System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation *

Xiliang Zhong and Cheng-Zhong Xu
Department of Electrical and Computer Engineering
Wayne State University, Detroit, Michigan 48202
{xlzhong, czxu}@wayne.edu

ABSTRACT

We present a dynamic voltage scaling (DVS) technique that minimizes system-wide energy consumption for both periodic and sporadic tasks. It is known that a system consists of processors and a number of other components. Energy-aware processors can be run in different speed levels; components like memory and I/O subsystems and network interface cards can be in a standby state when they are active but idle. Processor energy optimization solutions are not necessarily efficient from the perspective of systems. Current system-wide energy optimization studies are often limited to periodic tasks with heuristics in getting approximated solutions. In this paper, we develop an exact dynamic programming algorithm for periodic tasks on processors with practical discrete speed levels. The algorithm determines the lower bound of energy expenditure in pseudo-polynomial time. An approximation algorithm is proposed to provide performance guarantee with a given bound in polynomial running time. Because of their time efficiency, both the optimization and approximation algorithms can be adapted for online scheduling of sporadic tasks with irregular task releases. We prove that system-wide energy optimization for sporadic tasks is NP-hard in the strong sense. We develop (pseudo-) polynomial-time solutions by exploiting its inherent properties.

1. INTRODUCTION

Power management is important for battery-powered embedded systems. Dynamic voltage scaling (DVS) is one of the most effective techniques in power-aware design because the energy consumption of a CMOS circuit has a superlinear dependency on the supply voltage, whereas an approximately linear relationship between voltage and delay.

Conventional DVS algorithms consider only processor energy, with a focus on optimizing dynamic power consumption. More recently, as leakage power is growing rapidly with each technical generation, researchers study the problem of reducing both the static and dynamic power of a processor [10] [15] [8]. In addition to processors, most applications use other system components during execution, such as memory, flash drives, and wireless interface. Most of the components support multiple power states, such as active, standby (active but idle), and shutdown. If the components need to be active during application execution, processor speed slowdown may lead to an increase of the standby time of other components. The power of those components in a standby state can be significant. For examples the standby power of memory and wireless interface can be as large as 0.4W and 1W in comparison with 2W CPU power [7]. Since the overall system energy consumption could be adversely affected by processor slowdown, it is necessary to consider resource standby power to compute energy efficient slowdown factors.

Existing studies on system-wide energy optimization focus on periodic tasks with complete timing information known in advance. Even for periodic tasks, it is still hard to get an accurate analysis on a processor with discrete voltage levels. Researchers have tried to solve a simplified version of the problem with relaxed timing requirement [3], apply heuristics to get approximated solution [7], assume a processor with continuous speed levels [17], and focused on the effect of non-preemptive shared resources on system-level energy conservation [2].

The first contribution of this paper is system-wide energy optimization solutions for periodic tasks on processors with limited number of speed scales. We prove that the minimization problem is an instance of the NP-hard Multiple-Choice 0-1 Knapsack Problem (MCKP) with non-integer coefficients. Existing solutions to MCKP assume integer coefficients, which is invalid in energy optimization. We develop a dynamic programming algorithm to solve the energy optimization problem in pseudo-polynomial time. As the worst case running time may grow rapidly with large number of tasks, we propose a fully polynomial time approximation scheme (FPTAS) whose worst case performance can be bounded by a predefined value. Experimental results show that the optimal solution leads to an energy consumption bound much lower than energy consumption of the heuristic by Jejurikar and Gupta [7]. The performance degradation of the approximation solution can be bounded by the predefined value with a much smaller average error.

Many typical real-time systems have both periodic and sporadic tasks. A sporadic task model is effective for applications with unpredictable arrival times and external events such as operator's commands. The irregular releases of sporadic tasks call for online decision making with all timing information known only after task releases. Studies of online sporadic tasks energy saving based on DVS were limited to reduction of dynamic energy consumption of processors, rather than system; see [16] for a recent review.

The second contribution of this paper is system-wide energy optimization solutions for online scheduling of sporadic tasks. The speed assignment is optimal in the sense that the decision is made online without assumptions about future task releases. We show

*This research was supported in part by U.S. NSF grants ACI-0203592, CCF-0611750, and NASA grant 03-OBPR-01-0049.

