Scheduling and Mapping of Periodic Tasks on Multi-Core Embedded Systems with Energy Harvesting

Jun Lu and Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, New York, USA
{jlu5, qqiu}@binghamton.edu

ABSTRACT- In this paper we propose a low-complexity and effective task mapping, scheduling and power management method for multi-core real-time embedded systems with energy harvesting. The proposed method is based on the concept of task CPU utilization, which is defined as the worst-case task execution time divided by its period. This work mathematically proves that by allocating the new task to the core with the lowest utilization, we can achieve the lowest overall energy dissipation. This method, combined with a new dynamic voltage and frequency selection (DVFS) algorithm with energy harvesting awareness and task slack management (TSM) forms the proposed UTilization Based (UTB) algorithm. With periodical tasks in a multi-core platform, this partitioned scheduling method is optimal for energy dissipation if the proposed utilization-based scheduling and DVFS algorithm is applied on each core. Experimental results show that new algorithm achieves better performance in terms of deadline miss rate in a single-core environment, comparing to the best of existing algorithm. When applied on a multi-core platform, the UTB algorithm achieves better efficiency in utilizing the harvested energy and overflowed energy.

Keywords-energy harvesting; multi-core; power management; task scheduling; real-time embedded system

I. Introduction

Embedded systems have evolved tremendously in recent years as high frequency and low-power integrated circuits research advances forward. However, there is no significant extension in the lifetime of battery-powered embedded systems because of the limitation of the battery technologies. In recent years, the energy harvesting/scavenging [1] technologies has been explored to provide renewable energy to the portable electronic systems. It is expected that with the energy-harvesting technologies, we will eventually achieve energy autonomy in portable computing and communication.

Many techniques have been proposed in the area of power management of energy harvesting real-time embedded systems (EH-RTES). They range from task scheduling and workload distribution to dynamic voltage and frequency selections. In [2], the tasks are executed as late as possible at full speed by using a lazy scheduling algorithm (LSA). Compared to the earliest deadline first (EDF) scheduling, the LSA maintains the same deadline miss rate with much smaller battery size. The authors of [3] proposed an algorithm named energy aware dynamic voltage and frequency selection (EA-DVFS) that achieves better energy saving by slowing down tasks when the harvested energy is not sufficient. The algorithm was further improved in [4] by using an adaptive scheduling and DVFS algorithm (AS-DVFS), in which the processor operation voltage and frequency are adjusted under timing and energy constraints to achieve the energy efficiency. The AS-DVFS algorithm exploits the tasks slack for energy saving by evenly distributing workload over time. In [7], the authors proposed a reliable solar harvesting prediction algorithm with energy management considering both solar and weather conditions, to achieve better energy utilization. The authors of [8] proposed a task scheduler based on a linear regression model with DVFS to achieve health monitoring accuracy and measurements.

All above research works are targeted at task scheduling and DVFS techniques for EH-RTES with a single-core processor. Recent research work has started to move towards multi-core processor architectures. Many studies have been done on multi-core systems with DVFS-capability to minimize energy dissipation. Aydin *et al* [9] showed that multi-core scheduling is an NP-Hard problem, and developed a framework with minimum energy consumption objective by using the variable voltage earliest deadline first algorithm. In [10], the authors proposed an adaptive minimal bound first-fit (AMBFF) algorithm with realistic constraint consideration to save more energy. The authors of [11] proposed a new approximation algorithm for energy-efficient homogeneous multi-core processor with the 1.21-approximation factor. Ref. [12] gave general suggestions of interfacing energy harvesting model with multi-core scheduling. The work in [13] introduced the task movement algorithm (TMA) to schedule tasks on a multi-core energy harvesting system. These research works provided coarse-grain scheduling framework and preliminary results, even though they are initial research on the multi-core scheduling problem for energy harvesting systems.

Multi-core scheduling, generally, can be divided into two categories, *global* or *partitioned* [9][10]. The global scheduling uses a global scheduler to assign task to each core and allows task migration among cores, while partitioned scheduling allocates each task to one core permanently. In this paper, we propose a new partitioned scheduling and CPU utilization based DVFS algorithm for multi-core EH-RTES. We start with introducing the utilization based DVFS algorithm and discuss its performance on a single-core EH-RTES. Then we present the least utilization based partitioned scheduling and algorithm for multi-core EH-RTES. The major contributions of this work include,

1) We propose a new scheduling algorithm with low implementation complexity for single-core EH-RTES.
2) We propose four Task Slack Management (TSM) algorithms to utilize the task slack to achieve energy efficiency for EH-RTES.
3) We prove analytically that the core energy dissipation is a convex function of the summation of tasks utilization, for periodical tasks.
4) We propose a *UTilization Based* (UTB) algorithm and analytically prove that it is the optimal partitioned scheduling method to schedule periodical tasks on a multi-core EH-RTES.

