IEICE TRANS. INF. & SYST., VOL.E94–D, NO.4 APRIL 2011

PAPER

Energy-Aware Task Scheduling for Real-Time Systems with Discrete Frequencies

Dejun QIAN[†a)], **Zhe ZHANG**[†], **Chen HU**[†], *Nonmembers*, *and* **Xincun JI**[†], *Student Member*

SUMMARY Power-aware scheduling of periodic tasks in real-time systems has been extensively studied to save energy while still meeting the performance requirement. Many previous studies use the probability information of tasks' execution cycles to assist the scheduling. However, most of these approaches adopt heuristic algorithms to cope with realistic CPU models with discrete frequencies and cannot achieve the globally optimal solution. Sometimes they even show worse results than non-stochastic DVS schemes. This paper presents an optimal DVS scheme for frame-based real-time systems under realistic power models in which the processor provides only a limited number of speeds and no assumption is made on power/frequency relation. A suboptimal DVS scheme is also presented in this paper to work out a solution near enough to the optimal one with only polynomial time expense. Experiment results show that the proposed algorithm can save at most 40% more energy compared with previous ones.
key words: *real-time system, low power design, dynamic voltage scaling, stochastic speed scheduling*

1. Introduction

Energy consumption minimization has become an important design issue for modern computing systems. As the peak performance is not needed all the time, dynamic voltage scaling (DVS) is introduced to slow down the processor to a just enough speed for a required performance and leads to energy savings.

For systems having real-time requirement, many DVS schemes have been designed to minimize energy consumption without making any task miss its deadline. Kim et al. [1] gives a method to classify and evaluate these schemes. Due to the real-time restriction, all the schemes are designed based on the assumption that each task executes worst-case execution cycles (WCEC) when there is no idea about the exact amount. However, a task might complete earlier than its worst-case estimation with some unused time (slack) left. These slacks could be reclaimed and allocated to latter tasks. Schemes in [2]–[5] focus on these slack reclamation techniques to reduce energy consumption. Shin et al. [6] developed an intra-task scheme which takes advantage of CFG obtained at compilation phase and recalculates WCEC through the execution path to reduce the energy further once branches advent. The resulting distribution information of the execution cycles obtained by profiling the tasks could also be used to help reducing energy consumption. Gruian [7], [8] or Lorch and Smith [9], [10] both derive stochastic intra-task schemes to reduce the expected energy consumption by executing a task at a lower frequency at the beginning and at higher frequencies for the rest. Yuan and Nahrstedt [11] use the accelerating strategy for soft real-time tasks. Lu et al. [12] applied Lagrange Multiplier Method to cope with systems with continuous probability density functions.

This paper focuses on frame-based hard real-time systems with variable and unpredictable workloads. Frame-based real-time systems are special cases of periodic real-time systems, where all tasks release at the same time, share the same period (also called the frame) and have the same deadlines which are equal to the end of the period. Many real-world real-time systems, especially embedded systems, are frame-based, such as tasks decoding MPEG video streaming. Decoding MPEG video streaming involves a series of steps: entropy decoding, IDCT (inverse discrete cosine transform), motion compensation, and dithering [13]. Many researches [14]–[20] have been done to cut down the energy bill for frame-based real-time systems. Gruian and Kuchcinski [16] and Leung et al. [20] develop heuristics for task ordering to reduce the expected energy consumption. Zhang et al. [14] and Xu et al. [15] consider distribution information and slacks concurrently and obtain the optimal speed schedule on the assumption of continuous processor frequency tuning. However, processors, now, can only provide discrete frequencies. Xian and Lu [17] and Xu et al. [15] deal with this problem by applying the research result from Ishihara and Yasuura [21] which uses two adjacent frequencies to execute task. Berten et al. [19] also present a technique to adapt a continuous-speed-based method to a discrete-speed system. However, the derived solution is shown to be not optimal by Chen [18]. Also Chen [18] develops a linear-programming approach to derive optimal solution for frame-based real-time tasks with discrete frequencies, but the solution suffers from exponential time increasing as the number of tasks increases.

This paper derives a new approach to minimize the expected energy consumption for frame-based real-time tasks with discrete probability density functions of their workload information when there are only limited frequencies with arbitrary power-speed relationship, which is more realistic. The approach explores the relationship between the expected energy consumption and the allowed time for the combination of multiple tasks and solves the problem recursively. Xu et al. [22] give a similar work, but it restricts the speed tunings to the boundaries of tasks. Differently, we

Manuscript received May 17, 2010.
Manuscript revised September 13, 2010.
[†]The authors are with Southeast University, Nanjing, China.
a) E-mail: qiandj@seu.edu.cn
DOI: 10.1587/transinf.E94.D.822

get rid of this limitation and present an algorithm to achieve optimal result when the execution order is specified. A sub-optimal DVS scheme is also presented to work out a solution near enough to the optimal one with only polynomial time expense. Experimental results show that the proposed algorithms can effectively reduce the expected energy consumption and can save at most 40% of the energy compared with previous schemes.

The rest of this paper is organized as follows: Sect. 2 provides system model and problem definition. Section 3 gives a motivational example and presents the optimal algorithm. The polynomial time algorithm is introduced in Sect. 4. Section 5 gives experimental results of the proposed algorithm. Section 6 is the conclusion.

2. Models and Problem Definition

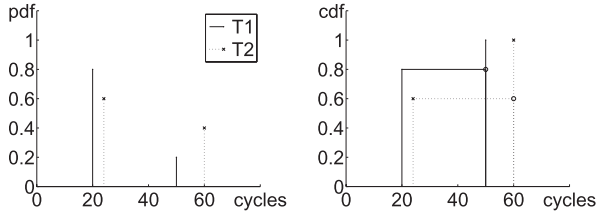
This section describes the model used in this paper along with the notations adopted to present the problem and our approach. The problem of frame-based system is also defined in this section.

2.1 Task Model

The task model used in this paper contains a task set including M periodic tasks. A periodic task is an infinite sequence of task instances, where each instance of a task comes in a regular period. This research focuses on frame-based task set in which task instances release at the same time, share the same period and have the same deadline at the end of the period. The task set is denoted by $\Gamma = \{T_1, T_2, \cdots, T_M\}$. Because of the periodic property, we only study the first period (also called frame), which starts at time 0 and ends at time D.

Each task T_i is characterized by its worst-case execution cycles (WCEC) W_i and the probability function of its execution cycles $p_i(\mathrm{x})$. $p_i(\mathrm{x})$ gives the probability that task T_i execute for x ($1 \le \mathrm{x} \le W_i$) cycles. Obviously, $\sum_{x=1}^{W_i} p_i(x) = 1$ and $p_i(W_i) \neq 0$. In practice, a histogram are used to represent the probability function considering that a task usually have millions of cycles. For each task T_i, the range (0, W_i] is divided into K_i bins. The j^{th} bin of task T_i is associated with its amount of cycle $X_{i,j}$ and its probability density function (PDF) $p_i(\mathrm{j})$. That is, the probability for task T_i with $\sum_{k=1}^{j} X_{i,k}$ cycles is $p_i(\mathrm{j})$. (Traditionally, profiling is done by using the same bin size to determine the probability. However, by eliminating the points with 0% probability during profiling, we can have bins with different sizes to reduce the input size.) The discrete cumulative density function (CDF) for task T_i to have cycles no more than $\sum_{k=1}^{j} X_{i,k}$ is $\psi_i(\mathrm{j}) = \sum_{k=1}^{j} p_i(k)$. By definition, $\psi_i(K_i) = 1$. For notational brevity, we define $\psi_i(0)$ as 0. Therefore, the probability that the schedule has to execute the first $\sum_{k=1}^{j} X_{i,k}$ cycles of task T_i is $1 - \psi_i(j-1)$, denoted by $\Psi_i(\mathrm{j})$.

