Thermal-Aware Semi-Dynamic Power Management for Multicore Systems with Energy Harvesting

Yi Xiang and Sudeep Pasricha
Department of Electrical and Computer Engineering,
Colorado State University, Fort Collins, CO, USA
E-mail: {yix, sudeep}@colostate.edu

Abstract

In this paper, we focus on power and thermal management for multicore embedded systems with solar energy harvesting as the power source and a periodic hard real-time task set as the workload. We design a novel semi-dynamic scheme, which reschedules tasks at the beginning of specified time epochs. By rejecting job instances of certain tasks until the next rescheduling point, our scheduler dispatches a subset of tasks that comply with the predicted energy budget and thermal conditions. Our approach reacts to run-time energy harvesting power variation without losing the consistency of the periodic task set, which helps to scale processor speed evenly by utilizing slack time efficiently without the need for complex slack reclamation algorithms, as in prior work. When applied to a multicore platform, our approach offers a chance to shut down cores and reassign tasks for superior energy efficiency. As a result, experimental results show up to 70% miss rate reduction compared to prior work. Unlike any prior work, our approach also integrates thermal management to reduce peak temperature while minimizing miss rate for energy harvesting embedded systems.

Keywords

Energy harvesting, task scheduling, power management

1. Introduction

Power, energy and thermal constraints have enforced significant change in the design of contemporary computing systems. Due to unacceptable power dissipation at higher clock rates and more potent power leakage with technology scaling, single-thread performance has been slowing down. Thus thread-level parallelism (TLP) to improve performance within a given power budget is widely practiced across various computing platforms, ranging from high-end servers to desktops, as well as embedded devices. Recent years have also seen a significant increase in popularity of multi-core processors in low power embedded devices [1]. With advances in parallel programming and power management techniques, embedded devices with multicore processors and TLP support outperform single-core platforms in terms of both performance and energy efficiency [2]. But as core counts rise to cope with increasingly complex applications, techniques for workload distribution and power management are the key to achieving significant energy savings in emerging multi-core embedded systems.

For some applications, we need energy autonomous devices that utilize ambient energy to perform computations without relying entirely on an external power supply or frequent battery charges. As the most widely available energy source, solar power harvesting has attracted a lot of attention and is rapidly gaining momentum. Several embedded systems with intelligent energy harvesting and transfer have been proposed, such as Heliomote [3] and Prometheus [4]. There has been some research on Maximum Power Point Tracking (MPPT) [5]-[7] as well, which focuses on applying the proper load on circuitry to draw the maximum load for solar panels. To fully exploit the capability of these energy harvesting systems, a considerable amount of work has also explored task scheduling, primarily for embedded systems with real-time task sets. Most of these efforts are derived from the earliest deadline first (EDF) algorithm, as it has been shown to be optimal for scheduling a known task set with timing constraints in preemptive system. An early work in [8] called lazy scheduling (LSA) executed tasks as late as possible, reducing deadline miss rates when compared to the EDF algorithm. However, it does not consider DVFS frequency selection and always executes tasks at full speed. Because a processor's dynamic power is generally a convex function of frequency, operating the processor at a lower frequency often results in a higher energy efficiency. Liu et al. [9] proposed the EA-DVFS technique that takes processor DVFS into consideration. EA-DVFS utilizes task slack to slow down execution speed for energy savings and thus outperforms LSA in miss rate when total task utilization is low. Later the same authors proposed a more intelligent technique called HA-DVFS [10], which improves energy efficiency mainly by distributing multiple arriving tasks as evenly as possible over time and executing them with more uniform frequency. However, these works focus on uni-processor systems and have not considered execution on multi-core platforms. Recently, a utilization-based technique (UTB) was proposed in [11] to better address periodic task scheduling in energy-harvesting system. UTB takes advantage of predictability provided by periodic task information for better task distribution. Moreover, UTB proposed a simple extension to support multi-core platforms by allocating a subset of tasks to each core and executing the single-core UTB algorithm separately on each core.

In this paper, we propose a thermal-aware semi-dynamic algorithm (TA-SDA) to reduce the deadline miss rate of periodic tasks under variant and insufficient energy harvesting. Compared to prior work, TA-SDA reacts to run-time energy shortage and fluctuations proactively to find significantly greater scope for energy savings, especially for multi-core platforms. At the system level, TA-SDA is triggered at specified time epochs to adjust inter-core task allocation and set a per-core execution strategy based on an averaged energy budget estimated for a period of few minutes until the next specified reschedule time. In addition, TA-SDA offers the flexibility to tackle processor overheating by re-allocating workload or proactively shutting down cores. At the core level, a novel dual-speed method is designed for periodic tasks to address discrete frequency levels and DVFS switching overhead. Our experimental studies show that TA-SDA outperforms the best known prior work, achieving superior drop rate reduction and energy efficiency. *To the best of our knowledge, this work is also the first to integrate thermal-awareness during resource allocation in energy-harvesting embedded systems.*

2. Problem formulation

2.1. System model

In this paper, we focus on the problem of power and thermal management of real-time embedded systems with periodic tasks, powered by solar energy (Fig. 1). To cope with the unstable nature of the solar energy source, a supercapacitor is used to buffer solar energy collected by photovoltaic cells. In the rest of this section, we describe our target system and objective.

