
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

Ow 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.
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

$$E_H(t_1, t_2) = \int_{t_1}^{t_2} P_H(t)dt.$$  \hfill (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 levels, $E_{th-low}$ and $E_{th-high}$. When $E_C(t)$ is below or equal to $E_{th-low}$, the system is in energy depletion and the processor will enter energy saving sleep mode. The amount of remaining energy in 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 $E_{th-high}$. Based on the definition, during the normal operation mode we have

$$E_{th-low} \leq E_C(t) \leq E_{cap} \quad \forall t.$$  \hfill (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 $\nu$. 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., $E_S(t_1, t_2) = E_C(t_2) - E_C(t_1) + E_D(t_1, t_2)$, where $E_C(t_1, t_2)$ is the leakage energy of the storage from time $t_1$ to $t_2$. If $E_D(t_1, t_2) > E_S(t_1, t_2)$ then the EDM will be powered by both the EHM and the ESM and we have

$$E_D(t_1, t_2) = E_H(t_1, t_2) - \eta E_S(t_1, t_2) \quad \forall t_1 < t_2.$$  \hfill (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:

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

$$E_S(t_1, t_2) = \eta (E_H(t_1, t_2) - E_D(t_1, t_2)) \quad \forall t_1 < t_2.$$  \hfill (5)

C. Energy Dissipation Model

Assume that the DVFS-enabled processor has $N$ discrete operating frequencies $f_n : \{f_1 \leq f_2 \leq \cdots \leq f_N = f_{\text{max}}\}$, 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 $f_{\text{max}}$, that is

$$S_n = \frac{f_n}{f_{\text{max}}}. \hfill (6)$$

For convenience purposes, we use notations $f_n$ and $P_n$ interchangeably in this paper. Similarly for notations $P_n$ and $S_n$.

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

If task $\tau_m$ 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

\[ x = b_0 + b_1 z + \varepsilon. \] (7)

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

\[ \hat{b}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(z_i - \overline{z})}{\sum_{i=1}^{n}(z_i - \overline{z})^2} \] (8)

\[ \hat{b}_0 = \overline{x} - \hat{b}_1 \overline{z} \] (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 correspondent \( x \) and \( z \).

The fitted simple linear regression model is

\[ \hat{x} = \hat{b}_0 + \hat{b}_1 z. \] (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)

\[ \hat{x} = \frac{x(t) + x(t-1) + \cdots + x(t-N+1)}{N}. \] (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:

\[ \hat{x} = \alpha x(t) + (1 - \alpha) \hat{x}(t-1) \] (12)

where \( \hat{x}(t) \) is called the exponentially smoothed value, \( x(t) \) is the observed value at the same point of time. The fraction \( \alpha \) is called the smoothing constant, and \( \hat{x}(t-1) \) 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 ($M$th) 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 $\tau_M$, we have

$$
ift_M = a_M + d_M
$$
$$
ist_M = a_M + d_M - w_M.
$$

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

$$
ift_m = \min(a_m + d_m, ist_{m+1})
$$
$$
ist_m = \min\{\max(a_m + d_m - w_m, a_m), ist_{m+1} - w_m\}
$$

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 $\tau_m$ is not schedulable under the given timing constraint; so we have

$$
ist_m = \min\{a_m + d_m - w_m, ist_{m+1} - w_m\}
$$

where $a_m$, $d_m$, $w_m$ is the deadline of task $\tau_m$, minus its worst case execution time, and is the initial starting time of the next task (i.e., task $\tau_{m+1}$) minus the worst case execution time of task $\tau_m$. This scheduling is justified by the following facts: 1) task $\tau_m$ 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 $\tau_m$ 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 ($\tau_m$) 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.

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 $\tau_m$ for execution is determined by

$$
stm = \begin{cases} 
\max(a_1, current_time), & m = 1 \\
\max(a_m, fstm_{m-1}), & m = 2, \ldots, M.
\end{cases}
$$

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

$$
stm + \frac{w_m}{S(SI_m - 1)} < ftm
$$

the timing constraint can still be met after further stretching task $\tau_m$.