The remainder of the paper is organized as follows. In Section 2, the single/multi-core EH-RTES model is introduced and its major components are discussed. Section 3 gives a motivational example of improved task scheduling on EH-RTES, followed by the proposed algorithm. In Section 4, we propose algorithm called *UTilization Based* (UTB) multi-core scheduling by applying the improved task scheduling to the multi-core processor. Section 5 presents experimental setups and results with two DVFS processor models. Finally Section 6 gives the conclusions of the work.

II. System Model

In this work, we consider a typical EH-RTES with a single-core or multi-core processor. Figure 1 shows the major modules of a node-level EH-RTES platform: *energy harvesting module* (EH Module), *energy storage module* (ES Module), *real-time embedded system module* (RTES Module) and the Conversion Monitoring Circuitry. The EH module could be a solar panel or any other harvesting units (e.g. piezoelectric devices). The ultracapacitor and lithium-ion rechargeable battery are primary choices for the ES module. The examples of RTES module include the XScale [5][6] and PowerPC 405LP [10] microprocessor which has dynamic voltage and frequency capabilities for low power task scheduling.

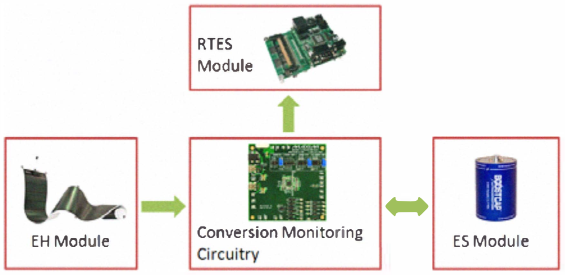

Figure 1. A simple EH-RTES platform.

A. *Energy Harvesting Module*

In this work, solar panels are chosen to be the primary energy harvesting technology for EH-RTES, since sunlight is the most well-known and prevalent renewable energy source. The energy harvesting rate is heavily dependent on the operating environment and fluctuates in real time, e.g., the output power of a solar panel is usually regarded as a random process affected by the light intensity, temperature, output voltage, manufacturing process variation, etc. We denote $P_H(t)$ as the power harvested by the EH Module, and the harvested energy $E_H(t_1, t_2)$ during time a particular interval $[t_1, t_2]$ is calculated as:

$$E_H(t_1, t_2) = \int_{t1}^{t2} P_H(t)dt \qquad (1)$$

Note that $P_H(t)$ it is time-vary variable, and it is not determined beforehand, but we can predict it by historic data using time series forecasting techniques [6].

B. *Energy Storage Module*

Energy storage is essential for EH-RTES to allow system to continue operation during periods of insufficient harvested energy. However, energy storage has limited capacity, and there is energy overhead when moving energy in or out of the ES Module. In this paper, the capacity of ES Module is denoted as E_{cap}, $E_C(t)$ is the remaining energy in the ES Module at time t, and η_s is the charging or discharging efficiency factor, E_S is energy flow at time interval $[t_1, t_2]$ in or out of the ES Module. We also define E_D as the energy dissipation by the RTES Module at time interval $[t_1, t_2]$. When the stored energy of ES Module reaches its capacity, the incoming energy overflows the ES Module, therefore we have,

$$0 \le E_c(t) \le E_{cap} \qquad (2)$$

If the ES Module is discharging during time interval $[t_1, t_2]$, then:

$$E_S(t_1, t_2) = (E_H(t_1, t_2) - E_D(t_1, t_2))/\eta_s \ \forall t_1 < t_2 \qquad (3)$$

Similarly, when the ES Module is charging the extra energy without overflow from the EH Module, we have:

$$E_S(t_1, t_2) = \eta_s(E_H(t_1, t_2) - E_D(t_1, t_2)) \ \forall t_1 < t_2 \qquad (4)$$

C. *Real-Time Embedded System Module*

We consider DVFS-enabled single-core or multi-core processor running real-time applications. Without loss of generality, a DVFS-enabled core is modeled to have N discrete variable voltage and frequency settings. And its speed S_n can vary from 0 to the maximum speed, where $1 \le n \le N$ .For simplicity, we normalize the processor speed with respect to maximum speed, therefore $0 < S_n \le S_{max} = 1.0$. At speed S_n, the power consumption of the processor is calculated by $g(S_n)$, where the $g()$ is a strictly convex function and increasing function on non-negative real numbers [9]. Consider a period of time $[t_1, t_2]$, if the speed is S_n, then the energy dissipation during this interval is

$$E(t_1, t_2) = \int_{t_1}^{t_2} g(S_n)dt \qquad (5)$$

The real-time task set **T** running on the processor consist of P periodic real-time tasks $\{T_1, \ldots, T_P\}$, the worst case execution time of a task T_p is denoted by w_p, D_p is denoted as the relative deadline of T_p, which is also equal to the period of T_p. All tasks are assumed to be independent and ready at the beginning of their period. The actual execution time of T_p is $\tau_p = w_p/S_p$, if it is executed at speed S_p. We define the *utilization* of task T_p as $u_p = w_p/D_p$ when running at maximum speed $S_p = S_{max} = 1$.

