Practical Voltage Scaling for Power-Aware Real-Time Systems

Saowanee Saewong and Ragunathan (Raj) Rajkumar
Real-time and Multimedia Systems Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213
{*ssaewong, raj*}*@ece.cmu.edu*

Abstract

Many embedded real-time systems exhibit power and/or heat constraints. For example, space-based systems and field-deployed battery-operated equipment are power-constrained, while on-demand multimedia servers have thermal limits which constrain the amount of heat that they can dissipate. In CMOS circuits, power consumption is proportional to the product of the frequency and the square of the supply voltage used. Hence, any reductions in the operating frequency of the processor and its supply voltage can lead to significant savings in energy consumption (and therefore heat dissipation) but cause longer execution times. The application of dynamic voltage scaling (DVS) techniques to real-time systems must therefore attempt to minimize energy consumption while guaranteeing the schedulability of the real-time tasks. In this paper, we address several problems that occur in practical implementations of power-constrained real-time systems. These problems include non-ideal power-frequency characteristics and the availability of a limited number of operating frequencies in modern processors. For example, due to the presence of leakage effects, some voltage levels in commercial processors may actually consume *less* energy than while running at lower voltage levels. We characterize some interesting properties of this non-ideal behavior, and present an efficient scheme for determining these "energy-inefficient frequencies". We also study the impact of limited frequency granularity on DVS schemes in real-time systems. Next, we propose two algorithms, called *Resource Kernel Static Voltage Scaling* (RK-SVS) and *Resource Kernel Dynamic Voltage Scaling* (RK-DVS). We evaluate the algorithms and find that for processors with a limited number of operating frequencies (up to 32 frequencies), our algorithms perform well both in terms of algorithmic complexity and energy consumption.

1 Introduction

The field of dynamic voltage scaling is currently the focus of a great deal of research interest with the goal of achieving energy efficiency in portable systems and servers. This is due to the fact that the power consumption of CMOS circuits is given by $P \propto C_L \times N_{cycle} \times V_{DD}^2$, where P is the power

consumption, C_L is the load capacitance, N_{cycle} is the number of executed cycles, and V_{DD} is the supply voltage. Since the power has a quadratic dependency on the supply voltage, scaling the supply voltage down is the most effective way to minimize energy consumption. However, lowering the supply voltage can also adversely affect the system performance due to increasing delays in CMOS circuits. Consequently, the maximum speed a processor can operate is a direct consequence of the supply voltage and is given by $f \propto \frac{(V_{DD}-V_{th})^{\alpha}}{V_{DD}}$, where V_{th} is the threshold voltage and α is the velocity saturation index of CMOS circuits [9].

A variety of dynamic voltage scaling techniques [1, 3, 8, 11, 12] has addressed the tradeoff between the system performance and the energy efficiency. These techniques make use of operating system information about the current workload and applications in order to reduce the processor voltage when the full system performance is not necessary. Some of these techniques target real-time systems where the timing constraints of real-time tasks must also be satisfied. All of the above techniques assume that the energy consumption in processors is minimized whenever the supply voltage is scaled down. Unfortunately, the power consumption in commercial processors such as the StrongARM processor (SA-1100) and the Crusoe processor, is not completely dominated by the dynamic power consumption, which results from switching capacitive loads. The static power consumption, which results from leakage current, also has a significant effect. Therefore, reducing the supply voltage does not always result in energy efficiency in these processors. In conclusion, simply running the processor at the lowest possible operating frequency with a lower supply voltage is not necessarily sufficient to minimize the energy consumption.

In this paper, we analyze the concept of *energy-inefficient operating frequencies*, first introduced as *critical power slope* in [7]. Energy-inefficient frequencies are those operating frequencies of a processor that consume *more* energy than their higher frequencies and thus *must* be excluded from the set of operating frequencies managed by the DVS algorithms. We will show that these energy-inefficient frequencies can be identified efficiently and off-line.

