Practical PACE for Embedded Systems [*]

Ruibin Xu, Chenhai Xi, Rami Melhem, Daniel Mossé
Computer Science Department, University of Pittsburgh
Pittsburgh, PA 15260
{*xruibin,chenhai,melhem,mosse*}*@cs.pitt.edu*

ABSTRACT

In current embedded systems, one of the major concerns is energy conservation. The dynamic voltage-scheduling (DVS) framework, which involves dynamically adjusting the voltage and frequency of the CPU, has become a well studied technique. It has been shown that if a task's computational requirement is only known probabilistically, there is no constant optimal speed for the task and the expected energy consumption is minimized by gradually increasing speed as the task progresses [11]. It is possible to find the optimal speed schedule if we assume continuous speed and a well-defined power function, which are assumptions that do not hold in practice. In this paper, we study the problem from a practical point of view, that is, we study the case of discrete speeds and make no restriction on the form of the power functions. Furthermore, we take into account processor idle power and speed change overhead, which were ignored in previous similar studies. We present a fully polynomial time approximation scheme (FPTAS), which has performance guarantees and usually obtains solutions very close to the optimal solution in practice. Our evaluation shows that our algorithm performs very well and generally obtains solutions within 0.1% of the optimal.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Algorithms

Keywords

Real-time, Dynamic Voltage Scaling, Power management, Processor Acceleration to Conserve Energy, Fully Polynomial Time Approximation Scheme

1. INTRODUCTION

Energy minimization is critically important for embedded portable and handheld devices simply because it leads to extended battery life. Dynamic voltage scaling (DVS) allows a processor to change the clock frequency and supply voltage on-the-fly to trade performance for energy conservation. Using DVS, quadratic energy savings [7, 19] can be achieved at the expense of just linear performance loss. Various chip makers, such as Transmeta, Intel, and AMD, manufacture processors with this feature.

When applied to real-time embedded systems, DVS schemes focus on minimizing energy consumption of tasks in the systems while still meeting the deadlines. Because most embedded systems are small and simple, a central problem is to statically (i.e., prior to runtime) find the best running speed/frequency of a task. If a task's computational requirement (i.e., number of cycles it will execute) is known deterministically, it has been shown that one can safely commit to a constant optimal CPU speed during the execution of a task to minimize the energy consumption [2], assuming continuous speeds. If a CPU only offers a fixed, limited set of valid speeds, using the two speeds which are immediate neighbors to the optimal speed will minimize the energy consumption [8]. Note that this requires intra-task speed change, which can be realized through a timeout mechanism, or by inserting system calls into the task at compile time.

However, more often than not, a task's computational requirement is only known probabilistically. In this case, it is impossible to determine a constant optimal speed and the expected energy consumption is minimized by gradually increasing the speed as the task progresses, which is an approach named as *Processor Acceleration to Conserve Energy* (PACE) [11]. Similar to the case where a task's computational requirement is known deterministically, the optimal speed schedule can be obtained assuming continuous speeds and well-defined power functions, such as perfectly cubic power versus frequency function.

These assumptions typically do not hold in practice. A number of schemes, such as PACE [11] and GRACE [20], have proposed to use well-defined functions to approximate the actual power function and solve the continuous version of the problem before rounding the speeds to the available

[*]This work has been supported by the Defense Advanced Research Projects Agency through the PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736 and by NSF through grants 0125704 and 0325353

EMSOFT'04, September 27–29, 2004, Pisa, Italy.

discrete speeds. The schemes are of great simplicity at the expense of the optimality of the solution. Although the authors of PACE predicted that rounding should work reasonably well, the effect of the approximations on the quality of the solution compared to the optimal solution has not been studied.

In this paper, we study the problem from a practical point of view, that is, we focus on efficient algorithms that apply directly to the case of discrete speeds and do not make any assumption about the power function. We also take into account processor idle power and speed change overheads. We design a fully polynomial time approximation scheme (FPTAS) that can obtain ϵ-optimal solutions, which are within a factor of $1+\epsilon$ of the optimal solution and run in time polynomial in $1/\epsilon$. That is, we explore the tradeoff between the accuracy of the solution and the computational effort needed to find it and show that in practice these algorithms are fast and extremely close to optimal algorithms. We also identify the conditions when fast optimal exact algorithms are applicable.

The remainder of this paper is organized as follows. Section 2 gives the problem formulation. A brief review of strongly related work (PACE and GRACE) is given in Section 3. In Section 4 we identify the conditions when simple, fast exact optimal algorithm is applicable. Section 5 presents our fully polynomial time approximation scheme and proof of correctness. Evaluation results are reported in Section 6. In Section 7 concluding remarks and future work are presented.

2. PROBLEM FORMULATION

Before we give the formulation of the problem solved in this paper, we show how the problem solved in [11] is formulated when the speed of the task can be changed continuously and the power function is well defined.

2.1 Theoretical Optimal Formulation

Assume the processor can attain any speed continuously between 0 cycles/sec and infinite cycles/sec, and the length (speed) of each cycle during the execution of the task can be adjusted. Assume that the number of cycles used by the task denoted is a random variable X, and that the task is allotted a certain amount of time, D, to execute (in the rest of this paper we think of D as the deadline of the task). Thus, the problem is to statically find the speed for each cycle such the expected energy consumption of the task is minimized while still meeting the deadline.

The cumulative density function, cdf, associated with X is defined as $cdf(x) = Pr(X \leq x)$. This function reflects the probability that the task finishes before a certain number of clock cycles, x. Let WC be the worst case number of clock cycles for the task, thus we have $cdf(WC) = 1$. For each cycle y $(1 \leq y \leq WC)$, a specific energy, e_y, will be consumed depending on the frequency, f_y, adopted for this cycle. However, each of these cycles is executed with a certain probability, and hence, the expected energy consumed by cycle y is $(1 - cdf(y-1)) \cdot e_y$. Let Z be the energy consumed by the whole task, which is also a random variable. Thus the expected value of Z is

$$\mathbb{E}[Z] = \sum_{1 \leq y \leq WC} (1 - cdf(y-1)) \cdot e_y$$

Assume that the power function is well defined by $e_y = f_y^\alpha$, where $\alpha \geq 2$. Then, we have the following mathematical programming problem:

$$\texttt{Minimize} \quad \sum_{1 \leq y \leq WC} (1 - cdf(y-1)) \cdot f_y^\alpha$$

$$\texttt{Subject to} \quad \sum_{1 \leq y \leq WC} \frac{1}{f_y} \leq D$$

which can be solved by Lagrangian technique [13] or Jensen's inequality [9].