For a multi-core processor, we assume there are M identical cores $\{C_1, \ldots, C_M\}$. The total utilization of the task set **T** is, $U_{tot} = \sum_{i=1}^{P} u_i = \sum_{i=1}^{M} U_i$, where U_i is the total task utilization of C_i. The real-time system is considered to be a preemptive system, and the task with the earliest deadline has the highest priority to be executed. The scheduler preempts any other task if necessary.

D. *Conversion and Monitoring Circuitry*

As mentioned in 2.1, the renewable energy source can have high random variations due to environmental changes. This randomness causes the uncertainty in harvested energy by the EH module. Therefore we need the conversion and monitoring (CM) circuitry to measure and modulate the output from the EH Module and direct the energy flow among the EH, ES and RTES modules. Another important aspect is that we assume that the CM circuitry is capable of performing automatic *maximum power point tracking* (MPPT) [5] to yield the maximum output power by the solar panel. The conversion (from solar irradiation to electricity) efficiency is denoted as η.

III. Utlization based DVFS for Single-Core

In this section, we will first introduce the utilization-based scheduling algorithm for a single-core system, starting with a motivational example to illustrate the basic idea. Then, we will explain the algorithm in details.

A. *Motivational Example*

Assume that there are three periodical tasks T_1, T_2 and T_3 in the task set, and their worst case execution times are 2, 3 and 1 second, and the periods are 5, 10 and 20 seconds, respectively. Each task's relative deadline is the same as the task's period. During the one hyper-period, i.e., a 20-second period of time, the system will execute four instances of T_1: T_{11}, T_{12}, T_{13}, T_{14}, two instances of T_2: T_{21}, T_{22}, and one instance of T_3: T_{31}. Under this particular load, we compare the complexity and power consumption of two different scheduling methods. The first scheduling method is the AS-DVFS algorithm proposed in [4]. The other method does scheduling (and DVFS) based on task utilization defined in Section 3.2. We use the actual XScale processor power and frequency setting in this example, as shown in Table 1.

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
Normalized Speed	0.15	0.4	0.6	0.8	1.0

The AS-DVFS scheduling results are shown in Figure 2. At time instance 0, there are 3 tasks T_{11}, T_{21} and T_{31} to be scheduled. According AS-DVFS, they are scheduled at speed 0.4, 0.4 and 0.15, respectively, as shown in Figure 2(a). At time instance 5, T_{11} is finished and T_{12} is released, all tasks in task queue, T_{21}, T_{12} and T_{31}, have to be rescheduled. As shown in Figure 2(b) T_{21} and T_{12} are executed at speed 0.8 in order to meet their deadlines. We have to reschedule tasks at time instances 10 (Figure 2(c)) and 15 (Figure 2(d)) when a new task comes into the queue. Figure 2(d) also shows all the tasks with their start/finish times and execution speeds. The total energy consumption is calculated as: 5×170+5×1600+5×400+5×1600=18850mJ.

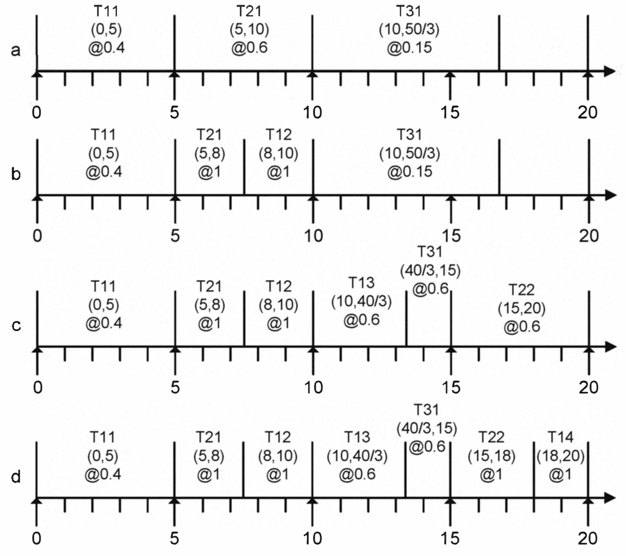

Figure 2. Scheduling results under AS-DVFS method.

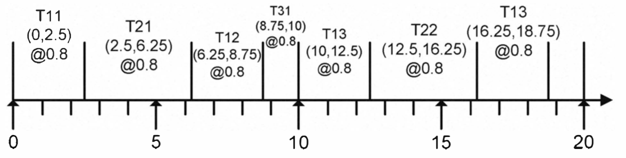

Figure 3. Scheduling result under utilization-based method.

