Harvesting-Aware Power Management for Real-Time Systems With Renewable Energy

Shaobo Liu, *Student Member, IEEE*, Jun Lu, *Student Member, IEEE*, Qing Wu, *Member, IEEE*, and Qinru Qiu, *Member, IEEE*

***Abstract*—In this paper, we propose a harvesting-aware power management algorithm that targets at achieving good energy efficiency and system performance in energy harvesting real-time systems. The proposed algorithm utilizes static and adaptive scheduling techniques combined with dynamic voltage and frequency selection to achieve good system performance under timing and energy constraints. In our approach, we simplify the scheduling and optimization problem by separating constraints in timing and energy domains. The proposed algorithm achieves improved system performance by exploiting task slack with dynamic voltage and frequency selection and minimizing the waste on harvested energy. Experimental results show that the proposed algorithm improves the system performance in deadline miss rate and the minimum storage capacity requirement for zero deadline miss rate. Comparing to the existing algorithms, the proposed algorithm achieves better performance in terms of the deadline miss rate and the minimum storage capacity under various settings of workloads and harvested energy profiles.**

***Index Terms*—Dynamic voltage and frequency selection (DVFS), embedded system, energy harvest, power management, real-time, task scheduling.**

I. Introduction

LOW power design remains one of the central issues for VLSI systems design, and it is particularly true for portable devices. Over the past decade, various power management techniques [1]–[6] have been developed to improve energy efficiency and prolong the operation time of battery-powered systems, subject to the timing and performance constraints. These conventional techniques can be classified into two categories based on the nature of energy dissipation reduction.

One of them is dynamic power management (DPM) [1]–[3], which achieves energy efficiency by switching the active component to the low power state or shutting down the idle components; the other is dynamic voltage and frequency selection (DVFS) [4]–[6], which lowers the operating frequency of the processor and reduces the energy dissipation.

Although DPM and DVFS techniques are both effective in reducing the system energy dissipation, any portable device will eventually exhaust the battery. Replacing the battery is required before the device can continue functioning. However, in some applications, replacing battery is either costly or impractical. Wireless sensor network is one of such applications. The sensor nodes are deployed in a wide wild area for environment surveillance and the deployment of sensor nodes contributes the majority of the cost for building the networked sensor nodes. Hence, ideally such a system should be designed to operate perpetually. With the battery being the only energy source, however, such design objective cannot be achieved.

Great interest has risen in powering these systems with renewable energy sources. *Renewable energy* is energy generated from natural resources such as sunlight, wind, rain, tides, geothermal heat, etc., which are naturally replenished. *Energy harvesting* (or *energy scavenging*) [7] refers to the process of collecting and converting renewable energy so that it can be utilized by electronic systems. Energy harvesting is a promising technology for overcoming the energy limitations of battery-powered systems and has the potential to allow systems to achieve energy autonomy. Several prototypes such as the *Heliomote* [8] and the *Prometheus* [9] have been designed to reveal the superiority of energy harvesting system.

Many technical challenges lie ahead in order to make an energy harvesting system work effectively. Among them is to develop novel power management methods and algorithms dedicated to energy harvesting systems, considering the following distinct features, comparing to the traditional battery-power systems.

1) An energy harvesting system is able to recharge its battery by the harvested power from the environmental energy source such as sunlight, wind, etc.
2) The energy source may present some type of periodic property. For instance, the sunlight has the high intensity at daytime and reduces to zero at nighttime.
3) The energy source is unstable and changing from time to time. The harvested power should be modeled as a time-varying variable. Some energy sources display both stochastic and periodic characteristics.
4) The uncertainty of energy availability. In a battery powered system, we are certain about how much energy is left in the storage for use. But for an energy harvesting system, we do not know beforehand exactly how much energy can be utilized by the system.

Manuscript received July 06, 2010; revised December 21, 2010 and May 10, 2011; accepted May 22, 2011. This work was supported by the National Science Foundation under Grant CNS-0845947.

S. Liu is with Marvell Semiconductor, Marlborough, MA 01752 USA (e-mail: lewtiob@gmail.com).

J. Lu and Q. Qiu are with the Electrical and Computer Engineering Department, State University of New York, Binghamton, NY 13902 USA (e-mail: jlu5@binghamton.edu; qqiu@binghamton.edu).

Q. Wu is with Air Force Research Laboratory, Rome, NY 13441 USA (e-mail: qwu2000@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2159820

The energy harvesting systems are exposed to new problems that do not exist in the conventional battery-powered systems. The conventional task scheduling and power management techniques are not designed for the energy harvesting systems and cannot handle the uncertainty in available energy. It is important to develop the novel power management techniques so that the energy harvesting systems are able to operate energy-efficiently and achieve energy autonomy.

In this paper, we propose a harvesting-aware scheduling algorithm for energy harvesting real-time embedded systems, which is designed to achieve the following two major objectives:

1) to schedule all tasks at the lowest possible speed and allocate the workload to the processor as evenly (over time) as possible;
2) to avoid the waste of harvested energy by preventing overflowing the energy storage.

Scheduling an evenly distributed workload not only reduces the delay and power overhead of processor voltage and operating frequency switches, but also allows DVFS techniques to achieve lowest energy dissipation [15]. Moreover, the overflow energy can be utilized for better performance instead of being simply wasted.

The major technical contributions of the proposed harvesting-aware algorithm can be summarized as follows.

1) It decouples the energy and timing constraints for the optimization process so that the power management and task scheduling algorithm can be designed with low complexity.
2) It fully explores the possibility of trading the task slack for energy saving by adaptively solving the problem when considering multiple tasks in the queue at the same time.
3) It adaptively reschedules tasks when the system predicts the overflow of energy storage will occur so that the system can take advantage of overflow for better performance.
4) The task scheduling and DVFS decisions are based on short term prediction of the energy harvesting rate. Three common techniques in online time series prediction are compared for their impact on the algorithm design and system performance.

The rest of this paper is organized as follows. Section II introduces the related research works. The energy harvesting system model and some assumptions are described in Section III. Section IV presents three *real-time sequence prediction* algorithms for predicting future energy availability. Section V introduces the proposed adaptive scheduling algorithm. Simulation results and discussions are presented in Section VI. Finally Section VII summarizes this paper.

II. Related Work

Energy harvesting system design recently has received substantial interests. Design considerations for energy harvesting systems are surveyed by the authors of [8] and [10]. Several techniques are proposed to maximize the rewards of the energy harvesting system in [18]–[22]. The authors assume that energy is consumed for obtaining certain level of service, measured in reward; and they focus on how to allocate and consume the energy such that the overall reward is maximized. However these techniques do not target at real-time systems and applications. In order to overcome this limitation, other methods are developed aiming at scheduling and power management techniques for energy harvesting real-time systems [11]–[14] to achieve better system performance and energy efficiency. An offline algorithm using DVFS is proposed in [11]. The optimization is implemented by assuming that harvested energy from the ambient energy source is constant, which is not the case in real applications. The work in [12] chooses the solar power as the harvesting energy source and models it as time variant. The solar energy source is assumed to work in two modes: daytime and nighttime. A lazy scheduling algorithm (LSA) is proposed in [13] and [17] that task execution is optimized based on as late as possible policy, however the task slack is not exploited for energy savings and DVFS was not considered.

In order to utilize the task slack for energy saving, the authors of [14] proposed an energy-aware DVFS (EA-DVFS) algorithm. It slows down the task execution if the system does not have sufficient available energy; otherwise, the tasks are executed at full speed. The main shortcomings of this work are as follows.

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 a task without depleting the energy. Then EA-DVFS algorithm schedules the task at full speed. That is not the desired behavior.
2) When the tasks are scheduled and the 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.

To further improve system performance and energy efficiency, we propose a harvesting-aware DVFS (HA-DVFS) algorithm in this paper. The HA-DVFS algorithm slows down the task execution whenever possible for energy saving. Meanwhile it will speed up the task execution in case of overflowing harvested energy. By speeding up the task execution using overflow energy, the current task will finish earlier than its scheduled time, giving the succeeding tasks more slack time to be slowed down further for energy saving. In this way, we not only prevent the waste of harvested energy, but also improve future energy savings.

Comparing to the LSA and EA-DVFS algorithms, the HA-DVFS 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 will be executed at a lower speed. Experimental results show that, comparing to the LSA and EA-DVFS algorithms, the HA-DVFS algorithm significantly reduces the deadline miss rate under various settings of processor utilizations and energy harvesting profiles. Also under this algorithm, the system requires less storage capacity for zero deadline miss rate.

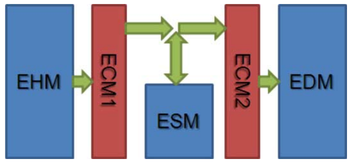


Fig. 1. Real-time system with energy harvesting.

III. System Model and Assumptions

As shown in Fig. 1, the energy harvesting real-time system under study consists of three major modules: energy harvesting module (EHM), energy storage module (ESM), and energy dissipation module (EDM). Two energy conversion modules (ECM) are used to regulate the voltage to the range which could be used by ESM and EDM. The required energy by EDM is drawn either from the energy source or energy storage or both.