2.2 Piecewise constant speed schedules

Obviously, computing speed for each cycle is too overwhelming considering that a task usually takes millions of cycles. Furthermore, the ability to change speed in any cycle is unreasonable since real-world operating systems have some granularity requirement for changing speeds [1, 12]. Hence, in practice, we should probably change the speed for a task no more than some reasonable number of times. Thus, we need a schedule with a limited number of transition points at which speed may change. This implies that the speed remains constant between any two adjacent transition points. Denote the i^{th} transition point by b_i. Therefore, given r transition points, we partition the range $[0, WC]$ into $r+1$ phases: $[b_0, b_1 - 1], [b_1, b_2 - 1], \ldots, [b_r, b_{r+1} - 1]$, where $b_0 = 1$ and $b_{r+1} = WC + 1$ are used to simplify the formula.

Define *speed schedule* as speeds for each phase. Our goal is to find a schedule that minimizes the expected energy consumption while still meeting the deadline. Assume that the processor provides M discrete operating frequencies, $freq_1 < freq_2 < \cdots < freq_M$. Let the frequency for the i^{th} phase be denoted by f_i $(0 \leq i \leq r)$, where $f_i \in \{freq_1, freq_2, \ldots, freq_M\}$. Let the energy consumed by a single cycle in phase i be denoted by $e(f_i)$, where $e(f_i) = \frac{p(f_i)}{f_i}$ and $p(f_i)$ is the processor power consumption when running at frequency f_i. Thus we obtain the following mathematical program:

$$\texttt{Minimize} \quad \sum_{0 \leq i \leq r} s_i \cdot F_i \cdot e(f_i) \qquad (1)$$

$$\texttt{Subject to} \quad \sum_{0 \leq i \leq r} \frac{s_i}{f_i} \leq D \qquad (2)$$

where $s_i = b_{i+1} - b_i$ and $F_i = \sum_{b_i \leq j < b_{i+1}} (1 - cdf(j-1))$. Since $e(f)$ is a discrete function, (1)-(2) cannot be solved by Lagrangian technique or Jensen's inequality.

It is not clear how to choose an optimal sequence of r transition points. If the task cycle distribution is estimated using a histogram, then assigning the boundaries of bins of the histogram as transition points seem to be a natural choice [20]. In [11] the authors proposed a heuristic to select a "good" sequence of transition points but only justified it under the condition of continuous speed. The choice of transitions during a program execution is a function of the overhead of the transition, the length of the program and the number of possible speeds in the system. This is still an open problem which is beyond the scope of this paper. The interested reader is referred to [1], which assumed the sequential form of program execution, where a program can be divided into n segments of equal length, and [16], which made use of compiler and placed the transition points on the edges of the control flow graph of the program.

2.3 The Impact of Idle Power

In general, if the CPU has no task to execute, it will put itself in an idle state, that is, the CPU is running at the minimum operating frequency $freq_1$ and only executing *no-op* instructions. Therefore, the idle power p_{idle} consumed by the CPU when idle is strictly less than the power consumption when running a task at the minimum operating frequency. However, it is important to notice that the idle power is not equal to zero.

The problem formulation described in (1)-(2) is based on the implicit assumption that the idle power is zero because our ultimate goal is to minimize the expected processor energy consumption and the objective function is the expected energy consumed by tasks (both PACE and GRACE ignore the idle power). Not only this assumption does not hold in practice[1], but also could lead to an abnormality called *inefficient frequency* [14]. For example, for the IBM PPC405LP (Table 1), the energy consumed by each cycle running at frequency 266MHz, $\frac{600}{266} \approx 2.2556$, is higher than the energy consumed by each cycle running at frequency 333MHz which is $\frac{750}{333} \approx 2.2522$. The frequency 266MHz is inefficient in the sense that it is slower than the frequency 333MHz yet consumes more energy. This implies that the frequency 266MHz will never be picked in the optimal speed schedule. However, this is not true. Although using 333MHz consumes less energy than using 266MHz for doing the same amount of computation, it finishes the computation faster and will spend more time consuming the idle power. Therefore, it might end up consuming more energy. In fact, if we take into account the idle power, which is 12mW for the IBM PPC405LP, then the frequency 266MHz becomes efficient [14].

In the presence of the idle power, we need to modify the problem formulation. More specifically, we need to modify the definition of the function $e(f)$. Assume that the idle power is fixed. Since the amount of idle power is always consumed by the CPU when it is on (when the CPU is running at frequency f, we can think of it consuming two portions of power: p_{idle} and $p(f) - p_{idle}$, where p_{idle} is a constant), we can eliminate the effect of the amount of idle power from the function $e(f)$, that is, we change the definition of $e(f)$ to

$$e(f) = \frac{p(f) - p_{idle}}{f}$$

and thus, the form of the objective function stay unchanged.

3. REVIEW OF PREVIOUSLY KNOWN SCHEMES

In this section, we give a brief review and discussion of the schemes provided by PACE and GRACE. Other researchers have also tackled this problem, but with a completely different solution [6], which is geared towards soft real-time tasks.

Table 1: PPC405LP speed settings and power consumptions

Speed (MHz)	33	100	266	333
Voltage (V)	1.0	1.0	1.8	1.9
Power (mW)	19	72	600	750

Table 2: 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

Both PACE and GRACE apply the well-known cubic-root rule of the power functions [3] and hence use $e(f) = \alpha f^2 + \beta$ (α and β are constants) to approximate the actual energy/cycle function. This is based on the fact that in CMOS circuits, the dominant component of power consumption is proportional to $V^2 f$, where V is voltage and f is frequency, and the highest frequency at which the CPU will run correctly drops approximately proportionally to the voltage ($f \propto V$) [15]. However, this approximation may not be accurate. We show that by two real-life examples of processors used in embedded systems. Tables 1 and 2 show the speed settings and power consumed by the IBM PPC405LP (from our actual measurements) and Intel XScale [17] processors. Figure 1 shows the least square curve fitting of the actual discrete power functions of these two processors with the form $af^3 + b$. As we can see, the XScale is quite accurate while the PPC405LP is not.