In practice, the number of operating points provided by a commercial processor is also limited. We will quantify the impact on the performance of the DVS algorithms by the use of limited operating voltage-scaling frequencies. For instance, a system with a limited number of operating frequencies may have to run at a higher frequency than the desired frequency and therefore consume more energy than what would be needed by an ideal processor. The amount of energy loss due to insufficient operating frequencies will be analyzed. We also introduce an operating frequency *grid* which minimizes the energy loss.

In addition to detailed analyses of the impact that practical processors can deliver to the performance of DVS algorithms, we present "Resource Kernel Static Voltage Scaling" (RK-SVS) and "Resource Kernel Dynamic Voltage Scaling" (RK-DVS) schemes. These new algorithms lead to significant energy consumption with low computational complexity.

This paper is organized as follows. Section 2 provides insight into the power-frequency dependency in practical processors and describes how energy-inefficient operating frequencies can be determined. Section 3 studies the influence of operating frequency granularity on the performance of DVS algorithms. Sections 4 and 5 describe our RK-SVS and RK-DVS schemes, respectively. Section 6 presents experimental results that evaluate RK-SVS and RK-DVS using simulations. Section 7 discusses related work, including DVS algorithms proposed in the literature. Concluding remarks and comments on future research appear in Section 8.

2 Energy-Efficient Operating Frequency

In this section, we show that scaling voltage down is not always appropriate for improving energy efficiency.

As an example, we consider the commercially available Crusoe processor from Transmeta. It can operate at six different frequencies/voltages as shown in Table 1 [2]. We show that, contrary to intuition, operating at 225 MHz is actually inefficient in terms of energy consumption. The processor

Frequency f (MHz)	Voltage V_{DD} (V)	Relative power (%)
600	1.60	100.00
525	1.50	70.00
450	1.35	45.00
375	1.22	33.33
300	1.20	26.67
225	1.10	23.33

Table 1: Clock frequency versus supply voltage of the Transmeta Crusoe processor.

consumption in the idle mode, P_{idle}, is about 0.3 watts at a clock frequency of 300 MHz, which is about 5% with respect to the maximum power consumption at maximum clock frequency, P_{max}. Suppose that a given processing workload is specified by the number of clock cycles needed on a processor. The total elapsed time to process a given workload W is given by $W/freq$. Hence, as the frequency increases, the elapsed time decreases but the processor must "idle" for the remaining time consuming P_{idle} every unit of remaining time. Assume that we can complete a job J within t_{225MHz} units of execution time at the operating frequency of 225 MHz. The energy consumption by this job is given by

$$\begin{aligned} E_{225MHz} &= P_{225MHz} \times t_{225MHz} \\ &= 0.233 \times t_{225MHz} \times P_{max} \end{aligned}$$

If we increase the processor frequency to 300 MHz, the energy consumption for completing the same

job J over the interval t_{225MHz} is given by

$$\begin{aligned} E_{300MHz} &= P_{300MHz} \times t_{300MHz} + P_{idle} \times (t_{225MHz} - t_{300MHz}) \\ &= 0.267 \times \frac{225}{300} \times t_{225MHz} \times P_{max} + 0.05 \times \frac{75}{300} \times t_{225MHz} \times P_{max} \\ &= 0.21275 \times t_{225MHz} \times P_{max} \\ &< E_{225MHz} \end{aligned}$$

Hence, 225 MHz is an example of an *energy-inefficient operating frequency*. It delivers less performance (meaning longer response time) and, even worse, consumes more energy than one of its higher operating frequencies (300 MHz in this case). Obviously, operating the system at such energy-inefficient operating frequencies must be avoided under all circumstances. To formalize the concept, consider