A. Energy Harvesting Model

We denote $P_H(t)$ as the net output power from the energy source. The harvested energy $E_H(t_1, t_2)$ at time interval $[t_1, t_2]$ can be calculated by

$$E_H(t_1, t_2) = \int_{t_1}^{t_2} P_H(t)dt. \tag{1}$$

The output power of the energy source (i.e., $P_H(t)$) depends on many parameters that vary from time to time. For example, the output power of a solar panel depends on the sun irradiation level, the operation temperature, and the angle of the sun. Thus it is not a constant value. But we can predict it either based on profiled information [15] or using a time series prediction algorithm [24]. At a given environmental condition (e.g., sunlight intensity), the energy harvesting device, such as a solar panel, usually has a "maximum power point" (MPP). It can be reached by proper control and load matching [23]. In this paper, we assume that the MPP tracking is controlled by the energy conversion module connecting to the output of the EHM and the energy harvesting device always works at the MPP.

B. Energy Storage Model (ESM)

The ESM is usually a rechargeable battery or an ultracapacitor with limited capacity, which is denoted by E_{cap}. The stored energy at time t is denoted by $E_C(t)$. When the stored energy reaches the capacity E_{cap}, the incoming harvested energy overflows the energy storage. We also define two threshold energy level, *energy threshold low* ($E_{\text{th-low}}$) and *energy threshold high* ($E_{\text{th-hi}}$). When $E_C(t)$ is below or equal to $E_{\text{th-low}}$ the system is in energy depletion and the processor will enter energy saving sleep mode. The amount of remaining energy in the ESM is reserved to save the memory content and to switch the device to sleep mode. The device will be turned on when the energy in the battery reaches $E_{\text{th-hi}}$. Based on the definition, during the normal operation mode we have

$$E_{\text{th-low}} \leq E_C(t) \leq E_{\text{cap}} \quad \forall t. \tag{2}$$

There is always energy overhead when charging or discharging energy storage [28]. We model this overhead by a parameter called *charging/discharging efficiency* and denote it as ν. Due to the overhead of charging and discharging the battery, our first choice is to power the EDM using the power coming from the EHM. If the EHM generates more energy than needed by the EDM, then the extra energy will be stored in the ESM. On the other hand, if the EHM cannot provide enough energy for the EDM, then the remaining energy will be drawn from the ESM.

Let $E_D(t_1, t_2)$ denote the processor energy dissipation during a given time interval $[t_1, t_2]$ and $E_S(t_1, t_2)$ denotes the change of the remaining battery capacity from time instance t_1 to t_2, i.e., $E_s(t_1, t_2) = E_C(t_2) - E_C(t_1) + E_l(t_1, t_2)$, where $E_l(t_1, t_2)$ is the leakage energy of the storage from time t_1 to t_2. If $E_D(t_1, t_2) > E_H(t_1, t_2)$ then the EDM will be powered by both the EHM and the ESM and we have

$$E_D(t_1, t_2) = E_H(t_1, t_2) - \eta E_S(t_1, t_2) \quad \forall t_1 < t_2. \tag{3}$$

Note that, because the battery is in discharge mode during the time interval $[t_1, t_2]$, $E_S(t_1, t_2)$ is a negative value and it can be calculated as the following:

$$E_S(t_1, t_2) = \frac{(E_H(t_1, t_2) - E_D(t_1, t_2))}{\eta}. \tag{4}$$

On the other hand, if $E_D(t_1, t_2) < E_H(t_1, t_2)$ then the battery will be charged. If there is no overflow, then the change of the remaining battery capacity should be computed from the following equation:

$$E_S(t_1, t_2) = \eta\left(E_H(t_1, t_2) - E_D(t_1, t_2)\right) \quad \forall t_1 < t_2. \tag{5}$$

C. Energy Dissipation Model

Assume that the DVFS-enabled processor has N discrete operating frequencies $f_n : \{fn | 1 \leq n \leq N, f_{\min} = f_1 < f_2 < \cdots < f_N = f_{\max}\}$; and the power consumption correspondent to clock frequency f_n is denoted as P_n. Here P_n is the overall power consumption of the EDM which is a combination of both dynamic power consumption and leakage power consumption.

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

$$S_n = \frac{f_n}{f_{\max}}. \tag{6}$$

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

The triplet (a_m, d_m, w_m) is used for characterizing a real-time task τ_m, where a_m, d_m, and 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, its actual execution time at frequency f_n will be w_m/S_n. Initially 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.

D. Energy Conversion Modules

As shown in Fig. 1, we consider two electrical energy conversion units in the energy harvesting real-time system. The ECM1 converts energy from the output of the EHM so that it can be used by the ESM. Depending on the type of energy harvesting technology, ECM1 can be either DC/DC or AC/DC converter. ECM2 is usually a DC/DC converter that regulates the supply voltage level of the EDM. For DVFS-enabled processors, the output voltage of ECM2 should be controllable.

IV. Predicting Harvested Energy

Accurate prediction of the near-future harvested energy is crucial to effective power management of the energy harvesting system. It has been acknowledged in [8], [16], and [24] that the efficiency of the optimization techniques of energy harvesting system largely depends on the accuracy of the energy harvesting profiling and prediction. A good prediction model for the energy harvesting system must have high accuracy, low computation complexity and low memory requirement. Some examples of simple energy prediction models can be found in [29]–[31].

In this paper, we investigate three different time series prediction techniques that meet the above mentioned conditions.

A. Regression Analysis

Regression analysis [26] is a statistical technique for modeling and investigating the relationship among observed variables. It is used to estimate and predict the value of one variable by taking into account the other related. Forecasting using simple regression is based on

$$x = b_0 + b_1 z + \varepsilon. \tag{7}$$

It is proven that the minimum least square estimation of b_0 and b_1 are given by the following equations:

$$\widehat{b_1} = \frac{\sum_{i=1}^{n}(z_\mathrm{i} - \overline{z})\sum_{i=1}^{n}(x_\mathrm{i} - \overline{x})}{\sum_{i=1}^{n}(z_\mathrm{i} - \overline{z})^2} \tag{8}$$

$$\widehat{b_0} = \overline{x} - \widehat{b_1}\overline{z} \tag{9}$$

where $(x_1, z_1), (x_2, z_2), \ldots, (x_n, z_n)$ are the n observations available. In this paper, they are the sunlight intensity and time, respectively. And $\overline{x}$ and $\overline{z}$ are the arithmetic mean of conrespondent x and z.

The fitted simple linear regression model is

$$\widehat{x} = \widehat{b_0} + \widehat{b_1} z. \tag{10}$$

B. Moving Average

The main objective of the *moving average* technique [26] is to predict future values based on averages of the past values. It is useful in reducing the random variations in the observation data. The simple moving average uses the N past observation data to calculate the next predicted time series value, as shown in (11)

$$\widehat{x} = \frac{x_{(t)} + x_{(t-1)} + \cdots + x_{(t-N+1)}}{N}. \tag{11}$$

The property of simple moving average depends on the number of past observations to be averaged, but it gives equal weight to all past data, of which a large number to be stored and used for forecasts.

C. Exponential Smoothing

The exponential smoothing approach [27] is widely used for short-time forecasting. Although it also employs weighting factors for past values, the weighting factors decay exponentially with the distance of the past values of the time series from the present time. Simple exponential smoothing can be obtained by:

$$\widehat{x} = x_{e(t)} = \alpha x_{(t)} + (1-\alpha)x_{e(t-1)} \tag{12}$$

where $x_{e(t)}$ is called the exponentially smoothed value, $x_{(t)}$ is the observed value at the same point of time. The fraction α is called the smoothing constant, and $x_{e(t-1)}$ is the previous exponentially smoothed value.

The quantitative comparison on prediction techniques in terms of their impact on system performance will be discussed in Section VI.

V. Harvesting-Aware Scheduling Algorithm

In this section we introduce the proposed HA-DVFS algorithm. The algorithm adaptively adjusts the processor speed to achieve system-wide energy efficiency based on the workload and available energy information. One of the key principles of the approach is that it decouples the energy constraints and timing constraints originated from a real-time system so that the complexity of the algorithm is kept low. The framework of the proposed algorithm consists of the following steps.

1) Create an initial schedule for all tasks in the ready task queue; that schedule is based on the lazy scheduling policy where the task with earlier deadline has higher priority. This step guarantees that timing constraints of the real-time system are met if the task set is schedulable and preemptible.
2) Distribute the workload as evenly (over time) as possible on the processor; DVFS technique is applied for slowing down the processor so that the slack time of tasks is sufficiently exploited for energy savings.
3) Tune the scheduling from Step 2 by taking into account the energy constraints. The schedule from Step 2 is the energy efficient schedule for the timing constraints, 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, if the system is able to harvest enough energy to finish the task before its deadline, the task is delayed until the system has sufficient energy. Otherwise, the task is removed from the queue. Removing the task gives the system a chance to save more harvested energy for future tasks.
4) Speed up the task execution when the algorithm predicts that the overflow of energy storage will occur. By speeding up task execution, the extra harvested energy is utilized to transfer the slack time from the current task to the succeeding tasks. As a result, the future tasks have chances to be slowed down further to save more energy.

