High-Level Power Management of Embedded Systems with Application-Specific Energy Cost Functions*

Youngjin Cho and
Naehyuck Chang†
School of CSE
Seoul National University
Seoul, Korea
naehyuck@snu.ac.kr

Chaitali Chakrabarti
Department of Electrical Engineering
Arizona State University
Tempe, AZ
chaitali@asu.edu

Sarma Vrudhula
CSE Department
Arizona State University
Tempe, AZ
vrudhula@asu.edu

ABSTRACT

Most existing dynamic voltage scaling (DVS) schemes for multiple tasks assume an energy cost function (energy consumption versus execution time) that is independent of the task characteristics. In practice the actual energy cost functions vary significantly from task to task. Different tasks running on the same hardware platform can exhibit different memory and peripheral access patterns, cache miss rates, etc. These effects results in a distinct energy cost function for each task.

We present a new formulation and solution to the problem of minimizing the total (dynamic and static) system energy while executing a set of tasks under DVS. First, we demonstrate and quantify the dependence of the energy cost function on task characteristics by direct measurements on a real hardware platform (the TI OMAP processor) using real application programs. Next, we present simple analytical solutions to the problem of determining energy-optimal voltage scale factors for each task, while allowing each task to be preempted and to have its own energy cost function. Based on these solutions, we present simple and efficient algorithms for implementing DVS with multiple tasks. We consider two cases: (1) all tasks have a single deadline, and (2) each task has its own deadline. Experiments on a real hardware platform using real applications demonstrate a 10% additional saving in total system energy compared to previous leakage-aware DVS schemes.

Categories and Subject Descriptors

D.4.1 [**Operating System**]: Process Management

General Terms

Algorithms

1. INTRODUCTION

Despite many advances over the last decade in semiconductor technology, together with circuit and system-level low-power design techniques, reducing power consumption continues to be a major challenge in the design of microelectronic systems. At the system level, a system can be viewed as collection of interacting components, and the accepted approach to power saving is to transition components to low power or standby states when they are idle, so as to reduce the wastage of energy. This approach is known as dynamic power management (DPM) [1], and the main challenge that is poses is to determine the occurrence and duration of idle periods. A large body of literature addresses this problem, and the solutions proposed range from fixed time-out policies [2] to a wide variety of predictive techniques. For detailed comparative surveys of DPM policies, the reader is referred elsewhere [3], [4].

Transitioning a component to a low-power state during an idle period reduces the energy wasted during that period, but does not reduce the active power. In many modern processors, a trade-off between active power and performance can be made by dynamic voltage and frequency scaling (DVS). In general, a processor's workload consists of a set of tasks, each with its own soft or hard deadline. The problem then is to determine the optimal factors by which the supply voltage and clock frequencies should be scaled to minimize energy consumption while meeting performance constraints. Like DPM, DVS has received a lot of attention from previous researchers [5–11]. The most common approach to DVS is to predict the workload expressed, for instance, as CPU utilization at the start of each scheduling interval, based on some measure of average utilization over past intervals, and then to scale the processor's voltage and frequency to a minimum feasible value that will allow completion of each task within its deadline, or simply before the start of the next cycle [9, 12]. Several researchers have considered the scheduling of real-time systems to target energy efficiency [6, 13–16]. The general approach is to estimate task execution times (e.g. by monitoring the cycle usage of tasks, constructing a probability distribution of execution times, etc.) and then to use heuristics to scale the processor's frequency.

Most of the existing DVS techniques have one important shortcoming: they do not use any explicit or measured relationship between the energy consumption and execution time of tasks. Task execution times are estimated in various ways, and then the factors used to scale the voltage and clock frequency are selected heuristically. This approach ignores the important and frequently observed fact [17] that the relationship between energy and execution time is strongly dependent on the characteristics of the application program, such as its memory and peripheral access patterns, cache miss rates, and the impact of scaling on the efficiency of the DC-DC converter [18]. Equally important is leakage, which can offset any dynamic power savings as the scaling factors become large. Accounting for leakage when determining the scale factors can lead to a better combination of DPM and DVS [19, 20]. Off-chip memory devices are dominant power consumers and they do not

*This research was funded in part by LG Yonam Foundation.