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 $\tau_m$ is decremented by 1; thus the operating frequency of task $\tau_m$ 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 $\tau_m$ if task $\tau_m$ can be executed by the processor at frequency $f_n$ subjected to the timing constraints.

Now the $ft_m$ can be calculated as

$$
ftm = stm + \frac{w_m}{S(SI_m)}.
$$

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 $\tau_m$ is decreased at most by 1 within 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.

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 \( m \)th task. Once we found that the slowdown factor for \( m \)th task is invalid, where \( m \), we no longer need to check for \( m \)th task. 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 \( m \)th 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

\[
E(X_m) = \sum_{k=1}^{M-m} k \cdot \Pr(X = k) = \frac{1 + M - m}{2}. \tag{21}
\]

Thus the average number of tasks checked in Line 8 is bounded by \( (1 + M - m)/2 \) for the \( m \)th iteration. The overall number of tasks that have been checked over \( M \) iterations is calculated as \( \sum_{m=1}^{M} (1 + M - m)/2 = (M^2 + M)/4 \). 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 \( m \)th task \( \tau_m \) in \( Q \) is the first task whose schedule is invalidated due to the energy shortage, which means

\[
E_C(s) - E_{th-low} + E_H(s) + f(t_m) < E_D(s) + d(t_m) \tag{22}
\]

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

\[
E_C(s) - E_{th-low} + E_H(s) + f(t_m + d(t_m)) = E_D(s) + d(t_m) + f(t_m + d(t_m)). \tag{23}
\]

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

\[
f(t_m + d(t_m)) \leq a_m + d_m \tag{24}
\]

and the slowdown factors for tasks indexed by \( m + 1, \ldots, M \) are still valid, then task \( \tau_m \) is executed at time interval \( [s(t_m) + d(t_m), t_m + d(t_m)] \) at the frequency \( f(S(t_m)) \); and the schedule for succeeding tasks is updated, as shown in Lines 6 ~ 9 in Algorithm 3; otherwise, task \( \tau_m \) 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 \( \tau_m \), therefore the time complexity of Algorithm 3 is \( O(M \log(a_m + d_m - s(t_m))) \).

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,
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 $\tau_1$ and $\tau_2$ 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.

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 ($E_{th-low}$) 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 $\tau_1$ 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 $\tau_1$ cannot be carried out, shown by the “lime” color long dash line in Fig. 2.

On the other hand, before running task $\tau_1$, the energy availability is checked by (22), and then task will be delayed by 2 time units; accordingly task $\tau_1$ is executed between time interval [52, 58], and the schedule for task $\tau_2$ 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 “l ime” color solid line in Fig. 2; energy is not a concern any more for the schedule of task $\tau_1$. The similar argument holds for task $\tau_2$.

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 $\tau_m$ and $\tau_{m+1}$ 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 $\tau_m$, if its finishing time $ft_m$ is earlier than the starting time $st_{m+1}$ of its successor $\tau_{m+1}$, the starting time $st_{m+1}$ of task $\tau_{m+1}$ is solely determined by its arrival time $a_{m+1}$.

Proof: From (18), we know the starting time $st_{m+1}$ of task $\tau_{m+1}$ 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 $\tau_{m+1}$ cannot be scheduled earlier than $st_{m+1}$ even if task $\tau_m$ is finished before $ft_m$. Assume that the energy overflow occurs at some point between $st_{m}$ and $st_{m+2}$, and the wasted energy through overflows 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 $\tau_m$, we cannot improve the system performance. The reasons are two-fold: 1) task $\tau_{m+1}$ cannot be scheduled earlier than $st_{m+1}$; 2) whether or not the $E_O$ is used for speeding up the execution of $\tau_m$, the energy storage is full at the time task $\tau_{m+1}$ 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 $\tau_m$, is being executed;
2) the successors of $\tau_m$ should be able to get more slack time after the overflowing energy is utilized for speeding up the execution of task $\tau_m$.

