A DYNAMIC TASK SCHEDULING ALGORITHM FOR BATTERY POWERED DVS SYSTEMS

Jameel Ahmed and Chaitali Chakrabarti

Department of Electrical Engineering, Arizona State University
ahmedjameel@asu.edu, chaitali@asu.edu

ABSTRACT

Battery lifetime enhancement is a critical design parameter for mobile computing devices. Maximizing battery lifetime is a particularly difficult problem due to the non-linearity of the battery behavior and its dependence on the characteristics of the discharge profile. In this paper we address the problem of dynamic task scheduling with voltage scaling in a battery-powered DVS system. The objective is to maximize the residual charge and the battery voltage after the execution of tasks. We present here a two phase algorithm: in the first phase (off-line) a battery-aware algorithm schedules the tasks in a hyper-period assuming WCET. In the second phase (on-line), the algorithm reassigns the voltage levels based on the additional slack generated due to the AET being less than the WCET. Simulation with 10,000 random examples shows that the proposed algorithm does significantly better than the competitive low power real time scheduling algorithm. We extend this procedure to handle scheduling on multi-processor environments.

1. INTRODUCTION

Battery-operated portable devices are widely used in mobile computing and wireless communication applications. Maximizing battery lifetime is the most important design metric for such systems. This problem is quite challenging due to the non-linear behavior of the battery. Since the amount of energy delivered by the battery depends on the discharge current profile [1], the battery life can be extended by controlling the discharge current level and shape. In this paper, we propose an approach based on modifying the discharge current profile of real time tasks during task scheduling on a DVS (Dynamic Voltage Scalable) processor such that the charge consumption or the drop in battery voltage (the parameters for evaluating battery performance) during the task execution is minimized.

In recent years, there has been significant amount of work done in studying battery characteristics [1] and using these characteristics to shape the discharge profile [4], [5]. All of the earlier work on battery aware task scheduling (including ours [5]) has been for static tasks where complete information about the tasks is known apriori. Task scheduling for real-time tasks has been investigated in the context of ideal power sources [6]. To the best of our knowledge, this is the first attempt at battery aware scheduling for real-time tasks.

The proposed algorithm for single processor systems works in two phases. In the off-line phase, a schedule is determined for each instance of the task based on the WCET (Worst Case Execution Time) of the task. This schedule is based on [5] and utilizes the non-linear properties of the battery to enhance the battery performance. In the run time phase, the voltage levels of the tasks are further adjusted based on the additional slack generated due to AETs (Actual Execution Time) being less than the WCETs. We have also extended this procedure to handle multiprocessor scheduling.

The rest of the paper is organized as follows. The paper begins with preliminaries like system configuration, battery models, etc in Section 2. In Section 3, the scheduling algorithm is presented with an example. The simulation with 10,000 random examples is described in section 3.4. A multi-processor scheduling strategy is presented in section Section 4.

2. BACKGROUND

2.1. System Level Configuration

The system configuration for the battery-operated processor under consideration is described in Figure 1. The system consists of one DVS processor driven by a single battery. The battery is used to power the processor through a DC-DC converter . The DC-DC converter has an efficiency η which is typically in the range [0.8,0.9]. The efficiency $\eta = \frac{I_{proc}V_{proc}}{I_{batt}V_{batt}}$, where V_{batt} and I_{batt} are the battery voltage and current and V_{proc} and I_{proc} are the processor voltage and current.

Fig. 1. System Level Configuration.

Throughout the paper, we assume that the change in voltage is always associated with a change in frequency. For a long channel CMOS gate with threshold voltage V_t and supply voltage V_{dd}, voltage scaling by a factor of s causes the processor current I_{proc} to scale by a factor of s^2. Assume that the DC-DC conversion efficiency η and the battery voltage V_{batt} are averaged constants for the duration of a task execution, then the battery current I_{batt} scales by ηs^3 [5]. Thus, scaling results in a large drop in the battery load and leads to significant maximization of the residual charge at the end of a task profile.

