subtasks: one implementing a mandatory execution flow and another implementing an optional flow. The mandatory flow is the hard real-time part of the task, and it must always be executed with in its deadline. The optional flow is the *best-effort* part of the task, which is only executed if the desired timing requirements can be met. The imprecise computation scheduler does not execute the optional subtasks when there is the possibility of any mandatory subtask deadline to be lost, thus reducing the demand for system processing. Moreover, in our scheduler, we propose that the optional subtasks be prevented from executing when the energy level will not be sufficient to meet the time specified by application. This control creates more idle periods in the system, and the scheduler can use power management techniques to reduce the energy consumption of components during these idle periods.

The proposed scheduler is based on EDF (*Earliest Deadline First*) [4] scheduler, which the tasks with the lowest deadlines have the highest priorities. A prototype of this proposal was implemented in EPOS [6], a component-based embedded operating system. EPOS provides a set of mechanisms for power management, such as an infrastructure which allows applications to achieve appropriate power management [3] and a power manager with different operating modes that realize power management for application [9]. Moreover, EPOS provides a battery monitoring system, which informs the remaining energy in the platform.

2 Background

This work aims at guaranteeing that the batteries used in an embedded system can last at least the time required by the application and yet preserving the deadlines of essential tasks, i.e., the deadlines of hard real-time tasks. Our scheduler starts to decrease QoS levels in order to save energy when it detects that batteries will not last long enough to satisfy a previously defined expected system lifetime. The decreased control of application QoS levels is based on imprecise computation mechanisms [5], which divide tasks into two subtasks: a mandatory one and an optional one. The proposed scheduler is based on the EDF scheduling algorithm.

2.1 Imprecise Computation

Imprecise computation is a scheduling technique originally proposed to satisfy timing requirements of real-time tasks through decreasing levels of QoS. The control of application QoS levels done by imprecise computation worsens quality of results by not executing optional subtasks in order to guarantee that no mandatory subtask deadlines will be lost.

With the division of each task into two parts, imprecise computation unites real-time computing and *best effort* techniques for, respectively, the mandatory and optional subtasks. The mandatory subtask of imprecise tasks generates imprecise results which reflect the minimum of QoS to guarantee that these results are useful. These imprecise results have their quality enhanced when the optional subtask executes, generating the precise results.

The imprecise computation showed us favorable to use in our proposal in relation to energy. Suppose that a task consumes X energy units obligatorily. When it is divided into mandatory subtask (Y energy units) and optional subtask (Z energy units) the scheduler can save Z energy units if the optional subtask is not executed.

2.2 EDF

The EDF (*Earliest-Deadline First*) [4] algorithm is a real-time scheduling mechanism based on dynamic priorities and widely used in the literature. EDF distributes the highest priorities to the tasks with the shortest deadlines. At project-time a schedulability test evaluates the possibility of any task lose its deadline. At execution-time a preemptive scheduler selects to execute the highest priority task in *READY* state.

An exact schedulability test of the EDF algorithm is presented below. The real-time system considered contains n periodic and independent tasks, $\boldsymbol{\tau} = \{\tau_0, \tau_1, ...,$ $\tau_{n-1}\}$. Each τ_i is characterized by three parameters, (P_i, D_i, C_i), where P_i is the period in which the task i is scheduled, D_i is the max relative deadline of conclusion in relation to instant of the task i release and C_i is the task i execution time in the worst case which included times waiting by the priorities reversal. In this test is supposed that $\forall \tau_i$, $D_i = P_i$. The utilization U_i of the task i in processing terms is represented by equation $U_i = \frac{C_i}{D_i}$. The processor's capacity is set to 1, i.e., 100%. A system with ω processors has ω capacity. Thus, in order to tasks to be schedulable in the EDF algorithm, the utilization sum of all the tasks must be less than or equal to the processors' capacity, i.e.,

$$\sum_{i=1}^{n}\left(\frac{C_i}{D_i}\right) \leq \omega \tag{1}$$

where $\omega = 1$ on a system with single-processor. If $\sum_{i=1}^{n} U_i > \omega$, the processor will be overloaded and the tasks will not be schedulable.

3 The Proposed Scheduling Strategy

Our scheduler, based on EDF, guarantees the execution of mandatory subtasks with their deadlines respectively met, independently of the system energy level. However, the optional subtasks execution is not guaranteed. The optional subtasks are executed only if the mandatory subtasks deadlines and the system's batteries lifetime desired by application are met.