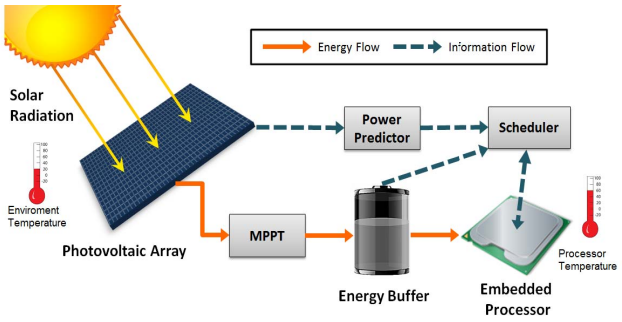

Figure 1: Real-time embedded processing with energy harvesting

2.1.1. Energy harvesting and storage model

A photovoltaic array is used as a power source for our embedded system, converting ambient solar energy into electric power. Naturally, the amount of harvested power varies over time due to changing environmental conditions, like angle of sunlight incidence, cloud density, temperature, humidity, etc. In our study, the converted solar power at time t is denoted as $P_H(t)$. The energy E_H charged into a supercapacitor between time instances t_1 and t_2 is given by

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

where η_{harv} is a coefficient between 0 and 1 to represent charging efficiency of the supercapacitor. We assume that a near ideal Maximum Power Point Tracking (MPPT) circuitry [5]-[7] is deployed between the photovoltaic array and supercapacitor to achieve high and constant recharging efficiency. The capacity of the energy storage device is limited, and denoted as E_{cap}. Clearly harvested energy will be wasted if the energy storage device is already fully charged. We assume that task execution must be halted when remaining energy is less than 10 percent, thus reserving enough storage to maintain system status and ensure graceful shutdown. In our work, we choose a supercapacitor as the energy storage medium, because it outperforms traditional electrochemical battery in terms of charge/discharge efficiency and does not require frequent replacements due to its high number of recharge cycles [12]. Although a battery does have some advantages, such as higher energy–weight ratio and better high voltage capability, it is not very beneficial to our problem as our processors operate at very low voltage levels (less than 2V) and an energy storage with small capacity is adequate to act as an energy buffer that helps smooth out transient variations in barely sufficient solar power harvesting.

2.1.2. Periodic real-time task model

We assume a task set of N independent periodic real-time tasks ψ: $\{\tau_1, \ldots, \tau_N\}$, in which each periodic task τ_i has a characteristic triplet (C_i, D_i, T_i), $i \in \{1, \ldots, N\}$. C_i is the maximum number of CPU clock cycles needed to finish a job instance of task τ_i, referred to as worst-case execution cycles (WCEC). The relative deadline of the task, D_i, is the time interval between a job's arrival time and its deadline. A job instance is missed if it is not finished before its deadline. T_i is the period of the task. At the beginning of each period, a new job instance of that task will be dispatched to the system. Like most recent works on periodic task scheduling (e.g., [11]) we assume that D_i equals T_i, with all jobs finishing before the arrival of the next job instance of the same task. In addition, we define an attribute X_i, which is the miss penalty associated with each task. Each time that a task's job misses the deadline, the job will be aborted and the penalty applied to the system. Thus, we can refine the triplet for task τ_i as (C_i, T_i, X_i). The relative importance of a task can be characterized by a *penalty density*, defined as the ratio of the task miss penalty and WCEC (X_i/C_i) [15].

Table I: XScale processor [13] power and frequency levels

Level	0	1	2	3	4	5
Voltage(V)	-	0.75	1.0	1.3	1.6	1.8
Power(mW)	40	80	170	400	900	1600
Frequency(MHz)	idle	150	400	600	800	1000
Energy Efficiency	0	1.875	2.353	1.5	0.889	0.625

2.1.3. DVFS-enabled processor model

We consider a homogeneous multicore processor with DVFS capability at the core level and support for task preemption. Each core has M discrete voltage and frequency levels: φ: $\{L_0, \ldots, L_M\}$. Each level is characterized by L_j: (v_j, p_j, f_j), $j \in \{1, \ldots, M\}$, which represents voltage, average power, and frequency respectively. We consider power-frequency levels of the Xscale processor as shown in Table I. Here level 0 represents the idle power of the processor when no task is executed while the system stays in active state. Typically, the dynamic power-frequency function is convex. Thus, a processor running at lower frequency can execute the same number of cycles with lower energy consumption. However, this is not always the case when static power is considered. To find an energy optimal frequency, we represent energy efficiency of a v-f level L_i by δ_i = cycles executed / energy consumed = f_i / p_i. From Table I we can conclude that level 2 is the most energy efficient because executing at this level consumes the least energy for a given number of cycles. We call the most energy efficient level as the *critical level* and thus $f_{\text{crt}} = f_2$. Although it is desirable to execute tasks at this critical frequency level, due to unique task timing constraints executing tasks at the critical level may end up being insufficient to finish all task instances by their deadlines.

We also define the utilization of a periodic task (U) with respect to the full speed provided by processor. That is, a task's utilization is its execution time under highest frequency divided by its period,

$$U_i = \frac{C_i/f_{max}}{T_i} \qquad (2)$$

The utilization for a task set is simply the accumulation of all tasks' utilizations. In preemptive real-time system, a task set is schedulable by EDF algorithm under a frequency level j if it meets the condition that

$$U_{total} \leq \frac{f_j}{f_{max}} \qquad (3)$$

When total utilization is known, the most energy efficient frequency can be deduced from this inequality, under the condition that $f_j \geq f_{crt}$.

Unlike many of the prior works discussed earlier, we consider DVFS switching energy overhead in our analysis. According to the model in [14], we set average switching energy overhead, E_{switch}, to be in the order of millijoules. The DVFS switching delay is in the range of a few microseconds, negligible when compared to task execution times, which are in the hundreds of millisecond range in our work.

Also, unlike any prior work, we consider thermal management in an energy harvesting multi-processing environment. We assume that each core in our multiprocessor has a unique Digital Thermal Sensor (DTS) implemented to monitor run-time temperature independently [18]. We set 85°C as the thermal setpoint at which throttling is initiated in the processor (i.e., throttling threshold = 85°C) [19]. When throttling is triggered, a core must halt execution and shift to idle state until its temperature drops to 80°C.

2.1.4. Power management modules

A scheduler module is an important component of the system for information gathering and execution control. The scheduler gathers information by monitoring the energy prediction module, energy buffer (supercapacitor), execution state, and core temperature (Fig. 1). The gathered data, together with profiled periodic task set information, informs a power management algorithm in our scheduler that coordinates operation of the multi-core platform. Each core is assigned a strategy by the scheduler to guide intra-core execution. The predictor keeps track of harvesting power history and provides prediction results to the scheduler for energy budget estimation. The functionality of these modules is implemented in software to simplify system implementation and configuration.

2.2. Scheduling Objective

Inspired by the conclusions of prior work, our primary objective is to reduce total task miss rate under variant and stringent energy and thermal conditions at runtime. Our technique should react to changing energy dynamics while incorporating periodic task information. Peak temperature should be minimized to avoid throttling which can significantly degrade performance. We also want processor speed to be as stable as possible for better energy efficiency. As we take DVFS switching overhead into consideration, our technique should also avoid frequent DVFS level switching.

