Tan and Yin *EURASIP Journal on Wireless Communications and Networking* (2016) 2016:114
DOI 10.1186/s13638-016-0602-8

EURASIP Journal on Wireless Communications and Networking

RESEARCH ARTICLE Open Access

A dynamic scheduling algorithm for energy harvesting embedded systems

Yonghong Tan[1*] and Xiangdong Yin[2]

Abstract

Energy harvesting embedded systems are embedded systems that integrate with energy harvesting modules. In this kind of systems, service tasks and energy harvesting tasks must be scheduled efficiently to keep the whole system working properly as long as possible. In this paper, we model an energy harvesting embedded system with an energy model, a task model, and a resource model and propose a dynamic task scheduling algorithm. The proposed algorithm is based on dynamic voltage and frequency scaling technique and dynamically concentrates all disperse free time together to harvest energy. We validate the efficiency and effectiveness of the proposed algorithm under both energy-constraint and non-energy-constraint situations with the Yartiss simulation tool.

Keywords: Energy harvesting embedded systems, Dynamic scheduling algorithm, Energy model

1 Introduction

In embedding systems, the energy is usually provided with batteries, and the requirements of real time and many extensive smart functions make embedding systems consume more and more energy [1]. If we run the embedding systems arbitrarily, the energy would run out very quickly; this would shorten the working time of smart devices. The techniques of dynamic voltage and frequency scaling (DVFS) [2] and dynamic power management (DPM) [3] could decrease the power consumption of embedding systems whereas satisfying the time constraints. However, once deployed, the embedded application will run a long time, and the energy of the battery will run out finally. Besides replacement of the battery, the embedded systems can harvest energy from the external environment [4], such as sunshine [5], wind [6], and vibration [7]. By applying these energy harvesting techniques [8, 9], the working time of embedded systems can be increased. While harvesting energy from the environment, both the harvest and storage of energy need processing time, and this needs to reschedule the processor and thus interrupts the working tasks. So, an energy harvesting embedded system is an embedded system with different kinds of energy harvesting modules inside.

In energy harvesting embedded systems [10], one needs to schedule tasks between harvesting tasks with working tasks, and the aim is to keep the energy of embedded systems in a reasonable level, while providing normal working services at the same time. During the scheduling of the tasks in energy harvesting embedded systems, both the energy output and storage units are the physical environment and the change of status is a continuous physical process [11]. The physical process provides much information for the computing environment, and the decision-making process of the computing environment affects the physical environment too. According to interaction and fusion [12] of the computing and physical environment, one can allocate the resources efficiently and thus optimize the performance of the whole system.

In this paper, we study the problem of task scheduling in energy harvesting embedded systems. In an energy harvesting embedded system, the system needs to harvest energy from the external environment during free or idle time. We model an energy harvesting embedded system with an energy model, a task model, and a resource model and propose a dynamic task scheduling algorithm. Based on dynamic voltage and frequency scaling techniques, the proposed algorithm concentrates

* Correspondence: 496081928@qq.com
[1]Experimental Training Center, Hunan University of Science and Engineering, YongZhou, Hunan Province, China
Full list of author information is available at the end of the article

all disperse free time together to harvest energy by dynamically scheduling harvesting tasks and service tasks.

The rest of the paper is organized as follows. In Section 1.2, we review related works about scheduling algorithms in embedded systems. In Section 1.3, we propose a model for energy harvesting embedded systems. In Section 1.4, we propose a dynamic task scheduling algorithm. Experiments and conclusions are given in Sections 1.5 and 2, respectively.

2 Related works

In this section, we review related works about scheduling algorithms in embedded systems, especially focus on energy saving and energy harvesting scheduling algorithms.

2.1 Energy saving scheduling algorithms

Energy saving task scheduling is a key research in embedded systems and sensor networks. The techniques of energy saving scheduling can be classified into traditional scheduling method and utility-based scheduling method. Traditional energy-saving scheduling methods [13, 14] can be applied to the simple task-arrival mode, such as period tasks, but it cannot assure the arrival tasks to be scheduled in real time.

In general, every completion of a task would bring some utility, and the utility depends on the running time of the task. The longer the running time, the smaller the utility is. Jensen et al. [15] proposed a time/utility function to describe the relationship between the running time and the utility of a task, and their aim is to maximize the total utility by finishing all tasks as quickly as possible. References [16–18] studied how to get the maximal utility with limited energy. In addition, in order to satisfy the utility acquirement and the energy budget, Wu et al. [19] proposed a unimodal arbitrary arrival with energy bounds algorithm (EBUA), and the EBUA solved the problem of task scheduling based on unimodal arbitrary arrival model.

