An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-time Energy Harvesting Systems

Shaobo Liu, Qing Wu, and Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, New York 13902, USA
{sliu5, qwu, qqiu}@binghamton.edu

Abstract – In this paper we propose an adaptive scheduling and voltage/frequency selection algorithm which targets at energy harvesting systems. The proposed algorithm adjusts the processor operating frequency under the timing and energy constraints based on workload information so that the system-wide energy efficiency is achieved. In this approach, we decouple the timing and energy constraints and simplify the original scheduling problem by separating constraints in timing and energy domains. The proposed algorithm utilizes maximum task slack for energy saving. Experimental results show that the proposed method improves the system performance in remaining energy, deadline miss rate and the minimum storage capacity requirement for zero deadline miss rate. Comparing to the existing algorithms, the new algorithm decreases the deadline miss rate by at least 23%, and the minimum storage capacity by at least 20% under various processor utilizations.

Categories and Subject Descriptors
B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aides

General Terms
Algorithms, Design, Performance

Keywords
Energy harvesting, Dynamic voltage and frequency selection

I. Introduction

The energy constraint remains a major issue for battery powered devices, despite that a lot of researchers have been working actively to solve this problem. Generally the research activities can be grouped into two categories: one is to focus on the reduction of the power consumption of the battery powered device, such as dynamic power management (DPM) [1-3] and dynamic voltage and frequency selection (DVFS) [4-6]. The other is to focus on seeking new energy sources for the device, such as energy harvesting [7-9]. Although both DPM and DVFS techniques are able to effectively reduce the power consumption of a device, the limited energy in the battery will be exhausted eventually; and then the battery has to be either recharged or replaced before the device can continue to function.

However, in some applications, neither recharging nor replacing batteries is practical. One example is the sensor nodes that are deployed in the radioactive surroundings and they are networked together for environment surveillance. In order to increase the lifespan of such application, the energy harvesting technologies [7-9] have been actively explored recently. Energy harvesting is considered as a promising method for overcoming the energy limitation for battery-powered systems and it could let systems achieve energy autonomy. Simply speaking, the energy harvesting system is a system that draws parts or all of its operating energy from its physical surroundings. Several prototypes have been proposed to demonstrate the effectiveness of energy harvesting system such as *Heliomote* [8] and *Prometheus* [9].

Several research works have been carried out in power minimization techniques for energy harvesting systems. An offline algorithm using dynamic voltage and frequency selection (DVFS) is proposed in [10] that targets at real-time tasks. The optimization is done by assuming that harvested energy from the ambient energy source is constant, which is not the case in real applications. The work in [11] chooses the solar power as the harvesting energy source and models it as time-variant. The energy source is assumed to work in two modes: daytime and nighttime. A lazy scheduling algorithm (LSA) is proposed in [12] that executes task as late as possible at full speed, in which the task slack is not exploited for energy savings.

In order to utilize the task slack for energy saving, the authors of [13] proposed an energy-harvesting-aware dynamic voltage and frequency selection (EA-DVFS) algorithm. The proposed algorithm slows down the task execution if the system does not have sufficient available energy; otherwise, the tasks are executed at the full speed. The main shortcomings of this work are:

1) The "sufficient available energy" is defined based on a single current task. As long as the remaining operation time of system at the full speed is more than the relative deadline of the task, then the system considers it has sufficient energy. However, there may be just as little as 1% energy left in the energy storage while the system can operate at full speed for more than the relative deadline of a task. Then EA-DVFS algorithm schedules the task at full speed. That is not the desired behavior.
2) When tasks are scheduled and operating voltages are selected, the EA-DVFS algorithm only considers one task instead of considering all tasks in the ready task queue. This results in that the task slacks are not fully exploited for energy savings.

In this paper we propose an adaptive task scheduling and DVFS algorithm for real-time energy harvesting systems. The goal of the proposed algorithm is to schedule all tasks in the ready queue at the lowest possible speed and allocate the workload to the processor as evenly as possible. The evenly distributed workload not only reduces the overhead from processor voltage and operating frequency switches, but also achieves system-wide energy efficiency [14]. The proposed algorithm also adaptively updates the scheduling and voltage/frequency selection when a new task arrives at the task ready queue. The main features of our approach can be summarized as follows,

1) It decouples the energy constraints and timing constraints for the real-time energy harvesting system so that the scheduling problem subjected to constraints both in timing domain and energy domain can be easily handled.
2) It fully explores the possibility of trading the task slack for

DAC'09, July 26-31, 2009, San Francisco, California, USA

energy saving by adaptively solving the problem when considering multiple tasks in the queue at the same time.