†Corresponding author

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

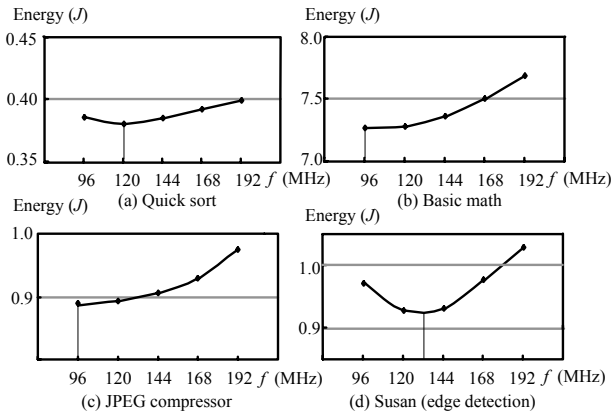


Figure 1: Energy consumption measured at different CPU speeds of the TI OMAP OSK platform, running for different applications.

generally allow supply voltage scaling. Although a larger scaling factor may achieve a greater reduction in CPU power, the longer execution time may increase the power consumption of memory devices. These and other factors can be more accurately accounted for if scaling factors are determined from an accurate understanding of the relationship between total energy consumption and execution time for each task.

To justify the preceding assertions, we used the TI OMAP5912 OSK board[1] to collect accurate energy estimates as a function of execution times for several real-world application programs. The board includes a TI OMAP processor, a DC-DC converter, memory and peripherals. Figure 1 shows total energy consumption measured at different clock frequencies for four different application programs. These graphs clearly differ significantly and thus we expect the optimal scale factors to be different.

The quicksort application (Figure 1(a)) is memory-intensive while the basic mathematics application (Figure 1(b)) is computation-intensive. More memory accesses increases the value of using the optimal scaling factors, an observation that is consistent with previous work [17]. Note that the total energy values are strongly dependent on execution times: in general, a longer execution time results in a larger energy requirement. The JPEG compressor[2] (Figure 1(c)) is also computation-intensive, and its energy function has a similar shape to the basic mathematics application. However, Susan, an image edge detector, shows a distinctly different energy function even though it is also somewhat computation-intensive. This is because Susan contains routines that read from and write to Flash memory before and after the edge-detection routines.

In this paper we present an analytical solution to the problem of determining the energy-optimal factors by which the voltage and clock frequency should be scaled so as to minimize the total energy for executing a set of tasks, each of which is associated with its own energy function (total energy versus execution time). We examine two scenarios: (1) a set of tasks that have to be executed in a given sequence before a shared deadline, and (2) a set of tasks to be executed, each with its own start time and deadline, in an environment that allows tasks to be preempted. Then we describe our experimental platform on which energy measurements of real-world applications can be made, providing the relationships

[1]For additional details on the experimental work and results see Section 5.

[2]The original JPEG compressor application includes flash memory read and write operations, but we measured the execution energy after the flash read routine has completed and before the flash write routine is initiated.

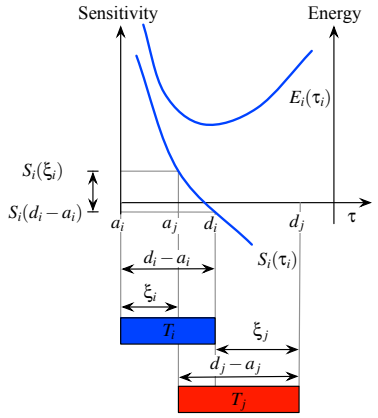


Figure 2: Determination of the sensitivity intervals.

between energy and execution time data for the applications. These results allow us to derive optimal scaling factors and we will go on to compare the resulting energy savings with existing DVS schemes.

2. PROBLEM FORMULATION

We are given a set of N tasks $\Gamma = \{T_1, \cdots, T_N\}$. Each task T_i is described by a four-tuple $(\tau_i, E_i(\tau_i), a_i, d_i)$, where τ_i is the execution time, $E_i(\tau_i)$ is the function that expresses the relationship between the energy and execution time, a_i is the arrival time, and d_i is the deadline. The energy function $E(t)$ is convex. The sensitivity of task T_i is defined as

$$S_i(\tau_i) = -\frac{dE_i(\tau_i)}{d\tau_i}. \tag{1}$$

Since $E(\tau)$ is convex, $S(\tau)$ is a monotonically decreasing function of τ (i.e., $S'(\tau) \leq 0$). A large positive value of $S_i(\tau_i)$ indicates that a relatively small decrease in the execution time would result in a large increase in energy and vice verse. Conversely, if $S_i(\tau_i)$ is negative, reducing the execution time will result in a lower the energy cost.