ICCAD'06, November 5-9, 2006, San Jose, CA

the problem is essentially a Multidimensional Multiple-choice 0-1 Knapsack Problem (MMKP). In general, there is neither pseudo-polynomial solutions nor FPTAS for an MMKP. In this paper, we present both a pseudo-polynomial algorithm and an FPTAS by exploiting inherent properties of sporadic task scheduling, in which the feasibility of a task does not depend on tasks finished later. Experimental results show that energy consumption of the optimal solution can improve upon a recently proposed competitive algorithm [16] by up to 57%. The approximation scheme offers a better trade-off between energy-efficiency and time complexity.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 provides power and task models of the system. We present the optimal and approximated solutions for periodic tasks and sporadic tasks in Section 4 and Section 5, respectively. We evaluate the performance of the algorithms in Section 6. Section 7 concludes the article.

2. RELATED WORK

Extensive studies on energy savings have been conducted for periodic tasks under DVS. An emphasis was on processor energy savings. System-wide energy expenditure considering standby power of various system components is a recent focus [3] [17] [2] [7].

Choi *et al.* proposed an interval based frequency setting policy that would minimize system power consumption of a program subject to a constraint on performance loss in terms of increased execution time [3]. Their approach can be best applied to applications with soft deadlines. Zhuo and Chakrabarti considered processors with continuous speed levels and proposed a static speed setting policy, and extensions to online slack distribution and preemption control [17]. Cheng and Goddard studied system-wide energy savings of periodic tasks with non-preemptive shared resources [2]. They considered the case when devices have non-zero transition delays and did not put devices in sleep in order to guarantee all jobs meet their deadlines.

A closely related work is due to Jejurikar and Gupta [7]. They considered periodic tasks on a processor with discrete speed levels and proposed a heuristic algorithm to the system-wide energy optimization problem. The algorithm starts by setting all tasks to their critical speeds and adjust s speed of a task that can lead to the minimum energy increase per unit time. The adjustment continues until a feasible schedule is found. Although the algorithm is more efficient than traditional DVS, it remains unclear how good the algorithm is and whether there exists a solution with more energy savings. In this paper, we answered these questions by proposing both an optimal solution and an approximation scheme with bounded performance degradation.

Furthermore, we adapt the optimal and approximated solutions to online scheduling of sporadic tasks. A general online DVS approach for sporadic tasks is to set processor to the lowest speed at which there is no deadline miss. For example, Zhong and Xu proposed an adaptive algorithm based on linear filter theories for sporadic tasks on processors with continuous speed levels [16]. The time-variant algorithm is able to find the lowest feasible speed without future task release knowledge. In this paper, we consider system-wide energy minimization for sporadic tasks on processors with a more practical discrete speed scale.

The connection between DVS for periodic tasks energy optimization and an MCKP was initially established by Mejía-Alvarez, Levner, and Mossé [11]. They modeled the problem as an MCKP, transformed it to a standard knapsack problem, and adapted a heuristic algorithm to solve the problem. Chen, Kuo, and Shih formulated energy minimization as a Subset Sum Problem, a special case of knapsack problem [1]. They then proposed an approximation solution with performance guarantee. Rakhmatov and Vrudhula presented an MCKP formulation for energy minimization with focus on the impact of battery recovery and discharging rate [12].

We note that all related formulations targeted at dynamic energy reduction of a processor. System-wide energy optimization involves other system components in addition to a CPU. Their standby states make the lowest feasible speed not necessarily energy efficient. In this paper, we prove that the energy optimization for periodic tasks is an MCKP and develop both optimal and approximated solutions. In addition, existing studies for system energy optimization are limited to periodic tasks. We make one step forward by connecting online DVS for sporadic tasks to an MMKP.

We note that there exist studies that model intra-task DVS, which has many scaling points during the execution of a task, as special cases of Subset Sum [14] and knapsack problems [13]. In contrast, we consider inter-task DVS due to its ease of implementation and less number of voltage switching.

3. PRELIMINARIES

3.1 Task Model

We study two types of hard real-time tasks, periodic and sporadic tasks. We assume all tasks to be independent and preemptive, scheduled by the Earliest Deadline First (EDF) policy. Consider n periodic tasks in a uniprocessor system. Task i is characterized by a tuple $\{C_i, T_i, D_i\}$, where T_i denotes task period and C_i is the execution time under the maximum frequency. The relative deadline D_i is assumed to be equal to task period T_i. To guarantee the task set is schedulable, we assume the cumulative utilization does not exceed one, i.e., $\sum_{i=1}^{n} C_i/T_i \leq 1$.

Another type of hard real-time task under consideration is sporadic tasks. Each sporadic task, denoted by a tuple $\{A_i, C_i, D_i\}$, is characterized by its release time A_i, execution time C_i, and deadline D_i. The arrival time of a sporadic task can be arbitrary with irregular intervals. We therefore do not assume knowledge of future task release at the time of speed assignment and parameters of a sporadic task are known only after its release.