Figure 1 is an example for a task set $\Gamma = \{T_1, T_2\}$ with $K_1 = K_2 = 2$. The PDFs are shown in Fig. 1 (a), where $X_{1,1} = 20$, $X_{1,2} = 30$, with $p_1(1) = 0.8$, $p_1(2) = 0.2$ and $X_{2,1} = 24$, $X_{2,2} = 36$ with $p_2(1) = 0.6$, $p_2(2) = 0.4$. Figure 1 (b) gives the discrete CDFs. Hence, in this example, $\Psi_1(1) = \Psi_2(1) = 1$, $\Psi_1(2) = 0.2$ and $\Psi_2(2) = 0.4$.



(a) probability density function (b) cumulative density function

Fig. 1 Workload distribution example.

2.2 System Model

The tasks are to be executed on a DVS system with N frequencies, denoted by $f_1, f_2, \cdots, f_N$, where $f_1 < f_2 < \cdots < f_N$. The power consumption function at frequency f_i is denoted by $P(f_i) = P_d(f_i) + P_i$, where $P_d(f_i)$ stands for the dynamic power consumption and P_i, which is assumed to be a constant, denotes the static power consumption [12]. When the system is idle, we assume the frequency to be 0 and assign $P_d(0) = 0$. Since the static power consumption is a constant, the power consumption function $P(f_i)$ could be rescaled by substituting P_i, while not affecting the analysis results. Distinct from many previous work [8], [10], [14], [15], [17] which assumes $P(f_i) = f_i^{\alpha}$ ($\alpha > 2$), this paper can cope with $P(f_i)$ as an arbitrary function.

The energy consumption in $(t_1, t_2]$ is $\int_{t_1}^{t_2} P(f(\mathrm{t}))\mathrm{dt}$, where $f(\mathrm{t})$ is the processor frequency at time t. Hence, executing a cycle at frequency f_i requires energy consumption $P(f_i)/f_i$. According to the results from [12], only frequencies meeting the following conditions are considered,

- $P(f_i)/f_i < P(f_{i+1})/f_{i+1}$, $\forall 1 \le i \le N-1$, since a schedule would not use a lower frequency with higher energy consumption.
- $\frac{P(f_i)/f_i - P(f_{i-1})/f_{i-1}}{1/f_{i-1} - 1/f_i} \le \frac{P(f_{i+1})/f_{i+1} - P(f_i)/f_i}{1/f_i - 1/f_{i+1}}$, $\forall 2 \le i \le N-1$, since $\frac{P(f_i)/f_i - P(f_{i-1})/f_{i-1}}{1/f_{i-1} - 1/f_i} > \frac{P(f_{i+1})/f_{i+1} - P(f_i)/f_i}{1/f_i - 1/f_{i+1}}$, indicates that f_i is inefficient.

2.3 Problem Description

A speed schedule function $S(\cdot)$ gives speed for each cycle of a certain task. For frame-based systems, a DVS scheme is composed of M sets of speed schedule functions $S_i^t(\cdot)$ (i = 1, 2, $\cdots$, M), each for a task. $S_i^t(\mathrm{x})$ denotes the speed to execute cycle x of task T_i, when T_i is scheduled to execute and there is time t remaining in the frame. Let $e(S, \mathrm{x})$ and $t(S, \mathrm{x})$ denote the energy consumption and time for executing a task using speed schedule S when the actual number of execution cycles of the task is x. $e(S, \mathrm{x})$ and $t(S, \mathrm{x})$ are computed by

$$e(S, x) = \sum_{y=1}^{x} \frac{P(S(y))}{S(y)} \quad (1)$$

$$t(S, x) = \sum_{y=1}^{x} \frac{1}{S(y)} \quad (2)$$

respectively, where $\frac{P(S(y))}{S(y)}$ and $\frac{1}{S(y)}$ are the energy and time consumed by the y^{th} cycle respectively. The expected energy consumption for executing T_i, T_{i+1}, $\cdots$, T_M, when there is time t left for execution, can be computed by

$$E_i(t) = \sum_{j=1}^{K_i} p_i(j)\left(e\left(S_i^t, \sum_{k=1}^{j} X_{i,k}\right) + E_{i+1}\left(t - t\left(S_i^t, \sum_{k=1}^{j} X_{i,k}\right)\right)\right) \quad (3)$$

and $E_{M+1}(\mathrm{t}) = 0$. Thus, the problem is to find the speed schedule functions S_i^t that minimize $E_1(\mathrm{D})$.

It's hard to work out a general mathematical solution to this problem, as the power-speed relationship is arbitrary and the frequency is not continuous. This paper tries to give an algorithm for optimal solution and a suboptimal method with polynomial time expense.

3. Optimal Scheme

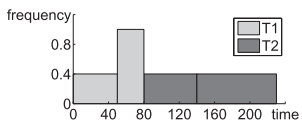
3.1 A Motivational Example

Before going into the detail of our solution, we give a motivational example to help getting a preliminary knowledge adopting the task set introduced in Fig. 1. Of course, the tasks have the same period (we take 230 as an example), as we focus on frame-based systems in this paper. Suppose the system has three frequencies: 0.2, 0.4 and 1.0, where $P(f_j) = f_j^3$. Figure 2 (a) to Fig. 2 (d) show the result by applying algorithm M-GREEY proposed by Chen [18] along with slack reclamation. Figure 2 (a) gives the schedule result by using algorithm M-GREEDY. Following this schedule result, the first 20 cycles of T_1 and the whole cycles of T_2 are assigned frequency 0.4, while the rest cycles of T_1 are assigned frequency 1. Figure 2 (b) shows the situation in which T_2 only execute 24 cycles. When T_1 completes earlier than worst case situation, slack could be used to lower the energy consumption further. Figure 2 (c) and Fig. 2 (d) are the results after using the slack, where the first 12 cycles of T_2 are lowered to frequency 0.2. The expected energy consumption could be calculated as $0.08 \cdot 42.8 + 0.12 \cdot 37.04 + 0.32 \cdot 11.36 + 0.48 \cdot 5.6 = 14.192$. Although the approach works, it does not derive an optimal solution which minimizes the expected energy consumption. When speed scheduling in Fig. 2 (e) to Fig. 2 (h) are applied, the expected energy consumption will be $0.08 \cdot 42.8 + 0.12 \cdot 11.84 + 0.32 \cdot 11.36 + 0.48 \cdot 5.6 = 11.168$. It gets the same result of the Linear-Program method [18], which is optimal for frame-based systems, with less time effort (this will explained later), and saves energy as much as 21.3% compared to M-GREEDY. The following text will present how this scheduling result is calculated.

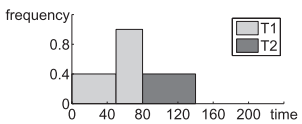
3.2 Problem Redefinition

In fact, the speed schedule function $S^t(\cdot)$ introduced in Sect. 2.3 is a two-variable function. Its value depends on both the time t and the cycle x, where t is a positive real and x is a positive integer. As t is continuous and the domain of x could be large (usually a task may have millions of cycles), it's hard to express $S^t(\cdot)$ in computer. Fortunately, the problem could be redefined to an easy form by bringing the concept of virtually continuous speed introduced in [23].

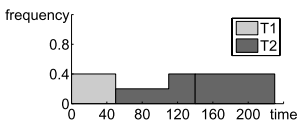
Virtual continuous frequency extends the discrete frequency set $\{f_1, f_2, \cdots, f_N\}$ to a continuous region $(0, f_N]$



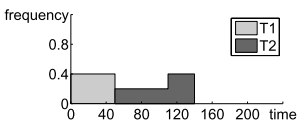
(a) Speed schedule when T_1 and T_2 both execute the whole cycles. The probability for this case is $0.2 \cdot 0.4 = 0.08$. The energy consumed is $20 \cdot 0.4^2 + 30 \cdot 1^2 + 60 \cdot 0.4^2 = 42.8$.