2.2 Energy harvesting scheduling algorithms

The task scheduling problem in energy harvesting embedded systems was first proposed by Allavena and Mossé in [20], and then some algorithms for solving this problem were proposed. The lazy scheduling algorithm (LSA) proposed by Moser et al. [21] adjusts the worst-case execution time by adjusting frequency of the CPU according to tasks' energy consumption. The LSA is based on a strong assumption that the worst-case execution time of a task is related with its energy consumption directly, and this assumption is impractical [22]. Chandarli et al. [23] proposed an ALAP scheduling algorithm, and it was a fixed-priority scheduling policy that delayed the executions of jobs as long as possible. Abdeddaïm et al. [24] proposed an ASAP algorithm, and this algorithm scheduled tasks as soon as possible when there was available energy in the battery and suspended execution to replenish the energy needed to execute one time unit. Under constraints of energy and time, ASAP is proved to be optimal.

In addition, in order to get better system performance and less energy consumption, some scheduling algorithms [25, 26] take dynamic voltage frequency scaling into consideration. The EA-DVFS algorithm [25], proposed by Liu et al., adjusts the CPU frequency by the remaining energy of the system. If there is not enough energy to run the task, then it decreases the CPU frequency; otherwise, it runs the task with maximal CPU frequency. The HA-DVFS algorithm [26] optimizes the system performance and improves the energy utilization further based on the EA-DVFS algorithm. However, in some real-time embedded systems with high reliability, DVFS and related algorithms can extend the running time of tasks, affect the run-time attribute of the system, and thus decrease its reliability.

3 Energy harvesting embedded system model

In this section, we model the embedded system with an energy model, a task model, and a resource model.

3.1 The energy model

In this paper, we applied the system-level energy model proposed by Martin [27] in his doctoral thesis. We let the CPU value be 1, if it runs with maximal speed. So, the amount of computation is also the number of clock period under maximal speed. We assume that the embedded system supports DVFS and the CPU has d discrete speeds (or frequencies) $f_i (1 \le i \le d)$. The CPU speed f_i means that the CPU runs f_i clock periods per second. When the CPU runs with speed f, the energy consumption can be described as

$$\mathrm{PC} = S_3 f^3 + S_2 f^2 + S_1 f + S_0, \tag{1}$$

where S_3, S_2, S_1, and S_0 are constants.

In the above model, the system-level energy consumption includes dynamic energy consumption, static energy consumption, and the energy consumed by other components. So, when the system runs with frequency f, the energy consumption of every clock period is

$$E(f) = S_3 f^2 + S_2 f + S_1 + \frac{S_0}{f} \tag{2}$$

3.2 The task model

We assume that the embedded system is a preemptive real-time system and the task set is $\Gamma = \{\tau_1, \cdots, \tau_n\}$. Every arrival task is an instance of its corresponding task, and the jth instance of task τ_i is denoted as $\tau_{i,j}$. In a unimodal arrival model, each task is related with one tuple $<a_i, P_i>$, and the tuple means that the maximal number of arrival instances is a_i in any sliding window P_i. The periodic real-time task mode is a special case of the unimodal arrival model, where the value a_i equals to 1.

We use $U_i(\cdot)$ to denote the time/utility function of the task τ_i, and thus, $U_i(\cdot)$ is the time/utility function of any instance of τ_i. If $\tau_{i,j}$ finishes with time t, then the utility is $U_i(t)$. In addition, we define the beginning time of U_i with the arrival time of τ_i and define the ending time of U_i with the sum of its beginning time and the length of the sliding window P_i We let the number of clock periods be c_i and the relative deadline be D_i, and both computations of c_i and D_i can be found in [19].

3.3 The resource model

We define the sharing resource set except for the CPU as $\mathrm{SR} = \{\mathrm{SR}_1, \mathrm{SR}_2, \cdots \mathrm{SR}_r$, where each sharing resource SR_i can be shared among all tasks but can only be accessed by one task at one time. Once some task has been authorized to access sharing resource, then this task executes a critical section. After executing of the critical section, the task releases the sharing resource. We denote the *j*th critical section of task τ_i as $z_{i,j}$, then the amount of computation is $c_{cs}(z_{i,j})$ and the accessed resource is denoted as $s(z_{i,j}) \in \mathrm{SR}$. In addition, let $N_{cs}(\tau_i)$ be the number of critical sections for τ_i, and the total amount of computation for τ_i under all non-critical sections be $c_{ns}(\tau_i)$, then the amount of computation for task τ_i is

$$c_i = c_{ns}(\tau_i) + \sum_{j=1}^{N_{cs}(\tau_i)} c_{cs}(z_{i,j}). \tag{3}$$

The critical sections of a task can be nested, that is for critical sections $z_{i,j}$ and $z_{i,k}$ of task τ_i, we can have $z_{i,j} \subset z_{i,k}$, $z_{i,k} \subset z_{i,j}$, or $z_{i,j} \cap z_{i,k} =$ null.

