PreDVS: Preemptive Dynamic Voltage Scaling for Real-time Systems using Approximation Scheme[*]

Weixun Wang and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA
{wewang,prabhat}@cise.ufl.edu

ABSTRACT

System optimization techniques based on dynamic voltage scaling (DVS) are widely used with the aim of reducing processor energy consumption. Inter-task DVS assigns the same voltage level to all the instances of each task. Its intra-task counterpart exploits more energy savings by assigning multiple voltage levels within each task. In this paper, we propose a voltage scaling technique, named PreDVS, which assigns voltage levels based on the task set's preemptive scheduling for hard real-time systems. Our approach is based on an approximation scheme hence can guarantee to generate solutions within a specified quality bound (e.g., within 1% of the optimal) and is different from any existing inter- or intra-task DVS techniques. PreDVS exploits static time slack at a finer granularity and achieves more energy saving than inter-task scaling without introducing any extra voltage switching overhead. Moreover, it can be efficiently employed together with existing intra-task scaling techniques. Experimental results demonstrate that PreDVS can significantly reduce energy consumption and outperform the optimal inter-task voltage scaling techniques by up to 24%.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time systems and embedded systems

General Terms

Algorithm, Design

Keywords

Real-time systems, energy-aware scheduling, dynamic voltage scaling, approximation algorithm

1. INTRODUCTION

Energy conservation has been the main concern in embedded system optimization for a long time since these systems are generally limited by battery lifetime. Real-time embedded systems require unique design considerations since task execution must meet their deadlines in order to ensure correct system behavior. A task set is said to be *schedulable* if all the tasks can finish execution within their time constraints. Periodic task set is of major interest since it is most popular in typical real-time systems. A task in such systems arrives in regular intervals and normally has a deadline equal to its period. Earliest Deadline First (EDF) [1] and Rate

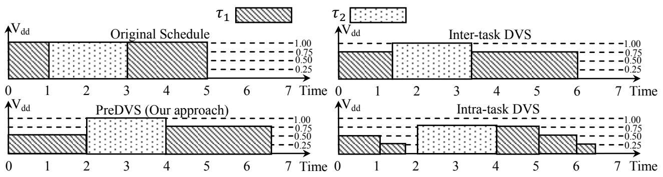


Figure 1: Inter-task DVS, PreDVS and Intra-task DVS.

Monotonic (RM) [2] are two most commonly used scheduling algorithms in real-time systems. EDF employs a dynamic priority scheme and has been proved to be optimal with an utilization bound of 1 [2]. Dynamic voltage scaling (DVS) [3] is widely acknowledged as one of the most effective processor energy saving technique. The reason behind its capability to save energy is that linear reduction in the supply voltage leads to approximately linear slow down of performance while the power can be decreased quadratically. Dynamic cache reconfiguration techniques are recently proposed for real-time embedded systems to reduce cache hierarchy's energy consumption [4] [5] [6].

In this paper, we propose a novel voltage scaling technique which generates voltage assignment based on the preemptive schedule of the target task set. Unlike heuristics which may generate inferior results in certain cases, we develop a fully polynomial approximation scheme which can guarantee to give solutions within specified quality bounds. Our approach, named PreDVS, differs from inter-task and intra-task DVS as illustrated in Figure 1. Specifically, PreDVS differs from existing inter-task scaling techniques [7] [8] [9] [10] in that we adjust processor voltage level multiple times throughout each task's execution to potentially achieve more energy savings. For example, in Figure 1, if the deadline of task τ_1 is 7, inter-task DVS cannot further lower any task's voltage level otherwise deadline will be missed since it requires reduction of voltage for all task instances. However, PreDVS is able to further reduce the energy consumption by lowering the voltage level for the first segment of τ_1. PreDVS differs from existing intra-task DVS techniques [11] [12] [13] in the following ways. Existing intra-task scaling methods assume that static slack allocation has already been done. They only consider one task instance (local optimization) and focus on exploiting dynamic time slacks generated at runtime due to early finished task execution. They require excessive analysis, runtime tracking and modification of the task source code, which is not always feasible in reality. Furthermore, they normally result in large number of additional voltage switching points and most of them assume continuous voltage levels. Our approach aims at static slack exploitation and is carried out during design time. In fact, our technique is complementary to existing intra-task DVS techniques. Any intra-task scaling can be applied after PreDVS to further reduce energy consumption at runtime. Our technique considers the problem globally and find a voltage assignment for all tasks so that the total energy consumption can be minimized while no task deadline is violated. Since our approach focuses on static slack exploitation, we assume every task takes its worst-case execution time to complete. We focus on hard real-time systems with preemptive periodic task sets in this work. The system can be executed on any processor with discrete voltage/frequency levels.

The rest of the paper is organized as follows. First, we for-

[*]This work was partially supported by NSF grant CCF-0903430.

mulate the problem in Section 2. Next, we describe our problem transformation scheme and an pseudo-polynomial algorithm which can give optimal solutions in Section 3. We then propose our fully polynomial-time approximation scheme. Section 4 presents our experimental results. Finally, Section 5 concludes the paper.

2. PROBLEM FORMULATION

In this section, we formulate our problem, prove its NP-hardness and then discuss its unique difficulties. We are given:

- A set of m independent periodic tasks $T\{\tau_1,\tau_2, ... ,\tau_m\}$.
- Each task $\tau_i \in T$ has known period p_i, deadline d_i and worst-case execution cycles (WCEC) c_i.
- A voltage scalable processor which supports l different voltage levels $V\{v_1,v_2, ... ,v_l\}$.
- Task $\tau_i \in T$ has energy consumption e_i^k and execution time t_i^k at processor voltage $v_k \in V$ in the worst-case.