3. Motivation

In this section, we present our motivation to apply a semi-dynamic framework and use its flexibility for thermal management in energy harvesting multi-core embedded system platforms.

3.1. Motivation for semi-dynamic framework

Most previous works deal with dynamic solar power variation by halting, dropping, or speeding up execution of a current task, changing instantly from an initial schedule deduced offline. For energy harvesting aware periodic task set scheduling, the most recent work, UTB [11], also follows this route. Although UTB deduces an optimal initial schedule offline, assuming sufficient energy provided, as shown in Fig. 2(a), there is still scope for improvement, as discussed below:

1) Most importantly, the task drop mechanism in UTB reacts to energy shortage *passively*, only when the current task lacks sufficient energy to finish in time. In the motivational example shown in Fig. 2, we assume a task set with four periodic tasks ($\tau_1 \sim \tau_4$), where each task has WCEC of 2.4 million CPU cycles and task period of 12ms. According to Table I, Eq. (2) and Ineq. (3), UTB initially sets execution frequency to 800MHz so that all tasks can finish with the best efficiency if energy is sufficient, as shown in Fig 2(a). However, the real issue lies in the run-time power management with insufficient energy budget. Let us assume remaining energy in the supercapacitor is 7200μJ and harvesting power in the next 36ms (3 periods) is 200mW, i.e., 200μW incoming energy per microsecond. According to Table I, executing at 800MHz has power consumption of 900mW, which is dramatically higher than 400mW for 600MHz due to quadratic relation between frequency and power consumption. The result of UTB for this scenario is shown in Fig. 2(b), where only 6 out 12 job instances are finished, resulting in a 50% miss rate. With the same energy budget, our proposed TA-SDA approach copes with energy shortage by *proactively* dropping tasks. It predicts energy budget before execution and then drops one task, τ_4, which helps to execute remaining tasks at a steady lower frequency. As can be seen in Fig. 2(c), all accepted job instances for $\tau_1 \sim \tau_3$ are finished and overall miss rate is 25%, significantly lower than 50% of UTB.

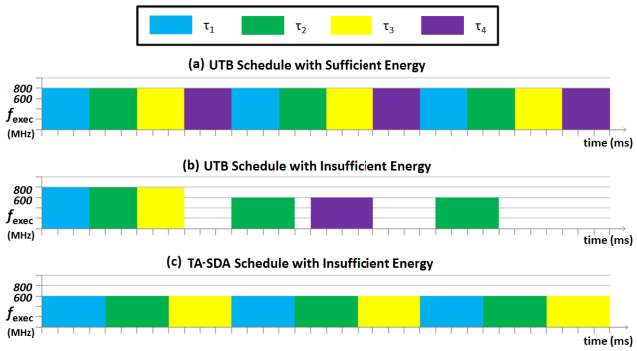

Figure 2: Motivation for proposed energy harvesting-aware scheduling

2) On multicore platforms, UTB partitions tasks into separate sets and then executes each set on a core using a single-core scheduling algorithm. However, as all cores are dependent on the same energy source, such isolated run-time adjustment is not amenable to learning upcoming energy requirements of other cores, leading to suboptimal or even faulty schedules. In addition, static task partitioning in UTB wastes the flexibility provided by a multicore platform. In contrast, TA-SDA triggers task rescheduling to exploit multicore flexibility.

In summary, there are many limitations with prior work. Our TA-SDA technique (described in Section 4) addresses these limitations.

3.2. Motivation for thermal management

Here we present our motivation for considering thermal management in energy harvesting embedded systems.

1) With limited power budgets for energy harvesting embedded systems, it is impractical to apply power-hungry air cooling systems. The very slow heat dissipation rate of natural convection in these systems can easily end up causing thermal emergencies during their long operation periods. Such overheating of processors is known to harm system reliability and stability. Perhaps most importantly, frequent thermal throttling that is initiated in processors to cope with thermal emergencies can disrupt scheduling strategies, reducing system performance and overall energy efficiency.

2) Due to the inherent nature of incident solar energy, our energy harvesting systems tends to receive abundant energy around middle of the day. During such times, heavy workloads can run at full speed. However, continuously executing at full-speed creates excessive heat in the processor package and leads to overheating issues. Around the same time, the ambient temperature is usually highest of the day, assuming no dramatic weather changing (Fig. 3). As the energy harvesting embedded system is deployed outdoors to enable efficient energy scavenging, the exposure to the ambient environment makes it

even more difficult for the processor to cool down around those hours without intervention.

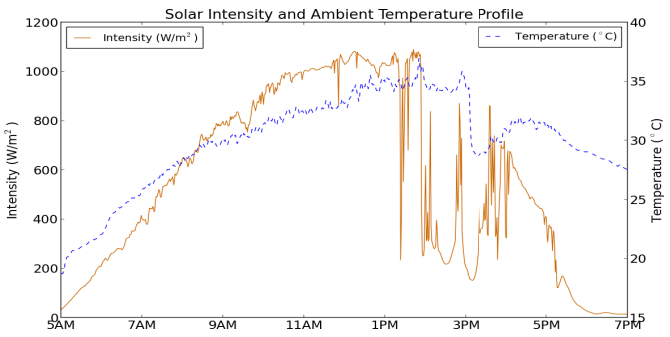

Figure 3: An example of solar intensity vs. ambient temperature

4. TA-SDA power management framework

In this section, we describe our thermal-aware semi-dynamic algorithm (TA-SDA), an improved power management technique for solar powered energy harvesting systems running periodic task sets.