We can associate a set of sub-intervals I_i of $[a_i, d_i]$ with each task in Γ, in such a way that no other task overlaps with task T_i in those sub-intervals. A set I_i may be empty. The sum of the lengths of all the sub-intervals in I_i, denoted by ξ_i, is called the preemption-free execution time of T_i. That is,

$$I_i = \left\{ [a_i, d_i] - \cup_{i \neq j} [a_i, d_i] \cap [a_j, d_j] \right\}, \tag{2}$$

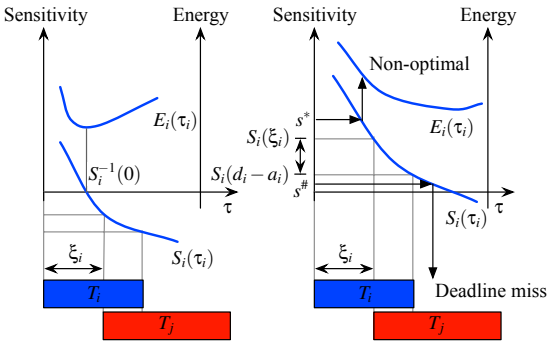
$$\xi_i = \sum_{\gamma \in I_i} |\gamma|. \tag{3}$$

Note that ξ_i represents the total dedicated time available for task T_i in the interval $[a_i, d_i]$. If there is no preemption then $\xi_i = d_i - a_i$, and if the whole time interval occupied by T_i overlaps the intervals of other tasks then $\xi_i = 0$. The sensitivity interval of task T_i is defined as $[S(d_i - a_i), S_i(\xi_i)]$, and will be discussed in the next section. Figure 2 shows examples of the preemption-free execution time and the sensitivity interval.

The problem is to determine the set of execution times $X = (\tau_1, \cdots, \tau_N)$ that minimizes the total energy $\sum_{i=1}^{N} E_i(\tau_i)$. Such a set will be called an energy-optimal schedule. Two different constraints apply:

1. For a given task sequence, the constraint applies to the total completion time of all tasks. That is

$$\sum_{i=1}^{N} \tau_i \leq D, \tag{4}$$



(a) An example of Property 2 (b) An example of Properties 3 and 4

Figure 3: Examples of Properties 2, 3, and 4.

where $D = \max(d_i) - \min(a_j)$, T_i and $T_j \in \Gamma$.

2. If each task has its own deadline, the constraint is simply $\tau_i \leq d_i - a_i, \forall T_i \in \Gamma$.

3. SOLUTION METHODS

3.1 Basic Properties of the Sensitivity Function

We start by clarifying the characteristics of the sensitivity function, which is the most important metric in finding a globally optimal energy consumption for all the tasks.

Property 1. *The sensitivity and the execution time of task T_i have a one-to-one correspondence.*

The sensitivity function is defined as the derivative of the energy cost function, which is convex. Thus the sensitivity function is monotonically decreasing, which proves Property 1. In other words, assigning an execution time to each task is virtually identical to assigning a sensitivity value to each task.

Property 2. *For a task i, if $S_i(\xi_i) < 0$, $\tau_i = S_i^{-1}(0)$ is the energy-optimal execution time of task T_i.*

Property 2 means that we can schedule task T_i without considering the other tasks, as long as the energy-optimal execution time lies within the preemption-free interval, as shown in Figure 3(a). In this case, obtaining the energy-optimal solution is straightforward and independent of the schedules of other tasks. However, if $S_i(\xi_i)$ is larger than zero, the energy-optimal execution time of the task must be larger than the preemption-free interval. In this case, the execution time of the task need to be scaled within a range extending from the preemption-free interval to the deadline, so as to meet both the energy-optimality and the real-time constraints. Since sensitivity has a one-to-one correspondence with execution time, sensitivity also has a scaling range, which is the sensitivity interval $[S(d_i - a_i), S_i(\xi_i)]$.

Let us suppose that a task T_i has a sensitivity value $s^\#$ or s^*, as shown in Figure 3(b).

Property 3. *If $s^\#$ is smaller than the minimum value $S_i(d_i - a_i)$ of the sensitivity interval, then the execution time $\tau^\#$ that corresponds to $s^\#$ is longer than the deadline constraint $d_i - a_i$. This violates the deadline constraint.*

Property 4. *If s^* is larger than the maximum value $S_i(\xi_i)$ of the sensitivity interval, then the execution time τ^* that corresponds to s^* is shorter than the preemption-free time. So, τ^* cannot be the energy-optimal execution time.*

3.2 Energy-optimality in Non-uniform Energy Cost Functions for Each Task

We will now show how we can obtain the energy-optimal solution for tasks that have non-uniform energy cost functions, by looking at their sensitivity. We begin by considering a task set without individual deadline constraints. We then go on to the more practical case where there is an individual real-time constraint for each task.