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 $\tau_m$ and $\tau_{m+1}$ 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 $\tau_m$ 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 $\tau_{m+1}$, 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 $\tau_m$ is transferred to task $\tau_{m+1}$. 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.
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 \( P_H = 1.5P_L \); 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 1.2\( P_L \) during the time interval [0, 5], and 0 afterwards. The energy storage is set to be full \( E_{cap} \) at time 0.

Tasks \( \tau_1 \) and \( \tau_2 \) are scheduled based on the workload balance algorithm, as shown in Fig. 5(a). From the scheduling we know that \( \tau_1 \) is executed at low frequency \( f_L \) at time interval [0, 6) and \( \tau_2 \) 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 \( \tau_1 \), 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 \( \tau_1 \) is finished, the available energy is \( E_{cap} - P_L \). Then the system runs task \( \tau_2 \), and the energy needed is \( P_H \times 6 = 2.5P_L \times 6 = 15P_L \). The energy in the storage decreases to \( E_{cap} - 16P_L \) when \( \tau_2 \) finishes.

To save the wasted overflowing energy, we can speed up the execution of task \( \tau_1 \), as shown in Fig. 5(b). (It is executed at high frequency \( f_H \) from time instance 0 to 4 and task \( \tau_2 \) at \( f_L \) during time interval [4, 13). Task \( \tau_1 \) finishes before its deadline and task \( \tau_2 \) finishes right at its deadline, as shown in Fig. 5(b). The power needed for executing task \( \tau_1 \) 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 \( \tau_1 \) finishes is \( E_C - (P_H - P_S) \times 4 - E_C = 2P_L - P_S \times 4 \times 1 = 5.2P_L \). Then the processor runs task \( \tau_2 \) 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 \( \tau_2 \). 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 \( \tau_m \) 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:

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

In order to prevent energy overflow, ideally the operating frequency of task \( \tau_m \) 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 \( \tau_m \) should be executed at a new speed \( f(SI_{m,new}) \) where the needed extra energy is no less than \( E_O \). That is

\[
E_D\left(st_m, w_m/f(SI_{m,new})\right) - E_D\left(st_m, w_m/f(SI_{m})\right) \geq E_O \tag{26}
\]

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

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

\[
E_D\left(st_m, w_m/f(SI_{m,new} - 1)\right) - E_D\left(st_m, w_m/f(SI_{m})\right) < E_O. \tag{27}
\]

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

\[
E_D(st_m, w_m) - E_D(st_m, w_m/f(SI_{m})) < E_O. \tag{28}
\]
In this case we schedule task $\tau_m$ at the full speed $f_{\text{max}}$.

After the new execution speed for task $\tau_m$ is decided, we need to compute the transferred slack $\text{slack}_{tf}$ from task $\tau_m$, which can be computed as

$$\text{slack}_{tf} = \frac{w_m}{f(SI_m)} - \frac{w_m}{f(SI_{m,\text{new}})}.$$  \hspace{1cm} (29)

In order to maximize energy efficiency of the transferred slack time $\text{slack}_{tf}$, ideally $\text{slack}_{df}$ should be distributed such that all tasks succeeding $\tau_m$ have the same frequency assignment. We have the procedure $\text{distribution}_{\text{slack}}()$ 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 ~ 5. Note that the starting time $s_{lm}$ of task $\tau_m$ keeps the same as before when it is executed at the new frequency $f(SI_{m,\text{new}})$, but $f_{lm}$ needs to be updated, as shown in Line 3. Task $\tau_m$ will finish earlier at the new assigned frequency, and the slack time from $\tau_m$ is transferred to tasks following $\tau_m$, as shown in Line 5. By utilizing the transferred slack time, tasks succeeding $\tau_m$ can run at lower frequencies, as show in procedure $\text{distribution}_{\text{slack}}()$, so that the system achieves better energy efficiency. The time complexity of Algorithm 4 is similar to Algorithm 2, which is $O(M^2)$ on average.