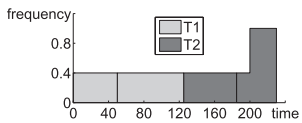
(b) Speed schedule when T_1 and T_2 execute 40 and 30 cycles respectively. The probability for this case is $0.2 \cdot 0.6 = 0.12$. The energy consumed is $20 \cdot 0.4^2 + 30 \cdot 1^2 + 24 \cdot 0.4^2 = 37.04$.



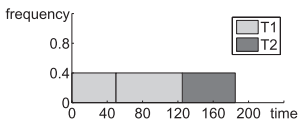
(c) Speed schedule when T_1 and T_2 execute 20 and 60 cycles respectively. The probability for this case is $0.8 \cdot 0.4 = 0.32$. The energy consumed is $20 \cdot 0.4^2 + 12 \cdot 0.2^2 + 48 \cdot 0.4^2 = 11.36$.



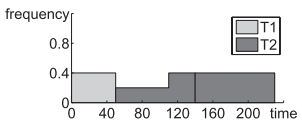
(d) Speed schedule when T_1 and T_2 execute 20 and 24 cycles respectively. The probability for this case is $0.8 \cdot 0.6$=0.48. The energy consumed is $20 \cdot 0.4^2 + 12 \cdot 0.2^2 + 12 \cdot 0.4^2 = 5.6$.



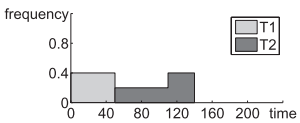
(e) Speed schedule when T_1 and T_2 both execute the whole cycles. The probability for this case is $0.2 \cdot 0.4 = 0.08$. The energy consumed is $50 \cdot 0.4^2 + 30 \cdot 0.4^2 + 30 \cdot 1^2 = 42.8$.



(f) Speed schedule when T_1 and T_2 execute 50 and 24 cycles respectively. The probability for this case is $0.2 \cdot 0.6 = 0.12$. The energy consumed is $50 \cdot 0.4^2 + 24 \cdot 0.4^2 = 11.84$.



(g) Speed schedule when T_1 and T_2 execute 20 and 60 cycles respectively. The probability for this case is $0.8 \cdot 0.4 = 0.32$. The energy consumed is $20 \cdot 0.4^2 + 12 \cdot 0.2^2 + 48 \cdot 0.4^2 = 11.36$.



(h) Speed schedule when T_1 and T_2 execute 20 and 24 cycles respectively. The probability for this case is $0.8 \cdot 0.6$=0.48. The energy consumed is $20 \cdot 0.4^2 + 12 \cdot 0.2^2 + 12 \cdot 0.4^2 = 5.6$.

Fig. 2 A motivational example.

by simulating a frequency that is not in the set with its two immediate neighbor frequencies. Using the immediate neighbor frequencies arrives at minimal energy consumption, though other frequencies may work also [21]. Suppose to execute a task with X cycles within time t, the frequency $f = \frac{X}{t}$ ($f_i < f < f_{i+1}$) could be simulated by executing λX cycles with frequency f_i and $(1-\lambda)\cdot$X cycles with frequency f_{i+1}, where

$$\lambda = \frac{1/f - 1/f_{i+1}}{1/f_i - 1/f_{i+1}} \tag{4}$$

$$1 - \lambda = \frac{1/f_i - 1/f}{1/f_i - 1/f_{i+1}} \tag{5}$$

Applying the virtually continuous frequency, the speed schedule function $S_i^t(\cdot)$ for task T_i could be rewritten as a K_i-dimension vector function $\vec{S}_i(\cdot)$, whose elements are denoted as $S_i^j(\cdot)$ ($1 \le j \le K_i$). $S_i^j(t)$ gives the speed for the j^{th} bin when task T_i is scheduled to execute with time t left in the frame. Instead of formula (1-3), the schedule problem could then be rewritten as following,

$$E_{i,j}(\vec{s}) = \begin{cases} \frac{P(f_q)}{f_q} \cdot X_{i,j}, & s^j = f_q \\ \frac{P(f_{q+1})}{f_{q+1}} \cdot (1-\lambda) \cdot X_{i,j} & \\ \quad + \frac{P(f_q)}{f_q} \cdot \lambda \cdot X_{i,j}, & f_q < s^j < f_{q+1} \\ \frac{P(f_1)}{f_1} \cdot X_{i,j}, & s^j < f_1 \end{cases} \tag{6}$$

$$t_{i,j}(\vec{s}) = \frac{X_{i,j}}{s^j} \tag{7}$$

$$E_i(t) = \sum_{j=1}^{K_i} p_i(j) \cdot \left(\sum_{k=1}^{j} E_{i,k}(\vec{S}_i(t)) + E_{i+1}\left(t - \sum_{k=1}^{j} t_{i,k}(\vec{S}_i(t)) \right) \right) \tag{8}$$

and $E_{M+1}(\mathrm{t}) = 0$. $E_{i,j}(\vec{s})$ and $t_{i,j}(\vec{s})$ represent the energy consumption and time for the j^{th} bin of task T_i when schedule function $\vec{s}$ is used. The certain speed scheduling functions $\vec{S}_i(\cdot)$ (i = 1, 2, $\cdots$, M) are to be found to minimize the expected energy consumption $E_1(\mathrm{D})$. This form of definition is used to present our solution in the rest of this paper.

3.3 The Optimal Solution

From the recursive description of the problem in Sect. 3.2, it is natural to compute $E_i(\cdot)$ and $S_i^j(\cdot)$ in reverse order, i.e., first compute $E_M(\cdot)$ and $S_M^j(\cdot)$, then $E_{M-1}(\cdot)$ and $S_{M-1}^j(\cdot)$, and so on. The computation of $E_i(\cdot)$ and $S_i^j(\cdot)$ only depends on $E_{i+1}(\cdot)$, as $E_{i+1}(\cdot)$ has already "summarized" functions $E_k(\cdot)$ and $S_k^j(\cdot)$ where k = i+2, $\cdots$, M. When the computation is done, all $E_i(\cdot)$ (i = 1, 2, $\cdots$, M) can be discarded because they are not needed for the operation of the system.

3.3.1 Solution for Bins

This section takes into account the energy consumption of the bins, which are elements of tasks. Without loss of generality, we try to find the speed schedule function $S_i^j(\cdot)$ for the j^{th} bin of task T_i, so that $E_{i,j}(t)$ equals the minimal energy consumption when time t is available. This is a well-known problem and has been studied by previous researches. We continue to explore this problem and examine it much further. Finally, some meaningful results will be achieved and become the basics of our later presented scheme.