The objective of this scheduler is not only save the energy consumed in the system — otherwise, the technique would simply never execute the optional subtasks — but to meet the battery lifetime specified by the application and to meet the mandatory subtasks deadlines with the execution of the maximum possible of the optional subtasks, thus optimizing the application utility.

Figure 1 presents proposed scheduler algorithm, which the subtasks are treated as tasks in terms of scheduling. π is the interval among battery charge measurements that can be specified by the application programmer and must take into consideration that each measurement consumes energy to be performed. This interval depends on the battery power state found in the last measurement.

```
1: For every task that enters in READY state:
2:     Determine the new absolute deadline in accordance with the elapsed time
3:     Determine the priority based on absolute deadline
4:     Add to the queue according to calculated priority
5:
6: For each $\pi$ time units:        /* $\pi$ specified by the programmer and it depends on the energy state */
7:     Measure the battery
8:     Check if there is enough energy to meet the time required by application
9:
10: For each rescheduling:
11:     Select the highest priority task in READY state
12:     if, task is hard real-time, then
13:         Execute the selected task
14:     else,                                                   /* task is best effort */
15:         if, there is enough energy to meet the system lifetime required, then
16:             Execute the selected task
17:         else,                               /* Battery does not have sufficient energy */
18:             Use power management techniques
19:
```

Fig. 1 Proposed scheduler algorithm.

3.1 Schedulability Tests at Project-Time

The proposed scheduler is based on the EDF algorithm, thus it is possible to follow the same logic to calculate the tasks schedulability at project-time with a few adjustments. Suppose that the real-time system considered has n periodic and independent tasks, $\mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$, where $\forall \tau_i$, $D_i = P_i$. In the imprecise computation model, each τ_i is divided into mandatory and optional subtasks with execution times in the worst cases of μ_i and θ_i, respectively. Therefore, the total execution time of τ_i, in the worst case, is $C_i = \mu_i + \theta_i$. In order to guarantee that no mandatory subtasks deadlines will be lost, equation (2) must be respected.

$$\sum_{i=1}^{n}\left(\frac{\mu_i}{D_i}\right) + \sigma \leq \omega \tag{2}$$

Where $\omega = 1$ for a system with a single-processor and σ represents the interference in the worst cases, which includes: time spent in the operating system, context switch, scheduler algorithm. Equation (2) must be met in order for the tasks to be schedulable in relation to mandatory subtasks deadlines, otherwise, the processor is overloaded.

With the inclusion of the optional subtask execution time in equation (2), we can determine if the tasks as a whole will be executed, mandatory and optional subtasks. However, it is important to note that equation (3) is not a obligatory requirement in our algorithm and only will be relevant when equation (2) is true, otherwise, the tasks are not schedulable.

$$\sum_{i=1}^{n}\left(\frac{\mu_i + \theta_i}{D_i}\right) + \sigma \leq \omega \tag{3}$$

Mandatory and optional subtasks are schedulable in relation to their deadlines when equation (3) is respected. Otherwise, a certain fraction χ of optional subtasks is discarded. Equation (4) presents how to find the fraction χ.

$$\chi = \frac{\sum_{i=1}^{n}\left(\frac{\mu_i+\theta_i}{D_i}\right) + \sigma - \omega}{\sum_{i=1}^{n}\left(\frac{\theta_i}{D_i}\right)} \tag{4}$$

The energy-related objective can be achieved by following the same kind of logic presented thus far, but taking into account the tasks' energy consumption rate. The τ_i energy consumption in the wort case, E_i, is given by the sum of the energy consumption in the mandatory and optional subtasks worst cases times $E_{\mu i}$ e $E_{\theta i}$, respectively, ($E_i = E_{\mu i} + E_{\theta i}$). We suppose that, as with worst cases times, the worst cases energy consumptions are previously known by the application developer. These values can be obtained by energy profiling or another techniques. The maximum number of possible executions η_i of τ_i in the time required by application T_t is given by division between the time required and the execution interval of τ_i, i.e., $\eta_i = \frac{T_t}{P_i}$. T_t is given by the application developer based on battery capacity. In order to meet at least the mandatory parts of the tasks, we have equation (5) which indicates if the set of tasks will be schedulable with respect to energy.

$$\sum_{i=1}^{n}\left(\frac{E_{\mu i}\times\eta_i}{E_t}\right)+\varepsilon\leq 1 \tag{5}$$