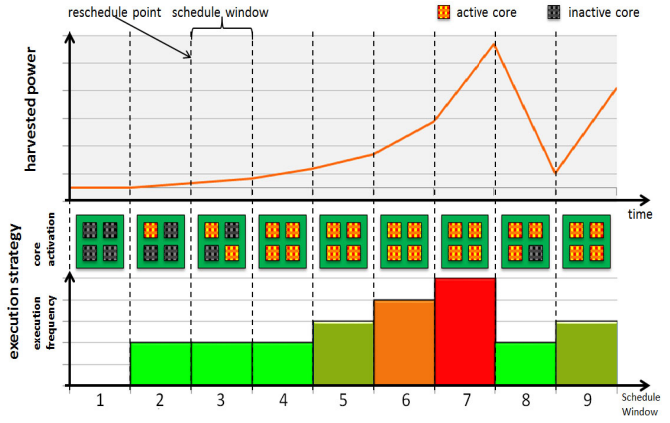

Figure 4: Illustration of semi-dynamic framework

One of the underlying ideas behind TA-SDA is to exploit time-segmentation during power management, as illustrated in Fig. 4. At each specified time interval (epoch), there is a reschedule point, where the execution strategy can be adjusted based on energy availability predication. A time frame between two reschedule points is called a *schedule window*, within which a strategy specified at the beginning is in effect until the next reschedule point. Thus reschedule points provide dynamic adaptivity needed by the energy harvesting aware system to adjust inter-core task execution strategy while the schedule window maintains a stable execution so that periodic task information can be utilized for better energy savings, as in Fig. 2(c). From schedule window 1 to 4 in Fig. 4, it can be seen that under low energy conditions, TA-SDA maintains execution at low (critical) frequency and increases the number of active cores to finish more tasks as harvested energy increases. Cores only execute at higher frequency when energy harvested is abundant. Thus TA-SDA has better energy efficiency as frequency within a schedule window remains stable.

At each reschedule point, our technique is composed of three stages, which attempt to address issues mentioned in the previous section. These three stages are: *(i) thermal-aware active core count selection*, which selects number of processing cores to activate; *(ii) thermal- and penalty-aware task rejection* to filter out subset of tasks that are less important and cannot be supported by the energy budget; *(iii) dual-speed switching method* to select optimal frequency from the frequency levels supported by processor. These three stages are organized in an order that successor stages make use of efforts made by previous stages, rather than diminishing them. Besides, stages *(i)* and *(ii)* can proactively slow down overheated cores to avoid frequent thermal throttling. The stages are described in the following sections.

4.1. Active core count selection

The main reason for having an active core count selection heuristic is that running a processor below its critical frequency actually decreases energy efficiency, as can be seen in Table I. This situation can occur when the energy budget is so low that only a small subset of tasks can be accepted, i.e., after evenly distributing these tasks to all cores, utilization on each core is smaller than maximum utilization supported by the critical frequency. With our active core count selection heuristic, we can shut down some cores at each reschedule point based on estimated energy budget. The power dissipated by inactive cores is negligible and the remaining cores can then receive enough workload to run at critical frequency. Also the associated power state switching overhead is minimal as we only trigger core shutdown at reschedule points. However, arbitrarily shutting down cores to reach a frequency higher than critical is not always optimal. Fig. 5 shows the maximum energy-efficiency for different frequencies in the XScale processor. Suppose cores execute at point *A* without shutdown. After shutdown of one core, the extra power budget allows us to run the remaining processor(s) at higher frequencies such as *B*, *C*, or *D*. But not every higher frequency is viable, e.g., frequency *D* leads to even lower energy efficiency than A, before shutdown! Thus, our heuristics should compare resulted efficiencies before making the shutdown decision.

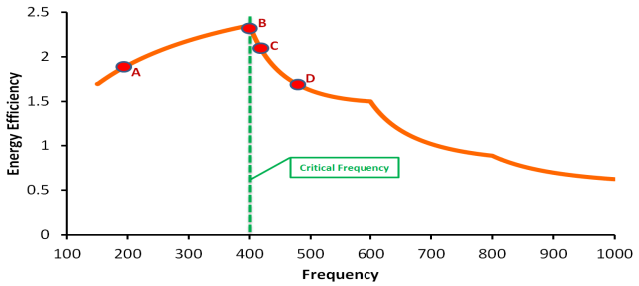

Figure 5: Energy-efficiency of XScale processor

In this heuristic, we also take core temperature into consideration. That is, we only shut down normal active cores after all overheated cores whose temperature exceeds a proactive reaction threshold, and is close to triggering thermal throttling. Also, overheated cores are set to run at critical frequency when activated, so as to finish tasks with highest energy efficiency and relatively low power dissipation.

The pseudo code of the active core count selection heuristic is given in Algorithm 1. Initially, the scheduler estimates energy budget based on harvesting power prediction for the upcoming schedule window (line 1). Then, the core shutdown procedure is triggered when energy budget is unable to support all active cores to execute at critical frequency (line 2). Subsequently (lines 3-15) if one less active core results in a better efficiency according to the profile, then the scheduler shuts down one core. Note that we keep count of active overheated cores separately

and calculate their energy based on tasks executing on these cores at critical frequency. If the energy budget for the current schedule window is extremely low, eventually all cores in the system will be shut down to save harvested energy for future execution. Recursively, these steps set the number of cores to keep active. Finally, the objective task-set utilization for penalty-aware task rejection is obtained by summing up utilization of each core (line 16).

Algorithm 1 Active Core Count Selection

Input: (i) harvesting power prediction for coming schedule window, P_{window}; (ii) energy budget to execute one core at critical frequency, E_{crt}; (iii) dual-speed method energy efficiency profile for task utilizations from 0 to 1, δ(U); (iv) number of active cores, num_core (v) number of active overheated cores, num_hot

1. $E_{window} \leftarrow P_{window} \times$ window size
2. **while** $E_{per\_core} < E_{crt}$ **and** num_core > 0 **do**
3. $E_{num\_core} \leftarrow (E_{window} - num\_hot * E_{crt}) / (num\_core - num\_hot)$
4. **if** num_hot > 0 **then**
5. $E_{num\_core-1} \leftarrow (E_{window} - (num\_hot-1)*E_{crt}) / (num\_core - num\_hot)$
6. **else then**
7. $E_{num\_core-1} \leftarrow E_{window} / (num\_core - 1)$
8. calculate f_{num_core-1} and f_{num_core}, maximum frequency supported by E_{num_core-1} and E_{num_core}
9. based on Inequality (3), calculate U_{num_core-1} and U_{num_core}, maximum utilization supported by f_{num_core-1} and f_{num_core}
10. Look up profile for δ (U_{num_core}) and δ (U_{num_core-1})
11. **if** $\delta(U_{num\_core}) < \delta(U_{num\_core-1})$ **then**
12. num_core ← num_core – 1
13. **if** num_hot > 0 **then**
14. num_hot ← num_hot – 1
15. $E_{per\_core} \leftarrow E_{num\_core-1}$, update U_{per_core}
16. $U_{obj} \leftarrow U_{per\_core} \times (num\_core - num\_hot) + U_{crt} \times num\_hot$

As a result of this selection, the number of cores activated is related to the energy budget available for the upcoming schedule window.