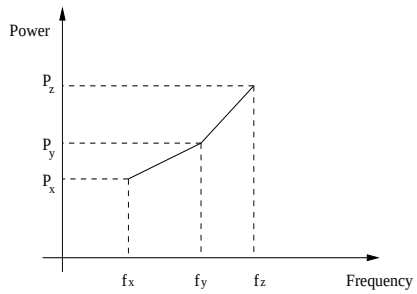


Figure 1: A convex non-decreasing function of power versus clock frequency

Figure 1. The clock frequency f_x is said to be energy-efficient if and only if the energy consumption by a workload operating at f_x is lower than operating at any higher frequencies f_y, f_z, etc. That is,

$$P_x < P_y \times \frac{f_x}{f_y} + P_{idle_y} \times (1 - \frac{f_x}{f_y})$$

For simplicity, we assume $P_{idle_y} = P_{idle_x}$. Let m_{xy} be the power-frequency slope from f_x to f_y and $P_y = P_x + m_{xy}(f_y - f_x)$. Therefore, for f_x to be energy-efficient,

$$\forall f_y > f_x : \quad m_{xy} > \frac{P_x - P_{idle_x}}{f_x} \tag{1}$$

Equation (1) is a necessary condition for any operating frequency f_x to be an energy-efficient operating point. In other words, f_x is considered to be an energy-inefficient operating point if there is at least one higher frequency which performs better but consumes less energy. Accordingly, f_x should be excluded from the set of operating frequencies managed by DVS algorithms. Figure 2 illustrates a pair-wise frequency comparison scheme to determine energy-inefficient operating frequencies. For each candidate frequency f_x, equation (1) must be tested. If equation (1) fails from some $f_y > f_x$, f_x is

energy-inefficient and can be excluded from consideration.

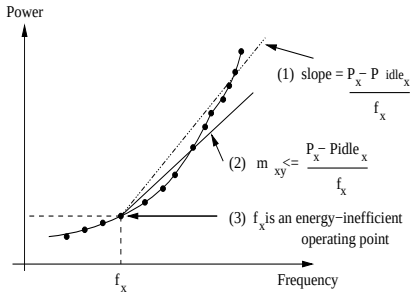


Figure 2: A scheme to determine energy-inefficient operating frequencies

The complexity of the above scheme is $O(n^2)$ where n is the number of available operating frequencies. Next, we will prove that this complexity can be reduced to $O(\lg n)$ in the case of processors which exhibit a convex non-decreasing power-frequency function. This property is usually true for practical processors using the voltage-scaling technique even though their power versus frequency function is not perfectly cubic.

Theorem 1 *For a convex non-decreasing power-frequency function, $P(f)$, if there is an operating frequency f which is energy-inefficient relative to a higher operating frequency, all operating frequencies lower than f are also energy-inefficient.*

Proof.

We prove the theorem by contradiction. Consider Figure 1. Let E_x, E_y and E_z be the energy consumption of a task at frequency f_x, f_y and f_z respectively. We assume that P_{idle} is constant for all operating frequencies. Let f_y be an energy-inefficient point relative to f_z. That is,

$$m_{yz} \leq \frac{P_y - P_{idle}}{f_y} \tag{2}$$

If we assume that f_x is energy-efficient, then

$$\begin{aligned} E_x &< E_y \\ P_x \times t_x &< P_y \times t_y + P_{idle} \times (t_x - t_y) \\ P_y - m_{xy}(f_y - f_x) &< P_y \times \frac{f_x}{f_y} + P_{idle}(1 - \frac{f_x}{f_y}) \\ \frac{P_y - P_{idle}}{f_y} &< m_{xy} \end{aligned} \tag{3}$$

Since $P(f)$ is a convex non-decreasing function, $m_{xy} \leq m_{yz}$. Therefore, equation (2) and equation (3) lead to a contradiction. Therefore, if f_y is an energy-inefficient operating frequency relative to $f_z > f_y$, any frequencies lower than f_y are also energy-inefficient relative to f_z. □

Corollary 1 *For a convex non-decreasing power-frequency function, $P(f)$, if there is an operating frequency f which is energy-efficient relative to a higher frequency, all operating frequencies higher than f are also energy-efficient.*

Proof.

This follows from Theorem 1.