The cubic-root rule of power versus frequency function naturally leads to using $e(f) = \alpha f^2 + \beta$ to approximate the actual energy versus cycle function, the minimization problem becomes

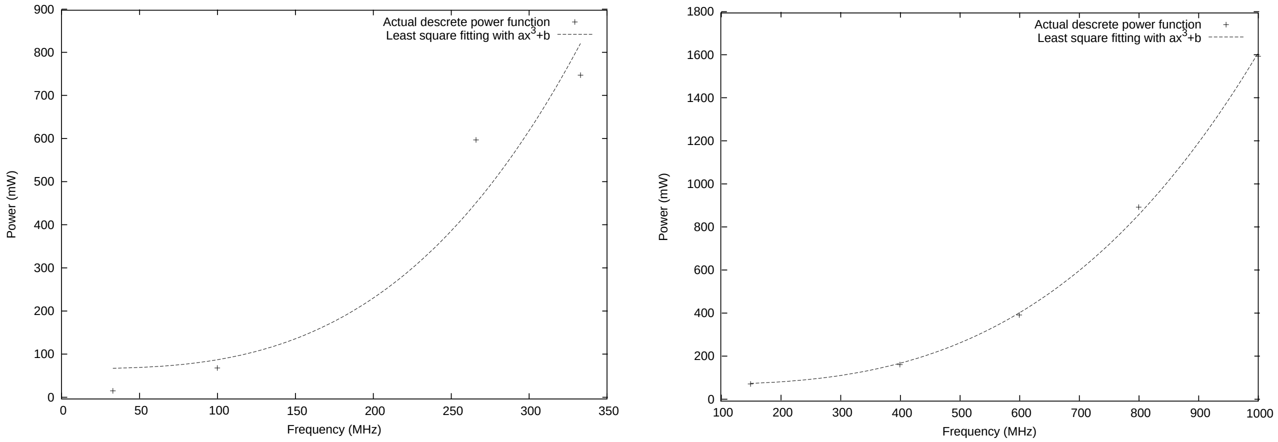
$$\begin{aligned} &\texttt{Minimize} && \sum_{0 \le i \le r} s_i \cdot F_i \cdot f_i^2 \\ &\texttt{Subject to} && \sum_{0 \le i \le r} \frac{s_i}{f_i} \le D \end{aligned}$$

Notice that α and β have no effect on deciding the speed schedule. Assuming that f_i is continuous, using the Lagrange technique or Jensen's inequality, the solution to the above mathematical program is

$$f_i = \frac{\sum_{0 \le j \le r} s_j \cdot F_j^{\frac{1}{3}}}{D \cdot F_i^{\frac{1}{3}}}$$

However, f_i needs to be rounded to some available discrete frequency. This is where PACE and GRACE differ. The GRACE scheme is conservative in the sense that it rounds f_i up to the closest higher discrete frequency, whereas the PACE scheme rounds f_i to the closest discrete frequency. Both schemes have shortcomings. For the GRACE scheme, f_i could be larger than the highest discrete frequency if F_i is very small. If this happens, f_i will have to be rounded down to the highest discrete frequency $freq_M$, and therefore the deadline could be missed (considering that most of f_i's do not have this problem and they are rounded up, the probability of missing deadline should be reasonably small). For the PACE scheme, chances of missing the deadline are high, because PACE may round down. To solve this problem, PACE proposes to linearly scan all the phases to adjust the speeds. But this still does not guarantee that the deadline will not be missed. If this happens, PACE will simply use a default speed schedule [10].

[1] It does not hold in practice because of overhead of turning the CPU on/off.



(a) PPC405LP

(b) Intel XScale

Figure 1: Applying the cubic-root rule to approximate the actual power functions

4. FAST EXACT OPTIMAL ALGORITHM

First, we state the sufficient condition for nondecreasing speeds as the task progresses (Theorem 1). Theorem 1 is a necessary condition for using the fast exact optimal algorithm.

THEOREM 1. *If $s_0 = s_1 = \cdots = s_r$, then there exists an optimal schedule in which $f_0 \leq f_1 \leq \cdots \leq f_r$.*

PROOF. Suppose there exist phase i and phase j such that $i < j$ and $f_i > f_j$ in the optimal speed schedule. The energy consumed by these two phases is $E_1 = s_i \cdot F_i \cdot e(f_i) + s_j \cdot F_j \cdot e(f_j)$ and the time spent in these two phases is $T_1 = \frac{s_i}{f_i} + \frac{s_j}{f_j}$.

Now we switch the frequencies of phase i and phase j to obtain a new speed schedule, that is, phase i uses frequency f_j and phase j uses frequency f_i. The energy consumed by these two phases becomes $E_2 = s_i \cdot F_i \cdot e(f_j) + s_j \cdot F_j \cdot e(f_i)$ and the time spent in these two phases is $T_2 = \frac{s_i}{f_j} + \frac{s_j}{f_i}$.

Obviously we have $T_1 = T_2$, which implies the new speed schedule is also feasible (i.e., tasks finish before deadlines). However, the energy consumed in the new schedule is larger than in the original schedule since

$$\begin{aligned} E_1 - E_2 &= s_iF_ie(f_i) + s_jF_je(f_j) - (s_iF_ie(f_j) + s_jF_je(f_i)) \\ &= (e(f_i) - e(f_j)) \cdot (s_iF_i - s_jF_j) \\ &\geq 0 \end{aligned}$$

The inequality holds because $e(f_i) > e(f_j)$ and $F_i \geq F_j$. and the equality holds if and only if $F_i = F_j$. If this is the case, the new schedule is another optimal schedule. If $F_i > F_j$, the new schedule achieves less expected energy consumption which contradicts the optimality of the original schedule. □

The task cycle distribution is generally estimated using nonparametric methods [10, 20]. In particular, histogram is used because of its simplicity and effectiveness. When a histogram is used, the boundaries of bins [20] are naturally chosen to be the transition points, where we have $s_0 = s_1 = \cdots = s_r$. In this case, the speeds are nondecreasing as the task progresses in the optimal schedule [2]. Furthermore, the number of available discrete frequencies is usually less than the number of transition points. Therefore there are at most $M - 1$ speed changes in the optimal schedule. Let the number of phases running at frequency $freq_i$ be n_i. Thus, we have $\sum_{1 \leq i \leq M} n_i = r+1$. From Theorem 1, it is clear that we can use a brute force approach to try all the possible values of n_i and compare the corresponding expected energy. Notice that once $n_1, n_2, \ldots, n_{M-2}$ are decided, n_{M-1} and n_M can be decided in constant time. This results in a straightforward algorithm running in $\Theta\left(\binom{r+M-1}{M-2} \cdot r\right)$ time.

The complexity of the algorithm can be further reduced to $\Theta\left(\binom{r+M-1}{M-2}\right)$ by using an extra $\Theta(r)$ space. This is done by computing a *cumulative cdf*, defined as $ccdf(k) = \sum_{i=1}^{k} cdf(i)$, before the brute force search.