3.2 System Energy Model

We consider a computing system with a DVS processor. The processor can scale its supply voltage and clock speed with m discrete levels within its operational ranges, $[f_{min}, f_{max}]$. We define a slowdown factor S as the normalized clock speed with respect to the maximum frequency. That is, $S = f/f_{max}$, $S_{min} \leq S \leq 1$ where $S_{min} = f_{min}/f_{max}$. Note that whenever the speed of a processor is scaled, its supply voltage is scaled according to a roughly linear relation. We assume negligible voltage switching overhead.

Let S_i denote the slowdown factor of task i. Its actual execution time is C_i/S_i and the scaled utilization is $\frac{C_i}{T_i S_i}$. As S_i can be set to any clock rate, we use S_{ij} to denote task i being executed at the jth speed. Similarly, we use u_{ij} and e_{ij} to denote scaled utilization and energy consumption of task i under speed j.

Let $P_{cpu}(S)$ denote the power consumption of the processor at a slowdown factor S. It includes both a dynamic power and a static (leakage) power consumption. Considering energy consumption per cycle, recent studies concluded the existence of *critical speed* [8] that minimizes processor energy. Beyond the speed, static energy becomes dominant and processor slowdown is no longer energy efficient. We denote the slowdown factor of the processor critical speed as S_{crit_p}.

In addition to a processor, the system under consideration consists of other resources. Power consumption of a resource includes both active power and standby power. It is assumed that active and

standby power of resources are independent of processor speed. If a resource is not used by an active task, it is shut down for energy saving with zero power. Task i requires a subset $\Re_i$ of the resources for execution. Standby energy of resource j for task i, denoted by E_{ij}^{std}, is equal to its standby power, denoted by P_{ij}^{std}, multiplied by task execution time, i.e., $E_{ij}^{std} = P_{ij}^{std} \cdot C_i/S_i$. Similarly, active energy of task i, denoted by E_i^{act}, is resource active power multiplied by access time. We assume access time remains constant with respect to S_i. As a result, the active energy consumption does not change with S_i. The system energy for task i under a slowdown factor S_i becomes

$$E_i(S_i) = P_{cpu}(S_i) \cdot \frac{C_i}{S_i} + \sum_{j \in \Re_i} P_{ij}^{std} \cdot \frac{C_i}{S_i} + E_i^{act}. \quad (1)$$

Similar system-level energy models have been defined in [17] [3] [7], as well.

In a processor with a limited number of speed levels, the critical speed of task i is the one that minimizes $E_i(S_i)$. We denote it as S_{crit_i}. Due to the effect of resource standby power, S_{crit_i} is no smaller than S_{crit_p}. The minimum slowdown factor of task i, denoted by S_{min_i}, is the larger value of the slowdown under minimum processor speed, S_{min}, and the critical slowdown factor, S_{crit_i}. That is $S_{min\_i} = \max\{S_{min}, S_{crit\_i}\}$. The set of speeds for task i, represented by M_i, is a subset of available slowdown factors from S_{min_i} to 1. Its cardinality m_i can be much smaller than the possible number of processor speeds m if the task has a large critical speed (or slowdown factor equivalently).

4. PERIODIC TASKS

4.1 Problem Formulation

For a periodic task set, we consider a hyper-period T, which is the Least Common Multiple of T_1, ..., T_n. As the schedule repeats every T time units, our objective is to minimize the overall energy consumption during the hyper-period. We formulate the optimal voltage scheduling problem as

$$\text{minimize} \sum_{i=1}^{n} \frac{T}{T_i} E_i(S_i) \quad (2)$$

$$\text{subject to} \sum_{i=1}^{n} \frac{C_i}{T_i S_i} \leq 1 \quad (3)$$

$$S_{min\_i} \leq S_i \leq 1,\ 1 \leq i \leq n. \quad (4)$$

The constraint (3) ensures the task set under EDF scheduling is feasible. We show in Theorem 4.1 that the optimization problem is NP-hard.

THEOREM 4.1. *The energy-minimization problem for periodic tasks defined by* (2)-(4) *is NP-hard.*

Proof: The theorem can be proved by a reduction from the Multiple-Choice Knapsack Problem (MCKP) [9] with 0-1 variables, which is NP-hard. Details are omitted. □

4.2 Optimal Solution

Studies showed that MCKP can be solved optimally in pseudo-polynomial time using dynamic programming [4]. Its optimal solution could be achieved if all its coefficients are integers. To apply this solution to the energy optimization problem with non-integer coefficients, a straight-forward approach is to scale and round the energy or utilization values to integers. However, it is only an approximation and it remains unclear to what extend we should scale the values. In the following, we develop a dynamic programming algorithm for the energy optimization problem with multiple choices of CPU speed.