Obviously, this is an inter-task schedule problem. The result could be solved from Eq. (6) and (7) directly,

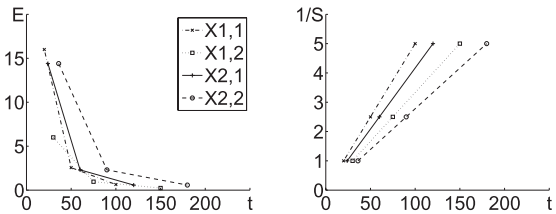
$$E_{i,j}(t) = \begin{cases} \frac{P(f_q)}{f_q} \cdot X_{i,j}, & t = \frac{X_{i,j}}{f_q} \\ \frac{P(f_{q+1})}{f_{q+1}} \cdot (1-\lambda) \cdot X_{i,j} & \\ \quad + \frac{P(f_q)}{f_q} \cdot \lambda \cdot X_{i,j}, & \frac{X_{i,j}}{f_{q+1}} < t < \frac{X_{i,j}}{f_q} \\ \frac{P(f_1)}{f_1} \cdot X_{i,j}, & t > \frac{X_{i,j}}{f_1} \end{cases} \tag{9}$$

$$S_i^j(t) = \frac{X_{i,j}}{t} \tag{10}$$

where, $\lambda = \frac{t/X_{i,j} - 1/f_{q+1}}{1/f_q - 1/f_{q+1}}$. No value is defined for both functions, when $t < \frac{X_{i,j}}{f_1}$. From the solution, we could see that $E_{i,j}(t)$ is a piecewise line with N line segments (N is the number of discrete frequencies) and the i^{th} segment has a slope of $\frac{P(f_i)/f_i - P(f_{i+1})/f_{i+1}}{1/f_i - 1/f_{i+1}}$ with the exception of the 0-valued slope of the N^{th} segment). $1/S_i^j(t)$ is also a piecewise line with N line segments with the same slope $\frac{1}{X_{i,j}}$ and the i^{th} segment shares the starting time and ending time with the i^{th} segment of $E_{i,j}(t)$. It is shown in Sect. 2.2 that $\frac{P(f_i)/f_i - P(f_{i+1})/f_{i+1}}{1/f_i - 1/f_{i+1}} < 0$ and $\frac{P(f_{i-1})/f_{i-1} - P(f_i)/f_i}{1/f_{i-1} - 1/f_i} \ge \frac{P(f_i)/f_i - P(f_{i+1})/f_{i+1}}{1/f_i - 1/f_{i+1}}$. Thus, $E_{i,j}(t)$ is seemed to be a non-increasing convex curve. Figure 3 (a) and Fig. 3 (b) illustrate the example for $E_{i,j}(t)$ and $1/S_i^j(t)$ respectively using the tasks in Fig. 1. Note that the graph when $t > \frac{X_{i,j}}{f_1}$ is ignored as it makes the figure too confused and provides no more information.

The piecewise linear functions play an important role in our approach. Thus, being able to represent and manipulate piecewise linear functions effectively is crucial for the viability of our approach.

Looking into Eq. (9), we could find that each line segment of $E_{i,j}(t)$ can be represented by its left end point because its right end point is the left end point of the line segment to its immediate right or is constant until infinity when it is the rightmost line segment of the piecewise line. We formally define piecewise linear function through the following two definitions.



(a) optimal energy consumption (b) optimal speed scheduling

Fig. 3 Solution for bin.

Definition 1: A point **P** is a 2-tuple (e, t), where e and t are nonnegative reals and denote energy and time respectively. We write the energy component as **P**.e and the time component as **P**.t.

Definition 2: A piecewise linear function $\mathbf{F}(\cdot)$ is defined as a point sequence $\mathbf{S} = [\mathbf{P}_1, \mathbf{P}_2, \cdots, \mathbf{P}_m]$ where $\mathbf{P}_1.t < \mathbf{P}_2.t < \cdots < \mathbf{P}_m.t$ and $\mathbf{P}_1.e > \mathbf{P}_2.e > \cdots > \mathbf{P}_m.e$. $\mathbf{F}(t)$ is undefined when $t < \mathbf{P}.t$, otherwise

$$\mathbf{F}(t) = \begin{cases} \mathbf{P}_i.e + \frac{\mathbf{P}_{i+1}.e - \mathbf{P}_i.e}{\mathbf{P}_{i+1}.t - \mathbf{P}_i.t} \cdot (t - \mathbf{P}_i.t), & \mathbf{P}_i.t \le t < \mathbf{P}_{i+1}.t \\ \mathbf{P}_m.e, & t \ge \mathbf{P}_m.t \end{cases} \quad (11)$$

In the rest of this paper, we use **F** and $\mathbf{F}.\mathbf{P}_i$ to denote a piecewise linear function $\mathbf{F}(\cdot)$ and the i^{th} point of **F** respectively. The number of the points in **F** is used to define the size of **F** and denoted by $|\mathbf{F}|$. Obviously, computing **F**(t) can be done in time $O(log|\mathbf{F}|)$ by using binary search. With this definition, $E_{i,j}(t)$ could be rewritten by $\mathbf{E}_{i,j} = [(\frac{P(f_M)}{f_M} \cdot X_{i,j}, \frac{X_{i,j}}{f_M}), (\frac{P(f_{M-1})}{f_{M-1}} \cdot X_{i,j}, \frac{X_{i,j}}{f_{M-1}}), \cdots, (\frac{P(f_1)}{f_1} \cdot X_{i,j}, \frac{X_{i,j}}{f_1})]$.

We now give some operations about the piecewise linear function.

Definition 3: The operator "$\cdot$" is defined between a real x and a piecewise linear function **F** such that $x \cdot \mathbf{F} = [(x \cdot \mathbf{F}.\mathbf{P}_1.e, \mathbf{F}.\mathbf{P}_1.t), (x \cdot \mathbf{F}.\mathbf{P}_2.e, \mathbf{F}.\mathbf{P}_2.t), \cdots]$ (i.e., the result is still a step function). This operator means to scale the piecewise linear function by a factor of x.

Operation "$\cdot$" takes time $O(|\mathbf{F}|)$ to complete with the size of the piecewise linear functions unchanged.

Definition 4: The sum operator "+" is defined between 2 piecewise linear functions, $\mathbf{F}_1$ and $\mathbf{F}_2$, such that $\mathbf{F}_1 + \mathbf{F}_2 = \mathbf{F}$ and $\mathbf{F}(t) = \mathbf{F}_1(t) + \mathbf{F}_2(t)$ with $\mathbf{F}.\mathbf{P}_1.t = max(\mathbf{F}_1.\mathbf{P}_1.t, \mathbf{F}_2.\mathbf{P}_1.t)$.

The time component of each point in **F** comes from either $\mathbf{F}_1$ or $\mathbf{F}_2$. As $\mathbf{F}.\mathbf{P}_1.t = max(\mathbf{F}_1.\mathbf{P}_1.t, \mathbf{F}_2.\mathbf{P}_1.t)$, the size of **F** is $|\mathbf{F}| \le |\mathbf{F}_1| + |\mathbf{F}_2|$. Because the points in $\mathbf{F}_1$ and $\mathbf{F}_2$ are already sorted, the time components of all points in **F** can be obtained by a procedure similar to merge sort in time $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$. The energy component of each point in **F** is acquired by computing the addition of $\mathbf{F}_1(\mathbf{F}.\mathbf{P}_i.t)$ and $\mathbf{F}_2(\mathbf{F}.\mathbf{P}_i.t)(1 \le i \le |\mathbf{F}|)$. It also takes time $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$ to complete. As a result, the time to fulfill the sum operation between $\mathbf{F}_1$ and $\mathbf{F}_2$ turns out be to $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$.

Definition 5: The merge operator "$\bigcup$" is defined between 2 piecewise linear functions, $\mathbf{F}_1$ and $\mathbf{F}_2$, such that $\mathbf{F}_1 \bigcup \mathbf{F}_2 = \mathbf{F}$, where **F** merges all the line segments of both $\mathbf{F}_1$ and $\mathbf{F}_2$ by sorting them with their slopes in ascending order and connecting them into a new piecewise line. The first point of **F** is defined by $\mathbf{F}.\mathbf{P}_1.e = \mathbf{F}1.\mathbf{P}_1.e + \mathbf{F}_2.\mathbf{P}_1.e$ and $\mathbf{F}.\mathbf{P}_1.t = \mathbf{F}_1.\mathbf{P}_1.t + \mathbf{F}_2.\mathbf{P}_1.t$.

The resulting piecewise linear function **F** by the merge operators over 2 piecewise linear functions $\mathbf{F}_1$ and $\mathbf{F}_2$ have $|\mathbf{F}_1| + |\mathbf{F}_2| - 1$ points. Computing the emerge operation contains calculating the slopes of all the piecewise segments and sorting them in an ascending order, which takes time $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$, and computing the time and energy components of the new points, which also takes time $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$. Thus, computing $\mathbf{F}_1 \bigcup \mathbf{F}_2$ takes time $O(|\mathbf{F}_1| + |\mathbf{F}_2|)$ too.