4 Dynamic task scheduling algorithm

In this section, we propose a dynamic resource scheduling algorithm. Here, "dynamic" means that the energy harvesting module is activated dynamically to maximize the harvested energy, and at the same time, the normal services cannot be disturbed. For convenience, we denote $\tau_{i,j}$ as τ_i in the following sections. The dynamic scheduling algorithm preserves the computing ability for the following tasks, and for each τ_i, constraints the number of instances in any sliding window to be *capacity*(τ_i). In addition, we keep the running speed in the critical section to be equal to the static frequency, and thus, the dynamic algorithm can acquire the speed in the non-critical section according to the current running status. In order to describe the proposed algorithm, we define some parameters and functions first:

- $c_{ns}^{r}(\tau_i, t)$ is the remaining computing ability in the non-critical section of task τ_i at time *t*.
- $c_{cs}^{r}(z_{i,j}, t)$ is the remaining computing ability in critical section $z_{i,j}$ at time *t*.
- $EET(\tau_i, t)$ is the required estimating time for the remaining part of task τ_i at time *t*, which can be computed with the remaining computing ability, the running speed in the non-critical section, and the static running speed in the critical section.
- $EEC(\tau_i, t)$ is the estimation of the required energy of task τ_i at time *t*, which can be computed with the remaining computing ability, the running speed in the non-critical section, the static running speed in the critical section, and the energy consumption defined in Section 1.2.1.
- $UER(\tau_i, t)$ is the estimated ratio of utility to the energy of task τ_i starting from time *t*, while it is not blocked, that is $UER(\tau_i, t) = U_i(t + EET(\tau_i, t))/EEC(\tau_i, t)$.
- UER' (τ_i, t) is the maximal speed of task τ_i starting from time *t*, while it is not blocked.
- *queue* is the waiting queue of arrival tasks.
- *recordtime*(τ_i) is the starting time of the sliding window of task τ_i, which initializes to be 1.
- $RC(\tau_i, t)$ returns the number of instances, which can be accepted in the future, of task τ_i at time *t*, and it equals to *capacity*(τ_i) minus the accepted number of instances of task τ_i in the current sliding window.
- *checkfeasible*(τ_i, t) checks whether or not the current sliding window can accept new instances of task τ_i at time *t*, and if *queue* = null, $t - recordtime(\tau_i) \geq D_i$ or $RC(\tau_i, t) > 0$, then this is the idle rate of task τ_i at time *t*, which can be computed by the following equation:

$$1-\left(\sum_{r=1}^{i}\left(\frac{c_{ns}(\tau_r)}{\eta_{ns}(\tau_r)\cdot D_r} + \sum_{k=1}^{N_{cs}(\tau_r)} \frac{c_{cs}(z_{r,k})}{\eta_{cs}(z_{r,k})\cdot D_r}\right) + \frac{B_i}{D_i}\right).$$

1. *runningstate*(τ_i, t) reflects the running status of task τ_i at time *t*, which returns true if τ_i is selected to run as a candidate task before *t*, and false, otherwise.
2. $f(\tau_i, t)$ is the current running speed of task τ_i at time *t*, which can be computed by Theorem 1.
3. *increaseslu*(τ_i, sl, t) increases the idle rate of task τ_i at time *t*, that is $SLU(\tau_i, t) = SLU(\tau_i, t) + sl/D_i$.
4. *decreaseslu*(τ_i, sl, t) decreases the idle rate of task τ_i at time *t*, that is $SLU(\tau_i, t) = SLU(\tau_i, t) + sl/D_i$.
5. *selectcandidate*(t) selects a candidate task to run.
6. *resource*(τ_i, t) is the resource accessed by task τ_i at time *t*, which is effective when τ_i is in the critical sections, and returns null otherwise.

Theorem 1. *If $\tau_i$is selected to run as a candidate task at time t, then resource(τ_i, t) returns null, and the running speed of $\tau_i$in the previous non-critical section is f(τ_i, t). In order to satisfy the schedulability of task τ_i, the current running speed is*

$$f(\tau_i,t) = c^r_{ns}(\tau_i,t)/\frac{c^r_{ns}(\tau_i,t)}{f(\tau_i,t^{'})} + \min_{\forall j\geq i}\{SLU(\tau_j,t)\times D_j\}). \tag{4}$$

Proof. As can be seen from the assumption, the task τ_i runs in the critical section, so we have $c^r_{ns}(\tau_i,t) = c^r_{ns}(\tau_i,t^{'})$. At time t, the available free time for the task τ_i is $\min_{\forall j<i}\{SLU(\tau_i,t)\times D_j\}$. At the same time, we have

$$\frac{c^r_{ns}(\tau_i,t^{'})}{f(\tau_i,t^{'})} + \min\nolimits_{\forall j\leq i}\{SLU(\tau_j,t)\times D_j\} = \frac{c^r_{ns}(\tau_i,t)}{f(\tau_i,t^{'})}.$$