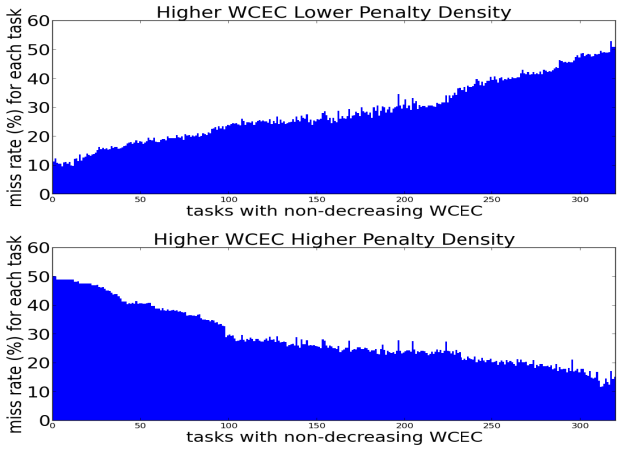

Figure 6: Miss rates for tasks with non-decreasing WCEC in TA-SDA

4.2. Penalty-aware task rejection and assignment

To add task priority control in TA-SDA, we distinguish a task's importance by assigning varied miss penalty to tasks. In this step, our scheme rejects tasks with lower penalty density (Section 2.1.2) first, rather than drop tasks with longer execution time to allocate limited energy budget to more important tasks for miss penalty reduction. In particular, for the case when all tasks are assigned an identical miss penalty, this scheme reduces miss penalty equivalent to miss rate.

We describe our task rejection heuristic with the pseudo code shown in Algorithm 2. In lines 1-7, we sort all tasks in non-decreasing order of tasks' penalty densities so that we can reject tasks one by one until the remaining tasks' total utilization is lower than the objective utilization given by Algorithm 1. The remaining tasks form the accepted task set and are assigned to all active cores using a simple but effective heuristic in lines 8-13. This heuristic not only enables priority control among tasks, but also evenly distributes workload to each core for execution under a stable frequency for better efficiency (except overheated cores, which are fixed to run at critical frequency only).

Algorithm 2 Penalty Aware Task Rejection and Assignment

Input: objective utilization from algorithm 1, U_{obj}

1. sort task set T in non-decreasing order of tasks' penalty densities
2. $T_{accepted} \leftarrow T$
3. **for** n = 1:N **do**
4. **if** $U_{accepted} > U_{obj}$ **then**
5. reject n^{th} task
6. **else**
7. done with task rejection, **break**
8. sort accepted task set $T_{accepted}$ in non-increasing order of task utilization
9. **for** n = 1:$N_{accepted}$ **do**
10. cur_core ← core with lowest utilization
11. **while** cur_core is overheated **and** $U_{cur\_core} + U(n^{th}\ task) > U_{crt}$ **do**
12. cur_core ← core with next lowest utilization
13. assign n^{th} task to cur_core

We explored two different scenarios for our penalty-aware rejection scheme: one assigns constant values to tasks so that miss penalty is equivalent to miss rate, while the other assigns penalties that are the square of tasks' WCECs to emphasis importance of longer tasks. Miss rates for all tasks arranged in non-decreasing WCEC order are plotted in Fig. 6. It can be seen that the first case has higher miss rate for longer tasks and the second case generates opposite results that actually comply with penalties assigned to the task set.

4.3. DVFS switching-aware dual-speed method

After distributing accepted tasks, based on Inequality (3) we can deduce a theoretical optimum execution frequency for the task set on each core. In most cases, however, those objective frequencies are unlikely to be supported directly by processors with discrete frequency levels. A solution for this issue is to use a dual-speed method, which approximates the objective optimal frequency by switching between its two adjacent discrete frequencies [16]. In this section, we describe the implementation details of our novel dual-speed method. For convenience, we denote the adjacent higher frequency as f_{high}, the lower one as f_{low}, and the objective optimal frequency as f_{obj}.

Firstly, to guide the switching between two adjacent discrete frequencies, we need to calculate the proportion of cycles to execute with f_{high}, denoted as α_{high}. Assume that the total number of cycles to be executed is C. Imitating f_{obj} with a combination of f_{low} and f_{high}, implies finishing C within the same amount of time, that is

$$\frac{C}{f_{obj}} = \frac{\alpha_{high} C}{f_{high}} + \frac{(1-\alpha_{high})C}{f_{low}} \quad (4)$$

From this equation, we can deduce the proportion α_{high} for each objective frequency as

$$\alpha_{high}(f_{\mathrm{obj}}) = \frac{1/f_{\mathrm{obj}} - 1/f_{\mathrm{low}}}{1/f_{\mathrm{high}} - 1/f_{\mathrm{low}}} \quad (5)$$

As f_{low} and f_{high} are determined by f_{obj}, there is a one-to-one correspondence between α_{high} and f_{obj}, and the values of $\alpha_{high}(f_{obj})$ can be calculated offline for a given task set. Based on the definition of energy efficiency in section 2.1.3, the theoretical efficiency of the dual-speed method $\delta_{dual}(f_{obj})$, can also be obtained offline as the objective frequency divide by the average power consumption.

However, it is non-trivial to get close to theoretical efficiency in a dual-speed method implementation, due to the following difficulties:

- Excessive DVFS switching results in massive switching overhead that considerably affects energy efficiency [14];
- Executing at f_{low} for too long causes task timing violations;
- Executing at f_{high} for too long results in timing slack before the arrival of new job instances of periodic tasks, which is wasted as idle cycles, thus reducing energy efficiency.