In new method, it schedules the tasks based on the utilization of tasks in the queue. Basically it chooses the lowest normalized core speed that is higher than or equal to the summation of the task utilizations. Therefore at time instant 0, the summation of task T_{11}, T_{21} and T_{31} is 0.75, the new method will schedule these task at 0.8. At time instance 5, T_{11} and T_{21} are finished, T_{12} is released, the summation of task utilization is still 0.75, so the tasks is still running at normalized speed of 0.8. Figure 3 shows the complete scheduling result. The energy consumption is 16875mJ, which is about 10.48% lower, comparing to AS-DVFS.

B. *Utilization Based Task Speed Selection*

The reason that the AS-DVFS gives inferiors scheduling results is that it ignores the recurring nature of tasks and therefore tends to underestimate the workload at the beginning.

In order to overcome this problem, here we propose a new algorithm that selects the task execution speed and supply voltage level based on the projected CPU utilization. Algorithm 1 shows the pseudo code of the algorithm. The new algorithm selects the lowest possible speed that is higher than or equal to the summation of utilization of all tasks. Therefore the core speed S is a non-decreasing function of total utilization. This speed is set for all current tasks in the task queue, and will not be changed until a new task is in. When tasks are scheduled at the same speed based on the total utilization, it also reduces the time and energy overhead of voltage and frequency scaling [10]. Algorithm 1 is solely decided by the number of recurrent tasks P in the queue, hence its complexity is $O(P)$. Please note that by selecting the core speed, we are also selecting the core supply voltage by default, due to the one-to-one correspondence between core speed and voltage.

Algorithm 1: Speed Selection Based on Utilization

```
1.  $util = 0$;
2.  for $i = 1:P$ {
3.    $util$ += get_util($T_i$)
4.  }
5.  choose lowest $S$ from{$S_1,\dots S_N$}, such that $util \le S$
6.  for $i = 1:P$ {
7.    $ft_i = st_i + w_i/S$
8.  }
```

C. *Avoid Energy Overflow and Shortage*

Due to the limited energy storage capacity and the uncertainty of the harvested energy and discharging current, overflow and underflow may happen on the energy storage. A good power management algorithm should be able to predict and avoid the battery overflow and underflow to reduce wasted energy and lower deadline misses.

Assume that task T_p is scheduled to execute at time interval $[st_p, ft_p]$ with speed S_p. If the energy overflow is predicted to occur between st_p and ft_p, we can calculate overall overflowing energy E_O until ft_p if no action is taken as follows:

$$E_O = E_C(st_p) + E_H(st_p, ft_p) - E_D(st_p, ft_p) - E_{cap} \quad (6)$$

Note that the value of $E_H(st_p, ft_p)$ are predicted based on the history energy harvesting rate. In order to prevent energy overflow, ideally the operating frequency of task T_p 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 we round up the execution speed of task T_p to the a higher speed $S_{p,new}$ where the needed extra energy is no less than E_O. That is:

$$E_D(st_p, w_p/S_{p,new}) - E_D(st_p, w_p/S_p) \ge E_O \quad (7)$$

where $w_m/S_{p,new}$ is the new execution time, and $E_D(st_m, w_m/S_{p,new}$ is the new energy dissipation for the task. In some cases, even if task T_p is executed at full speed S_{max}, E_O cannot be exhausted, then we schedule task T_p at the full speed S_{max}. As the task is executed at a higher speed, it will be completed earlier than expected and hence increase the slack of future tasks. After the new execution speed for task T_p is decided, we need to update the finishing time of the current task, as shown in line 3 to 4 in Algorithm 2.

Algorithm 2: Deal with Energy Overflow and Shortage

```
Require: maintain P tasks in Q;
1.  if (overflow energy){
2.    calculate new operating frequency for current task based on
      Eq(7);
3.    update the finish time of the current task;
4.    }
5.  } elsif (energy shortage) {
6.    remove the task;
7.    }
8.  }
```

When energy shortage is predicted to occur during T_p, if there is still slack available due to previous speed up, we utilize the slack to wait for the battery recharge otherwise the task will be removed from task queue even before it starts. This is because, even if we gathered enough energy by delaying the execution of T_p, then the successors of T_p is highly likely to miss deadline, because all the tasks are scheduled to execute at the lowest possible speed. One the other hand, removing task T_p can save the harvesting energy for successor tasks. Note that by removing T_p, more slack is created.

D. Task Slack Management

Following the above mentioned scheduling algorithm, the CPU will not be fully occupied. This is because,

1) The processor provides a finite number of discrete speeds (e.g. Table 1), and our algorithm selects the lowest possible speed that is higher than or equal to the summation of utilization of all tasks.
2) Some tasks will be executed at a higher speed and finished earlier if energy overflow is predicted.
3) Some tasks will be removed when energy underflow occurs.

All of these contribute to task slacks, which should be utilized to achieve energy efficiency in EH-RTES.

The slack can be utilized either by inserting idle period to let the system harvest more energy or by further slowing down the future tasks for lower energy consumption. In this work, we design four Task Slack Management (TSM) algorithms to utilize the slack, including three slack-consuming algorithms and one slack reclamation algorithm.