Where E_t is the total energy of the system (battery specification), i.e., battery capacity, ε represents energy consumption in the worst case of different factors such as the energy consumed by the operating system, the context switch, the scheduler algorithm itself. The battery's capacity is set to 1, i.e., 100 %. Substituting η_i in the equation (5) we have equation (6).

$$\sum_{i=1}^{n}\left(\frac{E_{\mu i}\times T_t}{P_i\times E_t}\right)+\varepsilon\leq 1 \tag{6}$$

The tasks are schedulable in relation to energy in our algorithm if equation (6) is respected. Otherwise, the system will not meet the battery lifetime required by application for this set of tasks. The inclusion of the energy consumed by optional subtasks in equation (6) allows us to check if the tasks as a whole will be executed. As discussed previously, this is not an obligatory requirement and equation (7) only should be calculated if equation (6) is respected, i.e., mandatory subtasks met.

$$\sum_{i=1}^{n}\left(\frac{(E_{\mu i}+E_{\theta i})\times T_t}{P_i\times E_t}\right)+\varepsilon\leq 1 \tag{7}$$

All mandatory and optional parts of the tasks are executed in relation to system energy if equation (7) is respected. Otherwise, a certain fraction γ of optional subtasks will not be executed because the system would not meet the battery lifetime specified by the application. Equation (8) provides a fraction γ of optional subtasks discarded in relation to energy.

$$\gamma=\frac{\sum_{i=1}^{n}\left(\frac{(E_{\mu i}+E_{\theta i})\times T_t}{P_i\times E_t}\right)+\varepsilon-1}{\sum_{i=1}^{n}\left(\frac{E_{\theta i}\times T_t}{P_i\times E_t}\right)} \tag{8}$$

In this algorithm, the objective is to meet the two parameters in relation to time and energy, i.e., the mandatory subtasks deadlines and battery lifetime specified by the application, respectively. Thus, (9) is the full equation of our scheduler that must be true in order to tasks will be schedulable.

$$\left[\sum_{i=1}^{n}\left(\frac{\mu_i}{D_i}\right)+\sigma\leq\omega\right]\wedge\left[\sum_{i=1}^{n}\left(\frac{E_{\mu i}\times T_t}{P_i\times E_t}\right)+\varepsilon\leq 1\right] \tag{9}$$

The mandatory subtasks have their executions guaranteed in our scheduler in relation to time and energy if equation (9) is respected. The maximum fraction λ possible of optional subtasks lost in relation to time and energy can be obtained by equation (10).

$$\lambda = \max(\chi, \gamma) \tag{10}$$

3.2 Schedulability Test at Execution-Time

In order to provide QoS in terms of energy and make better use the resources with the optional subtasks execution it is necessary periodically to check at execution-time if the battery lifetime specified by the application $T_{t\kappa}$ in the instant κ can be achieved. Therefore, $T_{t\kappa}$ is recalculated in the instant κ according to the elapsed time. The total energy of the system (battery charge) $E_{t\kappa}$ also must be recalculated in the instant κ. The embedded systems platforms usually provide mechanisms to get the battery charge. Equation (11) can be recalculated with the new values in order to check if $T_{t\kappa}$ can be met in the instant κ.

$$\sum_{i=1}^{n}\left(\frac{E_{\mu i} \times T_{t\kappa}}{P_i \times E_{t\kappa}}\right) + \varepsilon \leq 1 \tag{11}$$

All mandatory subtasks are executed and optional subtasks will be scheduled if equation (11) is respected because this equation indicates there is sufficient energy to meet $T_{t\kappa}$. Otherwise, some optional subtasks will be discarded. The scheduler calls a power manager in the time that the optional subtasks would be in execution. Thus, it takes the idle time of the system in order to save energy. The optional subtasks return to be executed when it is observed that equation (11) returns to be true.

4 Implementation

A prototype was developed in order to test the proposed scheduler using EPOS (*Embedded Parallel Operating System*) [6]. EPOS is a framework of hierarchically organized components that generates application-specific runtime support systems. To do that EPOS analyzes the set of dedicated applications it must support prior to system generation time, thus configuring the system accordingly. Furthermore, through the separation of system abstractions, hardware mediators and scenario aspects, EPOS allows the development of fully platform-independent applications.

In EPOS, every system component implements a uniform power management interface [3]. This infrastructure allows applications to interact with the system to implement proper energy consumption management for embedded systems. Through the use that EPOS provides a low-overhead dynamic power manager [9]. This power manager uses re-pluggable heuristics, allowing configuration and adaptability to specific applications. The EPOS power manager has different operation modes, such as the possibility to choose if the manager will be on or off, the possibility of configuring only the desired components by the application for the power management, and if the manager will be active or passive in the power management.