This algorithm is easy to implement and good for small M and r. For example, the system in [12] has only 3 available discrete frequencies; if we set $r = 100$, the number of possibilities is just 103. But for large M and r, this algorithm becomes impractical since $\Theta\left(\binom{r+M-1}{M-2}\right) = \Omega\left(\left(\frac{r+M-1}{M-2}\right)^{M-2}\right)$. We solve this problem with the algorithm provided in the next section.

5. A FULLY POLYNOMIAL TIME APPROXIMATION SCHEME

In this section, we present a fully polynomial time approximation scheme which we call PPACE (Practical PACE).

5.1 Preliminaries

DEFINITION 1. *A energy-time label l 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 $l.e$ and the time component as $l.t$.*

DEFINITION 2. *The "+" is defined as a binary operator on energy-time labels such that if (e_1, t_1) and (e_2, t_2) are two energy-time labels, then $(e_1, t_1) + (e_2, t_2) = (e_1 + e_2, t_1 + t_2)$.*

[2] If we use the heuristic described in [11] to choose transition points, where we cannot guarantee that $s_0 = s_1 = \cdots = s_r$, then the speeds are not necessarily nondecreasing in the optimal schedule, which contradicts with the results in [11]. Further investigation on this issue is warranted.

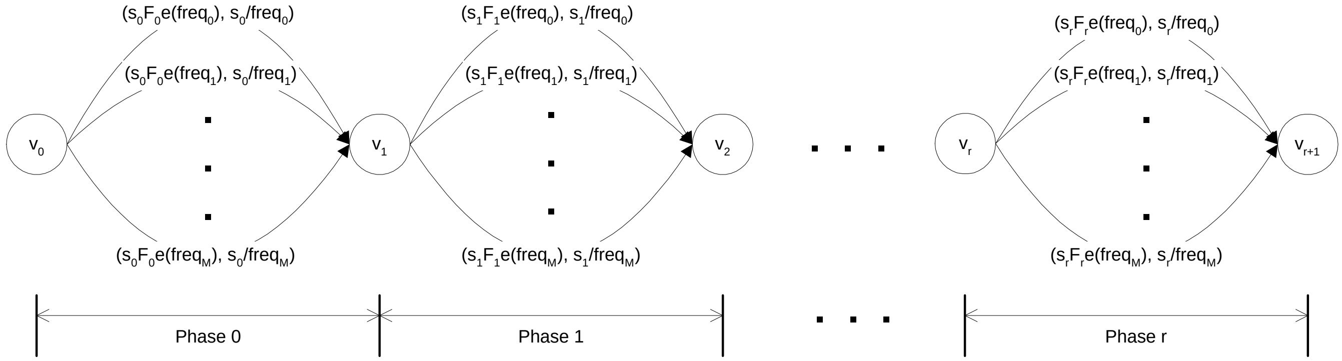


Figure 2: Graphical representation of the problem

The problem can be expressed as a graph $G = (V, E)$ shown in Figure 2. Vertex v_i $(1 \leq i \leq r+1)$ represents the point where the first i phases have been executed. The M edges between v_i and v_{i+1} $(0 \leq i \leq r)$ represent different choices (which frequency to use) for phase i. Each choice is represented by a energy-time label, indicating the expected energy consumption and the worst-case running time for that phase. We also associate each path in the graph with a energy-time label where the energy of a path is defined as the sum of the energy of all edges over the path and the time of a path is defined as the sum of the time of all edges over the path. Therefore, the problem is reduced to finding a path from v_0 to v_{r+1} such that the energy of the path is minimized while the time of the path is no greater than the deadline D.

Notice that a path now can be summarized as a energy-time label. A straightforward approach is to start from v_0 and working all the way from left to right in an iterative manner. Each vertex v_i is associated with a energy-time label set $LABEL(i)$. Initially all label sets are empty except $LABEL(0)$ which contains only one energy-time label $(0, 0)$. The whole process consists of $r+1$ iterations. In the i^{th} iteration, we generate all paths from v_0 to v_i from $LABEL(i-1)$ and store them in $LABEL(i)$. At the end, we just select the energy-time label with the minimum energy and with time no greater than D, from $LABEL(r+1)$, as the solution to the problem.

Unfortunately, the size of $LABEL(i)$ may suffer from exponential growth in this naive approach. To prevent this from happening, the key idea is to reduce and limit the size of $LABEL(i)$ after each iteration by eliminating some of the energy-time labels in $LABEL(i)$. There are two types of eliminations: one that does not affect the optimality of the solution and one that may affect optimality but still allows for performance guarantee.

5.2 Eliminations not Affecting Optimality

There are three eliminations for $LABEL(i)$ that do not affect the optimality of the solution:

1. For any energy-time label l in $LABEL(i)$, if $l.t + \frac{\sum_{i<j\leq r} s_j}{freq_M} > D$, we eliminate this label. This is because even if the maximum frequency is used after this point, the deadline will still be missed; this implies label l will not lead to any feasible solution.

2. For the second elimination, we need to compare two energy-time labels.

   Definition 3. *Let (e_1, t_1) and (e_2, t_2) be two energy-time labels. We say that (e_1, t_1) dominates (e_2, t_2)(denoted by $(e_2, t_2) \prec (e_1, t_1)$) if $e_1 \leq e_2$ and $t_1 \leq t_2$.*

   The dominance relation on a set of energy-time labels is clearly a partial ordering on the set. If $(e_2, t_2) \prec (e_1, t_1)$, this means (e_2, t_2) will not lead to any solution better than the best solution which (e_1, t_1) leads to. Therefore, we eliminate all energy-time labels that are dominated by some other energy-time label in the same label set.

   To facilitate this elimination, the energy-time labels in label sets are stored in decreasing order of the energy component, breaking ties with smaller time component coming ahead. Thus, this elimination can be performed using a linear scan of the label set.

3. For any energy-time label l in $LABEL(i)$ surviving the previous two eliminations, one can use some heuristic to compute the upper bound of the optimal solution. One simple heuristic is to first find the optimal continuous frequency assuming that the task will run for the worst-case cycles, round it up and then compute the expected energy consumption assuming that this frequency is used from this point on. This energy plus $l.e$ is an estimate of the upper bound of the optimal solution. Note that this operation can be done in constant time. We find the smallest estimated upper bound among all energy-time labels in $LABEL(i)$, say U, then we do a linear scan of $LABEL(i)$ to eliminate all energy-time labels whose energy is greater than U.

With the eliminations above, the size of $LABEL(i)$ would decrease substantially. Notice that at this point the optimal solution is guaranteed to be found. However, the running time of the algorithm still has no polynomial time bound guarantee. Inspired by the fully polynomial time approximation scheme (FPTAS) of the subset-sum problem [5], we obtain a FPTAS for our problem, presented next. The FPTAS, further reduces the size of $LABEL(i)$.