Comparing to the EA-DVFS algorithm, the proposed algorithm fully exploits the task slack for energy savings under timing and energy constraints. As long as the task can be slowed down for energy saving under given timing and energy constraints, the task is executed at a lower speed. The proposed algorithm results in more available/stored energy at any time, comparing to the EA-DVFS and LSA algorithms. Experimental results also show that, the proposed adaptive algorithm can significantly reduce the deadline miss rate under various processor utilizations; it also requires considerably less storage capacity for zero deadline miss rate, comparing to EA-DVFS and LSA algorithms.

The rest of this paper is organized as follows. The energy harvesting system model and some assumptions are presented in Section II. The proposed adaptive scheduling algorithm is described in Section III. Simulation results and discussions are presented in Section IV. Finally Section V gives the conclusions.

II. System Model and Assumptions

As shown in Figure 1, the energy harvesting system we consider in this paper consists of four major modules: energy source module, energy storage module, the uniprocessor module and the real-time task queue module. The energy source module harvests energy and feeds into the energy storage at power $Ps(t)$ at time instance t. The energy storage is the place to store energy and its capacity is denoted as C; the stored energy at time t is denoted by $E_C(t)$. When the stored energy reaches the capacity C, the incoming energy harvesting overflows the energy storage, so we have

$$0 \le E_C(t) \le C \quad \forall t \tag{1}$$

When the processor executes the real-time task, it draws energy from energy storage. If the energy storage is empty, the processor stops functioning.

2.1. Energy source

We denote $P_S(t)$ as the net power that the energy source feeds into the storage. The harvested energy $E_S(t_1, t_2)$ at time interval $[t_1, t_2]$ can be calculate by integration:

$$E_S(t_1,t_2) = \int_{t_1}^{t_2} P_S(t)dt \tag{2}$$

The power output of the energy source is a function of time, thus $P_S(t)$ can not be determined ahead. But we can predict it by tracing the energy source profile [15].

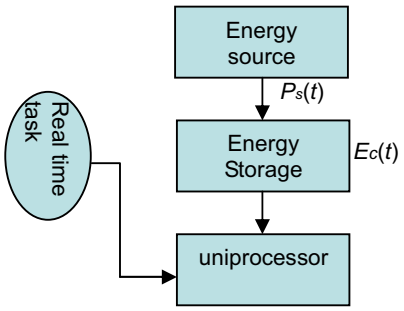


Figure 1 A real-time system with energy harvesting module.

2.2. Energy storage

The energy storage is assumed to be ideal. It can be fully charged and also fully discharged no matter how many charge/discharge cycles it has gone through. The energy that the processor demands only comes from the storage and harvested energy. Let $E_D(t_1, t_2)$ denote the processor energy dissipation from time t_1 to t_2, then we have:

$$E_D(t_1,t_2) \le E_C(t_1) + E_S(t_1,t_2) \qquad \forall t_1 < t_2 \tag{3}$$

Meanwhile, the stored energy should be the surplus that available energy deducts the energy dissipation by the processor if no overflow occurs, so we have:

$$E_C(t_2) \le E_C(t_1) + E_S(t_1,t_2) - E_D(t_1,t_2) \qquad \forall t_1 < t_2 \tag{4}$$

2.3. DVFS-enabled processor and real-time tasks

Assume the DVFS-enabled processor has N discrete operating frequencies f_n: $\{ f_n \mid 1 \le n \le N, \ f_{\min} = f_1 < f_2 < \ldots < f_N = f_{\max} \}$; and the power consumption with regards to f_n is denoted as P_n .

We define a slowdown factor S_n as the normalized frequency of f_n with respect to the maximum frequency $f_{\max}$, that is:

$$S_n = f_n / f_{\max} \tag{5}$$

For the sake of convenience, we use notations f_n, $f(n)$ interchangeably in this paper. Similarly for notations P_n & $P(n)$, and S_n & $S(n)$.

The triplet (a_m, d_m, w_m) is used for characterizing a real-time task τ_m, where a_m, d_m, w_m indicate the arrival time, the relative deadline and the worst case execution time of task τ_m, respectively. Before the real-time task τ_m is released, the triplet (a_m, d_m, w_m) is unknown. Once the task τ_m is released, the triplet is finalized, and τ_m is pushed into the ready task queue Q.

If task τ_m is stretched by a slowdown factor S_n, then its actual execution time at frequency f_n is w_m/S_n. All tasks are scheduled based on earliest deadline first (EDF) policy. The system is considered to be preemptive. The task with the earliest deadline has the highest priority and should be executed first; and it preempts any other task if needed.