EPOS also provides a battery charge monitor, which contributes to achieve the objectives of this work. The EPOS monitor is based on the battery voltage observation in order to get the battery charge, because the battery characteristic is to have its tension reduced as the use. However, there are some details to be observed, because the sampled voltage is not linearly related to battery discharge rate, the system does not have the ability to convert all provided tension in usable resource and also there is a minimum voltage that the system works. Thus, the monitor establishes a discreet relationship between the voltage and battery charge through the division of the obtained voltages in 10 time slices, which the voltages have different variations. Each slice corresponds to a nominal capacity percentage of the used battery.

The EPOS monitor does not implement a constant tracking of the real battery voltage, as each sampling consumes energy to be realized, in addition to considerably overhead for the application. In order to reduce these effects, the monitor uses a structure with information previously known which allows tracking the energy consumption in an approximate way. The information are in relation to specific characteristics of the battery and energy consumption by the system hardware components that will be monitored. The monitor verifies the battery charge through the voltage in the beginning of the execution, and during the execution updates the value with energy consumed by system peripherals.

We extended EPOS to support our scheduler with imprecise tasks and conditional executions to time and energy parameters. The tasks model in EPOS was based on monotone imprecise tasks. In this model, the monotone tasks improve the result quality at the time in execution and the worst case do not change the result. Thus, the mandatory subtasks generate results with the minimum QoS necessary to guarantee that these results are useful, and the optional subtasks realize successive refinements that results. The completion of these tasks can occur at any execution time without cause integrity problems in the results. Thus, the scheduler can decide at any instant to finalize the optional subtask execution. The application is responsible for the results integrity by different methods such as the use of control bits or the use of last data update timestamps.

The imprecise tasks implementation in EPOS was realized through the creation of two threads: one containing execution flow to handle the mandatory part and another with the execution flow to handle the optional part. The system creates these threads in a transparent manner to the programmer. This approach only expects the programmer to specify, when creating a imprecise thread, two entry points: one for the mandatory subtask and another for the optional subtask with their parameters.

The scheduler in execution always chooses the highest priority subtask in accordance with the deadlines as our algorithm is based in EDF. The optional subtasks are scheduled if there are not mandatory subtasks in *READY* state and if there is energy enough to meet battery lifetime specified by the application, i.e., optional subtasks have lower priorities than mandatory subtasks. When a mandatory subtask enters in *READY* state and its optional subtask is not yet finished the execution in the previous period, the scheduler immediately suspends this optional subtask execution. These characteristics prevent mandatory subtasks deadlines losses. The optional subtasks contexts are always restarted in a new task period.

The scheduler also updates at execution-time the $T_{t\kappa}$ with elapsed time and the $E_{t\kappa}$ using the EPOS energy monitor. Scheduler recalculates the equation (11) in periods π with these new values in order to check if the system is able to sustain the current workload without running out of battery before the required lifetime $T_{t\kappa}$ is achieved. π will depend on the last energy analysis. In the best case, equation (11) is respected and optional subtasks can be scheduled. Otherwise, optional subtasks are discarded and, taking advantage of the idle period created, the scheduler executes the EPOS power manager in passive mode. In addition to saving energy by not execute the optional subtasks, the power manager reduces the system energy consumption through the use of power management techniques. The optional subtasks return to execution when the scheduler identify $T_{t\kappa+\iota}$ can be met again in instant $\kappa+\iota$.

5 Related Work

GRACE-OS [11] is an energy-efficient operating system for mobile multimedia applications. This system uses a cross-layer adaptation technique to guarantee QoS on systems with adaptive software and hardware. It combines real-time scheduling with DVS mechanisms to dynamically manage energy consumption. It was implemented over the LINUX operating system and it only supports soft real-time tasks. GRUB-PA [8] is somehow similar to GRACE-OS. The main difference is GRUB-PA supports both soft and hard real-time tasks.

Niu [7] proposed to minimize energy consumed by soft real-time systems while guaranteeing QoS requirements. This goal is achieved by a hybrid static/dynamic scheduling algorithm that it uses DVS mechanisms and it partitions the set of tasks in mandatory and optional tasks. In this work, the QoS requirements are qualified by *(m,k)* constraints which it specifies that tasks must meet at least *m* deadlines in any *k* consecutive task releases. In a similar work, Harada [2] proposed to resolve the trade-off between QoS maximization and energy consumption minimization. It uses an allocation of processor cycles and frequency with QoS guarantees and it divides each task into mandatory and optional parts.