In the following sub-sections, we explain each step in detail.

A. Generate Initial Schedule

All tasks in the ready task queue Q are sorted in the ascending order according to their deadlines. 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 to be executed at full speed.

Then the lazy scheduling policy is used to schedule tasks in Q so that the tasks are executed as late as possible. In other words, the task in the tail will finish its execution right at its deadline. It starts being executed at the time instance that equals to its deadline minus 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 (Mth) in the tail. In order to get the initial schedule, the initial starting time (ist_m) and initial finishing time (ift_m) of each task $\tau_m(m = 1, 2, \ldots, M)$ are calculated in a reversed order. Hence, ist_M and ift_M are calculated first, while ist_1 and ift_1 last.

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

$$\mathrm{ift}_M = a_M + d_M \tag{13}$$

$$\mathrm{ist}_M = a_M + d_M - w_M. \tag{14}$$

In a reverse order, the initial schedules for other tasks are obtained by the following equations:

$$\mathrm{ift}_m = \min(a_m + d_m, \mathrm{ist}_{m+1}) \tag{15}$$

$$\mathrm{ist}_m = \min\left(\max(a_m + d_m - w_m, a_m), \mathrm{ist}_{m+1} - w_m\right) \tag{16}$$

where index variable m ranges from $M-1$ to 1. In order to make the schedule feasible, the ist_m cannot be smaller than a_m.

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

$$\mathrm{ist}_m = \min(a_m + d_m - w_m, \mathrm{ist}_{m+1} - w_m) \tag{17}$$

where $a_m + d_m w_m$ is the deadline of task τ_m minus its worst case execution time, and is the initial starting time of the next task (i.e., task τ_{m+1}) minus the worst case execution time of task τ_m. This 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 it by energy harvesting; 2) the timing constraint of task τ_m is guaranteed.

The procedure of generating the initial schedule is summarized in Algorithm 1. The time complexity of sorting algorithm in line 1 is $O(M \log M)$, and time complexity of the rest simple *for* loop is $O(M)$, therefore Algorithm 1 has a time complexity of $O(M \log M)$.

B. Balance Workload and Slow Down Task Execution

As long as each task (τ_m) finishes at its initial finishing time (ift_m), the timing constraint is met. However, in 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 by applying DVFS to stretch the execution time of each task with lower clock frequency and supply voltage for the processor.

Algorithm 1 Initial Scheduling

Require: M tasks in ready queue Q
1. sort tasks in the descending order of their deadline
2. **for** $m = M{:}1$ **do**
3. **if** $m == M$, **then**
4. $ift_m = a_m + d_m$,
5. **else**
6. $ift_m = \min(a_m + d_m, ist_{m+1})$
7. **end if**
8. $ist_m = ift_m - w_m$
9. **end for**

It is possible that some systems have mixture of tasks with or without DVFS potentials. For example, in a wireless sensor node, the sensing and communication operations usually cannot scale their operation frequency for energy saving while the digital signal processing operations can. A flag *stretchable* is introduced to indicate the DVFS capability of a task. The following discussion is mainly focused on those stretchable tasks. Those tasks that are not stretchable will always run at their full speed.

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.

Based on the initial schedule, all tasks are to be executed at the full speed, with the same slowdown factor index SI_m equal to N. In this step, all *stretchable* tasks in the ready queue are stretched by N rounds of DVFS policy, where N is the number of available operating frequencies to the processor.

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

$$st_m = \begin{cases} \max(a_1, \mathrm{current}_{\mathrm{time}}), & m = 1 \\ \max(a_m, ft_{m-1}), & m = 2, \ldots, M. \end{cases} \tag{18}$$

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

$$st_m + \frac{w_m}{S(SI_m - 1)} < ift_m \tag{19}$$

the timing constraint can still be met after further stretching task τ_m.

Second, we have to verify if the slowdown factors for tasks indexed from $m+1$ to M are still valid. If the answers to these two questions are yes, then SI_m for task τ_m is decremented by 1; thus the operating frequency of task τ_m is reduced to $f(SI_m - 1)$ from $f(SI_m)$. Otherwise, SI_m is kept unchanged.

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 the ft_m can be calculated as

$$ft_m = st_m + \frac{w_m}{S(SI_m)}. \tag{20}$$

The workload balance procedure is shown in Algorithm 2. Line 10 in Algorithm 2 tells us that the slowdown index SI_m for each task τ_m is decreased at most by 1 within a given round

Algorithm 2 Workload Balancing and DVFS

Require: Get the initial schedule for M tasks in queue Q

1. **for** n = 1:N **do**
2.   **for** m =1:M **do**
3.   **if** m == 1, **then**
4. $st_m = \max(a_m, current\_time)$
5.   **else**
6. $st_m = \max(a_m, ft_{m-1})$
7.   **end if**
8.   **if** (*stretchable flag*)
9.   **if** $st_m+w_m/S(SI_m-1) < ift_m$ && *no deadline miss for the other tasks*, **then**
10. $SI_m = SI_m - 1$
11.   **end if**
12.   **endif**
13. $ft_m = st_m + w_m/S(SI_m)$
14.   **end for**
15. **end for**

Algorithm 3 Scheduling Adjustment on Energy Availability

Require: The workload balanced schedule for M tasks in Q

1. **if** $E_C(st_m) - E_{th\text{-}low} + E_H(st_m, ft_m) < E_D(st_m, ft_m)$, **then**
2.   calculate dl_m from equation (23)
3.   **if** $ft_m + dl_m \le d_m$ && *the slowdown factors for tasks with lower priority is valid*, **then**
4. $st_m = st_m + dl_m$
5. $ft_m = ft_m + dl_m$
    //update schedule for succeeding tasks
6.   **for** $i = m+1$:M **do**
7. $st_i = \max(st_i, ft_{i-1})$
8. $ft_i = st_i + exe_i$;
9.   **end for**
10.   **else**
11.   remove task τ_m from queue Q
12.   **end if**
13. **end if**

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.

For a given processor, the number of available operating frequencies N is a constant, therefore the time complexity of Algorithm 2 is solely decided by the number of tasks M in the ready queue. In the inner for loop from line 2 to line 14, all other lines require constant time except line 9, when we need to check slowdown factor for $M-m$ tasks. Once we found that the slowdown factor for kth task is invalid, where $1 \le k \le M-m$, we no longer need to check for $(k+1)\text{th}, \ldots, (M-m)\text{th}$ tasks. In the best case, Line 8 just checks one task and finds its slowdown factor invalid. In the worst case, all m tasks are checked.

Let us define the random variable X_m to be the index of the first task whose slowdown factor checked to be invalid in the mth iteration. This means that X_m tasks have been checked in this round. Without loss of generality, we assume that the probability of event $\{X_m = k\}$ is $1/(M-m), \forall k \in [1, M-m]$. The mean value of checked tasks in this iteration is

$$E(X_m) = \sum_{k=1}^{M-m} k * \Pr(X = k) = \frac{1+M-m}{2}. \quad (21)$$

Thus the average number of tasks checked in Line 8 is bounded by $(1+M-m)/2$ for the mth iteration. The overall number of tasks that have been checked over M iterations is calculated as $\Sigma_{m=1}^{M}(1+M-m)/2 = (M^2+M)/4$. Therefore, the time complexity of the algorithm is $O(M^2/4)$.

C. Check Energy Availability and Fine-Tune Scheduling

One of the features of the energy harvesting systems is that the available energy not only is limited by the capacity of the energy storage, but also dynamically fluctuates with time due to the uncertainty in energy harvesting. In this step the HA-DVFS algorithm adaptively adjusts the task execution according to run-time energy availability.

When tasks are scheduled based on the workload-balanced algorithm in the previous subsection, the energy constraint is not considered. If the system energy reaches zero before a task finishes, the processor has to stop the task execution and the energy spent on that task is wasted. To avoid it, we have to check the energy availability on-the-fly and tune up the schedule adaptively.

If the schedule of a task is invalidated by energy shortage, it should not be removed immediately from the ready task queue. Instead we should try to delay the task's start time first.

For example, if we assume that the mth task τ_m in Q is the first task whose schedule is invalidated due to the energy shortage, which means

$$E_C(st_m) - E_{\text{th-low}} + E_H(st_m, ft_m) < E_D(st_m, ft_m) \quad (22)$$

where $E_S(st_m, ft_m)$ is the harvested energy between st_m and ft_m, and it can be estimated based on real-time prediction techniques discussed in Section IV or profiling of the energy harvesting source. Then task τ_m is rescheduled by delaying dl_m until the following equality holds:

$$E_C(st_m) - E_{\text{th-low}} + E_H(st_m, ft_m + dl_m) = E_D(st_m + dl_m, ft_m + dl_m). \quad (23)$$

In the above equations, $E_C(st_m) - E_{\text{th-low}}$ is the available energy for normal operations. If the deadline of task τ_m is not violated, that is

$$ft_m + dl_m \le a_m + d_m \quad (24)$$

and the slowdown factors for tasks indexed by $m+1, \ldots, M$ are still valid, then task τ_m is executed at time interval $[st_m + dl_m, ft_m + dl_m]$ at the frequency $f(SI_m)$; and the schedule for succeeding tasks is updated, as shown in Lines 6 $\sim$ 9 in Algorithm 3; otherwise, task τ_m is removed from task ready queue, as shown in Line 11. A binary search algorithm is used to find the delay time for the task τ_m, therefore the time complexity of Algorithm 3 is $O(M \log(a_m + d_m - st_m))$.