III. Adaptive Scheduling Algorithm

In this section we will introduce the proposed adaptive scheduling and voltage/frequency selection algorithm for real-time systems with energy harvesting module. This algorithm dynamically adjusts the processor speed to achieve system-wide energy efficiency based on the workload and available energy information. The key point is that the proposed algorithm decouples the energy constraints and timing constraints originated from a real-time system so that the problem can be easily tackled. The framework of the proposed algorithm consists of three steps:

1) Create an initial schedule for all tasks in the ready task queue; that schedule is based on the lazy scheduling policy with tasks having earlier deadline having higher priority. This step guarantees that timing constraints of the real-time system are met.
2) Distribute the workload as evenly as possible on the processor; dynamic voltage and frequency selection (DVFS) policy is used for slowing down the processor so that the system power consumption goes down under given timing and performance constraints.
3) Tune up the scheduling from step (2) by taking into account the energy constraints. The schedule from step (2) is the energy efficient schedule under the timing constraints [14], but it does not consider the available energy for energy-harvesting system. If the schedule from step (2) is invalidated due to energy shortage, we do not simply remove the tasks. Instead an adaptive policy is adopted: if the system is able to harvest enough energy to finish the task under its given timing constraints, then the task is delayed until the system has sufficient energy; otherwise, the task is removed. Removing the task gives the system a chance to purely accumulate energy by harvesting, which improves the available energy for future tasks.

In the following part, we will explain each step in more details.

3.1 Generate an initial schedule

All tasks in the ready task queue Q are sorted ascendingly in terms

of the task deadline. The task with earliest deadline is put in the head of the queue, and the one with latest deadline in the tail of the queue. In the initial schedule, all tasks are executed at full speed. Then the lazy policy is used to schedule tasks in Q and tasks are always executed as late as possible. In other words, the task in the tail gets executed right at its deadline, and it starts being executed at the time instance when its deadline minuses its worst case execution time.

Assuming that there are M tasks in the task queue, and the first task is located in the head, the last one (M-th) in the tail. In order to get the initial schedule easily, the initial starting time (ist_m) and initial finishing time (ift_m) of each task τ_m (m=1,2,…, M) are calculated in a reversed order . Hence, ist_M and ift_M are calculated first, while ist_1 and ift_1 last.

Based on lazy scheduling policy, for the last task τ_M, we have:

$$ift_M = a_M + d_M \tag{6}$$

$$ist_M = a_M + d_M - wcet_M \tag{7}$$

For all other tasks left, the initial schedule is easily obtained by the following equations,

$$ift_m = \min(a_m + d_m, ist_{m+1}) \tag{8}$$

$$ist_m = \min(\max(a_m + d_m - wcet_m,\ \ a_m),\ \ ist_{m+1} - wcet_m) \tag{9}$$

where index variable m ranges from M-1 to 1. In order to make the schedule practical, the ist_m can not be smaller than a_m.

Note that a_m+d_m-$wcet_m$ is no less than a_m; otherwise task τ_m is not schedulable under the given timing constraint; so we have

$$ist_m = \min(a_m + d_m - wcet_m,\ \ ist_{m+1} - wcet_m) \tag{10}$$

The indication of the above equation is clear that task τ_m starts being executed either at time instance $a_m + d_m - wcet_m$, when its deadline minuses its worst case execution time, or at the time instance, ist_{m+1}-$wcet_m$, when the starting executing time of its next task ist_{m+1} minus its worst case execution time $wcet_m$, no matter which one is earlier. That scheduling is justified by the following facts: 1) task τ_m is delayed as much as possible so that system may have more energy to execute task by energy harvesting; 2) the timing constraints of task τ_m is guaranteed.

3.2 Balance workload

As long as each task (τ_m) is finished at its initial finishing time (ift_m), the timing constraint is met. However, based on the initial schedule, all tasks are executed at the full speed of the processor, which is not an energy-efficient scheme. We need to make use of the task slacks for energy saving. The dynamic voltage and frequency selection (DVFS) [13] is applied to stretch the execution time of each task and slow down the processor.

The DVFS-enabled processor has multiple operating voltage and frequency levels. In order to achieve the maximum power savings, all tasks should be stretched uniformly [14]. In other words, the processor should avoid operating frequency switches as much as possible.

In terms of the initial schedule, all tasks are executed at the full speed, with the same slowdown factor index SI_m equal to N. Then all tasks in the ready queues are stretched by N rounds of DVFS policy, where N is equal to the number of available operating frequencies to the processor. The effort of using N round of DVFS policy is to make all tasks executed in the same frequency level so that the switch activity of the processor is minimized and the system-wide energy efficiency is maximized.

For a given round, the starting time (st_m) of task τ_m for execution is determined by:

$$st_m = \begin{cases} \max(a_1, current\_time), & m = 1 \\ \max(a_m,\ \ ft_{m-1}), & m = 2, \cdots, M \end{cases} \tag{11}$$

where m is from 1 to M.