5.3 Eliminations Affecting Optimality

The intuition for the FPTAS is that we need to further trim each $LABEL(i)$ at the end of each iteration. A trimming parameter δ $(0 < \delta < 1)$ will be used to direct the trimming. To trim a energy-time label set L by δ means to remove as many energy-time labels as possible, in such a way that if L' is the result of trimming L, then for every energy-time label l that was removed from L, there is an energy-time label $l' < l$ still in L' such that $l.t > l'.t$ and $\frac{l'.e - l.e}{l.e} \le \delta$ (or, equivalently, $l.e \le l'.e \le (1+\delta) \cdot l.e$). This means that the energy difference between these labels is smaller than δ.

Such a l' can be thought of as "representing" l in the new energy-time label set L'. Note that $L' \subseteq L$. Let the *performance guarantee* be ϵ ($0 < \epsilon < 1$), which means that the solution will be within a factor of $1 + \epsilon$ of the optimal solution. After the first type of elimination described in Section 5.2, $LABEL(i)$ is trimmed using a parameter $\delta = \frac{\ln(1+\epsilon)}{r+1}$. The choice of δ shall be clear later in the proof of Theorem 2.

The procedure TRIM (shown in Figure 3) performs the second type of elimination for label set L. Notice that at this point, the energy-time labels in label sets are stored in decreasing order on the energy component(or in increasing order on the time component).

```
PROCEDURE TRIM($L = [l_1, l_2, \ldots, l_{|L|}]$,$\delta$)
$L' := \{l_1\}$
$last := l_1$
for $i := 2$ to $|L|$ do
   if $last.e > (1+\delta) \cdot l_i.e$ then
      append $l_i$ onto the end of $L'$
      $last := l_i$
return $L'$
END
```

Figure 3: TRIM(L,δ)

5.4 The PPACE Algorithm

The PPACE algorithm is shown in Figure 4. For the sake of simplicity and clarity of presentation, the described algorithm is not the most efficient. We have a smarter implementation that makes the algorithm run faster but still has the same asymptotic behavior.

Considering the speed change overhead

The problem formulation in (1)-(2) cannot capture the speed change overheads. This means that GRACE and PACE cannot deal with the speed change overheads simply because their solutions are based on the mathematical solution to (1)-(2). However, the PPACE algorithm can take into consideration the speed change overheads with a slight modification. Since the time penalty and energy penalty for a speed change depend on the speed before changing and the speed after changing [4], we only need to modify Line 7 of the PPACE algorithm such that if the chosen frequency for the current phase is different from that for the previous phase, we add the energy overhead to the energy component and add the time penalty to the time component of the newly generated energy-time label.

In our experience with PPC750 embedded processors, we have observed (and the datasheets from the manufacture dictate) that the CPU will halt for a significant amount of time (10-50ms) each time the speed changes. The AMD PowerNow! processors have the same characteristic. For these types of processors it is important to take into account the speed change overhead. However, for fair comparison with PACE and GRACE, we assume the speed change overhead is zero throughout the paper.

```
    ALGORITHM PPACE($\epsilon$)
1.  for $i := 1$ to $r+1$ do
2.     $LABEL(i) := \phi$
3.  $LABEL(0) = \{(0,0)\}$
4.  for $i := 1$ to $r+1$ do
5.     for each label $l \in LABEL(i-1)$ do
6.        for each available frequency $f$ do
7.           $LABEL(i) := LABEL(i) \cup (l + (s_i F_i e(f), \frac{s_i}{f}))$
8.     remove all $l \in LABEL(i)$ such that
       $l.t + \frac{\sum_{i<j\le r} s_j}{freq_M} > D$
9.     remove all $l \in LABEL(i)$ such that $l \prec l'$
       where $l' \ne l$ and $l' \in LABEL(i)$
10.    estimate the upper bound $U$, remove all
       $l \in LABEL(i)$ such that $l.e > U$
11.    $LABEL(i)$ = TRIM($LABEL(i)$, $\frac{\ln(1+\epsilon)}{r+1}$)
12. return the label $l \in LABEL(r+1)$ with the
    minimum energy component
    END
```

Figure 4: PPACE algorithm

Next, we show some results on the algorithm PPACE. First, notice that line 11 of the algorithm corresponds to the TRIM procedure, that is, the second type of eliminations (see Section 5.3). Let $LABEL'(i)$ ($0 \le i \le r+1$) be the label set obtained if line 11 of the algorithm PPACE is omitted. Notice that $LABEL(i) \subseteq LABEL'(i)$ and the optimal solution is in $LABEL'(r+1)$. By comparing $LABEL(i)$ and $LABEL'(i)$, we have the following lemma:

LEMMA 1. *For every energy-time label $l' \in LABEL'(i)$, there exists a label $l \in LABEL(i)$ such that $l'.e \le l.e \le (1+\delta)^i \cdot l'.e$ and $l'.t \ge l.t$*

Lemma 1 shows how the error accumulates after each iteration when comparing label sets obtained with the second type of elimination and label sets obtained without the second type of elimination. For the details of the proof, see [18].

THEOREM 2. *PPACE is a fully polynomial-time approximation scheme, that is, the solution that PPACE returns is within a factor of $1+\epsilon$ of the optimal solution and the running time is polynomial in $1/\epsilon$.*

PROOF. Let l^* denote an optimal solution. Obviously $l^* \in LABEL'(r+1)$. Then, by Lemma 1 there is a $l \in LABEL(r+1)$ such that

$$l^*.e \le l.e \le (1+\delta)^{r+1} \cdot l^*.e$$

Since

$$\begin{aligned} (1+\delta)^{r+1} &= \left(1 + \frac{\ln(1+\epsilon)}{r+1}\right)^{r+1} \\ &\le e^{\ln(1+\epsilon)} \\ &= 1+\epsilon \end{aligned}$$

then

$$l.e \leq (1+\epsilon) \cdot l^{*}.e$$

Therefore, the energy returned by PPACE is not greater than $1+\epsilon$ times the optimal solution.