So, we can get the following result

$$f(\tau_i,t) = c^r_{ns}(\tau_i,t)/\left(\frac{c^r_{ns}(\tau_i,t)}{f(\tau_i,t^{'})} + \min\nolimits_{\forall j\geq i}\{SLU(\tau_j,t)\times D_j\}.\right.$$

Algorithm 1: Dynamic DVFS algorithm

```
While switch triggering event at time $t$ do
  Case $task\_completion(\tau_i)$
    Remove this instance from $queue$ and
    $adjustRC(\tau_i,t)$;
    If $queue \neq null$ then
      $\tau_i = selectcandidate(t)$;
      Calculate $f(\tau_i,t)$;//see algorithm 2
  Case $task\_arrive(\tau_i)$
    If $checkfeasible(\tau_i,t) = true$ then
      Check if a new sliding window opens;
      If $t - recordtime(\tau_i) \geq D_i$ or $queue = null$
      then
        $recordtime(\tau_i) = t$;
        let$initializeSLU(\tau_i,T)$ and $setRC(\tau_i,t)$
        Be $capacity(\tau_i)$ minus the number of
        Instances of $\tau_i$ in $queue$ at time $t$;
        Initialize the execution frequencies of critical Sections and non critical sections to their static ones
respectively;
        Append the new arrived instance to $queue$ and $adjustRC(\tau_i,t)$;
        If all new task arrivals and checked then
        $\tau_i = selectcandidate(t)$;
        Calculate $f(\tau_i,t)$;//see algorithm 2
```

Algorithm 1 describes the proposed dynamic DVFS algorithm. Line 3 rectifies the accepted number of instances in the current sliding window; line 5 selects a candidate task to run; and line 6 adjusts the running speed of the candidate task (details are in Algorithm 2). If a new sliding window is opened (line 10), we use line 11 to initialize the parameters and preserve the computing ability for the following tasks. When a new task arrives, line 14 restricts the number of the following tasks by adjusting $RC(\tau_i, t)$. From lines 11 to 13, we see that every sliding window of task τ_i can have $capacity(\tau_i)$ instances at most. Lines 15 to 17 reselect a candidate task and compute its running speed.

Algorithm 2: Compute $f(\tau_i,t)$ of task τ_i at time t

```
If $runningstate(\tau_i,t) = false$ then
  $increaseslu(\tau_i,sl,t)$, where $sl$ is the slack time that
  $B_i / capacity(\tau_i)$ minus the actual blocking time for task $\tau_i$;
If $resource(\tau_i,t) = null$ then
  Get $f(\tau_i,t)$ by theorem1;
  $f(\tau_i,t) = selectfrequency(\max\{UER^{'}(\tau_i,t), f(\tau_i,t)\})$;
  $m = c^{\tau}_{ns}(\tau_i,t)/f(\tau_i,t) - c^{\tau}_{ns}(\tau_i,t)/f(\tau_i,t^{'})$
  If $m > 0$ then
    $\forall_j \geq i, increaseslu(\tau_j,m,t)$ ;
  Else
    $\forall_j \geq i, increaseslu(\tau_j,-m,t)$
Else
  $f(\tau_i,t) = \eta_{cs}(z_{i,k})$,where $z_{i,k}$ is selected to run;
```

The aim of Algorithm 2 is to decrease the energy consumption by releasing/recycling free time and maximize the total utility. When a candidate task starts to run, lines 1 and 2 recycle the free time between the maximal blocking time with the practical blocking time. When the resource occupied by task τ_i at time t is null (line 3), line 4 computes the running speed of the candidate task in non-critical sections, line 5 selects the running speed for task τ_i at time t, and line 6 is equal to $\forall j \geq i\{SLU(\tau_j, t) \times D_j$. When we have the running speed of the candidate task, the available time of the task would be deducted (line 8) or released (line 10). Finally, we use line 12 to assure that the running speed of the candidate task in the critical section is equal to its static running speed.

5 Experiments

5.1 Experimental setup

In this experiments, we compare our proposed algorithm with the ASAP and ALAP algorithms and use the