2.2. Battery Models

In this work, we have considered two battery models: (1) a high-level analytical charge based model, and (2) a low-level electrochemical simulator, DUALFOIL [2]. The charge based high-level model gives an analytical relationship between the current load,

discharge time and the charge consumed due to the discharge. Under a time-varying discharge $i(t)$, the battery model is of the following form [4]:

$$\alpha = \int_0^L \left[1 + 2 \sum_{m=1}^{\infty} e^{-\beta^2 m^2 (L-\tau)} \right] i(\tau)\, d\tau \qquad (1)$$

where, L is the lifetime, and α and β are battery-specific parameters. Note that α is represents the charge capacity of the battery and is expressed in charge units (coulombs). β^2 is related to the diffusion rate within the battery and captures its nonlinear discharge profile. We define another variable σ to calculate the charge consumed for a given discharge profile of length T. σ is defined as:

$$\sigma = \int_0^T \left[1 + 2 \sum_{m=1}^{\infty} e^{-\beta^2 m^2 (T-\tau)} \right] i(\tau)\, d\tau \qquad (2)$$

The charge slack or residual charge at the end of a discharge is defined as $Q = \alpha - \sigma$. Our aim is to maximize Q. For validation purposes, our algorithm has been verified by a low level Li-ion battery simulator DUALFOIL [2]. For any input task profile, DUALFOIL gives a very accurate output in terms of battery voltage as a function of the discharge time.

Battery Configurations used:

Two batteries have been considered in this paper.

B1: This is a 2.2 Whr Li-ion Battery used in Compaq's ITSY pocket computer. α and β values have been estimated as 35220 mA-min and 0.637 $s^{-1/2}$ respectively [3].

B2: The parameters α and β have been estimated to be 40375 mA-min and 0.273 $s^{-1/2}$ respectively [3] .

The battery performance has been evaluated using residual charge for configurations B1 and B2, and drop in voltage for B2. The drop in battery voltage is defined as $(V_{oc} - V_{batt})/(V_{oc} - V_{cut})$, where V_{oc} is the open circuit voltge and V_{cut} is the cut off voltage. For B2, V_{oc} = 4.3V and V_{cut} = 3.2V.

2.3. Non-linear Properties of the Battery

There are several important properties of the battery with respect to voltage scaling that have been derived from the analytical model. We present two of the properties used for developing the real-time scheduling heuristics [3], [5]:

Property 1: *For a fixed voltage assignment (only task start times can be changed), sequencing tasks in the non-increasing order of their currents is optimal when the task loads are constant during the execution of the task.*

Property 2: *Given a pair of two identical tasks in the profile and a delay slack to be utilized by voltage down-scaling, it is always better to use the slack on the later task than on an earlier task.*

3. DYNAMIC TASK SCHEDULING ALGORITHM

Task Definition: A given task k is associated with the following parameters: the current I_k, the worst case execution time $WCET_k$, the arrival time a_k, the start time t_k, the actual execution time AET_k, the deadline d_k and the period P_k. The slack associated with a task is due to two factors: (1) the inherent slack due to the difference between the deadline and the WCET and (2) the slack generated due to the actual execution time being less than the worst case execution time. The task description is given in Figure 2.

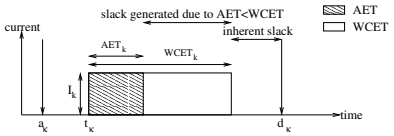


Fig. 2. Task Definition.

3.1. Overview of the algorithm

The basic idea of our algorithm is to exploit both the slacks to reduce the voltage levels of the tasks, so that the battery charge consumed or the drop in voltage is minimized. Our algorithm operates in two phases.

Phase I: Off-line task scheduling algorithm using WCET.

Phase II: On-line algorithm using AET.

In Phase I(off-line) we assume that the tasks execute at their WCETs. We determine a schedule for one hyperperiod (defined as the least common multiple of the periods of all the tasks in the task set) using the algorithm in [5]. In Phase II(run-time), the slack generated due to the AET being less than the WCET, is used to further scale the voltage levels of the tasks.

3.2. Phase I: Off-line task scheduling algorithm using WCET

The off-line scheduling algorithm, based on [5], determines the task ordering and the voltage level of each instance of a task in a hyperperiod. The assumption of WCETs in this phase guarantees that the tasks meet their deadlines. Furthermore, we assume equal currents for all the tasks in the given task set when operating at the reference voltage. This phase is done in two steps.