### E. Overall HA-DVFS Algorithm

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

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 ~ 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 over-flow 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 over-flow 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 , can be converted into the maximum possible power output of the solar panel, i.e., , using the following equation:

\[
P_H = Y \times U \times A \times \varphi
\]  

(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 “\( \varphi \)” is the conversion efficiency of the solar panel. In our experiment, we set \( U \) to be 250 \( W/m^2V \), \( A \) to be 0.01 m² and \( \varphi \) 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 \( E_{th-low} \) and \( E_{th-high} \), respectively. To speed up the simulation, the ESM has 500 J energy to start each simulation.

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 \( \tau_m \) is randomly drawn from the set \{10 s, 20 s, 30 s, …, 120 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 \( P_S \), and the task period is \( p \), the worst case energy consumption \( e \) of the task is uniformly drawn from interval \([0, P_S \times p]\), so \( e \) is a sample of a uniform-distributed random variable with distribution \([0, P_S \times p]\). 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

\[
U = \sum_{m} \frac{u_{m}}{p_{m}}
\]

(31)
where \( \tau_{\text{wc}} \) is the worst case execution time of task \( \tau_{\text{wc}} \), and \( p_{\text{m}} \) 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. 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 \( L_{\text{cap}} \) also have significant impact on the deadline miss rate.

### Table II

<table>
<thead>
<tr>
<th>Prof.</th>
<th>LSA</th>
<th>EA-DVFS</th>
<th>HA-DVFS-1</th>
<th>HA-DVFS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>16.95</td>
<td>6.76</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>31.27</td>
<td>14.78</td>
<td>8.79</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>47.65</td>
<td>24.46</td>
<td>17.75</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>61.37</td>
<td>35.27</td>
<td>28.04</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>9.15</td>
<td>3.64</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>21.17</td>
<td>8.12</td>
<td>6.12</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>37.20</td>
<td>16.34</td>
<td>11.62</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>50.93</td>
<td>28.46</td>
<td>22.19</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>10.38</td>
<td>4.03</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>23.72</td>
<td>8.86</td>
<td>6.46</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>42.76</td>
<td>17.35</td>
<td>13.36</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>54.39</td>
<td>29.67</td>
<td>22.96</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
<td>13.27</td>
<td>5.04</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>30.96</td>
<td>12.88</td>
<td>8.86</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>45.34</td>
<td>22.27</td>
<td>14.63</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>58.75</td>
<td>34.52</td>
<td>26.49</td>
</tr>
</tbody>
</table>
TABLE III
COMPARISONS OF DEADLINE MISS RATE WITH DIFFERENT PREDICTION ALGORITHMS

<table>
<thead>
<tr>
<th>Prof</th>
<th>$U$</th>
<th>HA-DVFS-2</th>
<th>HA-DVFS-2 Perfect prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RA</td>
<td>MA</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
<td>0.34</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.79</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>17.58</td>
<td>18.44</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>25.23</td>
<td>26.89</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.28</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>5.62</td>
<td>5.80</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>11.34</td>
<td>12.61</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>19.03</td>
<td>19.86</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.29</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>5.08</td>
<td>5.25</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>13.79</td>
<td>14.02</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>24.82</td>
<td>26.08</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
<td>0.40</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>8.93</td>
<td>8.34</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>16.75</td>
<td>16.69</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>28.91</td>
<td>27.49</td>
</tr>
</tbody>
</table>

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 2$P_H$, and between $E_{cap}$ of 500 and 1000 J. With further increasing $P_H$ (e.g., from 3$P_H$ and 4$P_H$) 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.

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

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

deadline miss under any processor utilization ratio. We use notations $C_{\text{min},\text{LSA}}$, $C_{\text{min},\text{EA-DVFS}}$, $C_{\text{min},\text{HA-DVFS-1}}$, and $C_{\text{min},\text{HA-DVFS-2}}$ 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 $C_{\text{min},\text{LSA}}$.

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 $C_{\text{min}}$ 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


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 refereed journals and conferences. Her works are supported by NSF, DoD, and Air Force Research Laboratory.