Theorem 2 *For a convex non-decreasing power-frequency function, the complexity to determine all energy-inefficient operating frequencies is $O(\lg n)$.*

Proof.

Using the knowledge from Theorem 1 and Corollary 1, one can use a *binary search* [10] to find all energy-inefficient operating frequencies. To find out if an operating frequency is energy-efficient, only one computation relative to the immediate higher frequency is needed. If the corresponding node is an energy-efficient operating point, we next search in the domain of its lower frequencies. Otherwise, we next search in the domain of its higher frequencies. The algorithm stops when the domain size is reduced to one. All higher frequencies relative to the last node are energy-efficient operating points and all lower frequencies are energy-efficient ones. The total number of computing iterations is $\lg n$.

3 The Effect of Finite Operating Frequencies

Commercial processors usually provide a finite operating frequency granularity. In other words, operating frequencies cannot be continuously varied, and are only available as a small number of discrete points. To sustain acceptable performance and timeliness guarantees, these processors have to operate at the next higher operating frequency if a desired frequency is not available. This inevitably results in more energy consumption. In the rest of this paper, we refer to such an energy loss due to finite operating frequencies as the *energy quantization error*.

We now investigate an operating frequency grid which minimizes the worst-case energy quantization error. An ideal processor with infinite operating frequencies can operate at any frequency from 0 to f_{max}. Let us initially consider a practical processor with two operating points, f_x and f_{max}, as shown in Figure 3(a). A frequency of 0 represents the processor in its *sleep* state. We will show that f_x should be located at $\frac{f_{max}}{\sqrt{2}}$ or $0.707 f_{max}$ to minimize the worst-case energy quantization error.

Let ΔE_1 and ΔE_2 be the worst-case energy quantization error when $f_{opt} \in (0, f_x]$ and $(f_x, f_{max}]$ respectively, where f_{opt} is the optimal operating frequency if the processor is ideal. The worst-case ΔE_1 occurs when $f_{opt} = \epsilon$, $\epsilon \to 0$, and the processor has to run at f_x and therefore use more energy. Similarly, the worst-case energy quantization error when $f_{opt} \in (f_x, f_{max}]$, ΔE_2, occurs when $f_{opt} = f_x + \epsilon$, $\epsilon \to 0$. The position of f_x affects both ΔE_1 and ΔE_2. If we increase f_x, ΔE_2 is decreased, whereas ΔE_1 is increased. The opposite occurs if f_x is decreased. Consequently, we should place f_x where $\Delta E_1 = \Delta E_2$ in order to minimize the overall worst-case energy quantization error.

(a) 2 operating points

(b) N operating points

Figure 3: A processor with limited number of operating points

Assume that an ideal processor can complete a task that needs λ cycles of execution at the frequency ϵ to guarantee the task timing constraints. Instead, a practical processor with two operating points has to run the same task at f_x to preserve timeliness guarantees. If we assume that both processors use the voltage-scaling technique to scale frequency, their relation of power versus frequency is given by $P(f) \propto C_L V_{DD}^2 f \approx k f^3$. Therefore, their energy consumption to complete *one* cycle is equal to $E = kf^2$. Let E_ϵ and E_{f_x} be the energy consumption by the ideal processor and the practical one to complete the task at f_ϵ and f_x respectively. P_{idle_f} represents the idle power at the operating frequency f. Then,

$$\begin{aligned} E_\epsilon &= k\lambda\epsilon^2 \\ E_{f_x} &= k\lambda {f_x}^2 + P_{idle_{f_x}}(1/\epsilon - 1/f_x) \\ \Delta E_1 &= E_{f_x} - E_\epsilon \end{aligned}$$

Similarly, if the ideal processor has to run the task at frequency $f_x + \epsilon$, the practical processor has no choice but to run that task at frequency f_{max}. Let E_{f_x} and $E_{f_{max}}$ be the energy consumption needed by the ideal processor and the practical one respectively. We get,

$$\begin{aligned} E_{f_x} &= k\lambda(f_x + \epsilon)^2 \\ E_{f_{max}} &= k\lambda {f_{max}}^2 + P_{idle_{f_{max}}}(1/(f_x + \epsilon) - 1/f_{max}) \\ \Delta E_2 &= E_{f_{max}} - E_{f_x} \end{aligned}$$