Consider a pair of two values $(\bar{u}_{ik}, \bar{e}_{ik})$ as a state, where $\bar{e}_{ik}$ denotes the energy sum of the first i tasks corresponding to the accumulated utilization $\bar{u}_{ik}$. All states of task i form a list

$$L_i = \langle (\bar{u}_{i1}, \bar{e}_{i1}), (\bar{u}_{i2}, \bar{e}_{i2}), ..., (\bar{u}_{in_i}, \bar{e}_{in_i}) \rangle,$$

where n_i is the number of states after task i is enumerated. Initially, we have list L_0 with zero energy and utilization values. List L_i is obtained in four steps, as shown in Algorithm 1. We first get a number of lists L'_{ij} by adding utilizations and energy values of task i under each speed $j \in M_i$ to states in list L_{i-1}. We denote the componentwise addition as $L'_{ij} = L_{i-1} \oplus (u_{ij}, e_{ij})$. Secondly, we merge the lists L'_{ij} into one list in non-decreasing order of energy sums. In the third step, we eliminate all non-feasible states in line 7. A state will not result in a feasible solution if the accumulative utilization of the partial solution and the minimum utilization (under the maximum speed) of the remaining tasks exceeds one. Finally, we prune the list by applying the following dominance criterion.

Dominance Criterion: If two states $(\bar{u}_{ij}, \bar{e}_{ij})$,$(\bar{u}_{ik}, \bar{e}_{ik})$ in list L_i satisfy $\bar{u}_{ij} > \bar{u}_{ik}$ and $\bar{e}_{ij} \geq \bar{e}_{ik}$, or $\bar{u}_{ij} \geq \bar{u}_{ik}$ and $\bar{e}_{ij} > \bar{e}_{ik}$, then the former state is said to be dominated by the latter.

A dominated state will not enter the optimal solution and can be removed from the list. Intuitively, if a state uses more energy and requires larger utilization than another state, it can always be replaced by the latter one in each iteration. The smallest state of a list is referred as the state with the smallest energy sum. The procedure Prune-Lists() finds the list of undominated states in L' and returns it as L''. Denote the smallest states in L' and L'' as $(\bar{u}', \bar{e}')$ and $(\bar{u}'', \bar{e}'')$, respectively. As we consider the states in non-decreasing order of energy sums, we have $\bar{e}' \geq \bar{e}''$. If $\bar{u}' < \bar{u}''$, neither state is dominated by the other and we add state $(\bar{u}', \bar{e}')$ to the end of list L''. If $\bar{u}' \geq \bar{u}''$, state $(\bar{u}', \bar{e}')$ is dominated by $(\bar{u}'', \bar{e}'')$ or the two states have the same energy utilization values. We skip the former state.

Algorithm 1 Energy minimization for periodic tasks using dynamic programming.

```
1: L_0 = ⟨(0, 0)⟩
2: for i = 1 to n do
3:     for all speeds j ∈ M_i do
4:         L'_ij = L_{i-1} ⊕ (u_ij, e_ij)
5:     end for
6:     merge L'_ij into a list L'_i in non-decreasing order of energy
7:     delete all states in L'_i with ū + Σ_{k=i+1}^{n} C_k/T_k > 1
8:     L_i = Prune-Lists(L'_i)
9: end for
10: return the smallest state in L_n
  procedure PRUNE-LISTS(L')
      add the state (0, ∞) to the end of list L'
      L'' = ∅
      repeat
          choose and delete the smallest state (ū', ē') from L'
          if (ē' < ∞) then
              let (ū'', ē'') be the smallest state in L'', ū'' = ∞ if L'' == ∅
              if (ū' < ū'') then
                  add (ū', ē') to the end of list L''
              end if
          end if
      until ē' = ∞
      return L''
  end procedure
```

Consider the running time of the ith iteration adding task i. Lines 3-5 are linear in the number of states in L'_i, $|L'_i|$, which is equal to

the multiplication of $|L_{i-1}|$ and the number of speeds for task i. Since L'_{ij} are sorted lists, the running time of the merging of the lists into one sorted list will multiply the sum of the lengths of the lists, $|L'_i|$, by the time to choose the lowest energy value. As the minimum energy value can be determined in no more than m_i comparisons, line 6 can be completed in $O(m_i|L'_i)|$. Lines 7 and 8 can be completed in $O(|L'_i|)$. As a result, the running time of adding task i is $O(m_i|L'_i|)$. Let the number of states in L'_i be bounded by U, i.e., $\max_{1\leq i\leq n}|L'_i| \leq U$. An upper bound of the running time adding n tasks is $O(\sum_{i=1}^{n} m_i U)$. Similarly, the necessary memory requirement of the algorithm is bounded by the number of states in a list. As we only need to keep the latest list in Algorithm 1, the space requirement would be in the order of $O(U)$.