Step I: Obtain a feasible schedule by using the earliest deadline first algorithm.

Step II: Utilize the available slack by voltage down scaling as much as possible starting from the end of the profile.

The slack utilization phase of the algorithm is based on Property 2 (see Section 2.3).

3.3. Phase II: On-line algorithm using AET

During the operation of the system, the actual execution time (AET) of a task could be a lot smaller than its WCET. The additional slack generated can be utilized to scale the tasks further. In our algorithm, the utilization of this slack is based on Property 2 (see Section 2.3) which suggests that it is best to use the slack as late as possible. We achieve this by a process called slack forwarding.

Slack forwarding is based on the observation that slack generated by early completion of a task can be made available to a later task if the later task is released prior to the time at which the slack originated. We explain this with an example in Figure 3. Consider two tasks T1 and T2. Assuming worst case execution times for the tasks. Task T1 starts at t1 and finishes at t4 and T2 starts at t4 and finishes at t7, as shown in Figure 3(a). Suppose T1 actually finishes earlier at time t2, generating a slack of t4-t2. All of this slack is available to T2 if it's arrival time is at t2 or before, as depicted in Figure 3(b). If the task T2 was released at t3, only a part of the generated slack is available to T2, as shown in Figure 3(c). If the task T2 was released at t4 none of the generated slack is available to T2 as shown in the Figure 3(d). Thus the decision of slack forwarding can be made by inspecting the arrival time of the subsequent task to be executed.

Figure 4(a) describes the on-line algorithm. The input to the online-algorithm consists of ordering of tasks as well as their voltage levels based WCET. The purpose of this algorithm is to readjust the voltage level of the task based on additional slack. The basic steps are as follows. After the completion of a task, the scheduler gets the next task from the run queue. The finish time of the task is estimated based on the voltage level determined in Phase I. If the finish time is before the release time of the next task in the queue, the voltage level of the task is readjusted. The over all algorithm is summarized in Figure 4(b).

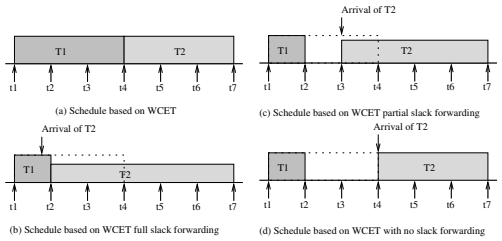
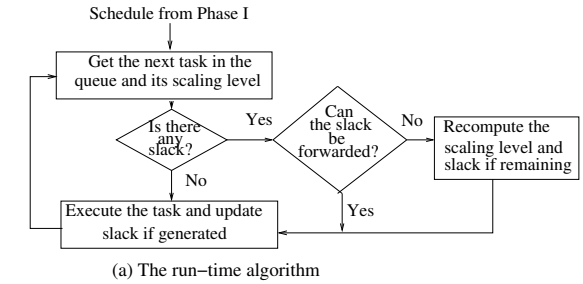


Fig. 3. Slack Forwarding.



(a) The run−time algorithm

Phase I: Get the task scaling level and finish times for every task
Phase II: Repeat for Every task
Get the scaling level of the next task ***k***
If the task is not released
then wait
else
If finish time of task ***k*** < release time of ***(k+1)***
then update the scaling level to absorb slack
end if
end if
Execute the task

(b) Summary of the over all algorithm

Fig. 4. On-line scheduling algorithm

Comparative Example

To demonstrate the effectiveness of our approach for battery-powered devices, we consider the task set in [6]. We scale all the numbers of the example in [6] by 10 to get significant voltage drops. There are three tasks with periods 5, 8 and 10 mins; the hyperperiod is 40 mins (L.C.M of 5, 8 and 10). The initial task specification is described in Table I in Figure 5. The set of operating voltages considered during voltage scaling is $S_V = \{3.3, 3.0, 2.7, 2.5, 2.0\}$V. We assume that the actual execution times of the tasks is drawn from a random Gaussian distribution with mean, denoted by μ, and standard deviation denoted by σ, given by

$$\mu = \frac{WCET + BCET}{2} \qquad \sigma = \frac{WCET - BCET}{6} \qquad (3)$$

where BCET is the best case execution time assumed to be 0.1 times the WCET.