Note that the tune-up procedure presented in Algorithm 3 is executed on the fly and the scheduler has to check the energy availability before any task is about to start.

The following is 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,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

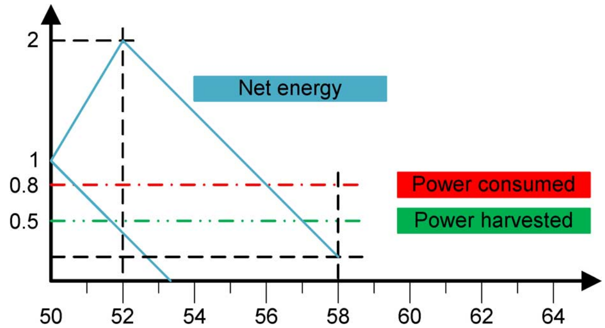


Fig. 2. Example of the adaptive tune-up algorithm.

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 $(st_1, ft_1, \text{deadline}_1) = (50, 56, 59)$, and $(st_2, ft_2, \text{deadline}_2) = (56, 62, 68)$. Both tasks are scheduled to execute at the lowest speed, and the power consumption of the processor is 0.8 at lowest operating frequency.

The available energy in the storage at time instance 50 is set to 1. To simplify the discussion, we assume that the efficiencies of the ESM and ECM are 1. We also assume that the energy threshold low $(E_{\text{th-low}})$ is 0. 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. Because the total energy the system provides at time instance 56 is $1 + 6 \times 0.5 = 4$; and the total energy needed for executing task τ_1 is $6 \times 0.8 = 4.8$. The energy shortage forces the processor to stop at time instance 53.3 and the schedule for task τ_1 cannot be carried out, shown by the "lime" color long dash line in Fig. 2.

On the other hand, before running task τ_1, the energy availability is checked by (22), 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 $(st_2, ft_2, \text{deadline}_2) = (58, 64, 68)$. After finishing task execution, the remaining energy is 0.2 shown by the "lime" color solid line in Fig. 2; energy is not a concern any more for the schedule of task τ_1. The similar argument holds for task τ_2.

D. Avoid Overflow and Transfer Slack Time

Due to the limited energy storage capacity, the harvested energy could overflow the storage in some cases, causing wasted energy. A good power management algorithm should recognize the overflow situation and try to prevent energy waste. Next we study when and how the overflowing energy can be utilized for better performance and more energy savings in the future.

Consider two tasks τ_m and τ_{m+1} in the task queue: the scheduling of these two tasks are shown in Fig. 3. Based on this scheduling, we would like to first introduce some observations, which establish the basis to simplify our discussions later.

Observation 1: Given a scheduling of two tasks in Fig. 3. For task τ_m, if its finishing time ft_m is earlier than the starting time st_{m+1} of its successor τ_{m+1}, the starting time st_{m+1} of task τ_{m+1} is solely determined by its arrival time a_{m+1}.

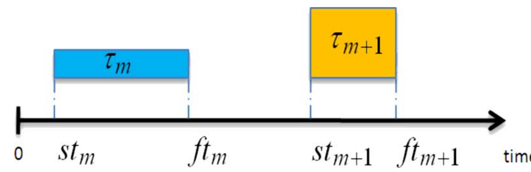


Fig. 3. Two scheduled tasks.

Proof: From (18), we know the starting time st_{m+1} of task τ_{m+1} is decided by: $st_{m+1} = \max(a_{m+1}, ft_m)$. Assume that ft_m is larger than a_{m+1}, then we have $st_{m+1} = ft_m$, which leads to a contradiction. Therefore, we have $st_{m+1} = a_{m+1}$. ■

From Observation 1, we know that task τ_{m+1} cannot be scheduled earlier than st_{m+1} even if task τ_m is finished before ft_m. Assume that the energy overflow occurs at some point between st_m and st_{m+1}, and the wasted energy through overflowing is E_O by time st_{m+1}. We claim that even if the overflowing energy E_O is utilized to speed up the execution of task τ_m, we cannot improve the system performance. The reasons are two-fold: 1) task τ_{m+1} cannot be scheduled earlier than st_{m+1}; 2) whether or not the E_O is used for speeding up the execution of τ_m, the energy storage is full at the time task τ_{m+1} is executed. Therefore, we cannot improve system performance in this situation. The above discussion is summarized in Observation 2.

Observation 2: Given the scheduling of two tasks in Fig. 3. If the energy overflow occurs at any time between st_m and st_{m+1}, the overflowing energy before time instance st_{m+1} cannot be exploited for performance improvement.

Conclusion: In order to utilize the overflowing energy for system performance improvement, both of the following two conditions must be satisfied:

1) the energy overflow must occur at the time when some task, say τ_m, is being executed;
2) the successors of τ_m should be able to get more slack time after the overflowing energy is utilized for speeding up the execution of task τ_m.

We know that *unusable overflow* cannot be traded for better performance; and no action is needed to deal with it. Hence, we only focus on the *usable overflow* in this paper.

Fig. 4 gives an example of the principle that usable overflow can be traded for better energy efficiency. There are two tasks τ_m and τ_{m+1} shown in Fig. 4. The original scheduling of these two tasks is presented in Fig. 4(a). Assume that the energy overflow occurs at some point between st_m and st_{m+1}. In order to avoid the overflow, task τ_m gets to run faster and finishes earlier than the original scheduling, as shown in Fig. 4(b). This gives more slack time to the successor τ_{m+1}, and it is executed at a lower frequency for energy saving. By the new scheduling, the system achieves better energy efficiency and improves performance. From this example we can see that the usable overflow is used as a media to transfer the slack to the future tasks. Fig. 4(b) shows that some slack for task τ_m is transferred to task τ_{m+1}. Since the slack transferring is enabled by the overflowing energy, the adjustment in scheduling does not reduce the available energy for future tasks. Therefore as long as the future tasks can be executed at lower frequency by utilizing this

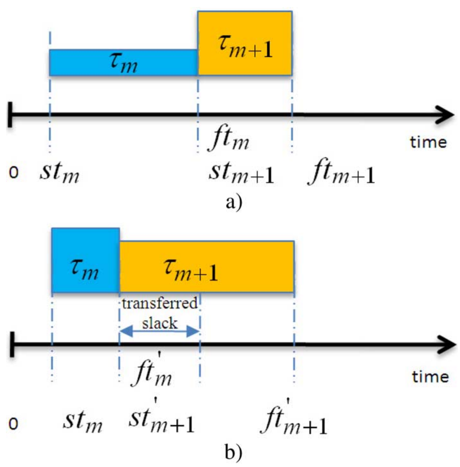


Fig. 4. Scheduling improvement with usable overflow. a) Original scheduling. b) Scheduling utilizing overflowing energy.

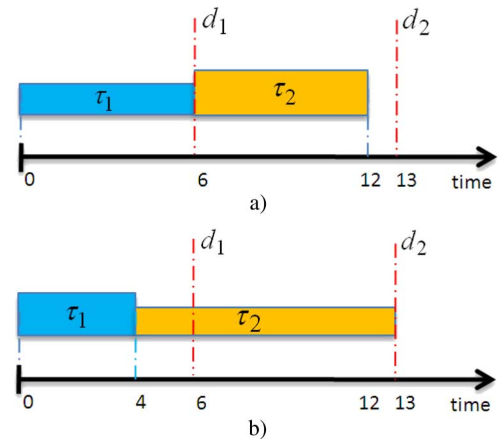


Fig. 5. Example of saving energy overflow. a) Scheduling with energy overflowing. b) Speed up the execution of task τ_1 to avoid overflowing.

transferred slack, this method achieves better energy efficiency for the system.

After qualitatively expounding why usable overflow improves system performance, we would like to quantitatively show how effective this mechanism can be through a concrete example.

Consider the following scenario: the processor can operate at high frequency f_H and low frequency f_L with $f_H = 1.5f_L$; and two correspondent power consumption levels P_H and P_L with $P_H = 2.5P_L$. The processor has two tasks $\tau_1 = (a_1, d_1, w_1) = (0, 6, 4)$ and $\tau_2 = (a_2, d_2, w_2) = (0, 13, 6)$ to execute. The harvesting power P_S is set to $1.2P_L$ during the time interval [0, 5], and 0 afterwards. The energy storage is set to be full $(E_{\rm cap})$ at time 0.