Solving $\Delta E_1 = \Delta E_2$ where ϵ is negligible with respect to f_x, we get

$$f_x = \sqrt{\frac{f_{max}^2}{2} + \frac{P_{idle_{f_{max}}}(1/f_x - 1/f_{max}) - P_{idle_{f_x}}(1/\epsilon - 1/f_x)}{2k\lambda}} \quad (4)$$

For large values of λ[1], the term $\frac{P_{idle_{f_{max}}}(1/f_x - 1/f_{max}) - P_{idle_{f_x}}(1/\epsilon - 1/f_x)}{2k\lambda}$ in equation (4) is negligible. Therefore, f_x should be equal to $f_{max}/\sqrt{2}$ to minimize the energy quantization error.

A similar approach can be employed to determine the optimal operating frequencies of a practical processor with N operating points as shown in Figure 3(b). For simplicity, we assume that ϵ is very small and λ is very large. We can then ignore the energy consumption during idle state. That is,

$$\begin{aligned} \Delta E_1 &= k\lambda f_1^2 \\ \Delta E_2 &= k\lambda f_2^2 - k\lambda f_1^2 \\ &\vdots \\ \Delta E_{N-1} &= k\lambda f_{max}^2 - k\lambda f_{N-1}^2 \end{aligned}$$

Solve $\Delta E_1 = \Delta E_2 = \ldots = \Delta E_{N-1}$. We obtain

$$\forall i \in [1, N-1] : f_i = \sqrt{\frac{i}{i+1}} f_{i+1} \quad (5)$$

$$f_N = f_{max} \quad (6)$$

Figure 4 gives an example grid of 10 normalized operating frequencies that minimize the energy quantization error.

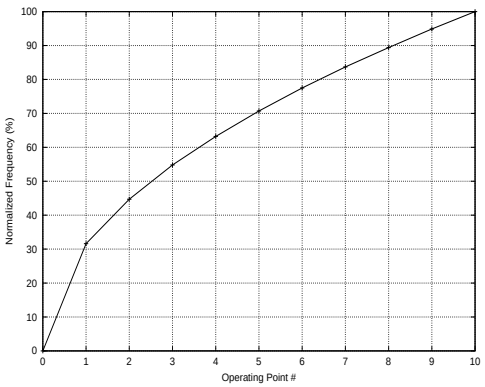


Figure 4: Optimal operating frequencies for a processor with 10 operating points

We now analyze how the number of operating points affects energy loss. Higher the number of

[1] This assumption is expected to be true in practice since battery/solar power can last for a few hours.

operating points, the less energy quantization error results. Since we equalize the energy quantization error at each level, we can just consider ΔE_{N-1} which is given by

$$\begin{aligned} \Delta E_{N-1} &= k\lambda f_{max}^2 - k\lambda f_{N-1}^2 \\ &= \frac{k\lambda f_{max}^2}{N} \end{aligned} \tag{7}$$

Let ΔE be the worst-case quantization error of a processor with N operating points whose frequency grid as defined by equations (5) and (6). Let E_{noDVS} be the energy consumed by the processor without any voltage-scaling. E_{noDVS} then is given by $E_{noDVS} = k\lambda f_{max}^2$. Therefore, from equation (7),

$$\Delta E = \frac{E_{noDVS}}{N} \tag{8}$$

According to equation (8), the deviation from minimal energy consumption is linear-inversely proportional to the number of operating frequencies. For example, if we want only 10% energy loss compared to having an infinite number of operating points, a processor will need at least $1/0.1 = 10$ operating frequencies.