To address these issues, we implement a simple and intuitive dual-speed method with inter-task switching, as described below:

1) In order to prevent unnecessary DVFS switching, we denote number of cycles continuously executed at f_{high} to be C_{high} and set a threshold C_{thresh}. When $C_{high} = C_{thresh}$, whether switching to f_{low} or staying at f_{high} brings about the same energy consumption, i.e.,

$$p_{high} \times \frac{C_{thresh}/\alpha_{high}}{f_{high}} = p_{opt} \times \frac{C_{thresh}/\alpha_{high}}{f_{opt}} + 2 \times E_{switch} \quad (6)$$

where E_{switch} is DVFS switching overhead. The value of $C_{thresh}(f_{obj})$, can be easily calculated offline. From Eq. (6), when $C_{high} < C_{thresh}$, the system forbids switching to f_{low}, as it leads to higher energy cost.

2) To avoid task timing violation at f_{low}, our dual-speed method always set execution speed to f_{high} initially. After finding a proper chance to switch to f_{low}, execution frequency jumps back to f_{high} as soon as a certain number of cycles have been executed at f_{low} such that $C_{high}/(C_{high}+C_{low}) = \alpha_{high}$, according to the specified proportion.

3) To avoid undesirable idle cycles at f_{high}, our dual-speed method switches to f_{low} if number of unfinished job instances is not greater than 1, indicating possible shortage of workload. On the other hand, this step also helps to reduce number of switches as it halts switching to f_{low} when job instances in the queue are sufficient.

The steps above are summarized in Algorithm 3. Note that at line 3, frequency switching is not triggered if f_{obj} is lower than critical frequency, as executing below critical frequency should be avoided.

Algorithm 3 Dual-Speed Method with Inter-Task Switching

Input: (i) objective optimal frequency, f_{obj}; switching proportion and threshold profile for f_{obj} from 400 to 1000 MHz, $\alpha_{high}(f_{obj})$ and $C_{thresh}(f_{obj})$

```
1. f_cur ← f_high
2. while true do
3.     if f_obj > f_crt then
4.         if f_cur = f_high then
5.             C_high ← C_high + 1
6.             if jobpool.size ≤ 1 and C_high > C_thresh then
7.                 f_cur ← f_low
8.         if f_cur = f_low then
9.             C_low ← C_low + 1
10.            if C_low > C_high × (1- α)/ α then
11.                f_cur ← f_high
12.                C_low←0, C_high←0
13.    if at reschedule point then
14.        update f_obj based on Inequality (3)
15.        find adjacent frequencies that f_low < f_opt < f_high
16.        fetch α_high(f_obj) and C_thresh(f_obj) from profile
```

To show the advantage of our dual-speed method with *inter-task* switching, we compare it to three alternatives: (i) *single-speed method* which just finds a higher than optimal frequency directly supported by the processor and does not switch frequency at all; (ii) *intra-task method* that aggressively toggles speed during executions for every task while considering switching overhead; and (iii) *ideal case* where intra-task method is applied, with switching overhead set to 0 to achieve theoretical best case efficiency. The comparison study calculates task miss rate and sets per core utilization to 100%. As can be seen in Fig. 7, the single-speed scheme shows the worst result as it keeps running at f_{high}. Intra-task method works better by switching between two DVFS levels. However, its miss rate is still significantly higher than the ideal case due to excessive DVFS switching overhead. By presetting switching threshold and monitoring available workloads in the job pool, our inter-task switching scheme finds appropriate switching points and results in a miss rate close to the ideal case.

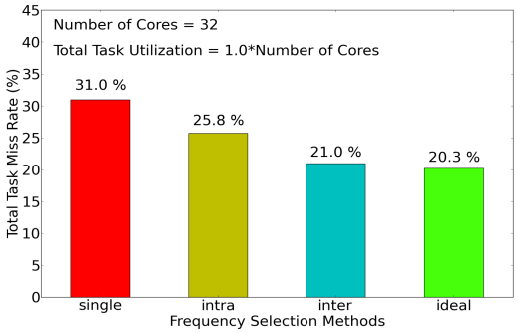

Figure 7: Performance comparison of frequency selection schemes

5. Experimental Studies

5.1. Overall miss rate reduction

We first study overall system miss rate reduction compared to previous works. We developed a C++ based simulator to evaluate the effectiveness of our proposed thermal-aware semi-dynamic power management algorithm (TA-SDA). We compared our scheme to the state of the art Utilization-Based Algorithm (UTB) [11], which we modeled in our environment. In addition, we also extended the energy harvesting-aware HA-DVFS [10] technique for multicore system with balanced tasks partitioning across multiple cores, to enable another comparison point. All tests are based on the processor power model shown in Table I. The energy harvesting power and ambient temperature profile is obtained from historical weather data of Golden, Colorado, US provided by National Renewable Energy Laboratory (NREL) [20]. Our system only executes during daytime over a span of 750 minutes, from 6:00 AM to 6:30 PM, when solar energy is available. Both UTB and HA-DVFS techniques assume a perfect prediction of the energy budget over the short term, based on the premise that solar energy profiles do not change notably over small windows of time (e.g., for a granularity of 1 minute). On the other hand, we use a simple moving average prediction for our TA-SDA technique. Our task sets are randomly generated, with the workload varying based on a given value of utilization for the task set. When comparing techniques under a specified setup, we make sure that they are working on the identical task set, eliminating possible impact of task set randomness. Finally, we set schedule window size for TA-SDA to be 5 minutes compared to the average task execution time of a few hundred milliseconds, not too short to cause frequent changes in the execution policy and not too long to make it hard to estimate energy budget accurately. Finally, we assign identical penalties to all tasks so as to compare TA-SDA with previous works that are penalty-oblivious.

5.1.1. Experiment Results

In our tests, we compare overall miss rates between HA-DVFS [10], UTB [11] and TA-SDA for different number of cores ranging from 1 to 32. With increasing number of cores, we scale harvesting power, number of tasks, and total task utilization linearly so as to keep a consistent and reasonable per core workload and energy budget. We generated 50 random task sets to test every configuration.