To show that its running time is polynomial in $1/\epsilon$, we first need to derive the upper bound on the size of $LABEL(i)$. Let $LABEL(i) = [l_1, l_2, \ldots, l_k]$ after trimming. We observe that the energies of any two successive energy-time labels differ by a factor of more than $(1+\delta)$ (otherwise, we have already eliminated it). In particular,

$$\begin{aligned} l_1.e &> (1+\delta) \cdot l_2.e \\ &> (1+\delta)^2 \cdot l_3.e \\ &\vdots \\ &> (1+\delta)^{k-1} \cdot l_k.e \end{aligned}$$

Moreover, $l_1.e \leq e(freq_M) \sum_{0 \leq j \leq i} s_j \cdot F_j$ and $l_k.e \geq e(freq_1) \sum_{0 \leq j \leq i} s_j \cdot F_j$. Let $\lambda = \frac{e(freq_M)}{e(freq_1)}$ (i.e., the ratio of the energies when running with the highest and lowest frequencies), then

$$(1+\delta)^{k-1} < \frac{l_1.e}{l_k.e} \leq \lambda$$

or, equivalently

$$\begin{aligned} k &< 1 + \frac{\ln \lambda}{\ln(1+\delta)} \\ &\leq 1 + \frac{\ln \lambda}{\frac{\delta}{1+\delta}} \\ &= 1 + \ln \lambda \cdot (1 + \frac{1}{\delta}) \\ &= 1 + \ln \lambda \cdot \left(1 + \frac{r+1}{\ln(1+\epsilon)}\right) \\ &= O(\frac{r \cdot ln\lambda}{\epsilon}) \end{aligned} \qquad (3)$$

Now we can derive the running time of PPACE. There are $r+1$ iterations (line 4). The processing time in each iteration is linear in the number of energy-time labels (line 5) generated, which, from Equation 3, is in $O(\frac{r \cdot ln\lambda}{\epsilon})$. The running time is also a function of the number of speeds, M (line 6). Thus, the total running time is $O(r^2 \cdot M \cdot \frac{ln\lambda}{\epsilon})$. □

In fact, the running time of PPACE depends entirely on the total number of energy-time labels stored in all the vertices which is $\sum_{i=0}^{r+1} LABEL(i)$. The approximation guarantee reflects the performance of the algorithm only on the most pathological instances. In our experimental results, we observe that using $\epsilon = 5\%$ usually gives a solution which is within 0.1% of the optimal solution and the total number of energy-time labels stored in all the vertices is much smaller than that acquired from the worst-case analysis.

The running time of PPACE is obviously higher than PACE and GRACE, which is $O(r \cdot M)$. However, as in PACE, the speed schedules can be precomputed and stored in a table [12]. Determining a speed schedule on-the-fly involves only a binary search through the table. Thus, PPACE is very suitable for dynamically discovering the close-to-optimal speed schedules.

The space complexity of PPACE, assuming the transition points are chosen as the boundaries of histogram bins (as in [20]) or according to PACE's heuristic, will be $O(M \cdot \frac{r \cdot ln\lambda}{\epsilon})$. This is because speeds are nondecreasing as the task progresses, so the speed schedule can be expressed using M integers, as mentioned in Section 4. We attach a speed schedule to each energy-time label in the label sets and hence the space complexity. If the transition points do not satisfy Theorem 1, then the space complexity becomes $O(\frac{r^2 \cdot ln\lambda}{\epsilon})$.

6. PERFORMANCE EVALUATION

We compare the three algorithms (GRACE, PACE, and PPACE) using different power models and cycle distributions for tasks. The power models used were IBM PPC405LP (Table 1), Intel XScale (Table 2), and a synthetic ideal processor (Table 3). The idle power for PPC405LP is 12mW. Since we do not have the idle power number for XScale, we assume that its idle power is one half of the power consumed at the minimum frequency, that is, 40mW. The synthetic ideal processor strictly conforms to the $p = f^3$ power-frequency relation and has 10 discrete frequencies ranging from 100MHz to 1000MHz with 100MHz step; its idle power is zero. With this synthetic processor, the power function in PACE and GRACE is exact and represents the best case for those algorithms.

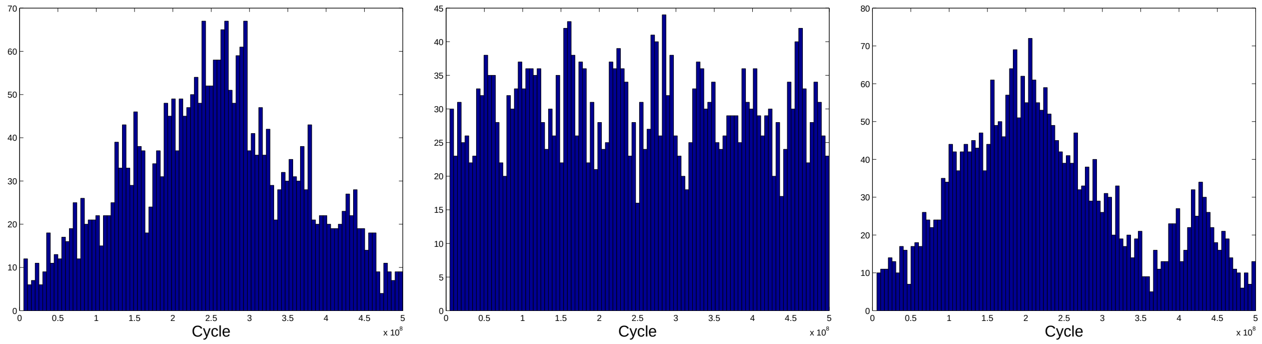
The length of tasks in cycles follow normal, uniform and bimodal distributions (Figure 5 shows the histograms of the three distributions). The worst-case number of cycles, WC, is $500,000,000$ and the minimum number of cycles is $5,000,000$. The number of transition points is 100 and they are placed evenly in the range of $[1, WC]$. This implies the distributions can be expressed with 100-bin histograms (according to our experience, 100 bins are enough to describe the cycle distribution).

We define the relative error for any algorithm that returns the expected energy consumption Egr to be $\frac{Egr - OPT}{OPT}$, where OPT is the optimal solution. We compute OPT using the PPACE algorithm without doing the trimming operation at the expense of much longer running time. As usual with FPTAS algorithms, we set $\epsilon = 0.05$ for PPACE when comparing it with GRACE and PACE. For all experiments, we vary the slack available for power management in the X axis. The slack is changed by varying the deadline from $\frac{WC}{freq_M}$ to $\frac{WC}{freq_1}$ (increasing the deadline will increase the slack, that is, will increase the allotted time for the task, thus allowing for better power management).

We do not show all the results (for all distributions) because results for all distributions were similar (for comprehensive results, see [18]). PPACE performs very well. As shown in Figures 6(b)-8(b), the measured relative errors are always below 0.1%, 1.5%, 2.5% for $\epsilon = 0.05, 0.1, 0.15$, respectively. This means that, in practice, the algorithms perform much better (approximately a 5-fold increase) than the performance guarantees they offer. While still meeting all deadlines, this knowledge allows system designers to set the parameter ϵ higher than required, in order to speed up the algorithm execution.