The piecewise line derived from the solution of bins, along with the defined operations will be used in the rest of the paper to introduce our schedule scheme.

3.3.2 Solution for T_M

Functions $E_M(\cdot)$ and $S^j_M(\cdot)$ are examined in this section. In case of i=M and $E_{M+1}(t) = 0$, formula (8) turns out to be,

$$E_M(t) = \sum_{j=1}^{K_M} \left(p_M(j) \cdot \sum_{k=1}^{j} E_{M,k}(\vec{S^j_M}(t)) \right) = \sum_{j=1}^{K_M} \Psi_M(j) \cdot E_{M,j}(S^j_M(t)) \quad (12)$$

Assume the j^{th} bin spend time t_j, when $E_M(t)$ is optimal. Using the result achieved in Sect. 3.3.1, we know that $S^j_M(t) = X_{M,j}/t_j$ and $E_{M,j}(S^j_M(t)) = E_{M,j}(t_j)$. Then, the problem could be rewritten as,

$$E_M(t) = \sum_{j=1}^{K_M} \Psi_M(j) \cdot E_{M,j}(t_j) \quad (13)$$

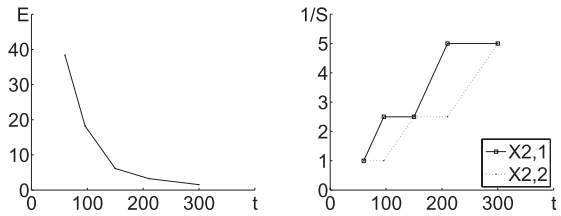
$$t = \sum_{j=1}^{K_M} t_j \quad (14)$$

Lemma 1: The optimal $E_M(t)$ is a piecewise linear function, and could be computed by $E_M = \bigcup_{j=1}^{K_M} (\Psi_M(j) \cdot E_{M,j})$.

Proof: This is an assignment problem. The aim is to find an low-power assignment of time t to each bin of task T_M so that $E_M(t)$ is minimized. Obviously, $dE_M(t) = \sum_{j=1}^{K_M} \left(\Psi_M(j) \cdot \frac{dE_{M,j}(t_j)}{dt_j} dt_j \right) \ge \min\{\Psi_M(j) \cdot \frac{dE_{M,j}(t_j)}{dt_j}\} \cdot \sum_{j=1}^{K_M} (dt_j) = \min\{\Psi_M(j) \cdot \frac{dE_{M,j}(t_j)}{dt_j}\} \cdot dt$. This means that the increased time dt should be assigned to the bin with lowest $\Psi_M(j) \cdot \frac{dE_{M,j}(t_j)}{dt_j}$ in order to achieve the largest power consumption decreasing. $E_{M,j}$ is piecewise linear, so E_M is piecewise linear too. Also, it is not difficult to find out that $E_M.\mathbf{P}_1.t = \sum_{j=1}^{K_M} E_{M,j}.\mathbf{P}_1.t$ and $E_M.\mathbf{P}_1.e = \sum_{j=1}^{K_M} E_{M,j}.\mathbf{P}_1.e$. As a result, The optimal $E_M(t)$ is a piecewise linear function and $E_M = \bigcup_{j=1}^{K_M} (\Psi_M(j) \cdot E_{M,j})$ □

The function $1/S^j_M(t)$ could be obtained following the process of computing $E_M(t)$ easily by merging and rearranging the segments of $1/S^j_M(t)$ accordingly during the sorting stage of the merge operation for $E_i(t)$. We will focus on discussing $E_i(t)$ in the following for its important position.

Applying the result of Lemma 1, we calculate the speed schedule functions for T_2 of the example introduced in Fig. 1 and obtain the result shown in Fig. 4. $E_M(\cdot)$, presented in Fig. 4 (a), is non-increasing and convex and contains $K_M \cdot (N-1)$ (M=2 and $K_M \cdot (N-1) = 4$ in this example) linear segments. Figure 4 (b) shows the functions of



(a) optimal energy consumption (b) optimal speed scheduling

Fig. 4 Solution for T_M.

$1/S_M^j(\cdot)$ $(1 \le j \le K_M)$, which are also piecewise linear. It is not surprising us that the functions keep non-descending, as more time leads to lower speed (longer cycle length). There exists an interesting fact that functions $1/S_M^j(\cdot)$ ascend in a mutually exclusive manner. The reason is that only the most energy-efficient bin is chosen to use the remaining time.

3.3.3 Solution for T_i

Now we examine functions $E_i(\cdot)$ and $\vec{S}_i(\cdot)$ $(1 \le i \le M)$ which involve multiple tasks. Similar to Sect. 3.3.2, we could rewrite formula (8) as,

$$E_i(t) = \sum_{j=1}^{K_i} \left(\Psi_i(j) \cdot E_{i,j}(t_j) + p_i(j) \cdot E_{i+1}\left(t - \sum_{k=1}^{j} t_k\right)\right) \quad (15)$$

Furthermore, we could continue to write Eq. (15) in a recursive manner as following,

$$E_i^{(j)}(t) = \Psi_i(j) \cdot E_{i,j}(t_j) + p_i(j) \cdot E_{i+1}(t - t_j) + E_i^{(j+1)}(t - t_j) \quad (16)$$

and $E_i^{(K_i+1)}(t) = 0$. When j = 1, $E_i^{(j)}(t)$ becomes $E_i^{(1)}(t)$, and equals Eq. (15).

Lemma 2: The optimal $E_i^{(j)}(t)$ is a piecewise linear function, and could be computed by $E_i^{(j)}(t) = (\Psi_i(j) \cdot E_{i,j}) \bigcup (p_i(j) \cdot E_{i+1} + E_i^{(j+1)})$.

Proof: This could be proved similar to lemma 1. □

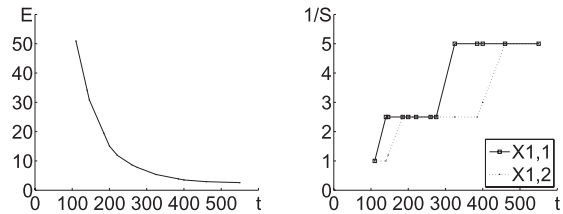
Using the formula in Lemma 2 recursively, we could get the optimal result for $E_i(t)$. Also, the schedule function $\vec{S}_i(\cdot)$ could be calculated easily while computing $E_i(t)$. The algorithm is named GLOBAL and described in Fig. 5.

An example is shown in Fig. 6, where Fig. 6 (a) and Fig. 6 (b) give $E_i(\cdot)$ and $1/\vec{S}_i(\cdot)$ respectively. Now, we try to present how we acquire the optimal result in Sect. 3.1. First, we calculate $1/\vec{S}_1(230) = [2.5, 2.5]$ by the function in Fig. 6 (b). Then, we execute task T_1 with this schedule result. If T_1 execute 50 cycles, it takes time $50 \cdot 2.5 = 125$, and has time $230 - 125 = 105$ left. Finally, we use function in Fig. 4 (b) to get $1/\vec{S}_2(95) = [2.5, 1.25]$ for the execution of T_2. As 1/1.25 is not the supported frequency, we simulate it with frequencies 0.4 and 1.0. The corresponding executing trace is in Fig. 1 (e) and (f). If T_1 execute 20 cycles, we get result in Fig. 1 (g) and (h) similarly.