Note that Algorithm 1 only gives the minimum energy value but does not explicitly return the corresponding optimal speed settings. We can get the speed setting by extending the state to a tuple of three values $(\bar{u}_{ik}, \bar{e}_{ik}, j)$, with j as the speed index of task i. The optimal speed settings is constructed by backtracking the n lists. Starting from task n, we find the speed with accumulative utilization closest to one, update utilization under the assigned speed, and repeat the procedure for the remaining tasks. A simple upper bound of the running time of backtracking is $O(nU)$, which does not change the order of the overall time complexity. However, we need to keep lists L_i of all iterations, which would lead to a space complexity as $O(nU)$. In short, the time and space complexities in getting the optimal speed settings are $O(\sum_{i=1}^{n} m_i U)$ and $O(nU)$. We include modifications for backtracking the speed settings in Algorithm 2.

4.3 An Approximation Scheme

States in Algorithm 1 can grow rapidly and it might be computationally expensive to get the optimal solution with a large number of tasks and speed levels. Practically, it is often not necessary to find the optimal solution with limited time and resources. An approximation algorithm is more desirable if it can be finished in reasonable time and can provide a nearly optimal performance, especially with a worst case performance bound. The quality of approximation is given by a relative performance ratio. An algorithm is said to be an r-approximation scheme if for a given value of $r \in (0, 1)$, we have $(E^A - E^*)/E^* \leq r$, where E^A and E^* are the energy consumption of the approximated and the optimal solutions. A desirable approximation scheme should have a running time which increases with polynomial time in both the number of tasks and the performance ratio. This leads to a classification of schemes called fully polynomial time approximation scheme (FPTAS) [9]. We will propose an FPTAS for the energy optimization problem.

The approximation scheme works by reducing the number of states in each iteration. We use E_{min} and E_{max} to denote the lower and upper bounds of energy consumption of n tasks, when all tasks run at their minimum and the maximum speeds respectively. The basic idea is to divide the energy values into a number of groups each of length $\bar{\epsilon}$. We will show how to determine the constant value $\bar{\epsilon}$ later. Each energy value will fall into one of the group. The scaled energy for task i under speed j and group size $\bar{\epsilon}$, denoted as $e_{ij}(\bar{\epsilon})$, is then rounded up to the next integer, i.e., $e_{ij}(\bar{\epsilon}) = \lceil \frac{e_{ij}}{\bar{\epsilon}} \rceil$, which will be used to represent the energy values in each group. As a result of the rounding up, the number of states can be reduced greatly so are the running time and space required to solve the scaled problem.

Let $s_i^{\bar{\epsilon}}$ be the optimal speed index of task i for the scaled problem, which is not necessarily the same as the optimal speed settings of the original problem s_i^*. Denote energy consumption of the scaled problem with group length $\bar{\epsilon}$ as $E^{\bar{\epsilon}}$. We have

$$E^{\bar{\epsilon}} = \sum_{i=1}^{n} e_{i,s_i^{\bar{\epsilon}}} \leq \sum_{i=1}^{n} \bar{\epsilon}\lceil \frac{e_{i,s_i^{\bar{\epsilon}}}}{\bar{\epsilon}} \rceil \leq \sum_{i=1}^{n} \bar{\epsilon}\lceil \frac{e_{i,s_i^*}}{\bar{\epsilon}} \rceil$$
$$\leq \sum_{i=1}^{n} \bar{\epsilon}(\frac{e_{i,s_i^*}}{\bar{\epsilon}} + 1) = \sum_{i=1}^{n}(e_{i,s_i^*} + \bar{\epsilon}) = E^* + n\bar{\epsilon}. \quad (5)$$

The second inequality is due to the fact that the minimum energy value of the scaled problem is no larger than the energy under the speed settings of the original optimal solution.

According to (5), the absolute error of the approximated solution is at most $n\bar{\epsilon}$. To bound the relative error below a given ratio r, we must have

$$\frac{E^{\bar{\epsilon}} - E^*}{E^*} \leq \frac{n\bar{\epsilon}}{E^*} \leq r.$$

Solving the inequality for $\bar{\epsilon}$, we get

$$\bar{\epsilon} \leq \frac{rE^*}{n}. \quad (6)$$

The right-hand side of (6) can be bounded by rE_{min}/n. Hence, choosing the constant $\bar{\epsilon}$ according to rE_{min}/n will satisfy the condition (6) and bound the relative performance error under r. We list an outline of the approximation in Algorithm 2.