However, its finishing time (ft_m) is more complicated to obtain. Before calculating ft_m, two questions need to be answered. First, check if the slowdown factor index SI_m for task τ_m can be reduced further. If the following inequality holds,

$$st_m + wcet_m/S(SI_m\text{-}1) < ift_m \tag{12}$$

then the timing constraint is still met after further stretching task τ_m. Second, check if the slowdown factors for tasks indexed from m+1 to M is still valid. If the answers to these two questions are yes, then SI_m for task τ_m is decremented by 1; in other words, the operating frequency of task τ_m is reduced to $f(SI_m\text{-}1)$ from $f(SI_m)$. Otherwise, SI_m is kept as it is.

The slowdown factor S_n is called valid for a given task τ_m if task τ_m can be executed by the processor at frequency f_n subjected to the timing constraints.

Now ft_m can be easily calculated as:

$$ft_m = st_m + wcet_m/S(SI_m). \tag{13}$$

The workload balance algorithm is shown in Figure 2. Line 10 in Figure 2 tells us that the slowdown index SI_m for each task τ_m is decreased at most by 1 at a given round of DVFS. The meaning is two-fold: 1) each task has the same opportunity to be stretched, which avoids some tasks getting overstretched by squeezing out the slack of other tasks; 2) the slack time of tasks is sufficiently exploited for energy savings.

```
1.  Require: get the initial schedule for M tasks in queue Q
2.  for n = 1:N do
3.     for m =1:M do
4.        if m = = 1, then
5.           st_m = max(a_m, current_time)
6.        else
7.           st_m = max(a_m, ft_m-1)
8.        endif
9.        if st_m+wcet_m/S(SI_m-1) < ift_m && the slowdown factors
          for tasks with lower priority is valid, then
10.          SI_m = SI_m – 1
11.       end if
12.       ft_m = st_m + wcet_m/S(SI_m)
13.    end for
14. end for
```

Figure 2: Workload balance algorithm.

3.3 Check energy availability and tune up schedule

One of the features of the energy harvesting systems is that the available energy is limited by the energy storage capacity. The available energy dynamically fluctuates with time in two opposite directions: increasing or decreasing. Therefore we need to tailor the workload-balanced schedule to that feature.

When tasks are scheduled based on the workload-balanced algorithm in Section 3.2, the energy constraint is not considered. If the energy availability invalidates the schedule, then the processor has to stop the task execution before the task can be finished. In order to overcome that problem, we have to check the energy availability after getting the workload balanced schedule; then tune up the schedule.

If the workload-balanced schedule is invalidated by energy shortage, tasks are not directly removed from the ready task queue. Instead the task execution is first delayed.

For example, if we define that mth task τ_m in Q is the first task whose schedule is invalidated by the energy shortage, so we have:

$$E_C(st_m) + E_S(st_m, ft_m) < E_D(st_m, ft_m) \tag{14}$$

where $E_S(st_m, ft_m)$ is the harvested energy between st_m and ft_m, and it can be estimated based on the profile of energy-harvesting source. Then task τ_m is rescheduled by delaying dl_m until the following equality holds,

$$E_C(st_m) + E_S(st_m, ft_m + dl_m) = E_D(st_m + dl_m,\ \ ft_m + dl_m) \tag{15}$$

If the deadline of task τ_m is not violated, that is:

$$ft_m + dl_m \le a_m + d_m \tag{16}$$

and the slowdown factors for tasks indexed by m+1, …, M are still valid, then task τ_m is executed at time interval $[st_m+dl_m, ft_m+dl_m]$ at the frequency $f(SI_m)$, obtained from workload-balanced schedule; and the schedule for tasks with lower priority is updated, as shown in lines 7~10 in Figure 3; otherwise, task τ_m is simply removed from task ready queue, as shown in line 12 in Figure 3.

Note that the tune-up algorithm presented in Figure 3 is executed on the fly and the scheduler has to check the energy availability before the task's execution. We would like to give an example to explain how the tune-up algorithm works. Assuming that the DVFS-enabled processor has 4 operating frequency levels with slowdown factor 1, 0.6, 0.4 and 0.15; and the corresponding power levels are 32, 10, 4, and 0.8. Also assume that there are 2 tasks τ_1 and τ_2 in Q, and they are scheduled by the workload-balanced schedule with (st_1, ft_1, $deadline_1$)=(50, 56, 59), and (st_2, ft_2, $deadline_2$)=(56, 62, 68). Both tasks are scheduled to execute at the lowest speed, and the power consumption of the processor is 0.8 at lowest operating frequency.

```
1.  Require: the workload balanced schedule for M tasks in Q
2.  if E_C(st_m)+E_S(st_m, ft_m) < E_D(st_m, ft_m), then
3.     calculate dl_m from equation (16)
4.     if ft_m+dl_m ≤ d_m && the slowdown factors for tasks with
         lower priority is valid, then
5.        st_m = st_m+dl_m
6.        ft_m = ft_m+dl_m
          //update schedule for tasks with lower priority in for
          loop
7.        for i = m+1:M do
8.           st_i = max(st_i, ft_i-1)
9.           ft_i = st_i + exe_i;
10.       endfor
11.    else
12.       remove task τ_m from queue Q
13.    endif
14. endif
```