Note that we use WCEC c_i here to reflect the worst-case workload of each task since it is independent of processor frequency level. e_i^k and t_i^k can be computed based on the underlying processor energy model. We assume that each task is released at the beginning of every period and the relative deadline is equal to the period. We prove that our voltage scaling problem is NP-hard by considering a **simplified version** of our problem – inter-task scaling – in which each task $\tau_i \in T$ is uniquely assigned a fixed voltage level throughout all its jobs. The system schedulability can be guaranteed by restricting the total utilization rate of task set under the scheduler's bound U. Note that it is sufficient to consider the task scheduling over its hyper-period P (equal to the least common multiple of all tasks' periods) since periodic task set has repetitive execution pattern during every P. To be more specific, the simplified problem can be stated as:

$$min(E = \sum_{i=1}^{m}\sum_{k=1}^{l}\frac{P}{p_i}\cdot x_i^k \cdot e_i^k) \tag{1}$$

subject to,

$$\sum_{i=1}^{m}\sum_{k=1}^{l} x_i^k \cdot \frac{t_i^k}{p_i} \leqslant U \quad ; \quad \forall i \sum_{k=1}^{l} x_i^k = 1 \tag{2}$$

In Equation (1), x_i^k is a 0/1 variable which denotes whether task τ_i is assigned with voltage level v_k. Equation (2) shows the sufficient condition of schedulability which must be satisfied and the assumption that only one voltage level is assigned to each task. According to [10], the inter-task scaling problem as formulated above is NP-hard.

Now let's switch back to our **original problem**. By assigning multiple voltage levels at different places throughout each task's execution, more energy savings can be achieved since we have more flexibility during decision making. Clearly, it is not feasible to consider all possible positions. Task preemption, which creates multiple segments of a single job, provides natural opportunities to assign different voltage levels to each task. In this paper, we examine the EDF schedule[1] when the system is executed without DVS and change the processor frequency whenever a job starts execution or resumes after being preempted. Since inter-task scaling techniques also have to perform voltage switching in all these occasions, our strategy does not introduce any additional runtime overhead. It is important since voltage scaling overhead can have negative impact on overall energy consumption and performance [11] [13]. Note that PreDVS leads to distinct energy consumption and execution time for each task's different jobs. As the simplified version has been shown to be NP-hard, the original problem is NP-hard as well.

[1]Our approach is also applicable with RM scheduling but is not discussed in this paper due to page limitation.

3. APPROXIMATION SCHEME

Since PreDVS problem has shown to be NP-hard, the best option is to devise an efficient method that can lead to approximate-optimal solutions. As described in Section 2, the original problem essentially adds another dimension (voltage/frequency selection for a task's each segment) to the simplified version. This fact prevents us from solving the problem directly by adapting approximation algorithms for MCKP or inter-task DVS [10]. In this paper, we make two primary contributions. First, we develop a problem transformation scheme that can eliminate this complexity. Next, we propose an fully polynomial-time approximation algorithm which can efficiently solve the problem.

3.1 Problem Transformation

This section describes four important steps of our problem transformation scheme.

Step 1: As in traditional systems without a voltage scalable processor, all the tasks are executed at a fixed frequency. We use the case in which the processor is running solely at the highest voltage as the baseline and further assume that the given task set is schedulable in this case otherwise applying DVS is not meaningful. We simulate and generate an EDF schedule of the task set on the target system. Each task is set to take its WCEC c_i to finish. During simulation, we let the scheduler generate the distinct block list and distinct block set list of each task, which are defined as:

DEFINITION 3.0.1. *A **distinct block** is an execution segment of a task with a distinct pair of start and end point.*

DEFINITION 3.0.2. *A **distinct block set** is a set of distinct blocks which compose a whole job of a task. Every distinct block set has a different set of distinct block(s) from another.*

Let b_i^j and s_i^j denote the j^{th} distinct block and distinct block set of task τ_i, respectively. Figure 2 illustrates these two terms. In this example, we consider three tasks: $\tau_1\{3,3,1\}$[2], $\tau_2\{5,5,2\}$ and $\tau_3\{12,12,4\}$. For task τ_1, it has only one distinct block b_1^1 that is of its entirety, which forms its only distinct block set $s_1^1 = \{b_1^1\}$. Task τ_2, however, has three distinct blocks: b_2^1 appears from time 1 to 3, which is of its entirety; b_2^2 appears from time 5 to 6, which is its first half; b_2^3 appears from time 7 to 8, which is its second half. Therefore, τ_2 has two distinct block sets: $s_2^1 = \{b_2^1\}$ and $s_2^2 = \{b_2^2, b_2^3\}$. Task τ_3, during its first period, experiences two preemptions which result in three distinct blocks: b_3^1 which is its first quarter, b_3^2 which is its second quarter and b_3^3 which is the rest half. These three distinct blocks compose a distinct block set $s_3^1 = \{b_3^1, b_3^2, b_3^3\}$. In practice, one needs to consider the whole hyper-period P to collect all the distinct blocks (and sets) for each task. We denote $|s_i^j|$ as the number of blocks in s_i^j and δ_i as the number of distinct block sets for task τ_i.