The slack-consuming algorithms consume the slack by inserting idle periods (i.e. halt the system). An idle period is a chance to refill the energy storage because the system power consumption is extremely low during this time. Note that a task will be speed up only when there is a predicted overflow in the energy storage, which means that right after the completion of the task, the energy storage is usually full. Therefore, it is better to save the slack to the future and consume it while the battery is low.

Based on when the slack time is consumed, we name the three TSM policies as, ASAP (As Soon As Possible)-TSM, ALAP (As Late As Possible)-TSM and MSTF(MoST Fitted)-TSM, The ASAP-TSM policy insert the idle period as soon as the battery is below 80% of the full capacity. As soon as the battery is fully charged, the CPU will resume its current execution and the remaining slack will be reserved for the future. The ALAP-TSM policy holds the slack until there is no more task to be executed and an idle period is automatically inserted. The MSTF-TSM policy will not consume the slack unless the energy harvesting rate exceeds a threshold E_{th}, This ensures that the battery to be charged at a higher rate and the slack is utilized efficiently.

Algorithm 3: Slack Reclamation

```
Require: Task slack is available after $T_p$ is executed;
1.  for $i = p + 1{:}P$ {
2.    $st_i = \min(st_i, ft_{i-1})$;
3.    if ($st_i + w_i/S_{id[i]-1} < ft_i$ && $(w_i/(w_i + slack) \le S_{id[i]-1}$);{
4.      slack = slack − ($w_i/S_{id[i]-1} - w_i/S_{id[i]}$);
5.      $S_i = S_{i,slow}$;
6.    }
7.    $ft_i = st_i + w_i/S_i$;
8.  }
```

The slack reclamation algorithm utilizes the slack to further slow down the future tasks for more energy saving. Its basic steps are described in Algorithm 3. Let *id*[*i*] be the index of the execution speed of the periodic task *i*. Its initial value is determined based on the overall utilization as discussed previously. Before a task is executed, we will first check the available slack. If there is enough slack to slow down the task execution to the next lower speed (i.e. $w_i/(w_i + slack) \le S_{id[i]-1}$) and the slowdown will not cause any energy overflow, then the task will be executed at speed $S_{id[i]-1}$ (Step 3~5 in Algorithm 3). The remaining slack will be reclaimed by other tasks or it will be consumed as idle period as late as possible. We reduce the execution speed of a task to the next lower frequency instead of extra lower frequency because this helps to distribute the slack to different tasks more evenly.

E. Overall Utilization-based Task Scheduling

The overall utilization-based task scheduling and DVFS method is shown in Algorithm 4, which consists of four steps. Step 1 generates the initial schedule by sorting tasks based on their deadlines, as shown in Line 3. Step 2 in Line 4 calls Algorithm 1 to schedule tasks based on the summation of utilization of all tasks. Step 3 in Line 6, is checking the energy availability of EH-RTES. If the overflow energy is predicted, the scheduler speedups the current task to eliminate the wasted energy; on the other hand, it proactively drops a task to save more energy and CPU time for other tasks, as shown in Algorithm 2. Step 3 in Line 7 is one of four Task Slack Management (TSM) algorithms that utilizes the task slack to achieve energy efficiency, once the scheduling is done, the tasks in the task queue will be executed under selected speed and removed from the queue afterward, as shown in Line 8 and 9. In general, because the new algorithm consumes less energy comparing to AS-DVFS, this should result in lowering deadline miss rate, which will be shown in experimental result section.

Algorithm 4: Overall Utilization Based Scheduling and DVFS

```
Require: maintain a ready task queue Q
1.   while (true) {
2.    if (incoming new task){
3.      push new task into $Q$, sort all task based on their
      deadlines;
4.      schedule task in $Q$ according Algorithm 1;
5.    }
6.    check energy availably according Algorithm 2;
7.    manage the task slack according to Task Slack Management
      Algorithm mentioned in section III.D
8.    execute current task in the task queue;
9.    remove finished task from $Q$;
10.  }
```

IV. Utilization Based Task Mapping for Multi-Core system

According to Algorithm 1, all tasks in the task queue execute at the same lowest speed S that is higher than or equal to the summation of utilization. The task execution time is given by w/S, thus it is easy to calculate that energy dissipation of all n tasks in task queue at every hyper-period D, where $D = \mathrm{LCM}(D_1, \ldots D_n)$,

$$E = \sum_i g(S_i)\frac{w_i}{S_i}\frac{D}{D_i} = D\sum_i \frac{g(S_i)}{S_i}\frac{w_i}{D_i} = D\frac{g(S)}{S}\sum_i u_i = Dg\left(\sum_i u_i\right) \quad (8)$$

where w_i is worst case execution time, S_i denotes the execution speed of task i. Because all tasks are executed at the same speed, S_i, $1 \le i \le n$, is also equal to S, where $S = \sum_i u_i$. Equation (8) indicates that the energy dissipation E is a convex function of summation of utilization of all tasks in the queue and also is an increasing function in any non-negative region.