Algorithm 2 A fully polynomial time approximation scheme for periodic tasks.

```
1: L_0 = ⟨(0, 0, 0)⟩
2: calculate E_min and set ε̄ = rE_min/n
3: for i = 1 to n do
4:     for all speeds j ∈ M_i do
5:         L'_ij = L_{i-1} ⊕ (u_ij, ⌈e_ij/ε̄⌉, j)
6:     end for
7:     merge L'_ij into a list L'_i in non-decreasing order of energy
8:     delete all states in L'_i with ū + Σ_{k=i+1}^{n} C_k/T_k > 1
9:     L_i = Prune-Lists(L'_i)
10: end for
11: backtrack the lists from L_n to L_1 to get the speed settings S^ε̄
12: find the smallest state in L_n, denoted as (ū, ē, j)
13: return speed settings S^ε̄ and state (ū, ē, j)
```

With chosen group length $\bar{\epsilon}$, we can determine the total number of undominated states by

$$U = \frac{E_{max} - E_{min}}{\bar{\epsilon}} = (\frac{E_{max}}{E_{min}} - 1)\frac{n}{r} = \frac{\gamma n}{r},$$

where $\gamma = E_{max}/E_{min} - 1$. Replacing U in the running time of the optimal solution, we get a time complexity of $O(\sum_{i=1}^{n} m_i \frac{\gamma n}{r})$, which is polynomial in the number of tasks, speed levels, and $1/r$. The approximated solution in Algorithm 2 is an FPTAS for the energy optimization problem for periodic tasks.

5. SPORADIC TASKS

5.1 Problem Formulation

Due to the irregular release times of sporadic tasks, we consider online scheduling with task timing parameters known only after task releases. When there is a new task release, we first determine whether the task should be admitted by an acceptance test as suggested in [5]. Suppose there are n ready tasks sorted in non-decreasing order of deadlines. Let $\bar{C}_i$, $\bar{D}_i$ denote the remaining execution time and deadline of task i. We define the maximum instantaneous utilization of all tasks U_{max} as $U_i = \sum_{j=1}^{i} \bar{C}_j/\bar{D}_j$

and $U_{max} = \max_{1 \le i \le n} U_i$. The task will not be admitted if $U_{max} > 1$.

Once the task is accepted, we will determine a speed assignment according to all ready tasks with the objective to minimizing energy consumption of all the tasks. For simplicity, we will consider a set of tasks released at time zero. In the general case for tasks released at different times, we can easily adapt the formulation by considering only ready tasks and by changing C_i, D_i to residue execution times and deadlines. The optimal speed assignment can be formulated as

$$\text{minimize} \sum_{i=1}^{n} E_i(S_i) \tag{7}$$

$$\text{subject to} \sum_{i=1}^{k} \frac{C_i}{S_i} \le D_k, 1 \le k \le n \tag{8}$$

$$S_{min\_i} \le S_i \le 1,\ 1 \le i \le n. \tag{9}$$

The set of constraints in (8) ensures that the schedule is feasible for all tasks. The problem is also NP-hard, as shown in Theorem 5.1.

THEOREM 5.1. *The energy-minimization problem for sporadic tasks defined by* (7)-(9) *is NP-hard in the strong sense.*

Proof: The theorem can be proved by a reduction from the Multidimensional Multiple-Choice Knapsack Problem (MMKP) with 0-1 variables, which is NP-hard in the strong sense. □

5.2 Optimal and Approximated Solutions

Theorem 5.1 implies that any optimal solutions to (7)-(9) will lead to strictly exponential computational times and there is no FPTAS. However, we will show that by investigating inherent properties of the energy minimization problem, we are able to find a pseudo-polynomial solution and an FPTAS with bounded performance degradation in moderate running time.

We assume the tasks have been sorted in non-decreasing order of their deadlines. Task k will finish before its deadline as long as the execution time sum of the first k tasks does not exceed D_k; it is not necessary to consider tasks completed later. This enables us to satisfy the constraints (8) one at each iteration. For example, after task i is added and we have a pruned list L_i, we find all feasible states in L_i satisfying the constraints of task i and its feasibility remains satisfied in later iterations. Therefore, after all the iterations, we get a list of states satisfying all the n constraints.

Algorithm 3 lists principles of sporadic tasks speed assignment extended from Algorithm 1. We use (t_{ij}, e_{ij}, j) to characterize states in a list, where t_{ij} is the execution time of task i under speed j. The presented algorithm has a worse case complexity $O(\sum_{i=1}^{n} m_i U)$ in running time and $O(nU)$ in space.