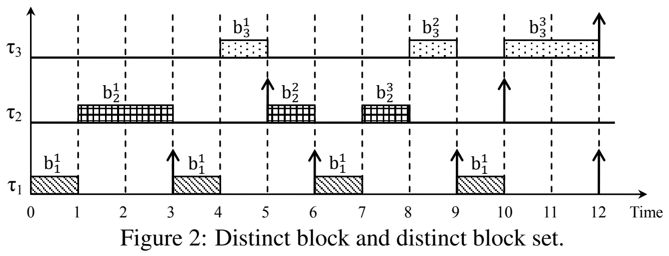


Figure 2: Distinct block and distinct block set.

Step 2: We calculate the energy consumption and execution time for each distinct block under all voltage levels in V. Let $e_i^{j,k}$ and $t_i^{j,k}$ denote these two values of task τ_i's j^{th} distinct block b_i^j under voltage v_k. Note that energy and time overhead for voltage transition are incorporated in them, respectively. In other words, we generate a *profile table* for every b_i^j which stores $e_i^{j,k}$ and $t_i^{j,k}$ for all l voltage levels in V.

[2]The three elements in the tuple here denote period, deadline and worst-case execution time, respectively.

Step 3: For each distinct block set s_i^j, different voltage assignment for every distinct block in it will effect the total energy consumption as well as execution time for the entire set, which essentially forms a whole job. In order to take all possible scenarios into account, we calculate the total energy consumption and execution time of all voltage level combinations, each of which comprises of one voltage level chosen for each distinct block in s_i^j. Let $E_i^{j,h}$ and $T_i^{j,h}$ stand for the total energy consumption and execution time of s_i^j using voltage level combination h. Each pair of $E_i^{j,h}$ and $T_i^{j,h}$ are stored in the *profile table* for s_i^j. Furthermore, non-beneficial voltage combinations, whose energy consumption and execution time are dominated by another combination in the same set, are eliminated. We use a dynamic programming based algorithm to generate the profile table for each distinct block set. The details are omitted due to limited space. Obviously, the number of Pareto-optimal combinations in s_i^j's profile table is $\pi_i^j \leqslant l^{|s_i^j|}$.

Step 4: So far we have obtained complete profiling information for all the jobs of each task. In order to guarantee the schedulability, we decide a *threshold* execution time t_i^{th} for each task to be used in schedulability test ($\sum_{i=1}^{m} \frac{t_i^{th}}{p_i} \leqslant 1$) that will act as the upper bound on each job's execution time. Now the original problem has become how to select one voltage combination for each distinct block set of a task so that the total energy consumption $E = \sum_{i=1}^{m} \sum_{j=1}^{\delta_i} \lambda_i^j \cdot E_i^{j,h'}$ (where h' is the selected profile table entry's index for s_i^j and λ_i^j is the number of times s_i^j occurs in the hyper-period P) is minimized while $\forall i \, \forall j \; T_i^{j,h'} \leqslant t_i^{th}$ is satisfied. In this step, we give every voltage combination a chance to use its execution time acting as the threshold for the corresponding task. Once a voltage combination of one distinct block set is picked as the threshold for task τ_i, all the other distinct block set's decisions can be made in a greedy manner, that is, the one with lowest energy consumption while execution time is less than or equal to the chosen threshold is selected. Note that if, for some other distinct block sets, no voltage combination can make its execution time under the threshold, the chosen threshold is infeasible and thus discarded. After all the decisions are made, we calculate the total energy consumption of that entire task and then put them along with the threshold execution time into the *aggregated profile table*, which is used as input to our approximation algorithm. Algorithm 1 illustrates this process. Note that h denote the index of the distinct block set profile table entry which is currently chosen to act as the threshold.

Algorithm 1 Aggregated profile table generation for τ_i.

```
Sort each $s_i^j$'s profile table in the ascending order of $E_i^{j,h}$
for $j = 1$ to $\delta_i$ do
    for $h = 1$ to $\pi_i^j$ do
        $isFeasible$ = true
        $h'_j = h$ {$h'_j$ denotes the selected index of $s_i^j$.}
        for $k = 1$ to $\delta_i$; $k \neq j$ do
            for $u = 1$ to $\pi_i^j$ do
                if $T_i^{j,u} \leqslant T_i^{j,h}$ then
                    $h'_u = u$ ; break
                end if
            end for
            if $h'_u$ is not updated then
                $isFeasible$ = false ; break
            end if
        end for
        if $isFeasible$ is true then
            $e_i = \sum_{j=1}^{\delta_i} \lambda_i^j \cdot E_i^{j,h'_j}$ {Total energy consumption of $\tau_i$}
            Add ($e_i$, $T_i^{j,h}$) into task $\tau_i$'s aggregated profile table
        end if
    end for
end for
```

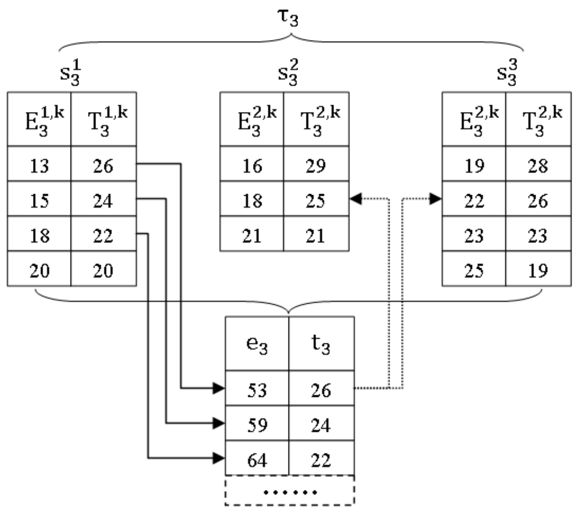


Figure 3: Aggregated profile table generation for each task.