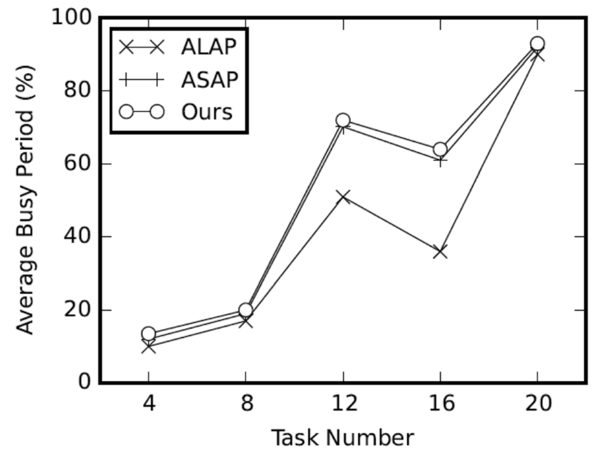

Fig. 1 Average busy period under non-energy constraints

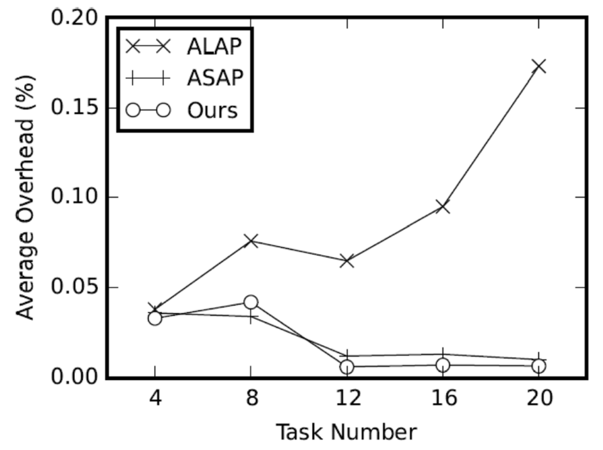

Fig. 3 Average overhead under non-energy constraints

simulation tool Yartiss [28] to implement these algorithms. Yartiss provides a simulation framework, and this framework can execute massive tasks of different algorithms with different parameters. We let the output of energy harvesting unit be equal to the energy supply rate e_{bat} of the system, and every time unit provides several energy units. The energy consumption of a task is linear, and every task consumes E_i/C_i energy units per time unit. In order to evaluate the performance of algorithms, we compare our proposed algorithm with the ASAP and ALAP algorithms under both energy-constraint and non-energy constraint situations. Under energy constraints, we evaluate the performance of algorithms by increasing the number of tasks; and under non-energy constraints, we design six application scenes with different amount of batteries and let $P_h = e_{\text{bat}} = 15$, $E_{\min} = 0$, and the running time $Duration = 2550$.

In simulation experiments, we use the following six metrics:

- 1) *Average busy period* is the average period of the CPU under simulation. The longer the average busy period, the higher the utility of the CPU is.
- 2) *Average idle period* is the average period of the CPU while it is free. Under energy constraints, it includes free time and relaxed time. The longer the average idle period, the higher the average energy level is, and thus, the lower energy constraints the system has.
- 3) *Average overhead* is the average time required to execute a task under simulation. The bigger the average overhead, the more likely the task will be missed before the stopping time.
- 4) *Average preemption* is the ratio of the preempted tasks to total tasks. The bigger the average preemption, the more context switches it has, and thus, the heavier the overhead is. The heavy overhead will decrease the performance of the