Tasks τ_1 and τ_2 are scheduled based on the workload balance algorithm, as shown in Fig. 5(a). From the scheduling we know that τ_1 is executed at low frequency f_L at time interval [0, 6) and τ_2 at high frequency f_H at time interval [6, 12). The energy storage is full in the beginning, and the harvesting power P_S, which is $1.2P_L$ at time interval [0,5), is larger than the demanded power P_L for executing task τ_1, so the energy overflows the storage at [0, 5). The wasted energy during overflow is $(1.2P_L - P_L) \times 5 = P_L$. After that, the energy is drawn from the storage. When task τ_1 is finished, the available energy is $E_{\rm cap} - P_L$. Then the system runs task τ_2, and the energy needed is $P_H \times 6 = 2.5P_L \times 6 = 15P_L$. The energy in the storage decreases to $E_{\rm cap} - 16P_L$ when task τ_2 finishes.

To save the wasted overflowing energy, we can speed up the execution of task τ_1, as shown in Fig. 5(b). It is executed at high frequency f_H from time instance 0 to 4 and task τ_2 at f_L during time interval [4, 13). Task τ_1 finishes before its deadline and task τ_2 finishes right at its deadline, as shown in Fig. 5(b). The power needed for executing task τ_1 is $2.5P_L$, which is larger than the harvested power P_S, so the harvested energy will not overflow the storage. The remaining energy after task τ_1 finishes is $E_C - (P_H - P_S) \times 4 = E_C - (2.5P_L - 1.2P_L) \times 4 = E_C - 5.2P_L$. Then the processor runs task τ_2 at low frequency f_L. From time instance 4 to 5, the system gains energy $(1.2P_L - P_L) \times 1 = 0.2P_L$ due to harvesting. After time 5, the energy storage is drawn at power P_L for executing task τ_2. The energy left in the storage is $E_C - 5.2P_L + 0.2P_L - P_L \times (9 - 1) = E_C - 13P_L$ when it finishes. Comparing to the scheduling in Fig. 5(a), the improved scheduling uses $(E_C - 13P_L) - (E_C - 16P_L) = 3P_L$ less energy.

Above example shows the basic mechanism of utilizing the "usable energy overflow" to transfer slack to future tasks to improve overall system energy efficiency. Saving energy also means better performance in fewer deadlines misses.

Our next step is to develop an algorithm to generalize this basic mechanism. Assume that task τ_m is scheduled to execute at time interval $[st_m, ft_m)$ with slowdown index SI_m. If the energy overflow occurs at some point between st_m and ft_m, we can calculate overall overflowing energy E_O until ft_m if no action is taken as follows:

$$E_O = E_C(st_m) + E_H(st_m, ft_m) - E_D(st_m, ft_m) - E_{\rm cap}. \tag{25}$$

In order to prevent energy overflow, ideally the operating frequency of task τ_m should be elevated to the level where E_O is "just" exhausted. However, the processor has discrete operating frequency-power levels and we may not be able to achieve it. So task τ_m should be executed at a new speed $f(SI_{m,new})$ where the needed extra energy is no less than E_O. That is

$$E_D\left(st_m, \frac{w_m}{f(SI_{m,\rm new})}\right) - E_D\left(st_m, \frac{w_m}{f(SI_m)}\right) \geq E_O \tag{26}$$

where $w_m/f(SI_{m,\rm new})$ is the new execution time, and $E_D(st_m, w_m/f(SI_{m,\rm new}))$ is the new energy dissipation for the task.

On the other hand, only part of E_O is used if task τ_m is executed at lower frequency $f(SI_{m,\rm new} - 1)$. So we have the following inequality:

$$E_D\left(st_m, \frac{w_m}{f(SI_{m,\rm new}) - 1}\right) - E_D\left(st_m, \frac{w_m}{f(SI_m) - 1}\right) < E_O. \tag{27}$$

In some cases, even if task τ_m is executed at the full speed $f_{\max}$, E_O cannot be exhausted; that is

$$E_D(st_m, w_m) - E_D\left(st_m, w_m/f(SI_m)\right) < E_O. \tag{28}$$

Algorithm 4 Utilizing Overflow Harvested Energy

Require: The workload balanced schedule for M tasks in Q
1. **if** $E_C(st_m)+E_H(st_m, ft_m) - E_D(st_m, ft_m) > E_{cap}$ & $a_{m+1} < ft_m$ **then**
2. calculate new operating frequency $f(SI_{m,new})$ for task τ_m
3. update $ft_m = st_m + w_m/S(SI_{m,new})$
4. compute transferred slack $slack_{tf}$
5. distribute $slack_{tf}$ among all tasks with lower priority than τ_m by calling *distribute_slack*($slack_{tf}$, τ_m)
6. **end**

Procedure: *distribute_slack*( )
Require: $slack_{tf}$, τ_m
7. **while** $slack_{tf} > 0$
8. **for** $k = m+1{:}M$ **do**
9. $st_k = \max(a_k, ft_{k-1})$
10. **if** $st_k+w_k/S(SI_k-1) < ift_k$ *&& the slowdown factors for tasks with lower priority is valid*, **then**
11. $SI_k = SI_k - 1$
12. **end if**
13. $ft_k = st_k + w_k/S(SI_k)$
14. update $slack_{tf}$
15. **end for**
16. **end while**

In this case we schedule task τ_m at the full speed $f_{\max}$.

After the new execution speed for task τ_m is decided, we need to compute the transferred slack slack_{tf} from task τ_m, which can be computed as

$$\mathrm{slack}_{tf} = \frac{w_m}{f(SI_m)} - \frac{w_m}{f(SI_{m,\mathrm{new}})}. \tag{29}$$

In order to maximize energy efficiency of the transferred slack time slack_{tf}, ideally slack_{tf} should be distributed such that all tasks succeeding τ_m have the same frequency assignment. We have the procedure $\mathrm{distribute\_slack}()$ to allocate the transferred slack for tasks $\tau_{m+1}, \tau_{m+2}, \ldots, \tau_M$, which is presented between Line 7 and Line 16 in Algorithm 4. This procedure is similar to the workload balance method used in Algorithm 2.

The overall algorithm handling the usable energy overflow is presented in Algorithm 4. Line 1 is used to decide two things: whether energy overflow occurs and whether the overflow is usable. If both are true, then the overflow energy is utilized to transfer slack time to future tasks, as shown in Lines 2 $\sim$ 5. Note that the starting time st_m of task τ_m keeps the same as before when it is executed at the new frequency $f(SI_{m,\mathrm{new}})$, but ft_m needs to be updated, as shown in Line 3. Task τ_m will finish earlier at the new assigned frequency, and the slack time from τ_m is transferred to tasks following τ_m, as shown in Line 5. By utilizing the transferred slack time, tasks succeeding τ_m can run at lower frequencies, as show in procedure $\mathrm{distribute\_slack}()$, so that the system achieves better energy efficiency. The time complexity of Algorithm 4 is similar to Algorithm 2, which is $\mathrm{O}(M^2)$ on average.

E. Overall HA-DVFS Algorithm

As we stated earlier, the proposed HA-DVFS algorithm comprises the following four steps:

Algorithm 5 Overall HA-DVFS Algorithm

Require: Maintain a ready task queue Q
1. set task queue Q empty
2. **while** (true) **do**
3. **if** new task coming, **then**
4. push new task in Q,
5. sort all tasks in Q in the ascending order based on the deadline
6. get initial schedule for tasks in Q using Algorithm 1
7. balance workload using Algorithm 2
8. **end if**
9. tune scheduling using Algorithm 3
10. utilize usable overflow using Algorithm 4
11. execute task in the task queue
12. **if** the task is finished, **then**
13. remove task from the ready task queue Q
14. **end if**
15. **end while**

1) generate the initial schedule;
2) balance workload and determine voltage and frequency;
3) adaptively tune up the schedule according to run-time energy availability;
4) avoid overflow and transfer slack time from the current task to the future tasks.

In this section, we put the previously discussed algorithms together to form the complete HA-DVFS algorithm for real-time energy harvesting systems, as shown in Algorithm 5.

At first, we assume that the ready task queue Q is empty, as shown in Line 1. Every time a new task comes, it is pushed into Q, as shown in Line 5, and then all tasks in Q are sorted in the ascending order according to their deadlines.

The arrival of a new task triggers the rescheduling of all tasks in Q, as shown from Lines 5 $\sim$ 7. Before each task is executed, we will check the energy availability and the overflow condition and adjust the scheduling accordingly. If there is a change in the energy harvesting rate, it will be detected before the task execution and the scheduling policy will be adjusted accordingly. Then tasks are executed based on the obtained schedule, as shown in Line 12, and are removed from the queue upon completion.

The core of the proposed algorithm is basically the sequential execution of Algorithms 1, 2, 3, and 4. The initial schedule guarantees that tasks meet their deadline requirements. The workload balance algorithm achieves the system-level energy efficiency by trading the task slack for energy savings by slowing down the execution speed. The tuning algorithm makes sure that the system has sufficient energy to execute the next task and it proactively drops a task to save more energy and CPU time for other tasks if there is an energy shortage. Finally Algorithm 4 looks for usable energy overflow and convert it to energy savings for future tasks. The overall complexity of HA-DVFS is the summation of complexity from Algorithm 1 to 4. The proposed HA-DVFS algorithm is targeted at novel power management techniques for embedded systems with energy harvesting capabilities, and it also can be extended to the networked real-time embedded systems that consist of individual nodes with energy harvesting capabilities.