Figure 3 shows an illustrative example of aggregated profile table generation. Suppose task τ_3 in Figure 2 has three distinct block sets in P and for each of them we have generated profile table in Step 3 as shown on the top-part of Figure 3. For simplicity, we assume that each of them only occurs once in P. The first entry in τ_3's aggregated profile table is calculated as follows. We choose execution time (26) of the first entry in s_3^1's profile table as the threshold. For s_3^2, the first entry cannot be selected since its execution time is higher than the threshold. Hence the second entry is chosen. For the same reason, the second entry is selected for s_3^3. The rest of the table can be generated similarly.

After applying Algorithm 1, non-beneficial entries in the aggregated profile table are filtered out. Note that Algorithm 1 assigns the same voltage combination to every occurrence of each distinct block set. Theorem 3.1 shows that it is reasonable and safe to do so in finding optimal assignments.

THEOREM 3.1. *An optimal solution of our problem must assign the same voltage combination for each occurrence of a distinct block set.*

PROOF. Suppose in the optimal assignment, distinct block set s_i^j is assigned two different voltage combinations vc_1 and vc_2. Assume that the threshold execution time chosen is t_i^{th}. Thus the execution time of both vc_1 and vc_2 are less than t_i^{th}. Since it is always safe to use voltage combinations with execution time under threshold, one can get a better solution by replacing the higher energy consuming one with the lower one, which contradicts the fact that the solution is optimal. $\square$

Complexity Analysis: We now analyze the complexity of our problem transformation scheme. *Step 1* performs scheduling of the task set. *Step 2* requires $\sum_{i=1}^{m} \sum_{j=1}^{\delta_i} |s_i^j|$ calculations. *Step 3* has a time complexity of $O(\sum_{i=1}^{m} \sum_{j=1}^{\delta_i} l \cdot |L_b'|)$, where $|L_b'|$ is the upper bound of the length of list L_b' in our dynamic programming algorithm. *Step 4* needs a running time of $O(\sum_{i=1}^{m} \sum_{j=1}^{\delta_i} \delta_i \cdot (\pi_i^j)^2)$. Each step takes only polynomial time. It is important to note that the problem transformation introduces only design-time computation overhead. Although the static overhead depends on the nature of the input task set, our experiments show that it normally takes only in the order of minutes for common task sets.

3.2 Approximation Algorithm

The program transformation described in the last section results in an aggregated profile table for each task. Each entry of the aggregated profile table (say j^{th} entry of task τ_i) represents one possible voltage assignment (of all distinct block sets) for task τ_i and keeps the corresponding total energy consumption as well as execution time, denoted by e_i^j and t_i^j, respectively. We divide each t_i^j

by period p_i to represent the utilization rate (t_i^j/p_i) of the task. Furthermore, let ρ_i denote the number of entries in task τ_i's aggregated profile table.

We now convert our problem from a minimization version to a maximization one. Let $e_i^{max} = \max\{e_i^1, e_i^2, \ldots, e_i^{\rho_i}\}$. For each e_i^j, we calculate energy saving $\overline{e}_i^j = e_i^{max} - e_i^j$. Now the objective becomes to maximize total energy saving $\overline{E} = \sum_{i=1}^{m} \overline{e}_i^{r_i}$ while satisfy the schedulability condition T = $\sum_{i=1}^{m} t_i^{r_i} \leqslant U$ by choosing one and only one entry from the aggregated profile table for each task (here r_i is the index of the chosen entry).

3.2.1 Dynamic Programming

Dynamic programming gives the optimal solution to our problem. Let $\overline{e}_i^{max}$ defined as $\max\{\overline{e}_i^1, \overline{e}_i^2, \ldots, \overline{e}_i^{\rho_i}\}$. Clearly, $\overline{E} \in [1, m\overline{e}_i^{max}]$. Let $S_i^{\overline{E}}$ denote a solution in which we make decisions for the first i tasks and the total energy saving is equal to $\overline{E}$ while the utilization rate T is minimized. A two-dimensional array is created where each element T[i][$\overline{E}$] stores the utilization rate of $S_i^{\overline{E}}$. Therefore, the recursive relation for dynamic programming is:

$$T[i][\overline{E}] = min_{j\in[1,\rho_i]}(T[i-1][\overline{E} - \overline{e}_i^j] + t_i^j) \tag{3}$$

Using this recursion, we fill up T[i][$\overline{E}$] for all $\overline{E} \in [1, m\overline{e}_i^{max}]$. Finally, the optimal energy saving $\overline{E}^*$ is found by:

$$\overline{E}^* = \{max\,\overline{E} \mid T[m][\overline{E}] \leqslant U\} \tag{4}$$

Dynamic programming achieves the optimal energy saving by iterating over all the tasks (1 to m), all possible total energy saving value (from 1 to $m\overline{e}_i^{max}$) and all entries in each task's aggregated profile table (from 1 to ρ_i). This algorithm fills the array in order so that when calculating the i^{th} row (T[i][]), all the previous (i - 1) rows are all filled. Hence, the time complexity is $O(m^2 \cdot max\{\rho_i\} \cdot \overline{e}_i^{max})$, which is pseudo-polynomial since the last term is unbounded.