4 Resource Kernel Static Voltage Scaling (RK-SVS)

In this section, we propose a static voltage scaling algorithm for hard real-time systems where timing constraints must be satisfied. This scheme computes the energy-saving frequency for each single task during admission control. In other words, the task operating frequency is set statically. At each context switch, a processor scales the operating frequency and the supply voltage based on the corresponding operating frequency of a task about to be scheduled. The uniqueness of our Resource Kernel Static Voltage Scaling (RK-SVS) scheme is its awareness of energy-inefficient frequencies and the limited number of operating frequencies. RK-SVS allows a deadline-monotonic (DM) scheduler to save energy consumption while maintaining schedulability and low computational complexity

4.1 System Model and Notation

We first define the system model and some terminology. Assume that there are n tasks. Each task specifies its requirements as $\{C_i, T_i, D_i\}$, where C_i is the worst-case execution requirement in the number of processor cycles, and T_i and D_i represent elapsed time of its period and deadline respectively. By convention, tasks are ordered and scheduled in priority order using the deadline-monotonic scheduling policy. That is $D_1 \leq D_2 \leq \ldots \leq D_n$. Throughout the rest of this paper, we assume that all processors have a convex non-decreasing relation between power and frequency. This assumption is expected to be true in practice.

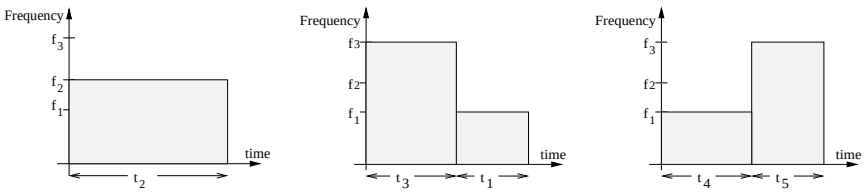
4.2 Energy-Saving Operating Frequencies

As previously discussed, the energy-inefficient operating frequencies must be avoided under all circumstances because the system can run at some higher frequency with the same or less energy cost and get the additional performance for free. Consequently, RK-SVS excludes all energy-inefficient frequencies from its target set of operating points. In the next theorem, we prove that the best solution to minimize energy usage is to run the processor at a constant operating frequency which is energy-efficient and exactly satisfies timing constraints.

Theorem 3 *The minimum energy usage for a processor which executes a given workload within a given completion time is achieved by operating the processor constantly at the lowest possible energy-efficient operating frequency that can satisfy the specified completion time.*

Proof.

Figure 5 shows three different clock frequency settings of a processor. Each setting completes the given workload in exactly the same time, given by t time units. Therefore, $t_2 = t_1 + t_3 = t_4 + t_5 = t$. The first one operates the task at a clock frequency f_2 constantly for t_2 time units. The second one operates the task at a higher clock frequency f_3 for t_3 time units and then scales the clock frequency down to f_1 for t_1 time units. The third one operates the task at a lower clock frequency f_1 for t_4 time units and then scales the clock frequency up to f_3 for t_5 time units. If $t_1 = t_4$ and $t_3 = t_5$, both speed-up and speed-down cases will consume the same amount of energy. In addition, all other patterns can be built from these general cases. Therefore, it is sufficient to just compare the first and second cases to draw a conclusion. Since all cases complete the same workload, we get $(f_2 \times t_2) = (f_1 \times t_1) + (f_3 \times t_3)$.



(a) Constant speed (b) Speed up and down (c) Speed down and up

Figure 5: Possible speed settings of a processor

Since $t = t_1 + t_3$, we have $t_1 = (\frac{f_3 - f_2}{f_3 - f_1})t$ and $t_3 = (\frac{f_2 - f_1}{f_3 - f_1})t$. Let $E_{constant}$ and $E_{varydown}$ be the energy usage of the first and second patterns, and $m_{f_1 f_2}$ and $m_{f_2 f_3}$ be the power-frequency slopes from f_1 to f_2, and f_2 to f_3 respectively. Since we assume a convex non-decreasing relation between power and