Algorithm 3 Energy minimization for sporadic tasks using dynamic programming.

```
1: L_0 = ⟨(0, 0, 0)⟩
2: for i = 1 to n do
3:     for all speeds j ∈ M_i do
4:         L'_ij = L_{i-1} ⊕ (t_ij, e_ij, j)
5:     end for
6:     merge L'_ij into a list L'_i in non-decreasing order of energy
7:     delete all states in L'_i with t̄ > D_i
8:     L_i = Prune-Lists(L'_i)
9: end for
10: backtrack the lists from L_n to L_1 to get the speed settings S*
11: return S* and the smallest state in L_n
```

Note that the speed assignment is optimal only in the sense that it minimizes the energy consumption in executing all tasks without assumption about future task releases. The energy consumption could be further reduced with more knowledge about future task release times, which would lead to an offline algorithm.

It is not difficult to see that Algorithm 3 can be extended to an FPTAS in a similar way to the approximation technique in Section 4.3. We omit the details for brevity.

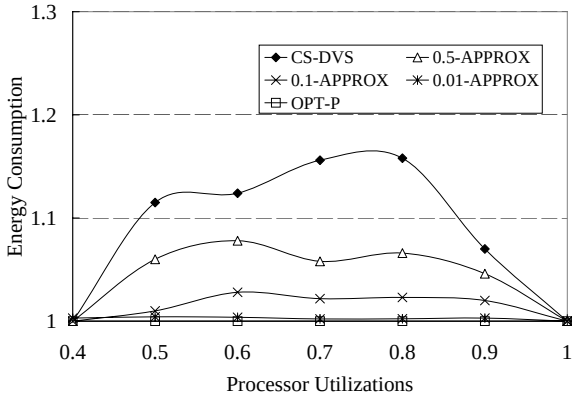

Figure 1: System energy consumption for periodic tasks.

6. EXPERIMENTAL RESULTS

In this section, we evaluate the effectiveness of the proposed algorithms in energy savings for both periodic and sporadic tasks. We consider a DVS processor with five processor speed levels similar to the Intel's XScale [6] 150MHz, 400MHz, 600MHZ, 800MHz, and 1000MHz. Power consumption is 80mW, 170mW, 400mW, 900mW, and 1600mW respectively. We assume negligible voltage switching overhead and CPU idle power. In addition to the processor, the system has three other resources with standby power consumption of 0.2W, 0.4W, and 1W, labeled in integers from 1 to 3. These are typical values for memory, flash drives, and wireless interface. Standby time for the resources as a percentage of task execution time is assumed to be in the range of [20%, 60%], [10%, 25%], and [5%, 20%], respectively. The values of standby power and time periods were based on the experimental setting of [7].

6.1 Periodic Tasks

We first evaluate the effectiveness of the proposed algorithms for periodic tasks. Tasks were assigned a random period T_i in the range [10ms, 120ms]. We varied the processor utilization from 0.1 to 1 in a step 0.1. For each utilization value, we generated 20 task sets, each containing 5 tasks. Utilization of each task, U_i, was randomly chosen with the total utilization. Execution time of each task at the maximum processor speed was set to $U_i \cdot T_i$. We assume the tasks have resource requirement $\emptyset$, $\{1\}$, $\{1, 2\}$, $\{1, 3\}$, and $\{1, 2, 3\}$.

We compare the proposed scheduling policy, referred to as OPT-P, with the heuristic algorithm CS-DVS [7] in Fig. 1. As critical slowdown factors of all tasks are no lower than 0.4, it is not energy-efficient to set CPU below the speed. Both polices set the processor speed to 0.4 for utilizations up to 40% by working at the critical speed. We have omitted those values to improve readability. The reported energy consumption is normalized with respect to OPT-P. For all utilization values, CS-DVS consumes up to 16% more energy than the optimal solution. The difference can be in part explained by the fact that CS-DVS terminates as soon as it finds a feasible solution. It may end up with a total processor utilization far less than 1. Another observation is that the approximation schemes can bound the performance degradation well below their worst case values. For example, with a given ratio 0.1, the average error relative to the optimal solution is no larger than 3%. A small ratio such

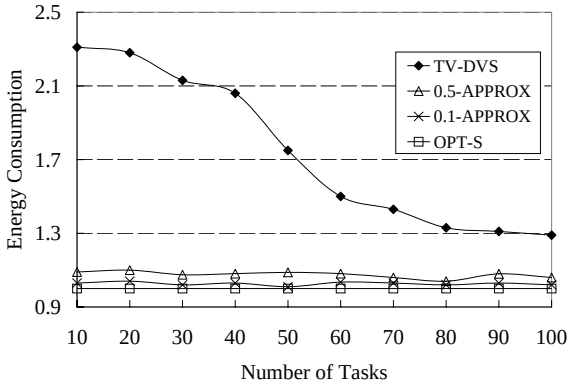

Figure 2: System energy consumption for sporadic tasks.

as 0.01 can provide a nearly optimal solution with a polynomial complexity.