Figure 3: The tune-up schedule algorithm to guarantee the energy availability.

The available energy in the storage at time instance 50 is set to 1. The harvesting power from time instance 50 to 68 is set to 0.5. If the system executes those two tasks based on the workload balance schedule, then the execution of both tasks will be suspended due to the energy shortage. The following calculation verifies our conclusion: The total energy the system provides at time instance 56 is 1+6*0.5=4; and the total energy demand for executing task τ_1 is 6*0.8=4.8. So the energy shortage forces the processor to stop running at time instance 53.3 and the schedule for task τ_1 can not be carried out, shown by the "lime" color long dash line in Figure 4.

On the other hand, before running task τ_1, the energy availability is checked by equation (15), and then task will be delayed by 2 time units; accordingly task τ_1 is executed between time interval [52, 58], and the schedule for task τ_2 is updated as (st_2, ft_2, $deadline_2$)=(58, 64, 68). After finishing task execution, the remaining energy is 0.2 shown by the "lime" color solid line in Figure 4; energy is not a concern any more for the schedule of task τ_1. The similar argument holds for task τ_2.

3.4 Put all together

As we stated earlier, the proposed algorithm comprises three steps:

1) generate the initial schedule;
2) balance workload for the processor;
3) tune up the schedule under the constraints of energy availability.

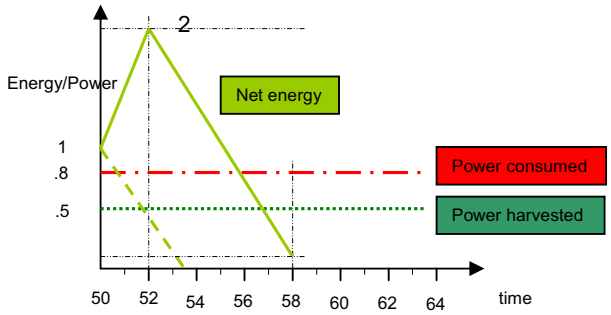


Figure 4: Tuning up scheduling algorithm illustration.

```
1.  Require: maintain a ready task queue Q
2.  set task queue Q empty
3.  while (true) do
4.     if new task coming, then
5.        push new task in Q,
6.        sort all tasks ascendingly in Q based on the deadline
7.        get initial schedule for tasks in Q,
8.        balance the workload based on algorithm shown in
          Figure 2
9.        tune up scheduling, shown in Figure 3
10.    endif
11.    execute task
12.    if the task is finished, then
13.       remove task from the ready task queue Q
14.    end if
15. end while
```

Figure 5: The proposed adaptive and voltage/frequency selection algorithm for real-time energy harvesting system.

In this section, we put the prior discussions together, and construct a complete adaptive scheduling and voltage/frequency algorithm for real-time energy harvesting system, presented in Figure 5.

In the beginning, we assume the ready task queue Q is empty, shown in line 2. Every time the new task comes, it is pushed into Q, as shown in line 5, and then all tasks in Q are sorted ascendingly based on their deadlines.

The event that the new task comes triggers rescheduling all tasks in Q, so that the task with higher priority (the earlier deadline) in the new task queue is scheduled to run earlier, as shown from lines 6~9 in Figure 5.

If there is no new task coming, the processor executes tasks based on the schedule obtained before, as shown in line 11. Once the task is finished, it is removed from Q, as shown in line 13.

The key of the proposed algorithm is in lines 6~9. The initial schedule guarantees that tasks meet the timing requirement. The workload balance scheduling algorithm achieves the system-level energy efficiency by two ways: 1) trading the task slack for energy savings by slowing down processor execution speed; and 2) the balanced workload reduces the frequency switch activities for the processor and then the correspondent overhead goes down. The tune-up algorithm makes sure the system has needed energy to execute each task based on the schedule.

IV. Simulations and Discussions

In this section, we evaluate the performance of the proposed scheduling algorithm based on simulations. We have developed a discrete event-driven simulator in C++ and implement the proposed scheduling algorithm. For comparison purposes, the lazy scheduling algorithm (LSA) in [12, 16] and energy aware DVFS algorithm (EA-DVFS) in [13] are also implemented as benchmarks.