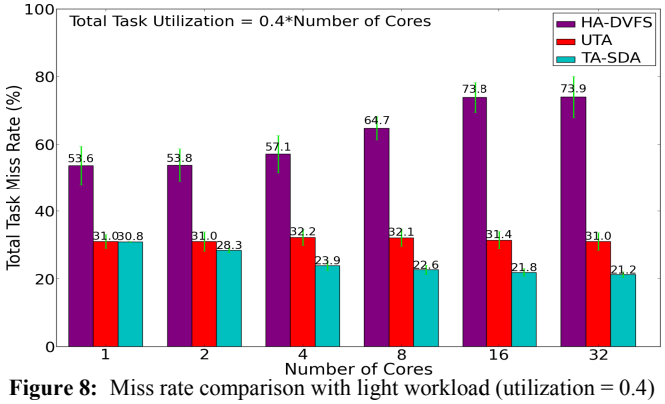

Figure 8: Miss rate comparison with light workload (utilization = 0.4)

First, we experiment on a workload with per core utilization set to 0.4, which has moderate energy requirements as system can execute at critical frequency for highest efficiency when energy is sufficient. The results are shown in Fig. 8. HA-DVFS can be seen to have a much higher miss rate as it does not make use of periodic task information and thus underestimates future workload. For the other two techniques, the advantage of TA-SDA over UTB is small for the single-core setup because task utilization is not very high. However, with increasing number of cores, TA-SDA's advantage expands considerably even though per-core workload and energy budget stays the same. One reason is that UTB uses an isolated task dropping scheme on each core, which is based on energy availability prediction for one upcoming task, ignoring workload on other cores that compete for the same energy source. In contrast, TA-SDA performs task rejection before assigning accepted tasks to different cores; thus the workload is adapted to a system-wide energy budget that has been predicted. Furthermore, TA-SDA actually benefits from increasing number of cores as it exploits the flexibility to shut down some cores for higher efficiency.

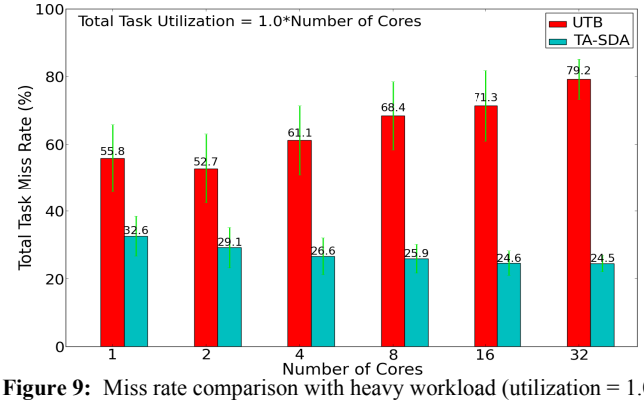

Figure 9: Miss rate comparison with heavy workload (utilization = 1.0)

We also compared UTB and TA-SDA under a much heavier workload, with per core utilization set to 1.0, results for which are shown in Fig. 9. Not surprisingly, the heavier workload expands the performance gap between UTB and TA-SDA. This trend is backed by observations for other utilizations (results for other utilizations are not shown for brevity). The reason for the performance gap is that the higher workload implies more stringent timing and energy constraints, under which TA-SDA's balanced run-time adjustment scheme becomes more effective, as discussed in Section 3. As a result, the most significant difference between these two techniques can be seen for the 32-core platform scenario, where TA-SDA has about 70% miss rate reduction compared to UTB. Additionally, the results of TA-SDA have less variation on multiple task sets compared to UTB, which indicates that task set randomness has less impact on TA-SDA as its dynamic adjustment is based on the scope of the entire task set, and not just individual tasks.

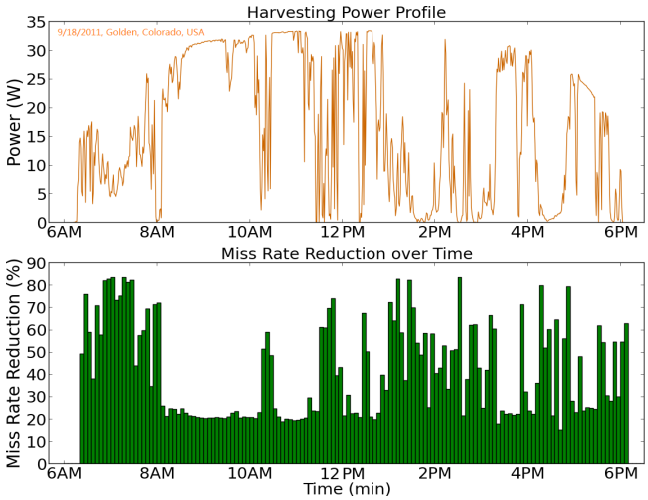

Figure 10: Miss rate difference over time

To support this conclusion from another perspective, we analyzed the advantage of TA-SDA over UTB as harvested power fluctuates over time (Fig. 10). As expected, higher miss rate reductions occurred for TA-SDA over UTB when harvesting power is low or changes dramatically.

5.2. Impact of thermal management

Here we study the impact of using different thermal management schemes in an energy harvesting environment. Our experimental setup for this study is similar to the one discussed in Section 5.1. To simulate a scenario with high overheating risk (as discussed in Section 3.2), we evaluate our approach for a very heavy workload with per core utilization set to 1.0. Our environmental profile considers high solar intensity and ambient temperatures from 9AM to 3PM. For thermal analysis, we integrated our simulator with HotSpot, an open-source thermal modeling and analysis tool [17]. We changed package parameters of the Hotspot tool to model a lower power processor, with no air cooling system.

5.2.1. Experiment Results

In our tests, we compare the performance of three schemes:

(i) Non-Throttling: We remove thermal awareness from our TA-SDA scheme. This is representative of current state of the art scheduling techniques for energy harvesting systems that ignore thermal issues;

(ii) Throttling: We again consider our TA-SDA scheme without thermal-awareness, but here we measure temperature and reactively enable throttling when temperature exceeds the throttling threshold;

(iii) Proactive: This is our TA-SDA approach from Section 4 that integrates proactive core slowdown to reduce instances of throttling.