In a homogenous multi-core system, if the execution speed and the supply voltage of each core are selected according to algorithm 1, then their energy dissipation has the following relations. In general, given 2 cores, core M_i and core M_j, if $\sum_i U_i \le \sum_j U_j$, then $E(M_j) \ge E(M_i)$. Because the energy dissipation of a core is a convex function of its utilization, it can also be proved that distributing the tasks to cores so that they have the same (or similar) CPU utilization minimizes the overall system energy [9].

Based on the above analysis, we proposed the *UTilization Based* (*UTB*) task mapping. All of the periodic tasks will be sorted based on descending order of their utilization. Starting from the first one (i.e. the one with the highest utilization), each task will be assigned to a core that has the lowest utilization.

Algorithm 5: Multi-Core UTB Partitioned Scheduling

```
Require: maintain a ready task queue Q
1.  sorted periodic tasks based on non-increasing order of their
    utilization;
2.  for $i$ = 1:$P$ {
3.    find the core $C_j$ with the lowest utilization;
4.    allocate the task $T_i$ to the core $C_j$;
5.  }
6.  execute Algorithm 2;
```

The key idea of this new algorithm is that it uses a simple and mathematically proven method to allocate tasks in a multi-core platform. This method not only reduces the system computation complexity, but also achieves the best energy dissipation in multi-core system. The overall UTB scheduling algorithm is shown in Algorithm 5. We assume that each core either executes Algorithm 4 by itself, or there is a centralized program that runs Algorithm 4 for each core.

V. PERFORMANCE EVALUATION

This section provides the experimental setup, and the performance evaluation of the proposed algorithm. A discrete event-driven simulator in C++ is developed in this experiment. We also implemented AS-DVFS for single-core processor [4] and Task Movement Algorithm (TMA) for multi-core processor [13] for comparison purpose.

A. *Experiment Setup*

As mentioned in 2.1, we consider solar energy as the source of energy harvesting in this paper. We use four different daytime solar radiation profiles collected in [5][6]. As it is pointed out in reference [6], the *moving average* based predictor has better accuracy in predicting solar energy, comparing to other techniques. Here we adopt *moving average* as our prediction method to forecast the near future harvested energy.

Experiments are conducted on both single-core and homogenous multi-core processors. We evaluated our algorithm using two types of cores with different DVFS capabilities. The first one is an Intel XScale processor and while the second one is a PowerPC405PL [10]. Their frequency levels and corresponding power consumptions are given in Table 1 (section 3.1) and Table 2. The multi-core processor consists of 2 to 4 identical cores.

TABLE II. POWERPC405PL POWER AND FREQUENCY LEVELS.

Frequency(MHz)	33	100	266	333
Voltage(V)	1.0	1.0	1.8	1.9
Power(mW)	19	72	600	750
Normalized Speed	0.1	0.3	0.8	1.0

The workload on the single or multi-core processors are randomly generated task sets with different utilizations. To design different workload, we introduce the notation of average core utilization U_{ave}, which can be calculated as:

$$U_{ave} = \frac{U_{tot}}{M} = \frac{\sum_i \frac{w_i}{D_i}}{M} \le 1 \qquad (9)$$

where w_i and D_i are the worst case execution time and the period of task i respectively, $\sum_i \frac{w_i}{D_i}$ is the total utilization U_{tot} of all tasks in the system, and M is the number of cores. Simply speaking, average processor utilization U_{ave} is the total utilization divide by the number of cores. It is important to note that to have a feasible schedule is to have $U_{ave} \le 1$.

In the experiments, task sets with U_{ave} from 0.1 to 0.9 have been generated with a step of 0.1. The energy storage is assumed to be half full at the beginning of the simulation. Without loss of generality, the charging/discharging efficiency of ES Module of is fixed to be 0.9, and the efficiency of CM circuitry is 0.9. For each combination of solar profile and utilization setting, we simulate the system behavior from 7AM to 7PM. Each simulation is repeated for 1000 times, each time with a new random task set. Experiments with different architectures have also been conducted, due to the space limitation, only the results for PowerPC405PL are presented. However, the results for XScale have very similar trends.

B. *Simulation result of Single Core Processor*

For real-time embedded systems, deadline miss rate (DMR) is one of the most important performance metrics, which is the ratio of the number of tasks missing their deadline to the total number of tasks. First, we examine the DMR of EH-RTES with single-core processor. We conducted experiments with 4 different solar power profiles in [5][6], and recorded the correspondent DMR in Figure 4.

From Figure 4, we can see that, our utilization-based algorithm achieves lower DMR than AS-DVFS, thus improve the system performance. We can see that, our approach provides the most improvement at workloads with medium utilization (i.e. 0.4~0.6) and our algorithm performs consistently well all four profiles.