We compare PPACE, GRACE, and PACE in Figures 6(a)-8(a). We can see that (as mentioned above) PPACE's performance is very close to optimal.

On the other hand, the relative errors of GRACE and PACE depend on the power model, cycle distribution, and the deadline. For the ideal processor (Figure 6(a)) when the relative errors of GRACE and PACE only result from

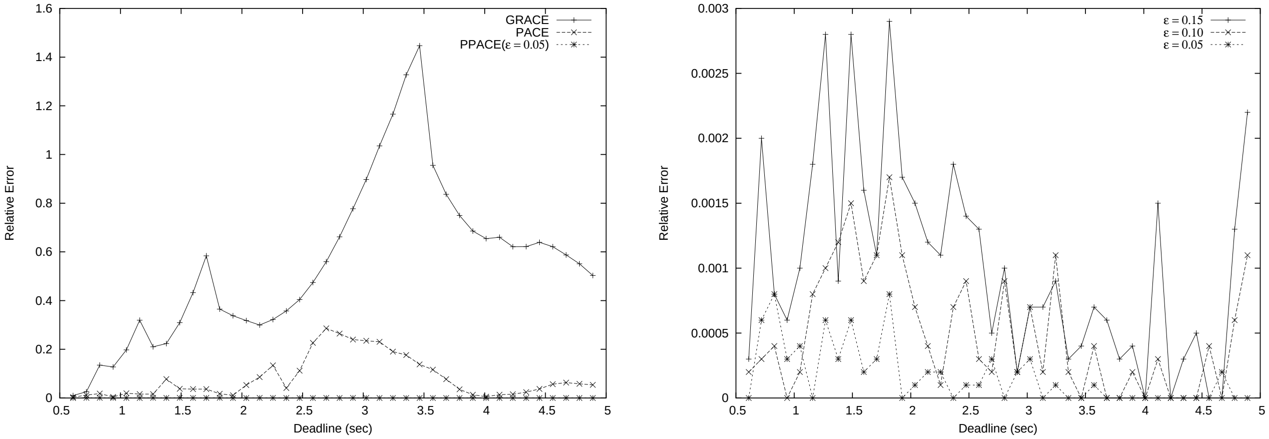


(a) Normal

(b) Uniform

(c) Bimodal

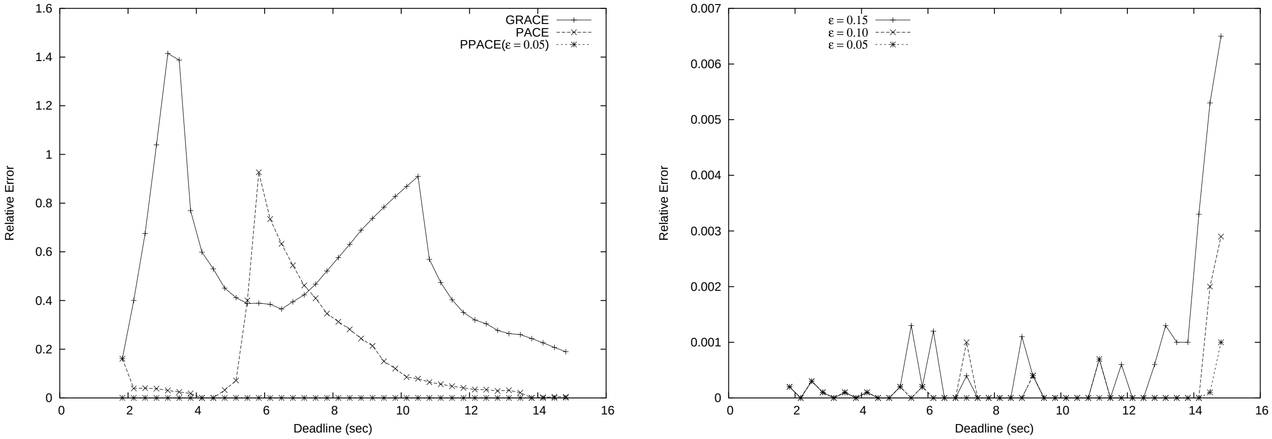
Figure 5: Cycle distributions



(a) Comparison of GRACE, PACE, and PPACE

(b) Effect of different ϵ on the relative error

Figure 6: Ideal processor, bimodal distribution



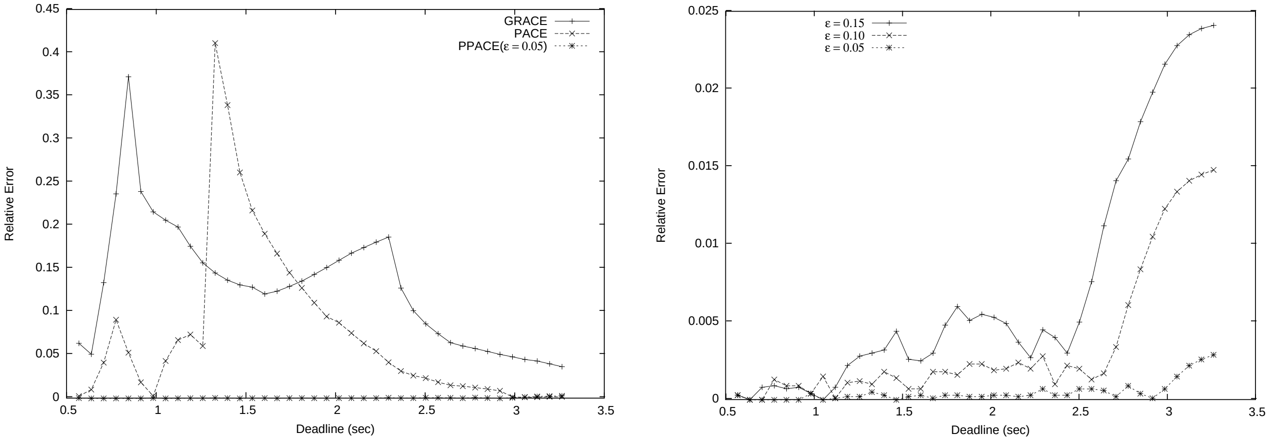
(a) Comparison of GRACE, PACE, and PPACE

(b) Effect of different ϵ on the relative error

Figure 7: PPC405LP processor, bimodal distribution

Table 3: Ideal processor speed settings and power consumptions

Speed (MHz)	100	200	300	400	500	600	700	800	900	1000
Power (mW)	100^3	200^3	300^3	400^3	500^3	600^3	700^3	800^3	900^3	1000^3



(a) Comparison of GRACE, PACE, and PPACE

(b) Effect of different ϵ on the relative error

Figure 8: XSCALE processor, bimodal distribution

rounding, PACE always performs better than GRACE; we note that even in the ideal processor case, PACE's relative error can be up to 40% [18]. But for the PPC405LP and XScale, neither PACE nor GRACE are clear winners. For example, for PPC405LP and the bimodal cycle distribution (similar results are obtained for the uniform and normal distributions), when the deadline is small or large, PACE does better than GRACE. But when the deadline is in the middle range, it is the other way around. GRACE's relative error can be up to 140% and PACE's can be up to 100%. In fact, the effect of two approximations(i.e., using well-defined functions to approximate the actual power function and rounding) could compound or cancel each other out. Thus the end effect is unpredictable. In other words, GRACE and PACE are not as stable as PPACE.