We design three sets of experiments. The first set is designed to show how the system remaining energy is improved by the

proposed scheduling algorithm, comparing to LSA and EA-DVFS. The second set of experiments reports how much the deadline miss rate is reduced based on the proposed algorithm, comparing to the other two; the third set of experiments shows the minimum energy storage capacity requirement in order to maintain zero deadline miss rate for three different scheduling algorithms.

4.1 Simulation setup

We choose the solar energy as the energy harvesting source in our simulations. In order to exhibit the stochastic and periodic characteristics of the solar energy source, we multiply a random number generation function with two deterministic cosine functions together [13, 17], as shown in:

$$P_S(t) = \left| 10 \cdot N(t) \cdot \cos(\frac{t}{70\pi}) \cdot \cos(\frac{t}{120\pi}) \right| \qquad (17)$$

where $N(t)$ is a random number generator and it is subject to the normal distribution with mean 0 and variance 1.

A DVFS-enabled processor similar to Intel's Xscale processor [18] is used in the simulations. The processor has five operating frequencies: 150MHz, 400MHz, 600MHz, 800MHz and 1000MHz. Accordingly, the processor also has 5 power levels: 80mW, 400mW, 1000mW, 2000mW and 3200mW, each corresponding to a specific operating frequency. The overhead from the processor operating frequency switching is ignored in the simulations.

The task set consists of limited specific periodic tasks and the cardinality of the task set is arbitrary. The period of the specific task is uniformly drawn from a set {10, 20, 30, …, 120}; the relative deadline of the task is set to its period; and the worst case execution time is calculated based on its period and harvesting power. Assume that the average harvesting power is $\overline{P_S}$, and the task period is p, the worst case energy consumption e of the task is uniformly drawn from interval $[0, \overline{P_S} \cdot p]$; so e is a sample of a uniform-distributed random variable with distribution $[0, \overline{P_S} \cdot p]$. Then the worst case execution time of the task can be computed as $e/P_{\max}$.

We define the processor utilization U as

$$U = \sum_m \frac{w_m}{p_m} \qquad (18)$$

where w_m is the worst case execution time of task τ_m, and p_m is the period. The processor utilization stands for the ratio of its busy time over the summation of its busy time plus its idle time when the processor operates at full speed. So the utilization U cannot be larger than 1. To obtain a specific U, we need scale the worst case execution time of each task in a task set in the same ratio in some cases.

The energy storage is assumed to be full in the beginning of the simulation. The simulation terminates after 10,000 time units. For a specific utilization, we repeat experiments for 5,000 task sets.

4.2 Remaining energy comparison

In this set of experiments, we focus on comparing the remaining energy for systems using proposed scheduling algorithm, against systems using two baseline scheduling algorithms, LSA and EA-DVFS. In order to have fair comparison, all simulations are performed under the same condition, except for scheduling algorithms.

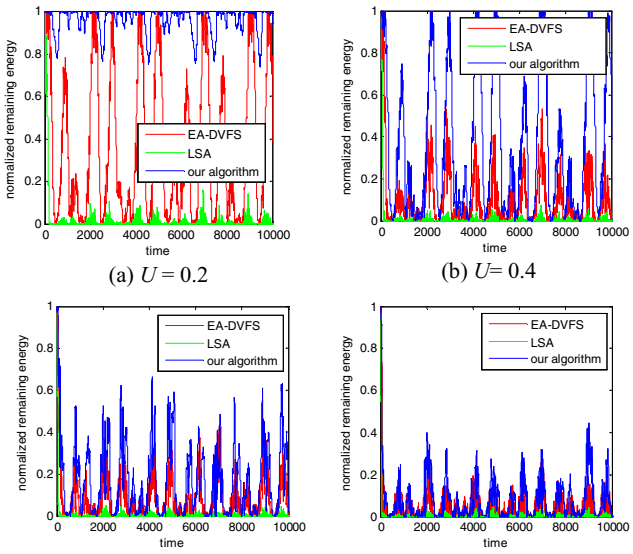
From the discussion of the proposed scheduling algorithm, we know that the processor utilization has significant impact on the remaining energy. We do experiments sweeping utilization U from 0.2 to 0.8 with a step of 0.2.

When the utilization U is given, all experiments show the similar trend in the remaining energy, we present the results with 6 periodic tasks in a task set. Note that the reported remaining energy is normalized to the storage capacity.

The simulation results with 4 different processor utilization ratios are plotted in Figure 6. As shown in Figure 6, the proposed algorithm has the most available remaining energy in various processor utilization, comparing to the other algorithms.