Agorithm: GLOBAL

```
1:  for i = 1 to M do
2:      for j = 1 to K_i do
3:          // compute for each bin
4:          compute E_{i,j}(·)
5:          compute S_i^j(·)
6:      end for
7:  end for
8:
9:  E_{M+1} = 0
10: for i = M downto 1 do
11:     E_i^{(K_i+1)} = 0
12:     // compute for task T_i
13:     for j = K_i downto 1 do
14:         E_i^{(j)} = (Ψ_i(j) · E_{i,j}) ⋃ (p_i(j) · E_{i+1} + E_i^{(j+1)})
15:         update S_i^j(·) correspondingly
16:     end for
17: end for
```

Fig. 5 The optimal algorithm.



(a) optimal energy consumption (b) optimal speed scheduling

Fig. 6 Solution for T_i.

3.4 Space and Time Complexity

We now analyze the time complexity and space complexity for computing E_i. The key operation in computing $E_i^{(j)}$ is the merge operation over 2 piecewise linear functions, with the size of $|E_{i,j}|$ and $|E_{i+1}| + |E_i^{(j+1)}|$ respectively. Thus, the time to compute $E_i^{(j)}$ is $O(|E_{i,j}| + |E_{i+1}| + |E_i^{(j+1)}|)$ and the number of points in $E_i^{(j)}$ is

$$|E_i^{(j)}| \le |E_{i,j}| + (|E_{i+1}| + |E_i^{(j+1)}|) \quad (17)$$

The operation for computing E_i is to compute $E_i^{(j)}$ for K_i times recursively. Thus, the time to compute E_i is $O(K_i \cdot (|E_{i,j}| + |E_{i+1}|))$ and the number of points in E_i is computed as

$$|E_i| = |E_i^{(1)}| \le K_i \cdot N + K_i \cdot |E_{i+1}| \quad (18)$$

Since the base case is $|E_{M+1}| = 1$, we can obtain the closed forms of the time complexity to be $O(N \cdot (\sum_{i=1}^{n}(\prod_{j=1}^{i} K_j)))$ and the number of points in E_1 is

$$|E_1| \le N \cdot \left(\sum_{i=1}^{n} \left(\prod_{j=1}^{i} K_j \right) \right) \quad (19)$$

Formula (19) shows that the number of points in E_1 is exponential to the number of tasks and polynomial to the number of discrete frequencies and the number of bins of the

tasks. The number of points in $\vec{S_1}$, which reflects the space needed to store the assistance data for scheduling, equals E_1. It should be lowed before the algorithm could be used in an embedded system especially when the number of tasks is high. Time overhead is composed of two parts: 1) the time used to find the scheme for each task before the task is to be executed, and 2) the time needed to change the frequency within the execution of a task. The former time is proportional to $log(|\vec{S_i}|)$ and the latter time is proportional to the number of discrete frequencies.

4. Suboptimal Scheme

Section 3 introduced the optimal algorithm which could lead to speed schedule with the minimal expected energy consumption. However, formula (19) has exposed that the number of line segments in the functions may suffer exponential growth, so does the time expense. This section presents an approximate method which brings out an algorithm with polynomial time complexity and maintains controllable error compared with the optimal expected energy consumption. The idea behind the technique is to merge the two points into one if they are close enough to each other.

4.1 Operation

The approximation method is shown in Fig. 7. It's called TRIM procedure as it aims at trimming away the less important points in a piecewise linear function. The procedure checks every point orderly in line 1. Line 2 examines the distance between $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$. If the distance is relatively small, i.e. $\frac{\mathbf{P}_i.e-\mathbf{P}_{i+1}.e}{\mathbf{P}_{i+1}.e} < \delta$ (δ is a parameter to quantify the difference), point $\mathbf{P}_{i+1}$ will be trimmed away as shown in line 3.

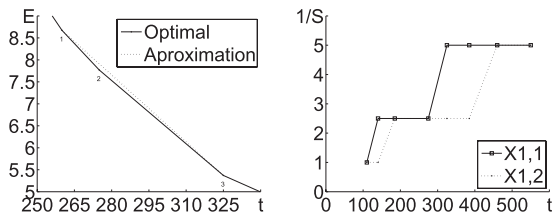
Using this approximation technique could greatly reduce the points in the functions, and thus the time and the space complexity. In fact, the number of turning points is reduced and upper bounded by a polynomial in $\frac{1}{\delta}$.

Figure 8 (a) shows an amplified part of Fig. 6 (a) with the time region of (250, 340). There are three points in the figure, (e_1, t_1), (e_2, t_2) and (e_3, t_3), where $e_1 > e_2 > e_3$ and $t_1 < t_2 < t_3$. It's assumed that $\frac{e_1-e_2}{e_2} < \delta$. After applying the TRIM procedure, point (e_2, t_2) will be eliminated and the graph changes from the solid line to the dotted line which contains only two points. Obviously, the resulting function is only an approximation of the original function (i.e., the elimination induces error). This is because when time t where $t_1 < t < t_3$ is available, we use the dotted line instead of the solid line and result in expected energy greater. However, because of the way we eliminate the points, the difference between the resulting function and the original function is guaranteed to be no more than δ times the original function. We will explain this in detail in the next section.

This approximation technique could guarantee the schedulability and meet the real-time demand, as $\vec{S_i}(\cdot)$ is not affected by the TRIM procedure. In fact, we could eliminate some non-end points locating on the line segments to reduce the size of $\vec{S_i}(\cdot)$ further. Figure 8 (b) shows an example of the result of Fig. 6 (b).

Procedure: TRIM
1: **for** $i = 1$ to $|\mathbf{F}| - 1$ **do**
2: **if** $|\mathbf{P}_i.e - \mathbf{P}_{i+1}.e| < \delta \cdot \mathbf{P}_{i+1}.e$ **then**
3: eliminate $\mathbf{P}_{i+1}$
4: **end if**
5: **end for**

Fig. 7 The TRIM procedure.



(a) energy consumption (b) speed scheduling

Fig. 8 Aproximation.

We apply the above function approximation technique to function $E_i(\cdot)$ before computing $E_{i-1}(\cdot)$. Thus, the error accumulates and increases as i decreases. Let the error of $E_1(\cdot)$ be denoted by ε. The expected energy consumption of the system, $E_1(D)$, is within a factor of $1 + \varepsilon$ of the optimal expected energy consumption. If we let ε be a parameter set by system designers, we can derive the value of δ to be used for each function approximation. The technical detail will be described in the following.

4.2 Details

This section details how to reduce the size of function E_i within a polynomial bound while maintain a controllable energy consumption error. The function approximation achieved by the trimming procedure is similar to the label elimination technique used in [24]. To do that, we trim (i.e., remove some points) function E_i after it is computed after Line 14 in Fig. 5 using the TRIM procedure introduced in Fig. 7. A trimming parameter δ ($0 < \delta < 1$) is used to direct the trimming. After function E_i is trimmed, the distances of any adjacent points differ by at least a factor of δ. The operations from line 11 to line 16 in Fig. 5 transport the error without any change. This could be obtained by analyzing Formula (15).

Let $E_i^{'}(i = 1, 2, \cdots, M)$ be the piecewise linear functions obtained after the TRIM procedure is applied for E_i. That is, $E_i^{'}$ is the functions returned by the approximation algorithm. By comparing $E_i^{'}$ and E_i, we have the following important lemma:

Lemma 3: $E_i(t) \le E_i^{'}(t) \le (1+\delta)^{M+1-i} \cdot E_i(t)$ for any value of t.

Proof: The proof is by induction on i and the base case for i=M+1 obviously holds from Line 9 in Fig. 5. In the induction step for $E_i^{'}$, we inspect Line 11 to Line 16 in Fig. 5 along with Formula (15). From the hypothesis, $E_{i+1}^{'}(t)$ is

within a factor of $(1+\delta)^{N-i}$ of $E_{i+1}(t)$. All the operations at Line 11 to Line 16 will preserve this property. After the trimming operation, the factor will be only increased by $(1+\delta)$, which will make $E_i'(t)$ with a factor of $(1+\delta)^{N-i+1}$ of $E_i(t)$. □

Using functions E_i' will lead to expected energy consumption of $E_1'(D)$ and using functions E_i will lead to expected energy consumption of $E_1(D)$. From Lemma 3, we have $E_1'(D) \leq (1+\delta)^N \cdot E_1(D)$. If we choose δ to be $\sqrt[N]{1+\varepsilon}-1$, we have $E_1'(D) \leq (1+\varepsilon)\cdot E_1(D)$.