Lemma 1. *If there are no individual task deadlines, but there is a constraint D on the total completion time, and if $X = (\tau_1, \cdots, \tau_N)$ is an energy-optimal solution, then $S_i(\tau_i) = S_j(\tau_j)$; and the energy-optimal solution is one of the following:*

$$S_i(\tau_i) = 0 \text{ and } \sum_{i=1}^{N} \tau_i < D, \forall T_i \in \Gamma, \tag{5}$$

$$S_i(\tau_i) = \lambda > 0 \text{ and } \sum_{i=1}^{N} \tau_i = D, \forall T_i \in \Gamma. \tag{6}$$

Proof: [Lemma 1] The objective function and the completion time constraint are combined to form the Lagrangian. Let β denote a slack variable that converts the inequality in Equation 4 into an equality. We require that $\beta \geq 0$, and so we introduce the slack variable as a square term[3]. The Lagrangian is given by

$$L(X, \lambda, \beta) = \sum_{i=1}^{N} E_i(\tau_i) + \lambda \left(\sum_{i=1}^{n} \tau_i - D + \beta^2 \right). \tag{7}$$

The partial derivatives of L with respect to τ_i, β and λ have to vanish at the minimum. This results in three equations which must be solved simultaneously:

$$\frac{\partial L}{\partial \tau_i} = -S_i(\tau_i) + \lambda = 0, \ \forall i, \tag{8}$$

$$\frac{\partial L}{\partial \beta} = 2\lambda\beta = 0, \tag{9}$$

$$\frac{\partial L}{\partial \lambda} = \sum_{i=1}^{N} \tau_i - D + \beta^2 = 0. \tag{10}$$

The solution of Equation (9) can be either $\lambda = 0$ or $\beta = 0$. If $\lambda = 0$, then the simultaneous equations have an unconstrained solution, given by Equation (8), in which case $S_i(\tau_i) = 0, \ \forall i$. If $\lambda \neq 0$ and $\beta = 0$, then the equations to be satisfied are

$$-S_i(\tau_i) = -\lambda, \ \forall i, \tag{11}$$

$$\sum_{i=1}^{N} \tau_i = D. \tag{12}$$

In either case, S_i is a constant. □

In summary, we note that there are two possibilities for the optimal execution times: (1) the optimal solution is the unconstrained solution, in which case the execution times are simply the roots of the sensitivity function, or (2) the optimal solution is a zero-slack solution and the values of the sensitivity functions of the individual execution times are the same.

3.3 Solution

In this section, we apply the energy-optimal solution of Lemma 1 to real cases where each task has its own deadline constraint, which is different from the assumption in the previous section. This means that the solution of Lemma 1 is only applicable to real cases when additional conditions are met, as follows,

[3]Any non-negative function of β whose minimum is zero can be used.

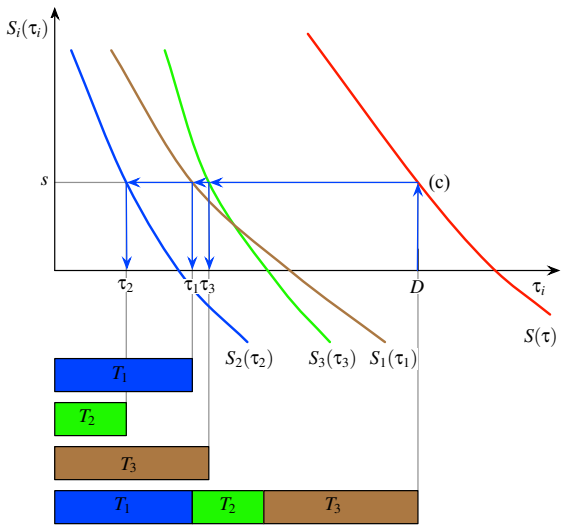


Figure 4: Scheduling of a sequence of tasks with an end-deadline constraint.

Condition 1. $\cup_{i\neq j, T_i \text{ and } T_j \in \Gamma}[a_i,d_i]\cap[a_j,d_j] = [\min(a),\max(d)]$.

Condition 2. $\cap_{T_i\in\Gamma}[S_i(d_i-a_i),S_i(\xi_i)] \neq \phi$.

Intuitively, Conditions 1 and 2 mean that the union of the timing constraint intervals of all the tasks must not be split, and that the sensitivity intervals should have a common range, shared by all tasks.

Condition 3.
(a) $0 \leq S_{min}$ and $\sum_{T_i\in\Gamma} S_i^{-1}(S_{max}) \leq D \leq \sum_{i\in T_s} S_i^{-1}(S_{min})$, or
(b) $S_{min} \leq 0 \leq S_{max}$ and $\sum_{T_i\in\Gamma} S_i^{-1}(0) \leq D$,

where S_{min} and S_{max} are the minimum and the maximum of $\cap_{T_i\in\Gamma}[S_i(d_i-a_i),S_i(\xi_i)]$. That means that the solutions of Lemma 1 are applicable to the system with an individual deadline for each task only when either Condition 3 (a) or (b) as well as Conditions 1 and 2 are satisfied.