When utilization is set to 0.2, the proposed algorithm is able to keep the energy storage almost full for most of the time shown in Figure 6(a). The reason is that most of tasks are scheduled at the lowest speed and the energy consumption goes down considerably; furthermore, the harvesting energy replenishes to the energy storage. The EA-DVFS algorithm only slows down task execution when the system available energy is not sufficient. Therefore, some tasks are executed at full speed, and the others at reduced speed, which leads to the EA-DVFS-based system has less available energy. LSA-based system always executed tasks at full speed, and it accordingly has least available energy at any time.

With the processor utilization increasing from 0.4 to 0.8, the proposed algorithm has less task slack to take advantage for energy savings, therefore the available energy difference among LSA, EA-DVFS and the proposed algorithm decreases, as shown in Figure 6(b-d).



(a) U = 0.2 (b) U= 0.4

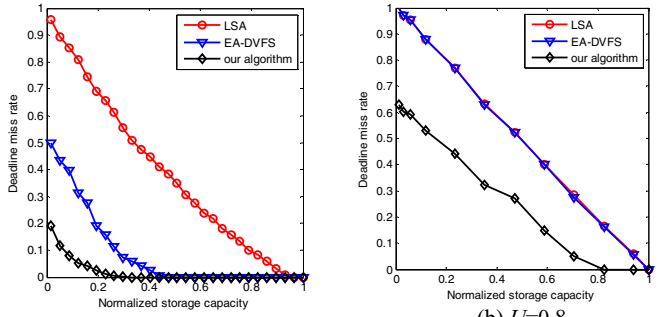
(c) U=0.6 (d) U=0.8

Figure 6: Remaining energy with different utilizations.

4.3 Deadline miss rate comparison

From the last sub-section, we know that the system based on the proposed scheduling algorithm has significantly more available energy than LSA-based/EA-DVFS-based systems no matter the processor utilization is low or high. The more available energy helps reduce the deadline miss rate due to energy shortage for future tasks. Hence, the proposed algorithm significantly decreases the deadline miss rate, comparing to LSA-based/EA-DVFS-based system, as shown in Figure 7, when U is set to 0.4 and 0.8, respectively.

Note that when utilization is set to 0.8, EA-DVFS performs the same as LSA, as shown in Figure 7(b); nevertheless our algorithm is able to reduce the deadline miss rate by at least 23% on average, comparing to LSA and EA-DVFS. When utilization is high, EA-DVFS schedules almost all of tasks at the full speed, and it can not achieve much energy efficiency. However, the proposed algorithm is able to squeeze out the task slack for energy savings and still improves the system energy efficiency. Hence, the proposed algorithm decreases the deadline miss rate even when the processor utilization is high.



(a) U=0.4 (b) U=0.8

Figure 7: Deadline miss rate with two workloads.

4.4 Storage capacity comparison

Finally, we compare the minimum storage capacity requirement for three scheduling algorithms in order to avoid any deadline miss for any task. Notations $C_{\text{min-LSA}}$, $C_{\text{min,EA-DVFS}}$ and $C_{\text{min,our}}$ represent the minimum storage requirement for LSA, EA-DVFS and our algorithm, respectively. They are all normalized to $C_{\text{min-LSA}}$.
We sweep the processor utilization U from 0.2 to 0.8 to run experiments with a step 0.2. The results are reported in table 1.

Table 1 Normalized storage capacities.

U	0.2	0.4	0.6	0.8
$C_{\text{min-LSA}}$	1	1	1	1
$C_{\text{min,EA-DVFS}}$	0.43	0.73	0.92	0.98
$C_{\text{min,our}}$	0.04	0.34	0.62	0.79

To visualize the storage capacity difference, the normalized storage capacities for LSA , EA-DVFS and the proposed algorithm, are presented in a bar chart in Figure 8.

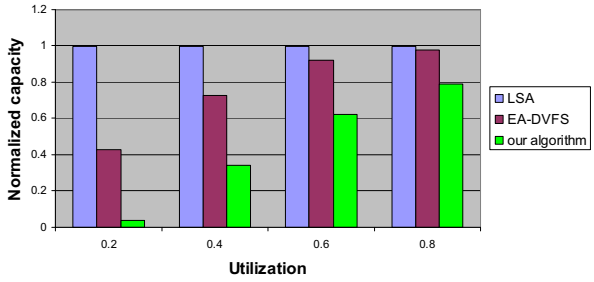


Figure 8: Normalized storage capacity comparison

Observing the results in Table 1 and in Figure 8, we can see that the proposed algorithm requires less storage capacity to achieve zero deadline miss rate in all cases. When utilization is set to 0.2, our algorithm needs as little as 1/10 of the storage capacity that EA-DVFS needs, and as little as 1/23 of the storage capacity that LSA needs. With the utilization going up, the difference between $C_{\text{min-LSA}}$, $C_{\text{min,EA-DVFS}}$ and $C_{\text{min,our}}$ reduces. Note that when utilization is set to 0.8, $C_{\text{min,our}}$ is about 21% less than $C_{\text{min-LSA}}$, while $C_{\text{min-LSA}}$ and $C_{\text{min,EA-DVFS}}$ are almost the same.