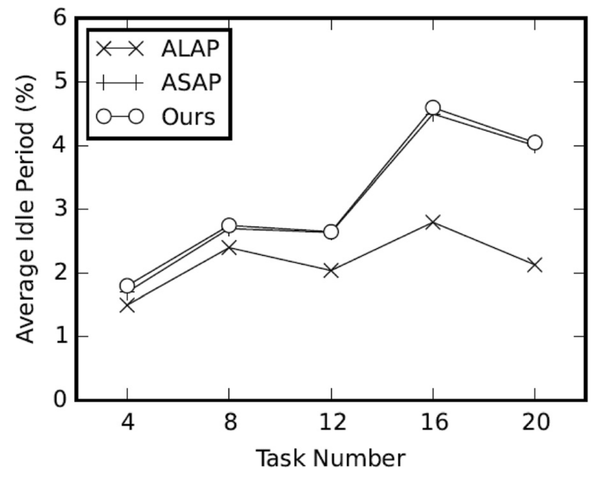

Fig. 2 Average idle period under non-energy constraints

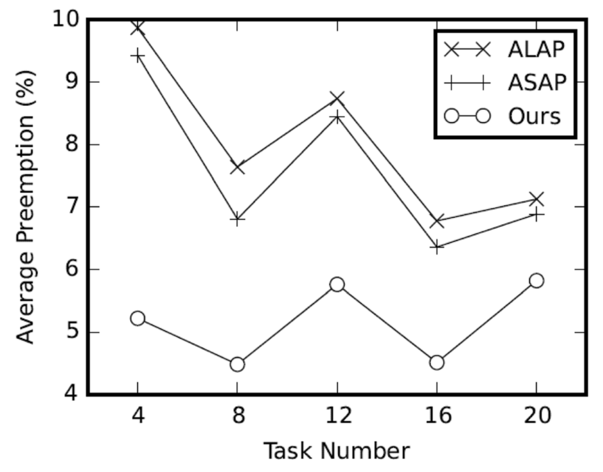

Fig. 4 Average preemption under non-energy constraints

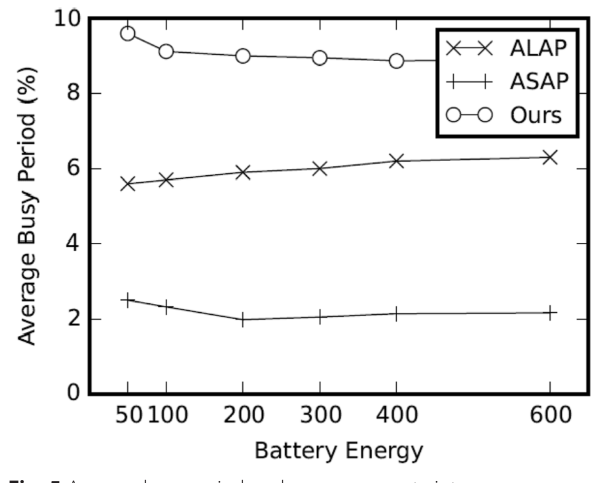

Fig. 5 Average busy period under energy constraints

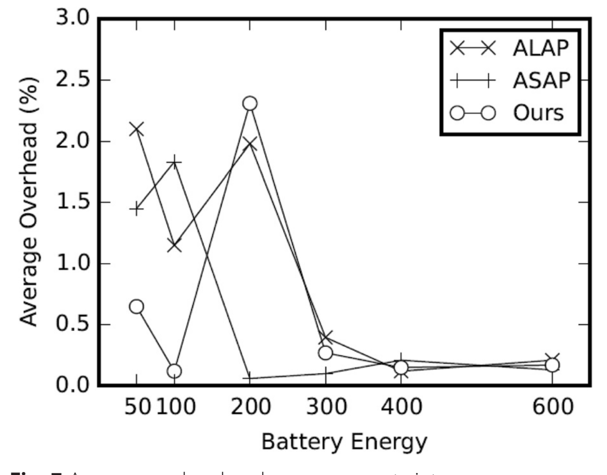

Fig. 7 Average overhead under energy constraints

whole system and make the scheduling algorithm unpracticable.

- 5) *Average energy level* is the average energy percent of batteries. The higher the average energy level, the lower energy constraints the system has.
- 6) *Average battery switch mode* is the ratio of battery switch modes to total tasks. The more the average battery switch mode, the lower the energy utility is. Low energy utility would make the system work un-properly.

5.2 Experimental results

We compare our proposed algorithm with the ASAP and ALAP algorithms under both the non-energy-constraint and the energy-constraint situations, and the results are in Figs. 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

Under non-energy constraints, the comparisons of average busy period, average idle period, average overhead, and average preemption are in Figs. 1, 2, 3, and 4, respectively. The ALAP algorithm postpones the execution of tasks as long as possible, and it generates massive free time, so both the average busy period and idle period are minimum. However, with the increase of the number of tasks, the average overhead of the ALAP algorithm increases obviously. The ASAP algorithm executes tasks as soon as possible, and it is equal to the scheduling of fixed-priority preemption. Our algorithm is based on fixed-priority preemption, and increases the threshold of preemption, so it has similar average busy period, average idle period, and average overhead to the ASAP algorithm. In addition, with the introduction of preemption threshold, our algorithm avoids the tasks with high priorities preempted, while keeping these tasks finished in time. So, our algorithm has the least number of preemptions.

Under energy constraints, the comparisons of average busy period, average idle period, average overhead,