To compute the upper bound of the number of points in E_i, we note that after the trimming procedure, the energy components of any adjacent points differ by at least a factor of δ. Let the leftmost point in E_i be denoted by $\mathbf{P}_l$ (which is upper bounded by the energy consumption when all tasks use the maximum speed) and the rightmost point in E_i be denoted $\mathbf{P}_r$ (which is lower bounded by the energy consumption when all tasks use the minimum speed). Thus, we have

$$\mathbf{P}_l.e > (1+\delta)^{|E_i|-1} \cdot \mathbf{P}_r.e \tag{20}$$

By some algebraic manipulations, we will obtain $|E_i| = O(\frac{log\lambda}{\delta})$ where $\lambda = \frac{\mathbf{P}_l.e}{\mathbf{P}_r.e} \leq \frac{P(f_N)/f_N}{P(f_1)/f_1}$. Thus, the number of points in E_i is upper bounded by a polynomial in $\frac{1}{\delta}$, so is the space complexity. The time overhead is proportional to $log(|E_i|)$ and N, and is neglectable compared to the execution time of the tasks.

5. Performance Evaluations

This section describes the simulation setup and presents the simulation results comparing the energy savings from our method and the existing solutions.

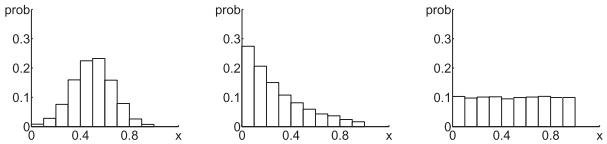
5.1 Simulation Setup

We use the frequency / voltage settings and power consumption of Intel XScale [25] (as shown in Table 1) for finite frequencies. The power consumptions listed in Table 1 are obtained by measuring the processor power when running certain benchmarks. Using polynomial fitting, we could work out the idle power to be 78 mW ($1.537\times10^{-6}\cdot f^3+78$).

A frame-based real-time system is characterized by the number of tasks, the WCEC of each task, the probability distribution of the number of execution cycles of each task, and the frame length. We simulated systems consisting of 5 and 10 tasks. We only show the results for the systems with 5 tasks because the results for systems with 10 tasks are similar. We assume the WCEC for each task is 10,000,000 and the minimum number of cycles is 1,000,000, which is typical for the real multimedia programs like MPEG video decoder and H.263 video decoder [11], [18]. We consider three types of distributions (Gaussian, exponential and uniform) for cycle demand as suggested in [14], [25]. The bin width of the histograms denoting the probability functions

Table 1 XScale speed settings and power consumptions.

Speed (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



(a) normal (b) exponential (c) uniform

Fig. 9 Probability functions.

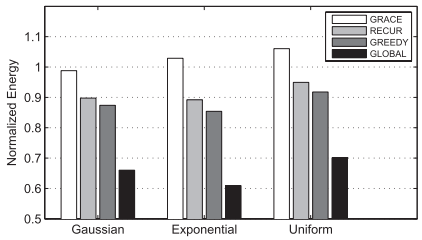


Fig. 10 Energy with different distribution.

is 1,000,000 cycles. Figure 9 shows the histograms of three distributions respectively. We experiment with 20 frame lengths chosen evenly from $\frac{5\times WCEC}{f_M}$ (no slack) to $\frac{5\times WCEC}{f_1}$. 4 DVS schemes, (1) GRACE with $\rho = 0.95$ [11], (2) RECURE [17], (3) GREEDY [18] and (4) GLOBAL with $\delta = 0.5$, are evaluated. GRACE and RECURE are two typical schemes extended from PACE [9] using different methods to cope with processors with discrete frequencies. GREEDY, which is better, treats the discrete property directly like our approach. The energy consumption of all these schemes are normalized to that of STATIC [2]. When evaluating a DVS scheme on a simulated system, we performed a run in which we generated 100,000 frames and computed the average energy consumption per frame as the energy consumption for that scheme.

5.2 Energy Savings

Figure 10 shows the normalized energy for different kinds of task set when frame length is 95 ms. The three groups of bars show the result for task sets with Gaussian, exponential and uniform distributions respectively. For each group, the four bars represent the normalized energy consumption by the four schemes. As shown in the figure, our approach GLOBAL always saves more energy then others no matter which distribution of task set is applied. The energy savings range from about 30% to 40% (in fact later discussion will show the maximum saving of 55% when frame length is 65 ms) compared to STATIC, while GREEDY (which is the best of the other three) save at most 15% energy than STATIC. The reason why GLOBAL shows such a great experience achievement is that GLOBAL considers the whole

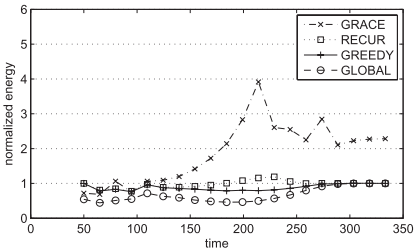


Fig. 11 Energy with different distribution.

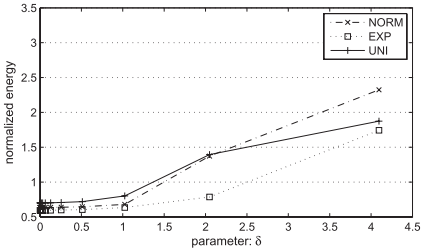


Fig. 12 Energy with different δ.

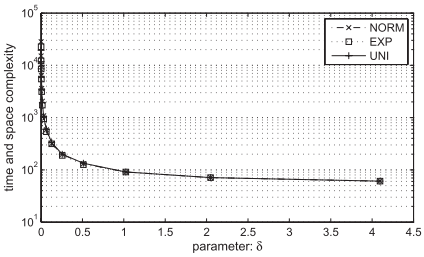


Fig. 13 Time and space overhead with different δ.

tasks globally and uses the slack in a more efficiently way than others. Figure 10 also shows that tasks with exponential distribution leads to more energy saving than the ones with Gaussian or Uniform distribution. Furthermore, the energy savings decrease following the sequence of tasks with exponential, Gaussian and uniform distributions. This is because tasks with exponential distribution have more probability to execute few cycles and produce larger slacks. Tasks with Gaussian distribution overcome those with Uniform distribution for the same reason. Applying GRACE, the tasks with distributions of Gaussian and uniform have normalized energy greater than 1. This is counter-intuitive, as GRACE uses more information than STATIC and introduces intra-task technique for a better effect. This is due to the aggressive strategy in which frequencies are rounded up to match the limited set of frequencies of the processor. The strategy causes the task to run at an unnecessarily high frequency which leads to more energy consumption.

The effect of frame length on energy saving is examined. Figure 11 only shows the result for tasks with Gaussian distribution as others have the similar result. As illustrated in the figure, GLOBAL overcomes other algorithms at any frame length and saves at most 55% of the energy compared to the STATIC at the frame length of 65 ms. When frame length is bigger than 285 ms, all the algorithms (except GRACE) bring out the same effect of energy saving. This is because processor has an extremely low occupation in this circumstance and runs at the lowest frequency even the tasks have the cycles of WCEC. There exists no room for the sophisticated algorithms to work well and they all degenerate to STATIC. Figure 11 also shows that GREEDY and GLOBAL outperform STATIC at any situation, while RECUR and GRACE are not the case. In fact, when frame length is larger than 175 ms, RECUR and GRACE both work no better than STATIC which is simple and easy to realize. In this experiment, GRACE performs not well. Its energy consumption is even 4 times the energy consumption of STATIC when frame length is near 210 ms. As mentioned before, the reason is the rounding up strategy.