VI. Experimental Results and Discussions

In this section, we evaluate the performance of the proposed scheduling algorithm based on simulation. In order to evaluate how much the overflow can be used to improve the system performance, we have implemented two versions of proposed algorithm. One consists of the first three steps [14], as shown in Section IV-E, referred as "HA-DVFS-1"; the other is the completed Algorithm 5, referred as "HA-DVFS-2". The difference between HA-DVFS-2 and HA-DVFS-1 algorithms is that the overflow in HA-DVFS-2 is utilized for improving performance whereas wasted in HA-DVFS-1.

We have developed a discrete event-driven simulator in C++ and implement the HA-DVFS algorithms. For comparison purposes, the LSA [13], [17], EA-DVFS algorithm in [14] are also implemented as benchmarks.

We designed three sets of experiments. The first set is designed to show the deadline miss rate (DMR) comparisons among LSA, EA-DVFS, HA-DVFS-1 and HA-DVFS-2 algorithms, and compare the three prediction algorithms discussed in Section V. In the second experiment, we apply the HA-DVFS-2 algorithm to schedule the tasks on a wireless sensor node with energy harvesting capability. The application consists of both tasks with DVFS potential (e.g., digital signal processing tasks) and without DVFS potential (e.g., sensing and communication tasks). In the third experiment setup, we study the performance trend of HA-DVFS-2. Finally, the fourth set compares the minimum energy storage capacity requirement for maintaining zero deadline miss rate for these four algorithms.

A. Simulation Setup

We choose the solar energy as the energy harvesting source in our simulations. Fig. 6 shows four different daytime (7:00 AM ~ 7:00 PM) solar irradiation profiles that we have collected during February and March of 2010 by using a pyranometer [32]. The data are collected every 5 s and the readings are in volts. We use linear interpolation to generate more data points to fit the simulation step size. The maximum output power of a solar panel is linearly proportional to the sun irradiation level. The readings of the pyranometer, which is denoted as Y, can be converted into the maximum possible power output of the solar panel, i.e., P_H, using the following equation:

$$P_H = Y * U * A * \varphi \tag{30}$$

where "U" is a constant value with unit of $W/(m^2V)$ determined by the solar sensor [32], "A" is the area of the solar panel and "φ" is the conversion efficiency of the solar panel. In our experiment, we set U to be 250 W/m^2V, A to be $0.01m^2$ and φ to be 10%.

A DVFS-enabled processor similar to the XScale processor [24] is used in the simulations. The actual XScale processor power and frequency setting is shown in Table I. The overhead from the processor voltage and frequency switching is ignored in the simulations.

The ESM is assumed to be a rechargeable battery or a super capacitor. Without loss of generality, we assume that the charging/discharging efficiency of the ESM is fixed to be 0.9, the conversion efficiency of ECM1 and ECM2 are 0.9. The capacity of the ESM is set to be 1000 J. The *energy threshold* is set to be 5% and 10% of ESM capacity for $E_{\text{th}-\text{low}}$ and $E_{\text{th}-\text{high}}$, respectively. To speed up the simulation, the ESM has 500 J energy to start each simulation.

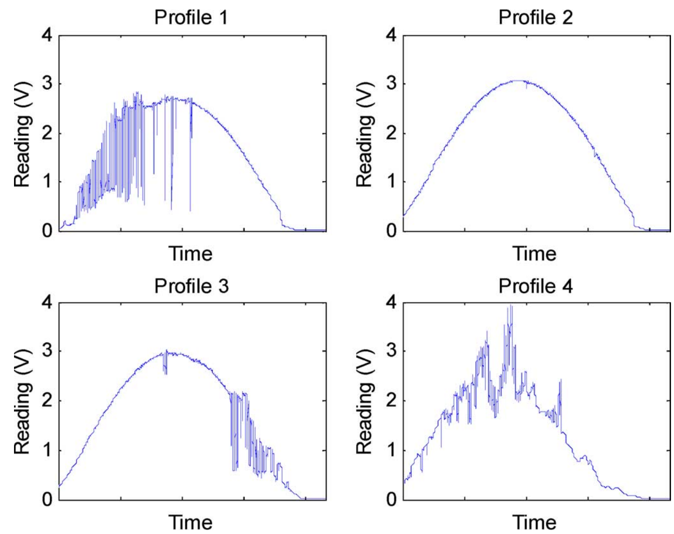


Fig. 6. Solar irradiation profiles.

TABLE I
XScale Processor Power and Frequency Levels

Frequency (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
Power (mW)	80	170	400	900	1600
Slowdown factor	0.15	0.4	0.6	0.8	1.0
Idle Power (mW)	45				

Similar to most of previous research work in this area, we use synthetic task sets for simulation [13], [17]. Each synthetic task set contains the arbitrary number of periodic tasks. In the given synthetic task set, the period p_m of a specific task τ_m is randomly drawn from the set $\{10\text{ s}, 20\text{ s}, 30\text{ s}, \ldots, 120\text{ s}\}$, and the relative deadline d_m is set to its period p_m; and the worst case execution time is calculated based on its period and harvesting power. Note that we assume large (in the time magnitude of seconds) tasks to maintain reasonable CPU time for simulation. However, the duration of the task can be arbitrary and it does not affect the application of the scheduling and DVFS algorithm. 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 $[0, \overline{P_S} * p]$, so e is a sample of a uniform-distributed random variable with distribution $[0, \overline{P_S} * p]$. Then the worst case execution time of the task can be computed as $e/P_{\max}$, where $P_{\max}$ is the power consumption of the system when running at the highest frequency and voltage level.

We define the processor utilization U as

$$U = \sum_m \frac{w_m}{p_m} \tag{31}$$

TABLE II
COMPARISONS OF DEADLINE MISS RATE (%) FOR DIFFERENT ALGORITHMS

Prof.	U	LSA	EA-DVFS	HA-DVFS-1	HA-DVFS-2
1	0.2	16.95	6.76	0.69	0.26
	0.4	31.27	14.78	8.79	5.93
	0.6	47.65	24.46	17.75	15.02
	0.8	61.37	35.27	28.04	23.15
2	0.2	9.15	3.64	0.52	0.17
	0.4	21.17	8.12	6.12	4.00
	0.6	37.20	16.34	11.62	8.92
	0.8	50.93	28.46	22.19	15.74
3	0.2	10.38	4.03	0.55	0.20
	0.4	23.72	8.86	6.46	3.68
	0.6	42.76	17.35	13.36	10.12
	0.8	54.39	29.67	22.96	18.45
4	0.2	13.27	5.04	0.72	0.25
	0.4	30.96	12.88	8.86	6.31
	0.6	45.34	22.27	14.63	12.26
	0.8	58.75	34.52	26.49	23.01

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, which should be smaller than 1. To obtain a specific U, we scale the worst case execution time of each task in a task set in the same ratio. For our experiments, we test the algorithms under four utilization ratio settings: 0.2, 0.4, 0.6, and 0.8.

The simulation terminates after 10 000 time units. For a specific processor utilization setting, we repeat experiments for 5000 task sets.

B. Deadline Miss Rate Comparison

An important real-time system performance metric is the deadline miss rate. We conduct experiments with four different power profiles in Fig. 6 assuming that P_H can be accurately predicted, and record the correspondent deadline miss rates in Table II. In the Table II, the 3rd, 4th, 5th, and 6th columns report the deadline miss rate results for LSA, EA-DVFS, HA-DVFS-1 and HA-DVFS-2 algorithms, respectively, with the utilization ratio sweeping from 0.2 to 0.8 with a step of 0.2, while other parameters are fixed.

It is shown in the Table II that the proposed HA-DVFS algorithms achieve significant reduction in deadline miss rate, comparing to the LSA and EA-DVFS algorithms under all workload settings. Also, HA-DVFS-2 achieves lower deadline miss rate comparing to HA-DVFS-1.

When the utilization is set to 0.4, LSA algorithm records 31.27% of deadline miss rate, and EA-DVFS, 14.78%, and our proposed algorithms only generate a deadline-miss-rate of 8.79% in profile 1. As the utilization increases to 0.6 and 0.8, all algorithms record relatively high deadline miss rates; however, our algorithms still beats LSA and EA-DVFS algorithms by a large margin. The fundamentals that the proposed algorithms outperform the benchmark algorithms are that slack is exploited in the HA-DVFS algorithms to slow down task execution such that energy is saved for future tasks. If we consider the time when a task is dropped as the time when the service is not available, then reducing the deadline miss rate is actually extending the service time of the system. So what we are achieving is similar as extending the battery lifetime, which is the goal of conventional low power design.

Table II also reports the reduction of deadline miss rate between HA-DVFS-2 and HA-DVFS-1 varies. For example, when the utilization is set to 0.8, HA-DVFS-2 outperforms HA-DVFS-1 by 6.45% of solar Profile 2, while only 3.48% of Profile 4. The reason is that overall harvested power of Profile 2 is greater than harvested power of Profile 4 in Fig. 6; therefore, the HA-DVFS-2 can utilize more overflow energy under Profile 2 than Profile 4.

In the next we compare the performance of the three prediction methods discussed in Section IV when used in the HA-DVFS-2 algorithm. The time step of the prediction is set to be 1 second. Since these three prediction algorithms have low computational complexity, their overhead on system performance and energy dissipation can be ignored.