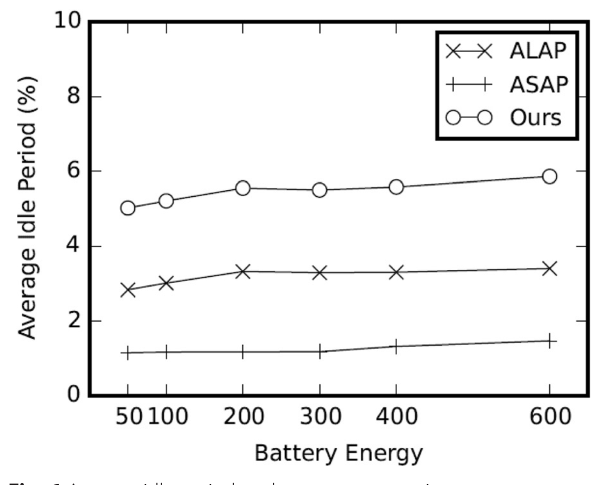

Fig. 6 Average idle period under energy constraints

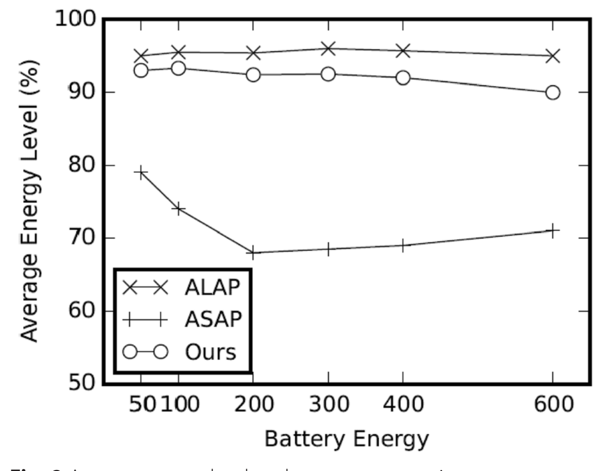

Fig. 8 Average energy level under energy constraints

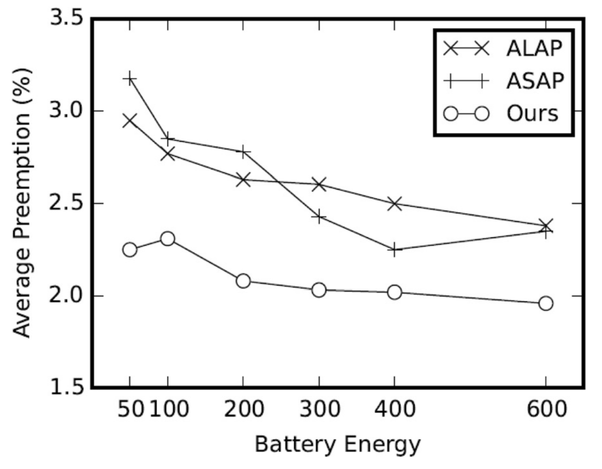

Fig. 9 Average preemption under energy constraints

average energy level, average preemption, and average battery switch mode are in Figs. 5, 6, 7, 8, 9, and 10, respectively. As we can see from these figures, our algorithm has the least preemption too. The reason is that the ASAP algorithm judges preemption after every execution time unit and our algorithm runs tasks concentratedly. When the energy is not enough, our algorithm concentrates all available relaxed time to harvest energy, which reduces the preemptions caused by lacking energy. Our algorithm reduces the battery switch mode, uses the relaxed time to harvest energy, and then maximizes the busy period and idle period. So, our algorithm provides higher energy level and decreases the energy constraints of the system. The overheads of all the above algorithms tend to be identical. Our algorithm keeps the battery being on charge or discharge mode all the time and has less battery switch mode than the ASAP algorithm.

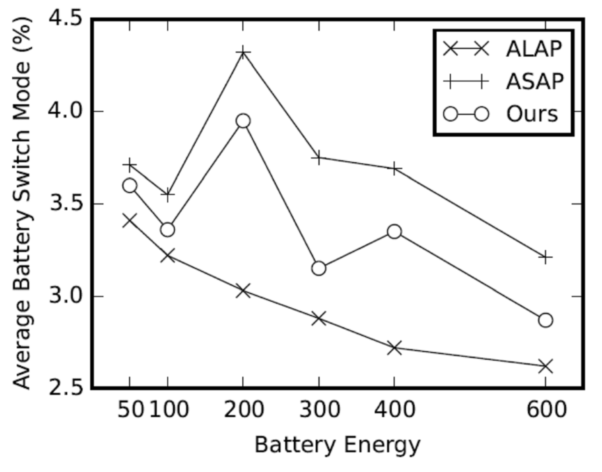

Fig. 10 Average battery switch mode under energy constraints

6 Conclusions

In energy harvesting embedded systems, the system needs to harvest energy from the external environment during free or idle time and schedule service tasks and energy harvesting tasks to keep the whole system working properly as long as possible. In this paper, we study the problem of task scheduling in energy harvesting embedded systems. We model an energy harvesting embedded system with an energy model, a task model, and a resource model and propose a dynamic task scheduling algorithm. Based on the dynamic voltage and frequency scaling techniques, the proposed algorithm concentrates all disperse free time together to harvest energy by dynamically scheduling harvesting tasks and service tasks. In the future, we will build a real energy harvesting embedded system, which implements the proposed algorithm. Currently, we only do some simulation experiments to validate the effectiveness of the proposed algorithm theoretically, and real improvement of lifetime on a real system will be our future work too.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The work was supported by the following funds: Hunan Provincial Natural Science Foundation of China (Grant No.2015JJ6043); Hunan University of Science and Engineering; Scientific Research Fund of Hunan Provincial Education Department(Grant No.12A054); and The Construct Program of the Key Discipline in Hunan University of Science and Engineering(Circuits and Systems).

Author details

[1]Experimental Training Center, Hunan University of Science and Engineering, YongZhou, Hunan Province, China. [2]School of Electronics and Information Engineering, Hunan University of Science and Engineering, YongZhou, Hunan Province, China.