We now find the energy-optimal solution when these conditions are met. We will assume that there is a given task set Γ which satisfies Conditions 1 and 2. The satisfaction of these conditions implies that there exists an $s \in \cap_{T_i\in\Gamma}[S_i(d_i-a_i),S_i(\xi_i)]$ such that $S_i(\tau_i) = S_j(\tau_j) = s$ (see Figure 4). Since each of the individual sensitivity functions are monotonically decreasing, their inverses are well-defined, allowing us to write the function $S^{-1}(s)$ as

$$S^{-1}(s) = \sum_{T_i\in\Gamma} S_i^{-1}(s) = \sum_{T_i\in\Gamma} \tau_i = \tau, \qquad (13)$$

where τ is the total execution time of all tasks in Γ, and $S(\tau)$ is the sensitivity of the task set. If the task set Γ satisfies Condition 3(b), then $S^{-1}(0) \leq D$. This means that there exists an unconstrained solution, in which the gradients of the energy cost functions become zero where the execution time of each task corresponds to $s = 0$. Therefore, the energy-optimal execution times are $\tau_i = S_i^{-1}(0)$.

On the other hand, if the task set Γ satisfies Condition 3(a), the energy-optimal execution times are $\tau_i = S_i^{-1}(S(D))$. What should the value of s be so that τ_i is the energy-optimal execution time for task i, $\forall T_i$? Suppose $s > S(D)$. Then (see Figure 4) $\tau_i = S_i^{-1}(s)$, and the solution has a positive slack, which cannot be energy-optimal, because $S(D) > 0$. If $s < S(D)$ then $\sum_{T_i\in\Gamma}\tau_i > D$, and the completion-time constraint is violated. Hence $s = S(D)$.

Table 1: Algorithm for energy-optimal scheduling with non-uniform energy cost functions.

```
01:While (G ≠ φ){
02:   Γ_i is a task set of G;
**:   \\Optional for performance
**:   If (Γ_i does not satisfy Condition 1)}
**:     Divide Γ_i into task sets that satisfy Condition 1;
**:     Remove Γ_i and insert such task sets into G;
**:     Continue; \\Go to the while loop.
**:   }
03:   Γ_temp is a temporary test set that is φ;
04:     Do until (Γ_i satisfies Condition 1, 2 and 3){
05:       \\ Find a task from Γ_i that has the highest S(d−a).
06:       T* = max_{T_k∈Γ_i}(S_k(d_k − a_k));
07:       If (Γ_i does not satisfy Condition 1){
08:         Move tasks whose timing constraint intervals
09:           part from that of T* to Γ_temp;
10:       }
11:       If (Γ_i does not satisfy Condition 2){
12:         Move tasks whose sensitivity intervals
13:           do not overlap that of T* to Γ_temp;
14:       }Else If (Γ_i does not satisfy Condition 3){
15:         \\ Remove a task from Γ_i, which has the lowest S(ξ).
16:         T# = min_{T_k∈Γ_i}(S_k(ξ_k));
17:         Move T# from Γ_i to Γ_temp;
18:       }
19:       \\ Do repeat, because Condition 1 may not be satisfied
20:       \\by the removal at lines 11-18.
21:   }
22:   If (Γ_temp ≠ φ) insert Γ_temp into G;
23:   Find the optimal sensitivity value of Γ_i using Lemma 1;
24:   Remove such Γ_i from G;
25:   Readjust the time intervals considering the removed tasks;
26:}
```

4. SCHEDULING OF TASKS WITH INDIVIDUAL DEADLINE CONSTRAINS

We now address the problem of more general tasks. We will say that a set of tasks Γ_i is compatible if they satisfy Conditions 1, 2 and 3. Ideally we would like to partition the set of tasks Γ into disjoint subsets $(\Gamma_1,\cdots,\Gamma_m)$ such that all the tasks within a subset are compatible. This would be possible if their compatibility is an equivalence relation. Unfortunately, it is not transitive. The situation is actually worse, in that a set Γ_1 may be compatible, while $\Gamma_2 \supset \Gamma_1$ is not compatible, but $\Gamma_3 \supset \Gamma_2$ is compatible. This makes the problem of partitioning the tasks into maximal compatible classes NP-hard. For this reason, we present a heuristic based on the properties of the sensitivity functions: we partition the set of tasks into subsets that are as large as possible so that the execution times of the tasks in each subset can be determined by the methods applied to a task sequence.