3.2.2 Approximation Algorithm

The approximation algorithm proposed in this section is based on dynamic programming. It reduces the time complexity by scaling down every $\overline{e}_i^j$ value by a constant K such that $\overline{e}_i^{max}/K$ can be bounded by m (as well as the approximation ratio ε) – which reduces the complexity to polynomial. By doing so, we actually decrease the size of the design space. Our goal is to guarantee that the energy saving achieved by our approximation algorithm is no less than $(1-\varepsilon)\overline{E}^*$.

In order to obtain the constant K, we need to get the lower and upper bound on $\overline{E}^*$. This is done by employing a LP-relaxation version of our problem by removing the integral constraint (choose only one entry out of each task's aggregated profile table), that is, "fractiona" entries are allowed to be chosen. Algorithm 2 shows the polynomial-time greedy algorithm which can give the optimal solution to the LP-relaxation problem. Note that $\widetilde{e}_i^j$ is the incremental energy saving – a measure of how much more energy saving can be gained if the j^{th} entry is chosen instead of the $(j-1)^{th}$ entry from task τ_i's table. Here, $\widetilde{p}_i^j$ represents the incremental energy saving efficiency ($\widetilde{e}_i^j/\widetilde{t}_i^j$). $\widetilde{U}$ keeps the residual utilization rate. The algorithm terminates when $\widetilde{U}$ is exhausted. The LP-relaxation optimal choice for task τ_i, found by the algorithm, is r_i where $x_i^{r_i} = 1$. The split task, τ_s, has two fractional entries being picked: r_s and $r_{s'}$ ($\widetilde{p}_s^{r_s} < \widetilde{p}_s^{r_{s'}}$). We have the following lemma:

LEMMA 3.1.1. *If the optimal solution S^{LP} to the LP-relaxation version of our problem has no split task, it is already the optimal solution to our original problem. Otherwise, S^{LP} has at most one split task τ_s in which the two chosen fractional entries must be adjacent in its aggregated profile table.*

PROOF. Since this scenario can be mapped to MCKP, we can reuse the proof shown in [14]. □

Algorithm 2 Greedy algorithm for LP-relaxation problem.

```
for $i = 1$ to $m$ do
    Sort $\tau_i$'s aggregated profile table in ascending order of $t_i^j$.
    $\widetilde{p}_i^1 = \overline{e}_i^1/t_i^1$
    for $j = 2$ to $\rho_i$ do
        $\widetilde{e}_i^j = \overline{e}_i^j - \overline{e}_i^{j-1}$; $\widetilde{t}_i^j = t_i^j - t_i^{j-1}$; $\widetilde{p}_i^j = \widetilde{e}_i^j/\widetilde{t}_i^j$
    end for
end for
$\widetilde{U} = U - \sum_{i=1}^{m} t_i^1$; $\widetilde{E} = \sum_{i=1}^{m} \overline{e}_i^1$
Sort all the entries from each task's aggregated profile table together in descending order of $\widetilde{p}_i^j$, associating with the original indices $i$ and $j$.
for each entry (i,j) in the sorted order of $\widetilde{p}$ do
    if $\widetilde{U} - \widetilde{t}_i^j < 0$ then
        $s = i$; $t = j$ {Indices of the entry to be split.}
        break {Utilization rate has exceeded above the bound.}
    end if
    $\widetilde{E} = \widetilde{E} + \widetilde{p}_i^j$; $\widetilde{U} = \widetilde{U} - \widetilde{t}_i^j$
    $x_i^j = 1$; $x_i^{j-1} = 0$ {entry (i,j) is chosen instead of (i,j-1)}
end for
$x_s^t = \widetilde{U}/\widetilde{t}_s^t$; $x_s^{t-1} = 1 - x_s^t$; $\widetilde{E} = \widetilde{E} + \widetilde{p}_s^t x_s^t$
return $\widetilde{E}$
```

Let $\overline{E_0}$ be the maximum of the following three values: 1) total energy saving when the split task τ_s is discarded; 2) energy saving generated by the first fractional entry; 3) energy saving generated by the second fractional entry. That is,

$$\overline{E_0} = max\{\sum_{i=1;i\neq s}^{m} \overline{e}_i^{r_i}\ ,\ \overline{e}_s^{r_s} x_s^{r_s}\ ,\ \overline{e}_s^{r_{s'}} x_{s'}^{r_{s'}}\} \tag{5}$$

Note that according to Lemma 3.1.1, the last two terms belong to the same task. Now we can give the upper and lower bound of $\overline{E}^*$, as shown in the following lemma:

LEMMA 3.1.2. *$\overline{E_0}$ dictates the lower and upper bound of the optimal energy saving as: $\overline{E_0} \leqslant \overline{E}^* \leqslant 3\overline{E_0}$.*

PROOF. If $\overline{E_0} = \sum_{i=1;i\neq s}^{m} \overline{e}_i^{r_i}$, we can safely obtain higher overall energy saving by adding $\overline{e}_s^{r_s}$. If $\overline{E_0}$ equals to either of the other two terms, more energy saving can be achieved by adding $\sum_{i=1;i\neq s}^{m} \overline{e}_i^{min}$ where $\overline{e}_i^{min} = min\{\overline{e}_i^1, \overline{e}_i^2, \ldots, \overline{e}_i^{\rho_i}\}$. Hence, we have $\overline{E}^* \geqslant \overline{E_0}$. Clearly, the solution to the LP-relaxation version must not be worse than the one for the original problem. In other words, $\widetilde{E} \geqslant \overline{E}^*$. Since $\widetilde{E} = \sum_{i=1;i\neq s}^{m} \overline{e}_i^{r_i} + \overline{e}_s^{r_s} x_s^{r_s} + \overline{e}_s^{r_{s'}} x_s^{r_{s'}} \leqslant 3\overline{E_0}$, we have $\overline{E}^* \leqslant 3\overline{E_0}$. □