Received: 26 September 2015 Accepted: 4 April 2016
Published online: 22 April 2016

References

1. TA Nguyen, M Aiello, Energy intelligent buildings based on user activity: a survey. Energy Build. **56**, 244–257 (2013)
2. EL Sueur, G Heiser, Dynamic voltage and frequency scaling: the laws of diminishing returns. International Conference on Power Aware Computing and Systems. USENIX Association, 1-8 (2010)
3. L Benini, A Bogliolo, G De Micheli, A survey of design techniques for system-level dynamic power management. IEEE Trans. Very Large Scale Integr. VLSI Syst. **8**(3), 299–316 (2000)
4. S Chalasani, JM Conrad, A survey of energy harvesting sources for embedded systems. Southeastcon. 442-447 (2008)
5. V Raghunathan, A Kansal, J Hsu, J Friedman, M Srivastava, Design considerations for solar energy harvesting wireless embedded systems. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, (IEEE Press, 2005), p. 64
6. S Li, J Yuan, H Lipson, Ambient wind energy harvesting using cross-flow fluttering. J. Appl. Phys. **109**(2), 26104 (2011)
7. SP Beeby, MJ Tudor, N White, Energy harvesting vibration sources for microsystems applications. Meas. Sci. Technol. **17**(12), 175 (2006)
8. A Kansal, J Hsu, S Zahedi, MB Srivastava, Power management in energy harvesting sensor networks. ACM Trans. Embed. Comput. Syst. **6**(4), 32 (2007)
9. RV Prasad, S Devasenapathy, VS Rao, J Vazifehdan, Reincarnation in the ambiance: devices and networks with energy harvesting. IEEE Commun. Surv. Tutorials **16**(1), 195–213 (2014)

10. V Raghunathan, PH Chou, Design and Power Management of Energy Harvesting Embedded Systems. International Symposium on Low Power Electronics and Design. 369-374 (2006)
11. T Tong, S Ulukus, W Chen, Optimal packet scheduling for delay minimization in an energy harvesting system. IEEE International Conference on Communications. IEEE, 4241-4246 (2015)
12. M Song, Y Zhang, M Peng, J Zhai, Low frequency wideband nano generators for energy harvesting from natural environment. Nano Energy **6**, 66–72 (2014). Tan and Yin Page 8 of 8
13. Y Zhang, XS Hu, DZ Chen, Task scheduling and voltage selection for energy minimization. Proceedings of the 39th Annual Design Automation Conference (ACM), 183–188 (2002)
14. G Quan, X Hu, Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. Design Automation Conference, 2001. Proceedings (IEEE), 828–833 (2001)
15. ED Jensen, CD Locke, H Tokuda, A time-driven scheduling model for real-time operating systems. RTSS **85**, 112–122 (1985)
16. L Huang, MJ Neely, Utility optimal scheduling in energy-harvesting networks. IEEE/ACM Trans. Networking **21**(4), 1117–1130 (2013)
17. A Jaleel, HH Najaf-Abadi, S Subramaniam, SC Steely, J Emer, Cruise: cache replacement and utility-aware scheduling. ACM Comp. Ar. **40**, 249–260 (2012). ACM
18. D Xue, R Murawski, E Ekici, Distributed utility-optimal scheduling with finite buffers. International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks. IEEE, 278-285 (2012)
19. H Wu, B Ravindran, ED Jensen, Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds. IEEE Trans. Comput. **56**(10), 1358–1371 (2007)
20. A Allavena, D Mossé, Scheduling of frame-based embedded systems with rechargeable batteries, in *Workshop on Power Management for Real-Time and Embedded Systems (in Conjunction with RTAS 2001)*, 2001
21. C Moser, D Brunelli, L Thiele, et al. Lazy Scheduling for Energy Harvesting Sensor Nodes[M]// From Model-Driven Design to Resource Management for Distributed Embedded Systems. (Springer US, 2006) 125-134
22. R Jayaseelan, T Mitra, X Li, Estimating the worst-case energy consumption of embedded software. Real-Time and Embedded Technology and Applications Symposium. Proceedings of the 12th IEEE (IEEE), 81–90 (2006)
23. Y Chandarli, Y Abdeddaim, D Masson, The Fixed Priority Scheduling Problem for Energy Harvesting Real-Time Systems. 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. IEEE Computer Society, 415-418 (2012)
24. Y Abdeddaïm, Y Chandarli, D Masson, Toward an optimal fixed-priority algorithm for energy-harvesting real-time systems, in *RTAS 2013 WiP*, 2013, pp. 45–48
25. S. Liu, Q. Qiu, Q. Wu, Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In: Design, Automation and Test in Europe, 2008. DATE'08 (IEEE, 2008), p. 236–241
26. S Liu, J Lu, Q Wu, Q Qiu, Harvesting-aware power management for real-time systems with renewable energy. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(8), 1473–1486 (2012)
27. TL Martin, *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999
28. Y Chandarli, F Fauberteau, D Masson, S Midonnet, M Qamhieh, Yartiss: a tool to visualize, test, compare and evaluate real-time scheduling algorithms, in *Proc. of the 3rd Int'l Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2012, pp. 21–26