frequency, $m_{f_2 f_3} \geq m_{f_1 f_2}$. That is,

$$\begin{aligned} E_{constant} &= P_{f_2} \times t_2 = P_{f_2} \times t \\ E_{varydown} &= (P_{f_2} + m_{f_2 f_3}(f_3 - f_2)) \times t_3 + (P_{f_2} - m_{f_1 f_2}(f_2 - f_1)) \times t_1 \\ &= P_{f_2} \times t + (m_{f_2 f_3} - m_{f_1 f_2}) \frac{(f_2 - f_1)(f_3 - f_2)}{f_3 - f_1} \\ E_{varydown} &\geq E_{constant} \end{aligned}$$

Even though both cases *may* consume the same amount of energy but switching frequencies will definitely cause some overhead, running the processor at a single lowest allowable frequency is the best choice. □

Remark: Theorem 3 also applies when $f_1 = 0$. That is, completing a workload quickly and idling the rest of the time is not the best way to minimize energy consumption.

4.3 Energy-Saving Schedulable Operating Frequencies

Theorem 3 indicates that for a given workload with a deadline, a processor will consume less energy if it runs at a constant clock frequency and completes the workload exactly at the deadline. This clock frequency can be determined from the number of processor cycles of the workload and the deadline. Consider a task set $\{\tau_1, \ldots, \tau_n\}$. Each task τ_i has $\{C_i, T_i, D_i\}$ requirements. We define *the energy-saving clock frequency* of task τ_i as a clock frequency that allows τ_i to complete as late as possible to save the processor energy (denoted as α_i).[2] In order to guarantee τ_i's timing constraint, τ_i's energy-saving clock frequency must be derived from the critical zone [6][3].

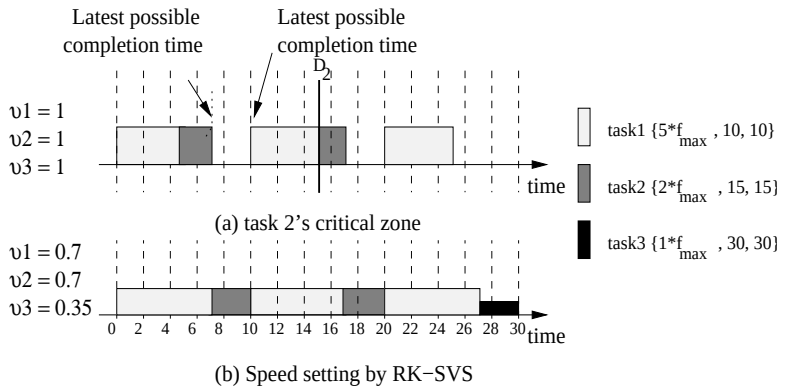


Figure 6: Optimal Task Operating Frequency

Figure 6(a) depicts a critical time zone of task τ_2 in a task set, $\{\tau_1, \tau_2, \tau_3\}$ with the specifications

[2] All operating frequencies are normalized to f_{max}, and therefore range from 0 to 1.0.
[3] The critical zone assumes that the requests of all tasks arrive simultaneously. This leads to the worst-case scenario.

of $(5 * f_{max}, 10, 10), (2 * f_{max}, 15, 15)$ and $(1 * f_{max}, 30, 30)$ respectively. This timeline assumes that the processor runs all tasks at full speed, f_{max}. As can be seen, the completion time of τ_2 is equal to 7, which will be referred to as *the earliest possible completion time*. Instead, if the processor operates at a lower clock frequency, the execution time needed by each task increases and this extends τ_2's completion time. Noting that we assume that a fixed-priority preemptive scheduling policy is used, any higher priority task $\tau_j, j < i$ will always preempt τ_i. As shown in Figure 6(a), τ_2 can be preempted by τ_1 at time 10 to 15 when the processor runs at full speed. This preemption will even be longer if the processor runs at a lower clock frequency. Therefore, if τ_2 cannot complete its job before 10, it will definitely miss its deadline. We refer to this last time instant that a task τ_i can complete and still meet its deadline as *the latest possible completion time* (denoted by δ_i). The total execution time needed by τ_i and its priority tasks to complete τ_i at its latest possible completion time when the processor runs at full speed is referred to as *the maximum aggregate demand until completion* (denoted by β_i). To save energy consumption, we now find the clock frequency that completes a task as late as possible, which is exactly at its latest possible completion time. Therefore, $\alpha_i = \beta_i/\delta_i = f_i/f_{max}$, which ranges from 0 to 1.0.