Figure 5 shows the final task profile with our algorithm after each phase as well as that generated with the low power fixed priority algorithm in [6]. Table II in Figure 5 shows the residual charges and drop in battery voltage for the two algorithms. It is clear that our approach gives a much better battery performance than the algorithm used in [6].

Table I

Task	Period	Deadline	WCET	Priority
T1	5	5	1	1
T2	8	8	2	2
T3	10	10	4	3

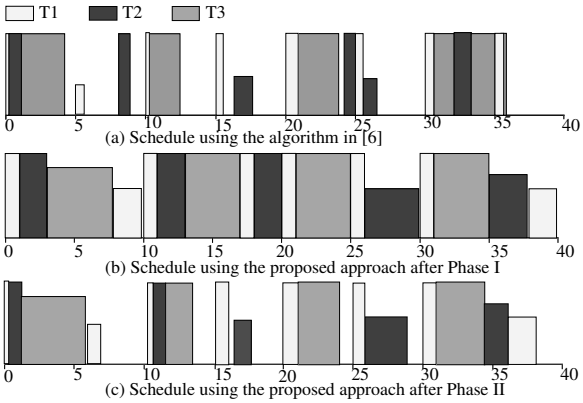


Table II

Algorithm	Analytical Q (mA−min)		Voltage	%Drop
	B1	B2	B2	in Voltage
In [6]	29428	31197	4.02V	25.5
Proposed	**30346**	**32816**	**4.07V**	**20.5**

Fig. 5. Task scheduling using low power fixed priority scheduling algorithm in [6] versus our algorithm

3.4. Results

We compare the battery performance of the proposed approach with the low power fixed priority scheduling algorithm in [6] for 10,000 random test cases. Each test case consisted of three tasks whose periods are randomly chose from 1 to 10 mins. The deadlines of the tasks were made equal to their periods. The WCET for a task was randomly chosen between 0 and the period of the task. All the test cases where the task set was not schedulable either with the algorithm in [6] or our algorithm were dropped. The AET of each instance of all the tasks was drawn from a random Gaussian distribution with mean and variance as given in Equation 3. We assumed a continuous operating voltage for the system, from 2.0 to 3.3 Volts and the B2 battery configuration. Figure 6 plots the difference in the residual charge obtained by our approach and with that in [6]. It is clear from the plot the the proposed algorithm consistently results in a better battery performance.

4. MULTI-PROCESSOR SCHEDULING

The system consists of multiple identical DVS (dynamic voltage scalable) processors driven by a single battery. There are n independent tasks to be run on m processors. Each task has its relative

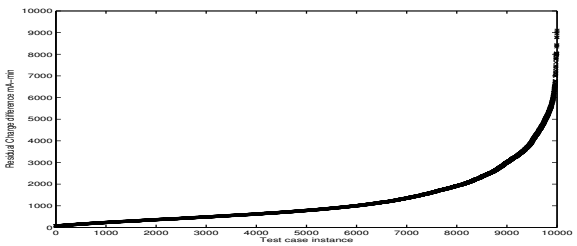


Fig. 6. Difference in the residual charge compared to algorithm in [6] for 10,000 random cases.

deadline equal to its period. For simplicity, we assume that all the tasks have the same period and that their arrival times are the same.

There are two strategies for scheduling real-time tasks on a multiprocessor system, namely, the global scheme and the partitioning scheme. In the global scheme, each occurrence of a real-time task may be executed on a different processor [7]. In contrast, in the partitioning scheme, all occurrences of a particular task are executed on the same processor. We choose the partitioning scheme because of its lower complexity. Furthermore, after the assignment is complete, any uniprocessor scheduling algorithm can be applied to each processor.

4.1. Proposed Algorithm

The proposed algorithm operates in two phases.

Phase I: Assignment of tasks to the processors.