A group of task sets is represented by $G = \{\Gamma_1,...,\Gamma_m\}$, where Γ_i is a task set. The initial task set is the group of all tasks $\Gamma_1 = \{T_1,...,T_N\}$. We start by removing trivial tasks: for all tasks $T_i \in \Gamma_1$, we check whether $S_i(\xi_i) < 0$; if so, we exclude these tasks from Γ_1 by the application of Property 2. An overview of the resulting energy-optimal scheduling algorithm is given in Table 1.

Figure 5 illustrates a scheduling example with six tasks. The vertical lines represent the sensitivity intervals. Arrowheads are shown on sensitivity intervals for which $\xi_i = 0$ and thus $S_i(\xi_i) = \infty$. The thick black lines represent the task or tasks that are currently being scheduled. The red line shows a sensitivity interval being eliminated by the scheduling of other tasks. The circles represent the resulting schedule.

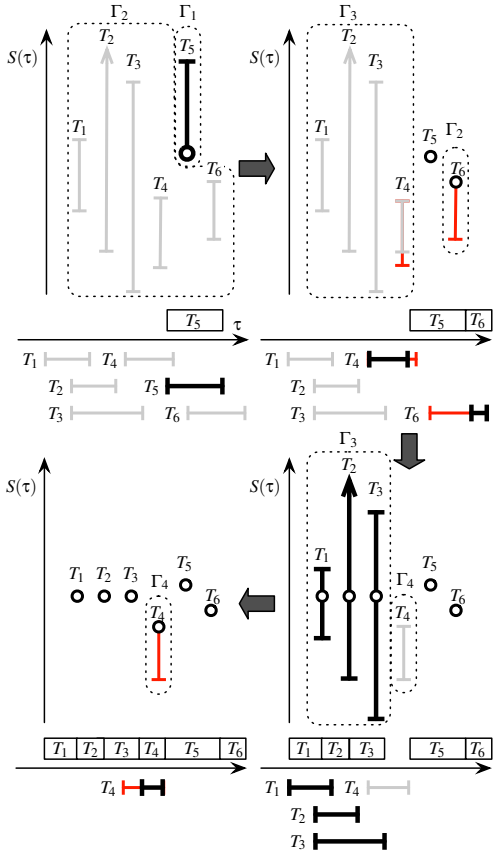


Figure 5: Illustration of multitask scheduling with individual deadlines.

The upper-left diagram in Figure 5 shows that the first pick is T_5 because $S_5(d_5 - a_5) = \min(S_1(d_1 - a_1), ..., S_6(d_6 - a_6))$. Since the sensitivity interval of T_5 does not overlap with any others, the schedule of T_5 has its minimal sensitivity. The upper-right diagram shows the next step. The time intervals of tasks T_4 and T_6 are readjusted because they have been preempted by T_5. This determines the schedule of T_6. Then T_1, T_2, T_3 and T_4 form the task set Γ_3. The lower-right diagram shows the next step. We are assuming that the tasks in Γ_3 are not compatible. We remove T_4 from the task set Γ_3 because T_4 has the smallest maximum sensitivity value. Now, assuming that T_1, T_2 and T_3, which are the tasks in Γ_3, are compatible, T_1, T_2 and T_3 can be scheduled with the same sensitivity value. The lower-left diagram shows the final step. Since T_4 has been preempted by T_3, it must be scheduled with the maximum sensitivity value. The final result is an energy-optimal schedule.

5. EXPERIMENTAL RESULTS

5.1 Experimental Platform

We used the TI OMAP5912 OSK platform which has isolated voltage islands for many different device groups, together with current measurement connectors. A data acquisition system was connected to these connectors in order to log the power consumption while we run several real application programs on the platform.

The supply voltage of the OMAP5912 processor can be adjusted between 1.2V and 1.6V in 0.1V steps by the use of an onboard DC-DC converter. The TI OMAP5912 OSK platform is capable of adjusting the clock frequency of the processor core and also the frequencies of the off-chip synchronous memory and peripherals.

Table 3: Clock frequencies of the JPEG pipeline optimized by VC-DVS.

D (s)	f_1 (s)	f_2 (s)	f_3 (s)	f_4
30	137.75	127.56	118.20	96.66
25	142.28	134.56	124.77	96.66
20	172.65	174.82	177.37	117.64

There is a DLL (delay-locked loop) clock generator that generates frequencies between 96MHz and 192MHz in 12MHz steps from a 12MHz crystal. The DLL output is connected to four 2-bit pre-scalers that can divide the input clock by 1, 2, 4 or 8. If we change the CPU clock frequency and the DLL output so that the memory and other peripherals maintain the same clock frequencies regardless of DVS, the possible scaled values of the clock frequency are 192MHz, 96MHz, 48MHz and 24MHz. As shown in Figure 1, the optimal clock frequencies are generally higher than 90MHz, so only two of these frequencies are useful. The alternative is to change the frequency of DLL, which has double the resolution of the DC-DC converter, while setting the pre-scalars to 1. This setup fully covers the clock frequency range of the CPU, allowing us to operate the CPU, together the synchronous memory and the peripherals from 96MHz to 192MHz in 24MHz step. Frequency scaling of the synchronous memory and peripherals is generally better than using fixed clock frequencies during DVS, because it allow us to reduce the effect of their dynamic energy consumption during idle mode [17].