We considered a test platform with 16 cores connected together in a mesh topology. Our test results for the three schemes can be seen in Fig. 11. We can see that the *Non-Throttling* scheme suffers from high peak temperatures for extended periods of time. Such high temperatures will significantly impact the system stability and reliability. Considering the fact that existing schemes proposed in literature for scheduling tasks on energy harvesting systems do not consider thermal issues, the result indicates that these schemes may not be practically viable for use in real energy-harvesting systems.

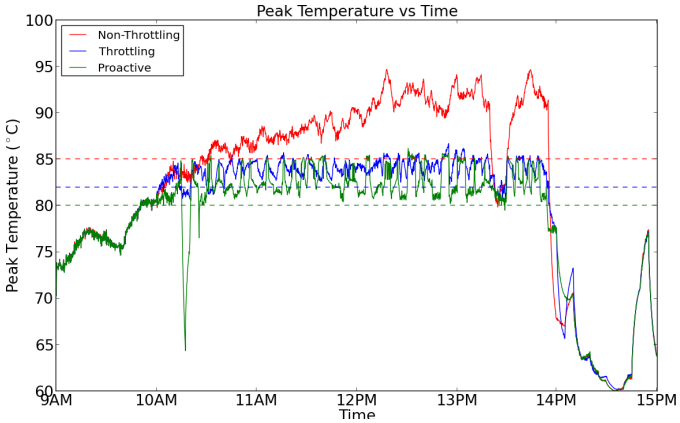

Figure 11: Peak temperature over time

Table II: Comparison between Throttling and Proactive schemes

Thermal management scheme	average peak temperature	number of throttlings	overall task miss rate
Throttling	79.60°C	94	35.92%
Proactive	78.53°C	74	35.33%

In contrast, the reactive *Throttling* scheme is able to control temperature to stay below throttling threshold for most of the time. In Fig. 11 the red dashed line indicates the throttling threshold at 85°C and the green dashed line shows the threshold at 80°C at which throttling terminates. Note that peak temperature seldom drops to 80°C in simulation. This is due to the fact that other un-throttled cores take over the role of thermal hotspots in the system from the throttled cores. Our TA-SDA *Proactive* scheme proactively performs core slowdown when temperature exceeds a proactive reaction threshold (set to 82°C, and shown with the blue dashed line shown in Fig. 11). This scheme helps to increase energy efficiency by avoiding unbalanced frequencies created by thermal throttling. Table II shows how our proactive approach not only reduces peak temperature, but also reduces the number of throttling instances, which allows more efficient scheduling management, culminating in an overall improved task miss rate. The results highlight the benefits of proactive thermal management, as practiced in our TA-SDA approach.

6. Conclusion

In this paper, we proposed a new thermal-aware semi-dynamic algorithm (TA-SDA) for real-time multiprocessor embedded systems with energy harvesting. Compared to the best known previous work, our scheme reduces miss rate up to 32 % for low intensity workloads and up to 70% for high intensity workloads. We also demonstrate, for the first time, the necessity of considering thermal issues in energy harvesting systems. Our TA-SDA approach not only outperforms prior work but also proactively solves the problem of thermal management in such systems to provide better performance and stability than ever before. Our ongoing work is looking at new prediction techniques that can further enhance the effectiveness of our approach in environments with highly varying solar profiles.

References

[1] Nvdia Tegra 3 processor, http://www.nvidia.com/object/tegra-3-processor.html.

[2] "The benefits of multiple CPU cores in mobile devices", http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf.

[3] V. Raghunathan et al., "Design considerations for solar energy harvesting wireless embedded systems," in IPSN, 2005, pp. 457-462.

[4] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," IPSN, 2005, pp. 463-468.

[5] M. Veerachary, T. Senjyu, and K. Uezato, "Maximum power point tracking of coupled inductor interleaved boost converter supplied PV system," IEE Proc. EPA, 2004, vol. 150, no. 1, pp. 71-80.

[6] M. Veerachary, T. Senjyu, and K. Uezato, "Voltage-based maximum power point tracking control of PV system," IEEE Trans. Aero. and Electron. Syst., vol. 38, no. 1, pp. 262-270, Jan. 2002.

[7] N. Femia et al., "Distributed maximum power point tracking of photovoltaic arrays: novel approach and system analysis," IEEE Trans. Indust. Electron., vol. 55, no. 7, pp. 2610-2621, Jul. 2008.

[8] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy-harvesting sensor nodes," in DIPES, 2006, pp. 125-134.

[9] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," in DATE, 2008, pp. 236-241.

[10] S. Liu, J. Lu, Q. Wu, and Q. Qiu, "Harvesting-aware power management for real-time systems with renewable energy," IEEE Trans. VLSI Syst., vol. 20, no. 8, pp. 1473-1486, Aug. 2012.

[11] J. Lu, Q. Qiu, "Scheduling and mapping of periodic tasks on multi-core embedded systems with energy harvesting," in IGCC, 2011, pp. 1-6.

[12] A. Mirhoseini, F. Koushanfar, "HypoEnergy: hybrid supercapacitor-battery power-supply optimization for energy efficiency," in DATE, 2011, pp. 1-4.

[13] Intel XScale, http://download.intel.com/design/intelxscale/27347302.pdf.

[14] J. Park, D. Shin, N. Chang, M. Pedram, "Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors," in ISLPED, 2010, pp. 419-424.

[15] J. Chen, T. Kuo, C. Yang, and K. King, "Energy-Efficient Real-Time Task Scheduling with Task Rejection", in DATE, 2007, pp. 1-6.

[16] D. Rajan, R. Zuck, and C. Poellabauer, "Workload-aware dual-speed dynamic voltage scaling," in RTCSA, 2006.

[17] W. Huang et al., "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," IEEE Trans. VLSI Syst., vol. 14, no. 5, pp. 501-513, May. 2006.

[18] I. Yeo, C.C. Liu, E.J. Kim, "Predictive dynamic thermal management for multicore systems," in DAC, 2008.

[19] A.K. Coskun et al, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs," IEEE Trans. VLSI Syst., vol. 16, no. 9, pp. 1127-1140, Sep. 2008.

[20] Measurement and Instrumentation Data Center, NREL, http://www.nrel.gov/midc.