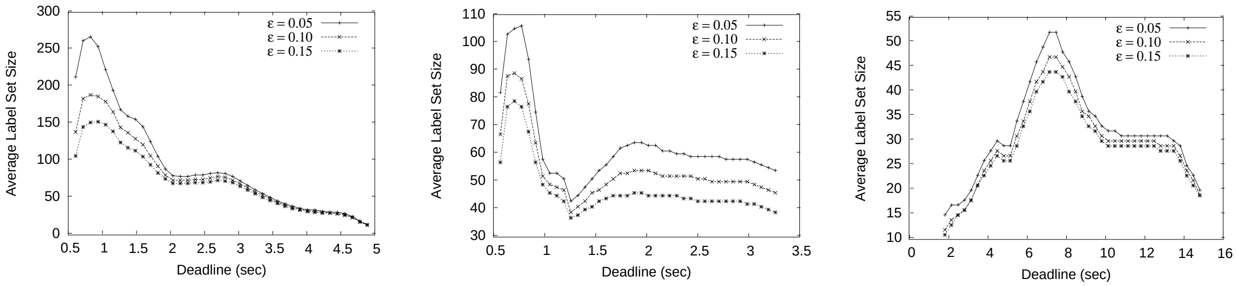
From Figure 9 we can also see that the average size of label sets increases when ϵ decreases, but they are relatively small (specially for the two real-life embedded processors shown in Figures 9(b) and 9(c)). It is also true that when M, the number of discrete speeds, increases, the label set size increases; this can be seen comparing the ideal processor, XScale, and PPC405LP results. (Note that the scale of the Y axis is different for each processor model.)

7. CONCLUSIONS

We studied, from a practical point of view, the problem of finding optimal speed schedules for embedded systems with tasks whose computational requirements are only known probabilistically. We took the processor idle power into account and thus avoided the abnormality of inefficient frequencies. We solved the problem by using efficient combinatorial algorithms that apply directly to the case of discrete speeds and do not make any assumption about the shape of the power function. In summary:

1. We provide an efficient approximation algorithm with performance guarantee for computing the near-optimal solution. We also identify the conditions when a simple, fast, exact algorithm is applicable;
2. We showed how our algorithms can consider non-zero idle power and the overhead of changing speeds;
3. By using our algorithm, we provide an accurate *estimation* on energy consumption of the optimal solution to the problem (because our FPTAS algorithms are so close to optimal, we deem this a good estimation);
4. We evaluated the effect of using well-defined functions and rounding in GRACE and PACE. We found that: (1) PACE outperforms GRACE in most cases; (2) When the actual power function is close to well-defined functions, PACE performs reasonably well; (3) When the deadline is at two extremes, that is, tight and loose, PACE performs much better than the other cases.

Future work will be evaluating our algorithm in the real-world embedded systems. Currently we are investigating two applications, known as *Event Extraction* and *Complex Ambiguity Function* (or *CAF* for short) with our industrial research partners. The computing system under consideration is a uniprocessor real-time system that serves incoming requests on a first-come-first-serve basis without preemption. In CAF, the requests arrive at fixed arrival rate while the arrival rate may vary in Event Extraction. Our algorithm is expected to work very well in CAF. However, the problem is more challenging in Event Extraction. This is because we have the situation where other requests could arrive during the execution of a request. Thus, we are faced with the problem of minimizing the expected energy consumption by multiple tasks subject to different deadlines. We will investigate how to extend our algorithm to deal with the new problem.



(a) Ideal processor

(b) XScale processor

(c) PP405LP processor

Figure 9: Average Label Set Size vs. deadline in PPACE with bimodal distribution

8. REFERENCES

[1] N. AbouGhazaleh, D. Mossé, B. Childers, R. Melhem, and Matthew Craven. Collaborative Operating System and Compiler Power Management for Real-Time Applications. In *The 9th IEEE Real-Time Embedded Technology and Applications Symposium(RTAS 2003)*, May 2003.

[2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of the 22nd Real-Time Systems Symposium (RTSS'01)*, 2001.

[3] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: design and modeling challenges for next generation microprocessors. *IEEE Micro*, 20(6), 2000.

[4] T. Burd and R. Brodersen. Design issues for Dynamic Voltage Scaling. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED-00)*, June 2000.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, 1990.

[6] F. Gruian. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *International Symposium on Low Power Electronics and Design*, Aug. 2001.

[7] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of the 19th IEEE Real-Time systems Symposium (RTSS'98)*, Madrid, Spain, December 1998.

[8] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *International Symposium on Low Power Electronics and Design*, pages 197–202, Aug. 1998.

[9] S. Krantz, S. Kress, and R. Kress. *Jensen's Inequality.* Birkhauser, 1999.

[10] J. Lorch. *Operating Systems Techniques for Reducing Processor Energy Consumption*. PhD thesis, University of California at Berkeley, 2001.

[11] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *ACM SIGMETRICS*, June 2001.

[12] J. Lorch and A. Smith. Operating system modifications for task-based speed and voltage scheduling. In *the First International Conference on Mobile Systems, Applications, and Services (MobiSys)*, May 2003.

[13] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading, Massachusetts, 1984.

[14] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, May 2003.

[15] N.H.E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, Reading, MA, 1993.

[16] F. Xie, M. Martonosi, and S. Malik. Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *Programming Language Design and Implementation (PLDI 2003)*, June 2003.

[17] Intel XScale Microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[18] R. Xu, C. Xi, R. Melhem, and D. Mossé. Discrete PACE. Technical Report TR-04-108, Department of Computer Science, University of Pittsburgh, 2004. http://www.cs.pitt.edu/∼xruibin/publications/dpace.pdf.

[19] F. Yao, A. Demers, and S.Shankar. A Scheduling Model for Reduced CPU Energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

[20] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *ACM SOSP'03*, October 2003.