6.2 Sporadic Tasks

We evaluated the proposed online scheduling algorithm for sporadic tasks in comparison with a recent time-variant DVS algorithm (TV-DVS) for online sporadic tasks [16]. Since the competitor is based on a continuous speed level, we rounded computed speed up to its nearest neighbor. We generated a group of sporadic tasks with 10 ms minimal interarrival time. The maximum utilization of each task was set randomly under the constraint that the accumulative utilization of all tasks at their maximum release rates does not exceed one. Task deadlines were drawn from [10ms, 120ms]. Execution time of a task is a multiplication of its maximum utilization and deadline. Interarrival times of a task were chosen from an exponential distribution with averages ranging from 20ms to 50ms. Each task was randomly assigned a subset of the available resources.

Fig. 2 shows the results due to TV-DVS, the proposed algorithm (OPT-S), and the approximation algorithms for a 10 minutes run. It is expected that TV-DVS consumes much more energy without considering the impact of critical speed. When the number of tasks is small, energy consumption of TV-DVS can be as large as 2.3 times that of OPT-S. The performance difference is mainly due to the inefficiency of TV-DVS in operating on speeds lower than critical speeds. As the number of tasks increases, system utilization increases so that TV-DVS improves its performance by running tasks at higher speeds. However, as TV-DVS assumes a processor with continuous speed levels and assumes identical power characteristics of all tasks, the performance gap can still be as large as 30%.

7. CONCLUSION

We have presented solutions to system-wide energy optimization for periodic real-time tasks. The energy minimization is proved to be NP-hard. We develop a pseudo-polynomial dynamic programming solution in getting the optimal solution. A fully polynomial time approximation scheme (FPTAS) is proposed to provide bounded performance guarantee with moderate running time even for large problem size. In addition to periodic tasks, we show that energy minimization for sporadic tasks is NP-hard in the strong sense, which is known to have strictly exponential running time in getting the optimal solution. However, we exploit inherent properties of the problem and show that we can have a pseudo-polynomial algorithm for exact solutions and an FPTAS with bounded performance guarantee. The exact solution is optimal in the sense it is online without any assumption about future task releases at the time of speed assignment decision. Evaluation results in comparison with existing approaches for periodic tasks [7], sporadic tasks [16] show superiority of the proposed algorithms in energy-efficiency and the effectiveness of the approximation schemes in providing bounded performance guarantee.

We finally point out that system-wide energy saving techniques on task procrastination in extending shutdown intervals [7], preemption control [17], and non-preemptive resources [2] can be incorporated into the policy. These issues are important yet beyond the scope of this paper.

References

[1] J.-J. Chen, T.-W. Kuo, and C.-S. Shih. $(1+\epsilon)$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *Proc. of EMSOFT*, 2005.

[2] H. Cheng and S. Goddard. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *Proc. of PARC*, 2005.

[3] K. Choi, W. Lee, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *Proc. of ICCAD*, 2004.

[4] K. Dudzinski and S. Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Oper. Research*, 28:3–21, 1987.

[5] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage voltage processor. In *Proc. of ICCAD*, 1998.

[6] Intel-XScale Microarchitecture (http://www.intel.com).

[7] R. Jejurikar and R. K. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proc. of ISLPED*, 2004.

[8] R. Jejurikar, C. Pereira, and R. K. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proc. of DAC*, 2004.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004.

[10] S. M. Martin, K. Flautner, T. N. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. of ICCAD*, 2002.

[11] P. Mejía-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embedded Comput. Syst.*, 3(2):284–306, 2004.

[12] D. N. Rakhmatov and S. B. K. Vrudhula. Energy management for battery-powered embedded systems. *ACM Trans. Embedded Comput. Syst.*, 2(3):277–324, 2003.

[13] F. Xie, M. Martonosi, and S. Malik. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Proc. of ISLPED*, 2005.

[14] R. Xu, C. Xi, R. G. Melhem, and D. Mossé. Practical pace for embedded systems. In *Proc. of EMSOFT*, 2004.

[15] L. Yan, J. Luo, and N. K. Jha. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. In *Proc. of ICCAD*, 2003.

[16] X. Zhong and C.-Z. Xu. Energy aware modeling and scheduling of real-time tasks for dynamic voltage scaling. In *Proc. of RTSS*, 2005.

[17] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proc. of DAC*, 2005.