5.3 Impact of δ and Computation Overhead

The value of δ is set to 0.5 in above experiments. This section gives the normalized energy and the computation overhead when δ is assigned with different values. After that, we explain the reason why we choose the value 0.5 for previous experiments. To launch the experiments, we change the parameter δ from 0 (TRIM procedure has no effect) to 4.096 (at most 4 points left after TRIM procedure is used) when tasks with Gaussian, exponential and uniform distributions are executed respectively.

The results of the normalized energy are shown in Fig. 12. Lemma 3 demonstrates that the upper bound of the energy increases rapidly as δ increases. However, the experiments exhibit that the actual energy increases much slower. This is because the TRIM procedure only influences parts of the piece-wise curve while the upper bound is obtained assuming the biggest differences in all circumstances. For example, in Fig. 8, the energy equals the optimal one and far smaller than the upper bound outside the range between point 1 and point 3. It is shown in the figure that the energy is near to the optimal one when δ is smaller than 1. However, when δ is bigger than 1, the energy of the tasks of Gaussian and uniform distribution is bigger than the result of static strategy.

As mentioned before, the value of $|E_i|$ is very important and a critical parameter to reflect the space and time complexity. Figure 13 shows the number of points in E_1. The figure presents that the number decreases rapidly as δ increases. When δ is smaller than 0.001, the number is over 10,000. But when δ is bigger than 0.5, the number decreases to around 100. According to the above experiments, we choose δ as 0.5 which leads to comparatively low space and time complexity and energy consumption.

6. Conclusion

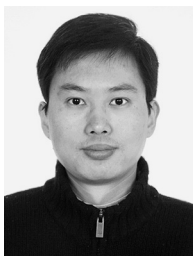
This paper presents an optimal stochastic DVS scheme and a suboptimal (provably close to optimal) stochastic DVS scheme for frame-based multitasking real-time systems with uncertain execution time. The probability distributions of

the cycle demands of tasks are used to help deriving our DVS scheme under the realistic power model. The experimental result shows that our method outperforms the existing solutions and can reduce the expected energy more effectively. For future research, we would like to extend the approach here to general periodic multitasking real-time systems.

References

[1] W. Kim, D. Shin, H.S. Yun, J. Kim, and S.L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," Proc. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), p.219, IEEE Computer Society Washington, DC, USA, 2002.
[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," Proc. 22nd IEEE Real-Time Systems Symposium, pp.95–105, 2001.
[3] P. Pillai and K.G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," Proc. 18th ACM Symposium on Operating Systems Principles, pp.89–102, 2001.
[4] W. Kim, J. Kim, and S. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," Proc. Conference on Design, Automation and Test in Europe, p.788, IEEE Computer Society Washington, DC, USA, 2002.
[5] C.Y. Yang, J.J. Chen, and T.W. Kuo, "Energy-efficiency for multi-frame real-time tasks on a dynamic voltage scaling processor," Proc. 7th IEEE/ACM International Conference Hardware/Software Codesign and System Synthesis, pp.211–220, 2009.
[6] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy, hard real-time applications," IEEE Des. Test Comput., pp.20–30, 2001.
[7] F. Gruian, "On energy reduction in hard real-time systems containing tasks with stochastic execution times," IEEE Workshop on Power Management for Real-Time and Embedded Systems, pp.11–16, Taipei, Taiwan, Citeseer, 2001.
[8] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," International Symposium on Low Power Electronics and Design, 2001.
[9] J.R. Lorch and A.J. Smith, "Pace: A new approach to dynamic voltage scaling," IEEE Trans. Comput., vol.53, no.7, pp.856–869, 2004.
[10] J.R. Lorch and A.J. Smith, "Improving dynamic voltage scaling algorithms with pace," ACM SIGMETRICS Performance Evaluation Review, vol.29, no.1, pp.50–61, 2001.
[11] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," Proc. 20th ACM Symposium on Operating Systems Principles (SOSP), pp.149–163, 2003.
[12] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron, "Procrastinating voltage scheduling with discrete frequency sets," Design, Automation and Test in Europe, p.6, 2006.
[13] J. Mitchell, MPEG video compression standard, Kluwer Academic Publishers, 1997.
[14] Y. Zhang, Z. Lu, J. Lach, M.R. Stan, and K. Skadron, "Optimal procrastinating voltage scheduling for hard real-time systems," Proc. 42nd Annual Design Automation Conference, pp.905–908, 2005.
[15] R. Xu, D. Mosse, and R. Melhem, "Minimizing expected energy in real-time embedded systems," Proc. 5th ACM International Conference On Embedded Software, pp.251–254, 2005.
[16] F. Gruian and K. Kuchcinski, "Uncertainty-based scheduling: Energy-efficient ordering for tasks with variable execution time," Proc. 2003 International Symposium on Low Power Electronics and Design, pp.465–468, Seoul, Korea, 2003.
[17] C. Xian and Y.H. Lu, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," Proc. 16th ACM Great Lakes Symposium on VLSI, pp.392–397, 2006.
[18] J.J. Chen, "Expected energy consumption minimization in dvs systems with discrete frequencies," Proc. 2008 ACM Symposium on Applied Computing, pp.1720–1725, Fortaleza, Ceara, Brazil, 2008.
[19] V. Berten, C.J. Chang, and T.W. Kuo, "Discrete frequency selection of frame-based stochastic real-time tasks," Proc. 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp.269–278, 2008.
[20] L.F. Leung, C.Y. Tsui, and W.H. Ki, "Minimizing energy consumption of hard real-time systems with simultaneous tasks scheduling and voltage assignment using statistical data," Proc. 2004 Asia and South Pacific Design Automation Conference, pp.663–665, Yokohama, Japan, 2004.
[21] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," Proc. 1998 International Symposium on Low Power Electronics and Design, pp.197–202, Monterey, California, United States, 1998.
[22] R. Xu, R. Melhem, and D. Moss, "A unified practical approach to stochastic dvs scheduling," ACM International Conference on Embedded Systems (EMSOFT), 2007.
[23] F. Dabiri, A. Vahdatpour, M. Potkonjak, and M. Sarrafzadeh, "Energy minimization for real-time systems with non-convex and discrete operation modes," Proc. Conference on Design, Automation and Test in Europe, pp.1416–1421, Nice, 2009.
[24] R. Xu, C. Xi, R. Melhem, and D. Mosse, "Practical pace for embedded systems," Proc. 4th ACM International Conference On Embedded Software, pp.54–63, Pisa, Italy, 2004.
[25] Intel XScale Microarchitecture: Benchmarks, http://web.archive.org/web/20050326232506/http://developer.intel.com/design/intelxscale/benchmarks.htm, 2005.

Dejun Qian received the B.S. and M.S. degrees in Electronic Engineering from Southeast University in 2004 and 2007, respectively. He is currently pursuing the Ph.D degree at School of Electronic Science and Engineering, Southeast University, China. His research is focused on embedded system and low-power computing.



Zhe Zhang received the B.S. and M.S. degrees in Electronic Engineering from Southeast University in 1997 and 2002, respectively. He is currently an associate professor in Southeast University. His research interests include embedded system designs, power-aware computing in real-time systems, as well as networking.

Chen Hu received the M.S. degree in Computer Science and Engineering from Nanjing University in 1994. He is currently a professor of Electronic Science and Engineering School at Southeast University, China. His research interests include SoC designs, embedded system designs, as well as networking.

Xincun Ji received the B.S. degrees in Electrical engineering from Suzhou University, Gainesville, FL, in 2006. He is currently pursuing the Ph.D. degree at school of Electronic Science and Engineering, Southeast University, China. His research is focused on frequency synthesizers.