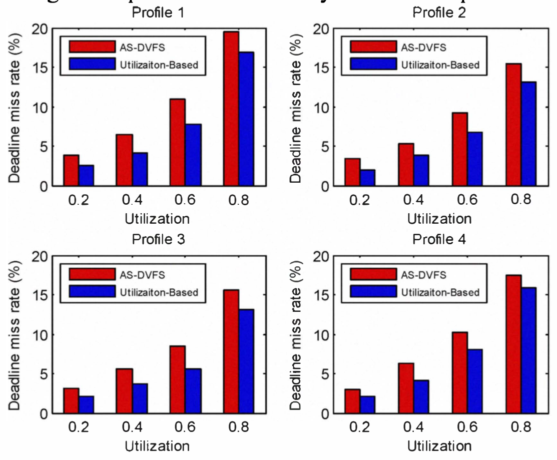

Figure 4. Deadline miss rate for system based on PowerPC core

The performance gain of our algorithm slightly reduces in systems with extremely low or high CPU utilization. This is because when the utilization is extremely low (i.e. U<0.2), in both algorithms, all tasks in the task queue are able to be executed at the lowest speed, which diminishes their difference in DMR. When the utilization is extremely high (i.e. U>0.8), all tasks in the task queue must be execute at the highest speed, this again will blur the different between two algorithms. When U is at medium level, comparing to our utilization-based algorithm, AS-DVFS may over stretch certain tasks, which results in consuming more energy; it further causes more DMR especially when system is at low energy availability.

C. *Task Slack Management Algorithms*

In the next experiment, we compare the performance of 4 different task slack management (TSM) policies. Figure 5(a) gives the 12-hour profiled sun intensity for this experiment. As we can see, in general, the sun radiation intensity increases from morning to noon and decreases from noon to evening. It has large variation in the morning due to weather condition. Figure 5(b) gives the recorded DMR during the day for those 4 TSM policies. The TSM1~4 represent the ASAP-TSM, ALAP-TSM, MSTF-TSM and the slack reclamation policy as described in Section III.D. Each data point in the figure shows the average DMR of 15 minutes. In this experiment, the utilization is set to be 0.5.

As we can see from Figure 5(b), for all the TSM algorithms, the DMR is zero at first because the initial battery is 50% full. The

DMR increases later because the sunlight intensity is too small to provide enough energy for tasks execution at the beginning of the day and extra energy will be drawn from the ES Module. As the battery becomes depleted, the DMR starts increasing. Then, the DMR remains stable although sunlight intensity varies significantly. As sunlight intensity continuous increasing, it soon becomes sufficient to power all tasks and the DMR gradually drops to 0. When sunlight intensity starts decreasing, the DMR begins to increase again. The results show that TSM-4 and TSM-3 provides up to 10.91% and 8.86% improvement compared to TSM-1 and TSM-2 respectively.

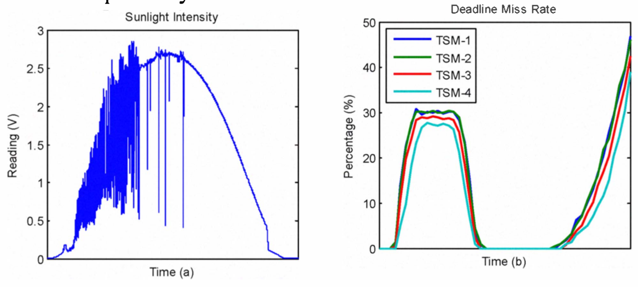

Figure 5. (a) Sun intensity and (b) DMR for four TSM algorithms

D. *Simulation result of Multi-core Processor with different partition algorithm*

In this set of experiments, we focus on comparing our proposed *UTilization Based* (*UTB*) multi-core partition algorithm with the random partition algorithm and task movement algorithm [13]. We test these three algorithms using harvesting power from Profile 1 [5][6] with 3 and 4 processor cores.

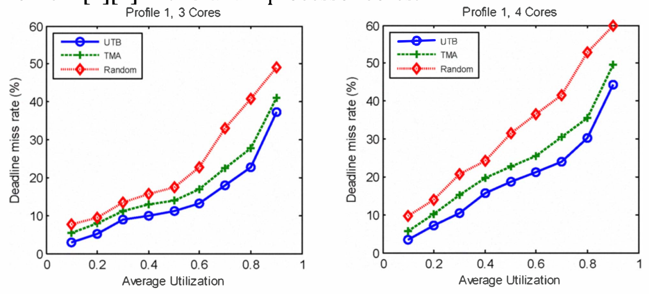

Figure 6. Deadline miss rate with different cores

Figure 6 shows that the DMR of an EH-RTES with 3 and 4 cores running random partition algorithm, TMA and our UTB partition algorithm. As shown from both Figure 6(a) and 6(b), the proposed UTB partition algorithm achieves lowest deadline miss rate, comparing to the other two algorithms. This is because the proposed UTB multi-core algorithm always assigns tasks to the core which results in lowest overall power consumption, therefore UTB algorithm achieves lowest deadline miss rate among three partition algorithms. The absolute deadline miss rate reduction increases when the utilization decreases. At lower utilization settings, the UTB algorithm achieves slightly reduction in deadline miss rate over the random algorithm and TMA, however at higher utilization settings, the UTB algorithm achieves significant reduction, about 25.33% and 10.21% in deadline miss rate over the random partition and TMA on 4 cores with U_{ave} is set to be 0.9.