Now we decide the scaling down factor K using the bounds described above. We prove that approximation ratio and polynomial time complexity can be guaranteed if we assign $K = \frac{\varepsilon\overline{E_0}}{m}$, as shown in the following lemmas:

LEMMA 3.1.3. *The K-scaled dynamic programming algorithm generates a $(1-\varepsilon)$ approximation voltage assignment.*

PROOF. Let the scaled energy saving value $\overline{e}'^j_i = \lfloor \overline{e}_i^j/K \rfloor$, we have $K\overline{e}'^j_i \leqslant \overline{e}_i^j < K(\overline{e}'^j_i + 1)$, hence $\overline{e}_i^j - K\overline{e}'^j_i < K$. Therefore, by accumulating all m tasks, we have:

$$\sum_{i=1}^{m} \overline{e}_i^{h_i} - K\sum_{i=1}^{m} \overline{e}'^{h_i}_i < Km \tag{6}$$

where h_i is the index of the selected aggregated profile table entry for task τ_i.

Note that the left term of Equation (6) is the approximation scaling error. Since $K = \frac{\varepsilon\overline{E_0}}{m}$, we have $Km = \varepsilon\overline{E_0}$. Since $\overline{E}^* \geqslant \overline{E_0}$, according to Lemma 3.1.2, we have $Km \leqslant \varepsilon\overline{E}^*$. Therefore, the approximation error $\sum_{i=1}^{m} \overline{e}_i^{h_i} - K\sum_{i=1}^{m} \overline{e}'^{h_i}_i < Km \leqslant \varepsilon\overline{E}^*$. Hence the approximation ratio ε holds. □

LEMMA 3.1.4. *The time complexity of the K-scaled dynamic programming algorithm is $O(\frac{m^2 \cdot max\{\rho_i\}}{\varepsilon})$.*

PROOF. Given the upper bound of $\overline{E}^*$ ($\overline{E}^* \leqslant 3\overline{E_0}$), the dynamic programming method can be improved to search in the range of $[1, 3\overline{E_0}]$, resulting in a time complexity of $O(m \cdot max\{\rho_i\} \cdot \overline{E_0})$. For the scaled version, the complexity is reduced to $O(m \cdot max\{\rho_i\} \cdot \frac{\overline{E_0}}{K})$. Given $K = \frac{\varepsilon \overline{E_0}}{m}$, we have $\frac{\overline{E_0}}{K} = \frac{m}{\varepsilon}$, thus the complexity becomes $O(\frac{m^2 \cdot max\{\rho_i\}}{\varepsilon})$, which is independent of any pseudo-polynomial energy values. □

THEOREM 3.2. *The proposed algorithm is a fully polynomial time (1 - ε) approximation scheme for the maximization version of our problem.*

PROOF. Directly follows from Lemma 3.1.3 and 3.1.4. □

Now let's evaluate the quality of the solution generated by the converted problem with respect to our original problem. Let E^* denote the optimal result (the minimum energy consumption) and α denote the approximation ratio for the original problem. Given an approximation ratio ε for the maximization version, α can be quantified as:

$$(1+\alpha)E^* = \sum_{i=1}^{m} e_i^{max} - (1-\varepsilon)\overline{E}^* \tag{7}$$

Hence,

$$\alpha = \frac{\sum_{i=1}^{m} e_i^{max} - \overline{E}^* + \varepsilon\overline{E}^* - E^*}{E^*} \tag{8}$$

Since for a specific solution, according to our conversion strategy, we have: $E^* = \sum_{i=1}^{m} e_i^{max} - \overline{E}^*$. Therefore,

$$\alpha = \frac{\overline{E}^*}{E^*}\varepsilon \tag{9}$$

Equation (9) illustrates that α is related to ε by the factor of $\overline{E}^*/E^*$, which is the ratio of the total energy saving to total energy consumption over all tasks. In the worst case, when the overall utilization is low enough so that entries with the lowest energy consumption are selected for each task, this ratio reaches maximum. Let v_{max} and v_{min} denote the maximum and minimum voltage available, respectively. We have,

$$\alpha \leqslant \frac{\sum_{i=1}^{m}(e_i^{max} - e_i^{min})}{\sum_{i=1}^{m} e_i^{min}}\varepsilon \leqslant \frac{{v_{max}}^2 - {v_{min}}^2}{{v_{min}}^2}\varepsilon \tag{10}$$

Let γ denote this maximum ratio (thus $\alpha \leqslant \gamma \cdot \varepsilon$). In practice, given a voltage scalable processor, we first calculate its γ value. If $\gamma \leqslant 1$, it means that by solving the converted maximization problem using approximation ratio ε, we can get a solution with an equal or better quality bound ($\leqslant \varepsilon$) to the original problem. Otherwise, if needed, we can set $\varepsilon = \alpha/\gamma$ so that the specified approximation ratio (α) to the original problem can be achieved firmly. As a result, the time complexity of our approximation scheme with respect to the original problem is $O(\frac{m^2 \cdot max\{\rho_i\} \cdot \gamma}{\alpha})$. As shown in Section 4.1, for common voltage scalable processors, γ is usually very small and in some cases (e.g. StrongARM [15]) is less than 1.