Whether the processor utilization is high or low, the proposed algorithm aggressively trades task slack for energy savings under timing and energy constraints. When utilization is low, the processor are scheduled to run task at lower speed; while utilization high, the task are executed at higher speed but not full speed. Under any circumstances the proposed algorithm saves more energy than LSA and EA-DVFS algorithms. That's why the proposed algorithm requires less storage capacity in all cases.

V. Conclusions

In this paper we have proposed an adaptive scheduling and voltage/frequency selection algorithm targeting at real-time systems with energy harvesting capability. The proposed algorithm consists of three steps:1) generate initial schedule; 2) balance workload; and 3) check energy availability for each scheduled task and tune up the schedule. The first step is to guarantee the timing constraints are met; the second step is to trade task slack for energy savings and the third step is to make sure the energy constraints are met. By dividing the original scheduling problem into three steps, we separate the constraints in timing and energy domains. So the problem can be easily handled.

Experimental results show that, the proposed algorithm aggressively trade the task slacks for energy saving. Hence, no matter the processor utilization is high or low, the proposed algorithm, comparing to LSA and EA-DVFS, increases the system available energy, decreases the deadline miss rate and reduces the energy storage capacity requirement for zero deadline miss rate.

References

[1] Lu, Y.-H., Benini, L., and De Micheli, G, "Low-power task scheduling for multiple device", Proc. Int. Workshop HW/SW Co-design, Mar.2000, pp. 39–43

[2] Mishra, R., Rastogi, N., Zhu, D., Mosse, D., and Melhem, R, "Energy aware scheduling for distributed real-time systems", Proc. Int. Parallel & Distributed Processing Symp., Apr. 2003

[3] S. Liu, Q. Qiu, Q. Wu, "Task merging for dynamic power management of cyclic applications in real-time multi-processor systems", *in Proc of ICCD*, 2006.

[4] F. Yao, A. Demers, et al,"A scheduling model for reduced CPU energy," in *IEEE symposium on Foundations of Comp. Science*, 1995.

[5] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. srivastava. "Power Optimization of Variable-Voltage Core-Based systems," *IEEE Trans. On Computer-Aided Design*, 1999.

[6] J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," *In Proc. Of Int. Conf. on VLSI Design*, pp.719-726, 2002

[7] S. Roundy, et.al, "Power sources for wireless sensor networks,". In *Proc. Of Wireless Sensor Networks, First Europeean Workshop*, 2004.

[8] V. Raghunathan, A. Kansal, et al, "Design considerations for solar energy harvesting wireless embedded systems", *In Proc. of the International Symposium on Information Processing in Sensor Networks*, 2005

[9] X. Jiang, J. Polastre, and D. E. Culler, "Perpetual environmentally powered sensor networks", *In Proc. of the International symposium on Information Processing in Sensor Networks* , 2005

[10] A. Allavena and D. Mosse, "Scheduling of frame-based embedded systems with rechargeable batteries," *In Workshop on Power Management for Real-time and Embedded Systems*, 2001

[11] C. Rusu, R. G. Melhen, and D. Mosse, "Multi-version scheduling in rechargeable energy-aware real-time systems", *In 15th Euromicro Conference on Real-time systems,* ECRTS 2003, , 2004.

[12] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy-harvesting sensor ndoes," *in Fifth Working Conference on Distributed and Parallel Embedded Systems* , 2006

[13] S. Liu, Q. Qiu, Q. Wu, "Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting", *In Proc. of DATE* 2008, 236-241.

[14] C. Xian, Y. Lu, Z. Li, "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time", *In Proc. of DAC 2007*: 664-669

[15] A. Kansal, J. Hsu, S. Zahedi and M. Srivastava, "Power Management in Energy Harvesting Sensor Networks," In *ACM Transactions on Embedded Computing Systems (in revision)* , 35 pages , May 2006.

[16] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," *in Proc. of the 18th Euromicro Conference on Real-time Systems* (ECRTS06),, 2006.

[17] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-Time Scheduling for Energy Harvesting Sensor Nodes,"*MICS Scientific Conference and SNF Panel Review*, 2006.

[18] Intel-Xscale Micro-architecture, available at **http://www.intel.com**