E. *Simulation result of Single-core and Multi-core Processor*

Finally, we compare the proposed UTB algorithm with different core while setting the total utilization U_{tot} equally. Table 3 shows the average utilization U_{ave}, average DMR, normalized consumed energy consumption and normalized overflow energy when the U_{tot} total utilization is set to be 60%. As we can see in Table 3, the amount of works that the each core performs decreases with increasing core numbers, thus the multi-core system is able to execute more tasks under the same solar panel and energy storage; on the other hand, multi-core system is able to utilize the overflowed energy that is wasted due to the limited energy storage capacity, therefore the consumed energy is increasing and the average DMR and overflowed energy is decreasing.

TABLE III. UTB ALGORITHM WITH DIFFERENT CORES

Core Number	1	2	3	4
U_{ave} (Average Utilization %)	60	30	20	15
Average DMR (%)	8.61	6.52	5.21	4.19
Normalized Consumed Energy	1	1.06	1.09	1.13
Normalized Overflowed Energy	1	0.94	0.89	0.84

VI. CONCLUSION

In this paper we proposed a low-complexity and effective task scheduling algorithm for EH-RTES based on task utilization, then we proposed the *UTilization Based* (UTB) partitioned methods to schedule periodic tasks on multi-core scheduling. Experimental results show that, the proposed algorithm has better performance on single-core processor in terms of task deadline miss rate comparing to existing methods, and UTB algorithm on multi-core scheduling is able to achieve less deadline miss rate than random and TMA multi-core scheduling. We have also illustrated the trend of DMR with different TSM algorithms, and compared the DMR, consumed energy and overflowed energy of single-core and multi-core processor.

REFERENCES

[1] J. Rabaey, F. Burghardt, D. Steingart, M. Seeman and P. Wright, “Energy Harvesting A Systems Perspective,” *IEEE International Electron Devices Meeting*, 2007, December, 2007, pp. 363-366

[2] C. Moser, D. Brunelli, L. Thiele, and L Benini, “Lazy Scheduling for Energy-Harvesting Sensor Nodes,” *Proc. of Fifth Working Conference on Distributed and Parallel Embedded Systems*, 2006.

[3] S. Liu, Q. Qiu, and Q. Wu, “Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting”, *Proc. of Design, Automation, and Test in Europe*, 2008.

[4] S. Liu, Q. Qiu, and Q. Wu, “An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-time Energy Harvesting Systems”, *Proc. of Design Automation Conference*, 2009.

[5] S. Liu, J. Lu, Q. Wu and Q. Qiu, “Load-matching adaptive task scheduling for energy efficiency in energy harvesting real-time embedded systems”, *Proc. of International Symposium on Low Power Electronics and Design*, August 2010.

[6] J. Lu, S. Liu, Q. Wu, and Q. Qiu, “Accurate Modeling and Prediction of Energy Availability in Energy Harvesting Real-Time Embedded Systems”, *The First International Green Computing Conference*, August 2010.

[7] J. Recas, C. Bergonzini, D. Atienza, and T. S. Rosing, “Prediction andmanagement in energy harvested wireless sensor nodes,” *Proc. VITAE’09*, May 2009.

[8] A. Ravinagarajan, D. Dondi and T. S. Rosing, “DVFS based task scheduling in a harvesting WSN for structural health monitoring,” *Proc. of Design, Automation and Test in Europe*, 2010.

[9] H. Aydin and Q. Yang, “Energy-Aware Partitioning for Multiprocessor Real-Time Systems”, *Proc. of International Parallel and Distributed Processing Symposium*, 2003

[10] G. Zeng, T. Yokoyama, H. Tomiyama and H. Takada, “Practical Energy-Aware Scheduling for Real-Time Multiprocessor Systems,” *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2009*.

[11] J.-J Chen and L. Thiele, “Energy-Efficient Scheduling on Homogeneous Multiprocessor Platforms,” the 25th *ACM Symposium on Applied Computing*, Switzerland, March 22-26, 2010.

[12] P. Koch, “How to Interface Energy Harvesting Models with Multiprocessor Scheduling Paradigms,” *the 1st International Conference on Wireless VITAE* 2009.

[13] T. Wei, Y. Guo, X. Chen and S.Hu, “Adaptive Task Allocation for Multiprocessor SoCs in Energy Harvesting Systems,” the *11th International Symposium on Quality Electronic Design*, March 2010.