In Table III, the cells in the 3rd, 4th, and 5th columns give the DMR of systems scheduled with the HA-DVFS-2 algorithm using different prediction methods, regression analysis (RA), moving average (MA) and exponential smoothing (ES), respectively. As a reference, the last column is the deadline miss rate of the HA-DVFS-2, which assumes that P_H can be accurately predicted.

From Table III, we can see that the ES has higher deadline miss rate, especially when utilization is higher, comparing to RA and MA. Also, the DMR values of RA and MA show minor differences in all profile and U settings. The results show that RA and MA have better accuracy in predicting solar energy, comparing to the ES technique. This is because, according to [27], single ES does not work efficiently when a remarkable trend component is present in the time series pattern. The results also show that compared to the system that applies HA-DVFS-2 with perfect future energy information, the system using RA or MA-based energy predictor has 25% higher deadline miss rate. This indicates the importance of accurate energy prediction to the system performance.

C. Performance Trend Study

In real applications, the size of solar panel and storage can vary due to application and technology, which causes the harvested power from solar panel and the capacity of storage to change in a wide range. In this set of experiments, we evaluate the HA-DVFS-2 algorithm with four different harvesting power settings derived from Profile 1 in Fig. 6: P_H, $2P_H$, $3P_H$, and $4P_H$. Also, four different capacity of storage are tested: 500, 1000, 1500, and 2000 J. The simulation results show that in addition to the utilization ratio U, the harvested power and the storage capacity (E_{cap}) also have significant impact on the deadline miss rate.

TABLE III
COMPARISONS OF DEADLINE MISS RATE WITH DIFFERENT PREDICTION ALGORITHMS

Prof.	U	HA-DVFS-2			HA-DVFS-2 Perfect prediction
		RA	MA	ES	
1	0.2	0.34	0.35	0.48	0.26
	0.4	6.79	7.05	8.49	5.93
	0.6	17.58	18.44	20.38	15.02
	0.8	25.23	26.89	30.79	23.15
2	0.2	0.28	0.26	0.34	0.17
	0.4	5.62	5.80	6.78	4.00
	0.6	11.34	12.61	13.98	8.92
	0.8	19.03	19.86	22.78	15.74
3	0.2	0.29	0.32	0.42	0.20
	0.4	5.08	5.25	7.15	3.68
	0.6	13.79	14.02	16.62	10.12
	0.8	24.82	26.08	29.79	18.45
4	0.2	0.40	0.34	0.49	0.25
	0.4	8.93	8.34	9.77	6.31
	0.6	16.75	16.69	19.44	12.26
	0.8	28.91	27.49	31.26	23.01

Fig. 7 shows the plots of sweeping both harvest power and storage capacity (E_{cap}) for four different utilization ratios of 0.2, 0.4, 0.6, and 0.8. We have the following observations.

1) With increase of the harvest power and/or the storage capacity (E_{cap}), the deadline miss rate decreases under all workload settings. The higher harvest power and/or storage capacity is, the lower the deadline miss rate is. It is obvious that tasks are able to be finished before deadline without causing deadline miss rate because of more energy coming from the harvested energy or storage or both.
2) Deadline miss rate reduction increases when the processor utilization ratio increases. This trend is also shown in Table II. At high utilization settings, less slack can be used to slow down the task execution and tasks are executed at high-speed and high-power mode. As a result, the system exhausts the available energy faster and causes more deadline misses.
3) Deadline miss rate reduction is not linear when harvest power or storage capacity (E_{cap}) increases. Also, it is easy to note that the most significant decreases happen between P_H and $2P_H$, and between E_{cap} of 500 and 1000 J. With further increasing P_H (e.g., from $3P_H$ and $4P_H$) and E_{cap} (e.g., from 1500 and 2000 J,), the improvements in the deadline miss rate are not as significant, as shown in the plots.

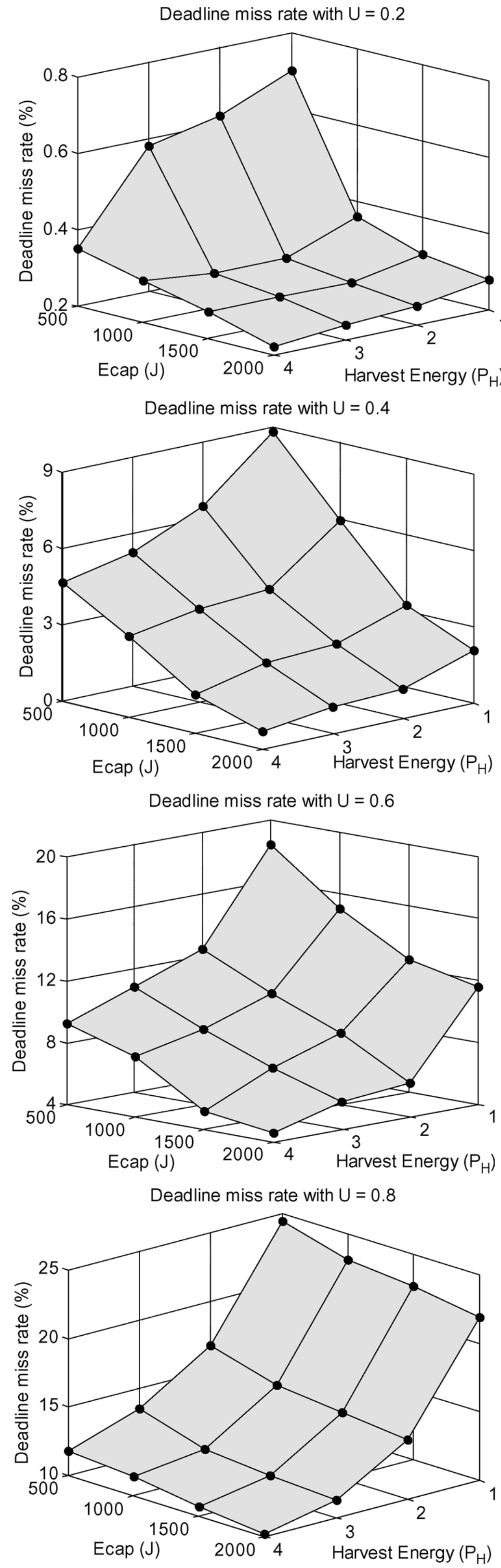


Fig. 7. Sweeping harvest energy and storage capacity.

D. Storage capacity comparison

In this section, we compare the minimum storage capacity requirement for the scheduling algorithms to achieve zero

TABLE IV
COMPARISONS OF MINIMUM STORAGE CAPACITY REQUIREMENTS FOR ZERO DEADLINE MISS RATE

Prof.	U	$C_{\text{min-LSA}}$	$C_{\text{min,EA-DVFS}}$	$C_{\text{min,HA-DVFS-1}}$	$C_{\text{min,HA-DVFS-2}}$
1	0.2	1	0.49	0.08	0.06
	0.4	1	0.72	0.45	0.38
	0.6	1	0.85	0.63	0.59
	0.8	1	0.95	0.85	0.79
2	0.2	1	0.46	0.07	0.05
	0.4	1	0.60	0.41	0.39
	0.6	1	0.85	0.64	0.57
	0.8	1	0.93	0.78	0.71
3	0.2	1	0.48	0.06	0.04
	0.4	1	0.66	0.44	0.31
	0.6	1	0.89	0.64	0.58
	0.8	1	0.95	0.80	0.74
4	0.2	1	0.54	0.07	0.06
	0.4	1	0.68	0.45	0.39
	0.6	1	0.84	0.70	0.66
	0.8	1	0.94	0.83	0.76

deadline miss under any processor utilization ratio. We use notations $C_{\min-\text{LSA}}$, $C_{\min,\text{EA}-\text{DVFS}}$ $\text{C}_{\min,\text{HA}-\text{DVFS}-1}$, and $\text{C}_{\min,\text{HA}-\text{DVFS}-2}$ to represent the minimum storage requirements for LSA, EA-DVFS and the two HA-DVFS algorithms, respectively. We run the simulations by sweeping the processor utilization U from 0.2 to 0.8 with a step 0.2 and the results are reported in Table IV. The values in Table IV are normalized to $C_{\min-\text{LSA}}$.

We can see that the HA-DVFS algorithms require much less storage capacity to achieve zero deadline miss rate in all cases. When processor utilization is low (e.g., 0.2), the HA-DVFS-2 algorithms needs almost 12% of the storage capacity that EA-DVFS needs, and 6% of LSA needs. When the utilization ratio increases, the difference among different algorithms reduces.

Also note that when the utilization ratio is at 0.2, the HA-DVFS-2 algorithm shows little improvement over HA-DVFS-1. With low utilization, all tasks have been scheduled to execute at lowest possible operating frequency. For the HA-DVFS-2 algorithm, although some tasks speed up their execution by utilizing the overflow energy and give more slack time for the succeeding tasks. The succeeding tasks cannot be further slowed down because they have already been scheduled to execute at lowest possible speed. Therefore, the HA-DVFS-2 algorithm has little improvement in $C_{\min}$ over HA-DVFS-1 at low utilization ratio.