Now we have obtained an approximated optimal solution based on the original EDF schedule which is generated without voltage scaling. As described in Section 3.1, we have ensured that the utilization bound of EDF is observed, the modified task set is guaranteed to be schedulable. Note that running the task set with the new voltage assignment could potentially result in a slightly different schedule since we are essentially changing the execution time of each block. Hence the solution we give is essentially with respect to the original schedule. Certainly, more iterations can be carried out based on the new schedule until it becomes steady. Based on our observations, such costly iterations contribute very little in overall energy savings, and therefore not beneficial.

4. EXPERIMENTS

4.1 Experimental Setup

To demonstrate the effectiveness of our approach, we consider two DVS processors: StrongARM [15] and XScale [16]. The former one supports four voltage - frequency levels (1.5V - 206MHz, 1.4V - 192Mhz, 1.2V - 162MHz and 1.1V - 133MHz) with $\gamma = 0.86$ and the latter one supports five levels (2.05V - 1000MHz, 1.65V - 800MHz, 1.3V - 600HMz, 0.99V - 400MHz and 0.7V - 200MHz) with $\gamma = 7.58$. We compare our results with two scenarios: when no DVS is used and when optimal inter-task scaling is employed [10]. In the former scenario, every task is running under the highest voltage level. While in the latter scenario, a dynamic programming based algorithm is used to obtain the optimal solution as discussed in Section 3.2.1. Approximation ratio α of 0.01, 0.05, 0.10, 0.15 and 0.20 are considered[3]. We implemented the EDF scheduling simulator along with all the algorithms in C++.

4.2 Results using Real Benchmarks

We first construct four task sets each of which consists of real benchmark applications selected from typical embedded system benchmark suites MediaBench [17] EEMBC [18] and MiBench [19] as shown in Table 1. Task Set 1 consists of tasks from MediaBench, Set 2 from EEMBC, Set 3 from MiBench, and Set 4 is a mixture from all three suites. We set each task's utilization rate (under the highest voltage level) randomly in the interval of $[\frac{0.5}{m}, \frac{1.5}{m}]$. The accumulated overall utilization rate is controlled to be within [0.7,0.9] for StrongARM and [0.5,0.7] for XScale.

Table 1: Task sets consisting of real benchmarks.

Task sets	**Tasks**
Set 1	cjpeg, djpeg, epic, mpeg2, pegwit, toast, untoast, rawcaudio
Set 2	A2TIME01, AIFFTR01, AIFIRF01, BaseFP01, BITMNP01, IDCTRN01, RSPEED01, TBLOOK01
Set 3	qsort, susan, dijkstra, patricia, rijndael, adpcm, CRC32, FFT, stringsearch
Set 4	cjpeg, epic, pegwit, A2TIME01, RSPEED01, qsort, susan, dijkstra

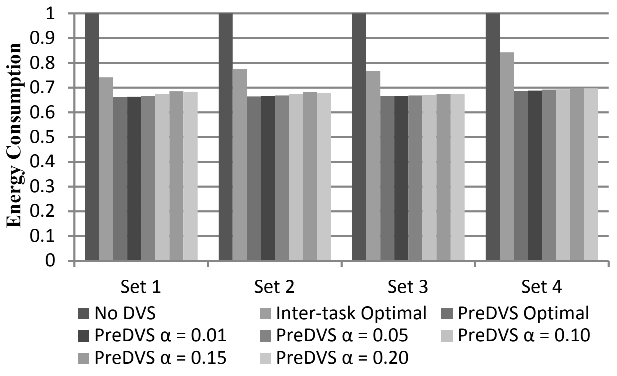
Figure 4 shows the results in both scenarios. On StrongARM processor, our approximation scheme saves up to 34% energy compared to no-DVS and outperforms the optimal inter-task scaling by up to 17% even when the approximation ratio is set to 0.2 ($\alpha = 0.2$). On XScale processor, due to larger span between available voltage levels and lower overall utilization rate, up to 67% energy saving is achieved over no-DVS scenario and on average 19% extra saving compared to optimal inter-task voltage scaling.

4.3 Results using Synthetic Tasks

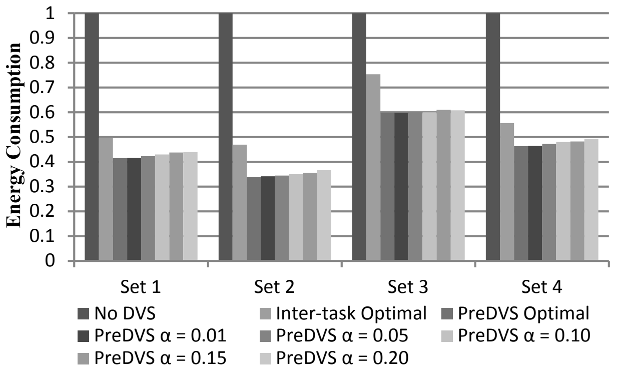
We also evaluated our approach by randomly generated task sets with 5 to 10 tasks per set with different overall utilization rates. We define the *effective bound* of a DVS processor as the following: any task set with an overall utilization rate equal to or lower than the effective bound can achieve the optimal voltage assignment by trivially choosing the lowest voltage for all the tasks. Clearly, the effective bound is the ratio of the lowest frequency to the highest one. In other words, the effective bound is 0.64 for StrongARM and 0.2 for XScale. Hence, in the former case, we vary the overall utilization rate of each task set from 0.65 to 0.95 at one step of 0.05 while from 0.3 to 0.9 at one step of 0.1. Given each overall utilization rate, we randomly generate task periods in the interval of [100,30000]. Similarly as in Section 4.2, each task's utilization rate is evenly distributed between $[\frac{0.5*U}{m}, \frac{1.5*U}{m}]$.