The dynamic power of an application can change because of many factors. For instance, each application has a unique device utilization profile, in terms of memory such as cache miss rates and memory access frequency. Furthermore, some peripheral devices have very different energy costs depending on the types of requests that they receive. For example, flash memory consumes little energy in reading data, but a lot in erasing and writing data. Thus operations which involve reading flash memory show a completely different energy cost function from writing operation to the same memory. So, in general, different applications will have different energy cost functions and different optimal scaling factors.

5.2 Task Sequence with a Single Deadline

We call our scheme *VC-DVS* (variable cost function DVS) in the summary of experimental results. Experiments on a sequence of tasks were done using a JPEG encoding application. The sequence involves four tasks: T_1 = Load, T_2 = IDCT, T_3 = Huffman encoding, and T_4 = Write-back. Three methods of DVS were compared and the results are shown in Table 2. The row labeled *DVS* corresponds to the conventional scheme that accounts only for the CPU energy consumption and uses the lowest possible frequency. CS-DVS is the method described by Jejurikar et al. [19], and VC-DVS is the method described in this paper. Note that the critical speed of CS-DVS is set to 120 MHz. Also note that all the other schemes use a single clock frequency for the whole JPEG application, whereas we adjust the clock frequency for each task. Table 3 shows our energy-optimal clock frequencies.

The results show that our method achieves a lower total system energy consumption than existing methods. In particular, the gain from introducing conventional DVS ranges from 10% to 14%; the improvement of CS-DVS over conventional DVS is in the range of 2% to 3%; and the improvement achieved by VC-DVS over CS-DVS is between 5% and 10%. We emphasize that all these values are based on accurate measurements making this a small but genuine improvement, since the CPU does not dominate the total system energy consumption in real embedded systems.

5.3 Tasks with Individual Deadlines

We now look at the system-wide energy reduction achieved by our method for multiple tasks with individual deadlines. We

Table 2: Measured energy saving for the JPEG pipeline (T_1: Load; T_2: IDCT; T_3: Huffman; and T_4: Write back).

D (s)	Method	E_1 (J)	τ_1 (s)	E_2 (J)	τ_2 (s)	E_3 (J)	τ_3 (s)	E_4 (J)	τ_4 (s)	E (J)	τ (s)	f (MHz)	Reduction
N/A	No DVS	1.48	2.47	2.55	4.24	3.53	6.20	2.40	4.16	9.96	17.06	192	0%
30	DVS	1.37	4.42	2.30	7.50	3.16	11.01	1.72	5.44	8.55	28.37	109.19	14%
	CS-DVS	1.34	4.00	2.27	6.83	3.13	10.02	1.76	5.19	8.51	26.03	120.00	15%
	VC-DVS	1.31	3.40	2.27	6.40	3.12	10.10	1.67	5.80	8.37	25.70	Table 3	16%
25	DVS	1.32	3.66	2.27	6.26	3.14	9.19	1.82	4.97	8.55	24.08	131.03	14%
	CS-DVS	1.32	3.66	2.27	6.26	3.14	9.19	1.82	4.97	8.55	24.08	131.03	14%
	VC-DVS	1.31	3.30	2.27	6.10	3.13	9.60	1.67	5.80	8.38	24.80	Table 3	16%
20	DVS	1.33	2.92	2.34	5.06	3.25	7.41	2.03	4.47	8.94	19.86	163.79	10%
	CS-DVS	1.33	2.92	2.34	5.06	3.25	7.41	2.03	4.47	8.94	19.86	163.79	10%
	VC-DVS	1.36	2.80	2.41	4.80	3.35	6.90	1.75	5.20	8.87	19.70	Table 3	11%

Table 4: Synthetic workload composed of multiple tasks with individual arrival and deadline times.