Other projects explore trade-off between application's QoS and energy consumption through adaptations in the applications aiming to meet the time specified by application. ODYSSEY [1] uses that idea. It monitors the energy budget and with this information it can select the correct state between energy saving and quality of application. This work also demonstrates how the applications can dynamically change their behavior ("fidelity" of the data) with the goal of saving energy.

ECOSYSTEM [12] is another operating system that supports application adaptation. This system is based in a "currency" that the applications use to allocate ("to pay") system resources (e.g., access to memory, network or disks), called *currentcy*. The system distributes *currentcies* periodically to tasks accordingly to an equation that defines the discharge rate that the system battery can assume to force the system to last for a defined period of time. This allows applications to adapt their execution based on their *currentcy* balance. This model unifies the calculation of energy on the various hardware devices and it provides a satisfactory energy allocation among the applications.

6 Conclusion

This work proposed an approach to exploit energy as a QoS parameter in order to guarantee that battery lifetime can last time desired by mobile embedded system and yet preserve the deadlines of hard real-time tasks. Our approach was inspired by imprecise tasks concepts, according to tasks can be divided into mandatory and optional parts. In this article, equations at project-time were presented with objective the of application programmer to check if a set of tasks will be schedulable in our algorithm in relation to two parameters desired, i.e., time and energy. At execution-time, our scheduler based on EDF algorithm ensures the mandatory subtasks deadlines and recalculates the equation of energy in order to check if the required battery lifetime will be met. The optional subtasks are prevented from executing, i.e, decreasing QoS levels if any required parameter will not be met. A prototype was developed in EPOS, which allowed the execution of a power manager in idle periods created by non-execution of the optional subtasks, thus reducing energy consumption by stopping or slowing down system components during these idle periods.

References

1. Flinn, J., Satyanarayanan, M.: Energy-aware adaptation for mobile applications. In: ACM SOSP '99, pp. 48–63. ACM Press, New York, NY, USA (1999)
2. Harada, F., Ushio, T., Nakamoto, Y.: Power-aware resource allocation with fair qos guarantee. In: IEEE RTCSA '06, pp. 287–293. IEEE Computer Society, Washington, DC, USA (2006)
3. Hoeller, A.S.J., Wanner, L.F., Fröhlich, A.A.: A Hierarchical Approach For Power Management on Mobile Embedded Systems. In: 5th IFIP DIPES, pp. 265–274. Braga, Portugal (2006)
4. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1), 46–61 (1973). DOI http://doi.acm.org/10.1145/321738.321743
5. Liu, J.W., Shih, W.K., Lin, K.J., Bettati, R., Chung, J.Y.: Imprecise computations. Proceedings of the IEEE **82**(1), 83–94 (1994)
6. Marcondes, H., Junior, A.S.H., Wanner, L.F., Fröhlich, A.A.: Operating Systems Portability: 8 bits and beyond. In: 11th IEEE ETFA, pp. 124–130. Prague, Czech Republic (2006)
7. Niu, L., Quan, G.: A hybrid static/dynamic dvs scheduling for real-time systems with (m, k)-guarantee. rtss **0**, 356–365 (2005)
8. Scordino, C., Lipari, G.: Using resource reservation techniques for power-aware scheduling. In: ACM EMSOFT '04, pp. 16–25. ACM Press, New York, NY, USA (2004)
9. Wiedenhoft, G.R., Hoeller, A.S.J., Fröhlich, A.A.: A Power Manager for Deeply Embedded Systems. In: 12th IEEE ETFA, pp. 748–751. Patras, Greece (2007)
10. Wiedenhoft, G.R., Hoeller, A.S.J., Fröhlich, A.A.: Quality-Of-Service: the Role of Energy. In: 9th Workshop on Real-Time Systems, pp. 107–110. Belem, Brazil (2007)
11. Yuan, W.: Grace-os: An energy-efficient mobile multimedia operating system. Ph.D. thesis, University of Illinois at Urbana-Champaign (2004)
12. Zeng, H., Ellis, C.S., Lebeck, A.R., Vahdat, A.: Ecosystem: managing energy as a first class operating system resource. In: ACM ASPLOS-X, pp. 123–132. ACM, New York, NY (2002)