Figure 5 shows the results which are the average of 10 randomly generated task sets for each overall utilization rate on both DVS processors. Clearly, in all cases, our approach achieves closely approximated overall energy consumption with respect to the optimal

[3]Approximation ratio ε for the maximization version of our problem is calculated as described in Section 3.2.2

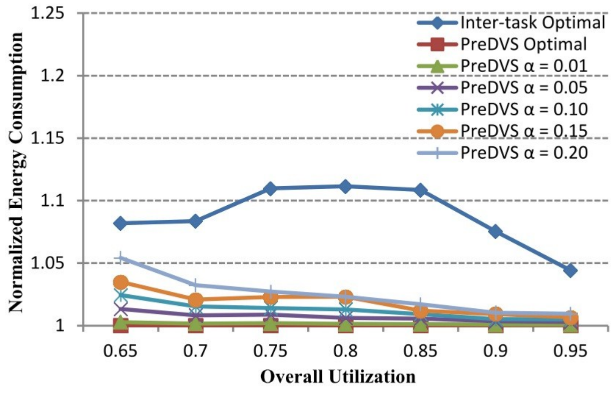


(a) StrongARM processor

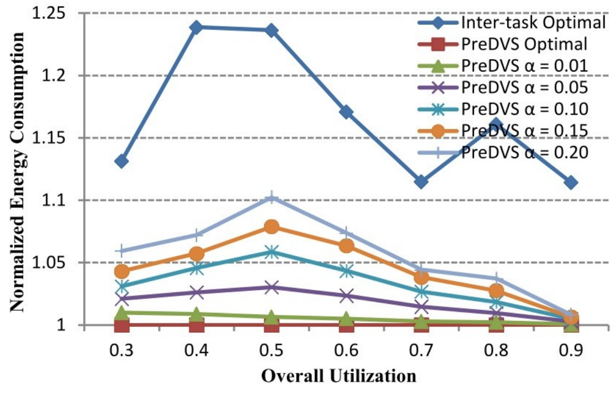


(b) XScale processor

Figure 4: Results for real benchmark task sets.



(a) StrongARM processor



(b) XScale processor

Figure 5: Results for synthetic task sets.

solution and outperforms inter-task optimal scaling consistently up to 24%. The running time of our algorithm is comparable with inter-task technique in the optimal solution case and much shorter in approximated solution cases. For example, inter-task DVS [10] and PreDVS optimal algorithms take 26442 and 27082 milliseconds, respectively, for one of the synthetic task sets, while PreDVS approximation algorithm only requires hundreds of milliseconds.

5. CONCLUSION

In this paper, we presented a dynamic voltage scaling technique for preemptive real-time systems using approximation scheme which achieves significant energy savings by assigning different voltage levels to each task. We proved that the problem is NP-hard and presented an approximation scheme by developing a novel transformation mechanism and a fully polynomial time approximation algorithm. Our approach does not introduce any additional voltage switching overhead compared to inter-task scaling techniques. Moreover, our approach exploits static time slack only and thus can be employed together with any existing intra-task scaling techniques. The approximate solutions given by our approach outperforms optimal inter-task scaling techniques by up to 24%. Our experimental results demonstrated that our approach can generate solutions very close to the optimal.

6. REFERENCES

[1] G. Buttazzo, *Hard Real-Time Computing Systems*. Kluwer, 1995.

[2] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[3] I. Hong et al., "Power optimization of variable-voltage core-based systems," *IEEE TCAD*, vol. 18, pp. 1702–1714, 1999.

[4] W. Wang et al., "SACR: Scheduling-aware cache reconfiguration for real-time embedded systems," *VLSI Design*, 2009.

[5] W. Wang et al., "Dynamic reconfiguration of two-level caches in soft real-time embedded systems," *ISVLSI*, 2009.

[6] W. Wang et al., "Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems," *VLSI Design*, 2010.

[7] R. Jejurikar et al., "Energy-aware task scheduling with task synchronization for embedded real-time systems," *IEEE TCAD*, vol. 25, pp. 1024–1037, 2006.

[8] G. Quan et al., "Energy efficient dvs schedule for fixed-priority real-time systems," *ACM TODAES*, vol. 6, pp. 1–30, 2007.

[9] X. Zhong et al., "System-wide energy minimization for real-time tasks: Lower bound and approximation," *ICCAD*, 2006.

[10] S. Zhang et al., "Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules," *ISLPED*, 2007.

[11] J. Seo et al., "Profile-based optimal intra-task voltage scheduling for hard real-time applications," *DAC*, 2004.

[12] D. Shin et al., "Optimizing intratask voltage scheduling using profile and data-flow information," *IEEE TCAD*, vol. 26, pp. 369–385, 2007.

[13] S. Oh et al., "Task partitioning algorithm for intra-task dynamic voltage scaling," *ISCAS*, 2008.

[14] H. Kellerer et al., *Knapsack Problems*. Springer-Verlag, 2004.

[15] Marvell, *StrongARM 1100 processor*, www.marvell.com.

[16] Marvell, *XScale microarchitecture*, www.marvell.com.

[17] C. Lee et al., "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," *Micro*, 1997.

[18] EEMBC, *EEMBC, The Embedded Microprocessor Benchmark Consortium*, http://www.eembc.org/.

[19] M. Guthaus et al., "Mibench: A free, commercially representative embedded benchmark suite," *WWC*, 2001.