Task set	Method	The sum of execution times (s)	The sum of slack times (s)	Energy (J)	Reduction (%)
$\mathbf{T_1}$	No DVS	491.51	522.32	280.93	0.0%
	DVS	835.80	178.03	252.34	10.2%
	CS-DVS	762.19	251.64	237.16	15.6%
	VC-DVS	774.67	239.16	227.70	19.6%
$\mathbf{T_2}$	No DVS	248.91	779.68	143.02	0.0%
	DVS	472.18	556.41	133.74	6.5%
	CS-DVS	381.69	646.90	119.32	16.6%
	VC-DVS	391.24	637.35	113.20	20.8%

explored the energy gain across the whole system with a synthetic workload corresponding to various possible situations. Table 4 summarizes the results. The task sets $\mathbf{T_1}$ and $\mathbf{T_2}$ each consist of 100 tasks. Task set $\mathbf{T_1}$ has exponentially distributed inter-arrival times with a mean of 10s, and the deadlines also follow an exponential distribution with mean 20s. Task set $\mathbf{T_2}$ has the same inter-arrival times and deadlines, but the execution time at the maximum speed is different. The worst-case execution times of $\mathbf{T_1}$ and $\mathbf{T_2}$ follow an exponential distribution with means of 5s and 2.5s respectively.

6. CONCLUSION

High-level power management including DVS is effective for embedded systems when correct energy cost functions are deployed. The challenge in achieving true optimal voltage assignment for DVS is the variability of energy cost functions across application programs. This paper introduces a systematic approach to voltage assignment for multitasking DVS which takes non-uniform energy cost functions into account. We have shown how the variable energy cost function of an embedded system can be found by actual hardware measurement, and we have also determined the individual energy cost function for each application program. For a sequence of tasks with a single deadline, which is typical of embedded applications, we developed an optimal voltage assignment solution. We defined the sensitivity functions of tasks, and showed that the optimal voltage assignment should have the same sensitivity values. We also extended our method to multiple tasks with individual deadlines, using a heuristic algorithm. Real measurements demonstrated savings of 10% of total system energy for a JPEG encoder pipeline, and up to 20% for synthetic task sets with individual deadlines, when compared with previous approaches. Our scheme can significantly extend the applicability of DVS in real-world power management practices.

7. REFERENCES

[1] S. Irani, S. Shukla, and R. Gupta, "Online strategies for dynamic power management in systems with multiple power-saving states," *ACM Trans. Embedded Computing Sys. (TECS)*, vol. 2, no. 3, pp. 325–346, 2003.

[2] P. M. Greenawalt, "Modeling power management for hard disks," in *Proc. Conf. Modeling, Analysis, Simulation of Computer and Telecomm. Sys. (MASCOTS)*, pp. 62–66, 1994.

[3] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. VLSI*, vol. 8, no. 3, pp. 299–316, 2000.

[4] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G. De Micheli, "Quantitative comparison of power management algorithms," in *Proc. Design Automation and Test in Europe Conf. (DATE)*, pp. 20–26, 2000.

[5] A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy/power," *IEEE Trans. VLSI*, vol. 11, pp. 270–276, April 2003.

[6] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. SIGOPS*, pp. 89–102, ACM Press, 2001.

[7] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. Intl' Symp. Low Power Electronics and Design (ISLPED)*, pp. 76–81, 1998.

[8] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey, and M. Neufeld, "Policies for dynamic clock scheduling," in *Proc. Symp. Operating Sys. Design and Implementation (OSDI)*, 2000.

[9] J. R. Lorch and A. J. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Trans. Computers*, vol. 53, pp. 856–869, July 2004.

[10] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. Symp. Operating Sys. Design and Implementation (OSDI)*, pp. 13–23, 1994.

[11] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," in *Proc. Intl' Conf. Mobile Computing and Networking (MOBICOM)*, pp. 13–25, 1995.

[12] A. Klaiber, "The technology behind Crusoe^{TM}processors," tech. rep., Transmeta Corp., 2000.

[13] T. Pering, T. Burd, and R. Brodersen, "Voltage scheduling in the IpARM microprocessor system," in *Proc. Intl' Symp. Low Power Electronics and Design (ISLPED)*, pp. 96–101, 2000.

[14] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proc. Symp. Operating Sys. Principles (SOSP)*, pp. 149–163, 2003.

[15] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling for portable systems," in *Proc. Design Automation Conf. (DAC)*, pp. 524–529, 2001.

[16] R. Jejurikar and R. Gupta, "Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-time Embedded Systems," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 78–81, 2004.

[17] Y. Cho and N. Chang, "Memory-aware Energy-optimal Frequency Assignment for Dynamic Supply Voltage Scaling," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 387–392, 2004.

[18] Y. Choi, N. Chang, and T. Kim, "DC-DC Converter-aware Power Management for Battery-operated Embedded Systems," in *Proceedings of the 42nd Conference on Design Automation*, pp. 895–900, 2005.

[19] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems," in *Proceedings of the 41st Conference on Design Automation*, pp. 275–280, 2004.

[20] L. Niu and G. Quan, "Reducing Both Dynamic and Leakage Energy Consumption for Hard Real-time Systems," in *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 140–148, 2004.