Phase II: Scheduling of the assigned tasks on each of the processor.

In Phase I, the task assignment is done such that the utilization of all the processors is as close to each other as possible. If $U_1, U_2, \ldots, U_m$ are the utilizations of the m processors, then the assignment that minimizes $\sum_{j=1}^{m} \sum_{k=1}^{m} abs(U_j - U_k)$ is chosen. The assignment problem can be framed as a simple MILP with the constraint that the utilization of each processor is less than 1.

In Phase II, each processor invokes the run-time algorithm described earlier. Among the tasks waiting to be executed, the task with the earliest deadline is selected first. If tasks have the same deadline, then the task with the maximum execution time is selected first. Voltage scaling is done on a task only if there is no other ready task.

4.2. Results

In this section, we describe an example consisting of a set of six tasks to be scheduled on a system consisting of two identical processors. Each task consumed a current of 100mA at 3.3 V. The voltage of the processors varied on a continuous voltage scale between 2V and 3.3 V. All the tasks had equal periods, same deadlines (equal to their periods) and same release times. The task periods were randomly chosen between 1 min and 10 min and the execution time was randomly chosen between 0.1 min and the task period. The task set was chosen such that the total utilization was between 1 and 1.75.

We simulated the battery performance for all possible schedules (task assignment followed by uniprocessor schedule). For each schedule, we noted the maximum value of the residual charge, the minimum value of the residual charge and the value obtained by the proposed algorithm. A plot of the the difference between the maximum residual charge and the residual charge obtained by the proposed approach for 500 random results is shown in Figure 7. The difference between the maximum residual charge and the minimum residual charge is also shown in the same plot. It is clear that the residual charge obtained by our approach is very close to the maximum, implying that our approach results in a discharge profile that has superior battery performance.

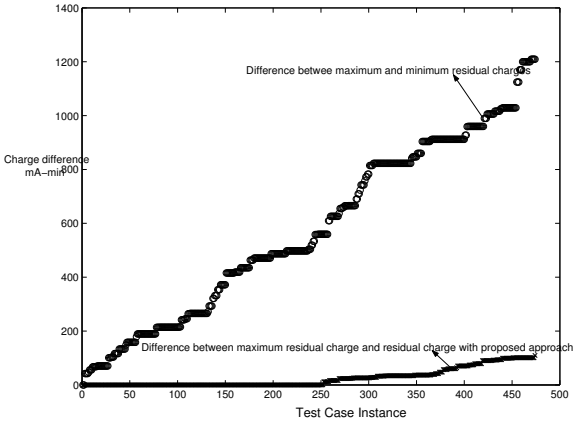


Fig. 7. Results for 500 random samples.

5. CONCLUSION

In this paper we addressed the problem of real-time task scheduling for battery operated DVS systems that maximize the residual charge and the battery voltage. The proposed algorithm for single processor system consists of an off-line phase and an on-line phase. In both phases, the slack is allocated in a battery-conscious manner. We also presented a multiprocessor scheduling algorithm. These algorithm have a superior battery performance compared to the other algorithms.

6. REFERENCES

[1] T. Martin, "Balancing batteries, power, and performance: System issues in CPU speed-setting for mobile computing," *Ph.D Dissertation*, August 1999.

[2] T. Fuller and M. Doyle, J. Newman, "Simulation and optimization of the dual lithium ion insertion cell," *J. Electrochemical Society 141(1)*, 1994.

[3] D. Rakhmatov, "Modeling and Optimization of Energy Supply and Demand for Portable Reconfigurable Electronic Systems," *Phd Thesis, University of Arizona*, 2002.

[4] D. Rakhmatov, S. Vrudhula, and C. Chakrabarti, "Battery-conscious task sequencing for portable devices including voltage/clock scaling," *Proc.DAC*, 2002.

[5] P. Chowdhury and C. Chakrabarti, "Battery-aware task scheduling for a system-on-a-chip using voltage/clock scaling," *Proc.SiPS*, 2002.

[6] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proc. DAC*, 1999.

[7] N.D. Thai, "Real-time scheduling in distributed systems," *International conference on parallel computing in electrical engineering*, 2002.