On the other side, when the utilization ratio is very high (e.g., 0.8), it is unlikely that the harvested energy will overflow the energy storage all the time because of the high energy demand by the system. Therefore the HA-DVFS-2 algorithm does not have significant improvement over HA-DVFS-1 at high utilization ratio either. As shown in Table IV, the best improvement comes at median utilization ratio settings, such as when U is 0.6.

VII. CONCLUSION

In this paper we have proposed a harvesting-aware scheduling and voltage/frequency selection algorithm targeting at real-time systems with energy harvesting capability. The proposed algorithm consists of four steps: 1) generate initial schedule; 2) balance workload; 3) check energy availability for each scheduled task and tune up the schedule; and 4) speed up task execution when the system detects the overflow will occur. The first step is to guarantee the timing constraints are met; the second step is to trade task slack for energy savings; the third step is to make sure the energy constraints are met; in order to avoid wasting the overflow energy, the last step utilizes the overflow and transfer the slack from the current task to the future task. By dividing the original scheduling problem into these steps, we separate the constraints in timing and energy domains, which lead to reduced computing complexity.

Experimental results show that, comparing to the LSA and EA-DVFS algorithms, the HA-DVFS algorithms significantly decrease the deadline miss rate and reduce the energy storage capacity requirement for zero deadline miss rate.

REFERENCES

[1] Y. Lu, L. Benini, and G. D. Micheli, “Low-power task scheduling for multiple device,” in *Proc. Int. Workshop Hardw./Softw. Codesign*, 2000, pp. 39–43.
[2] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem, “Energy aware scheduling for distributed real-time systems,” in *Proc. Int. Symp. Parallel Distrib. Process.*, 2003, pp. 21–29.
[3] S. Liu, Q. Qiu, and Q. Wu, “Task merging for dynamic power management of cyclic applications in real-time multi-processor systems,” in *Proc. Int. Conf. Comput. Design*, Oct. 2006, pp. 397–404.
[4] F. F. Yao, A. J. Demers, and S. Shenker, “A scheduling model for reduced CPU energy,” in *Proc. Symp. Foundations Comput. Sci.*, 1995, pp. 374–382.
[5] Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, “Power optimization of variable-voltage core-based systems,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 1702–1713, Dec. 1999.
[6] J. Luo and N. K. Jha, “Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems,” in *Proc. VLSI Design*, 2002, pp. 719–726.
[7] S. Roundy, D. Steingart, L. Frechette, P. K. Wright, and J. M. Rabaey, “Power sources for wireless sensor networks,” in *Proc. Euro. Workshop Wirel. Sensor Netw.*, 2004, pp. 1–17.
[8] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. B. Srivastava, “Design considerations for solar energy harvesting wireless embedded systems,” in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 457–462.
[9] X. Jiang, J. Polastre, and D. E. Culler, “Perpetual environmentally powered sensor networks,” in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 463–468.
[10] D. Brunelli, C. Moser, L. Thiele, and L. Benini, “Design of a solar harvesting circuit for battery-less embedded systems,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 11, pp. 2519–2528, Nov. 2009.
[11] B. A. Allavena, A. Allavena, and D. Mossé, “Scheduling of frame-based embedded systems with rechargeable batteries,” in *Proc. Workshop Power Management for Real-time Embed. Syst.*, 2001, pp. 279–284.
[12] C. Rusu, R. G. Melhem, and D. Mossé, “Multi-version scheduling in rechargeable energy-aware real-time systems,” *J. Embed. Comput.*, pp. 95–104, 2005.

[13] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy-harvesting sensor nodes," in *Proc. 5th Work. Conf. Distrib. Parallel Embed. Syst.*, 2006, pp. 125–134.

[14] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," in *Proc. Design, Autom., Test Eur.*, 2008, pp. 263–241.

[15] S. Liu, Q. Qiu, and Q. Wu, "An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems," in *Proc. Design Autom. Conf.*, 2009, pp. 782–787.

[16] Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 4, Sep. 2006, Art. No. 32.

[17] C. Moser, L. Thiele, L. Benini, and D. Brunelli, "Real-time scheduling with regenerative energy," in *Proc. Euromicro Conf. Real-time Syst.*, 2006, pp. 261–270.

[18] C. Moser, J.-J. Chen, and L. Thiele, "Reward maximization for embedded systems with renewable energies," in *Proc. Int. Conf. Embed. Real-Time Comput. Syst. Appl.*, 2008, pp. 247–256.

[19] C. Moser, J.-J. Chen, and L. Thiele, "Optimal service level allocation in environmentally powered embedded systems," in *Proc. ACM Symp. Appl. Comput.*, 2009, pp. 1650–1657.

[20] C. Moser, J.-J. Chen, and L. Thiele, "Power management in energy harvesting embedded systems with discrete service levels," in *Proc. Int. Symp. Low Power Electron. Design*, 2009, pp. 413–418.

[21] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Robust and low complexity rate control for solar powered sensors," in *Proc. Design, Autom., Test Eur.*, 2008, pp. 230–235.

[22] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management in energy harvesting systems," in *Proc. Design, Autom., Test Eur.*, 2007, pp. 773–778.

[23] D. Li and P. H. Chou, "Maximizing efficiency of solar-powered systems by loading matching," in *Proc. Int. Symp. Low Power Electron. Design*, 2004, pp. 162–167.

[24] J. Lu, S. Liu, Q. Wu, and Q. Qiu, "Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems," in *Proc. WIPGC*, 2010, pp. 469–477.

[25] A. Ravinagarajan, D. Dondi, and T. S. Rosing, "DVFS based task scheduling in a harvesting WSN for structural health monitoring," in *Proc. Conf. Design, Autom. Test Eur.*, 2010, pp. 1518–1523.

[26] D. C. Montgomery, L. A. Johnson, and J. S. Gardiner, *Forecasting and Time Series Analysis*. New York: McGraw-Hill, 1990.

[27] A. K. Palit and D. Popović, *Computational Intelligence in Time Series Forecasting*. New York: Springer, 2005.

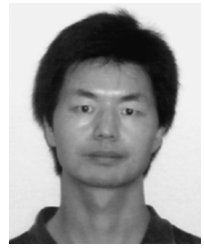
[28] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *Proc. Design Autom. Conf.*, 1999, pp. 861–866.

[29] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan, "Adaptive duty cycling for energy harvesting systems," in *Proc. ISLPED*, 2006, pp. 180–185.

[30] J. Recas, C. Bergonzini, D. Atienza, and T. S. Rosing, "Prediction and management in energy harvested wireless sensor nodes," in *Proc. VITAE*, 2009, pp. 6–10.

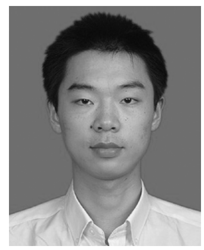
[31] D. K. NoH, L. Wang, Y. Yang, H. K. Le, and T. Abdelzaher, "Minimum variance energy allocation for a solar-powered sensor system," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst.*, 2009, pp. 44–57.

[32] Apogee Instruments, Inc., Logan, UT, "Silicon-cell photodiode pyranometers," 2010. [Online]. Available: http://www.apogeeinstruments.com/pyr_spec.htm



Shaobo Liu (S'07) received the B.S. degree in material science and engineering from Wuhan University of Technology, Wuhan, China, in 2001, the M.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2004, and the Ph.D. degree in electrical and computer engineering from State University of New York, Binghamton, in 2010.

He is currently with Marvell Semiconductor, Inc., Marlborough, MA. His research interests include power/thermal analysis and optimization, leakage estimation and minimization, energy harvesting system design, and energy aware computing.



Jun Lu (S'10) received the M.S. degree from the College of Biomedical Engineering and Instrument Science, Zhejiang University, Hangzhou, China, in 2008, and . He is currently pursing the Ph.D. degree from the Department of Electrical and Computer Engineering, State University of New York, Binghamton.

His research interests include low power design and power management of energy harvesting real-time embedded systems, high performance computing for many-core computing platform.

Qing Wu (M'96) received the B.S. and M.S. degrees from the Department of Information Science and Electronics Engineering, Zhejiang University, Hangzhou, China, in 1993 and 1995, respectively, and the Ph.D. degree in electrical engineering from the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, in 2001.

He is currently with the Information Directorate of United States Air Force Research Laboratory, Rome, NY. His research interests include neuromorphic computing algorithms and architectures, hardware/software optimization for massively parallel high-performance computing systems, low power design and power management for cloud computing and green data centers, low power design methodologies for energy harvesting mobile computing systems, power estimation of VLSI circuits and systems, FPGA-based hardware-accelerated computing.

Qinru Qiu (M'01) received the M.S. degree and Ph.D. degree from the Department of Electrical Engineering, University of Southern California, Los Angeles, in 1998 and 2001, respectively, and the B.S. degree from the Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 1994.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, State University of New York, Binghamton. Her research interests include energy efficient computing systems, energy harvesting real-time embedded systems, and neuromorphic computing. She has published over 40 research papers in referred journals and conferences. Her works are supported by NSF, DoD, and Air Force Research Laboratory.