The energy-saving clock frequency of a task τ_i, α_i, determines the normalized clock frequency the processor should use to operate all tasks in order to save energy consumption and let τ_i complete as late as possible. However, to maintain the schedulability of the task set, the operating frequency of a task may need to be larger than its energy-saving clock frequency. For example, suppose a higher priority task can run at a frequency of 50 MHz, but a lower priority task cannot run the processor at any frequency lower than 100 MHz to meet its deadlines and save energy (according to Theorem 3). In such a case, both tasks must be run at least at a frequency of 100 MHz. We refer to this energy-saving clock frequency that maintains schedulability of the task set as *the energy-saving schedulable clock frequency* of task τ_i. We now describe how RK-SVS determines the energy-saving schedulable clock frequency of each task by example. A formal algorithm description will follow.

Consider the same example in Figure 6. Since τ_1 is the highest priority task, it will be executed anytime it shows up. Hence, τ_1's latest possible completion time, δ_1, is equal to its deadline and its maximum aggregate demand, $\beta_1 = C_1/f_{max} = 5$. Therefore, the energy-saving operating frequency of τ_1, α_1, is equal to $\beta_1/\delta_1 = 0.5$.

For the case of τ_2, τ_2 can be preempted by τ_1 at time 10 which runs past τ_2's deadline of 15 as shown in Figure 6. The latest possible completion time of τ_2, δ_2, is equal to 10. The maximum aggregate demand (the computation needed by τ_1 and τ_2 before 10 when the processor operates at f_{max}), β_2, is equal to $(C_1 + C_2)/f_{max} = 7$. Therefore, $\alpha_2 = \beta_2/\delta_2 = 0.7$. Similarly, we get $\delta_3 = 30$, $\beta_3 = 20$ and $\alpha_3 = 0.67$. We now discuss how to pick the actual operating frequency of each task, such that *all* tasks in the system can be guaranteed to meet their deadlines (and not just τ_i).

With $\alpha_2 = 0.7$, it means that both τ_1 and τ_2 need to run at the operating frequency $0.7f_{max}$ to

meet τ_2's deadline. To maintain schedulability for all tasks, RK-SVS will set the relative operating frequency of τ_1, υ_1, to the maximum operating frequency required by itself and other lower priority tasks. That is $\upsilon_1 = max\{\alpha_1, \alpha_2, \alpha_3\} = 0.7$ and $\upsilon_2 = max\{\alpha_2, \alpha_3\} = 0.7$.

As in the above example, it is possible that some high-priority tasks (τ_1 and τ_2) may need to run faster than what a low-priority task (τ_3) really needs. This scenario occurs when

- a high-priority task needs to run faster to accommodate the timing constraint of a medium-priority task.
- a desired operating frequency is not available due to the limited number of available operating frequencies, or
- the combination of the above.

No matter what the cause, we refer to the artificially chosen higher frequency assignment as an *inflated frequency*. The use of an inflated frequency by a higher priority task leads to more available slack for a lower priority task. In our example, even though τ_3 only requires τ_1 and τ_2 to run at $0.67f_{max}$, both τ_1 and τ_2 must run at a faster operating frequency ($0.7f_{max}$) to satisfy τ_2's schedulability. τ_3's frequency can therefore be adjusted because of this additional slack. Since the processor needs to operate higher priority tasks at inflated frequencies, τ_3 can scale down the rest of the maximum aggregate demand which is not influenced by inflated frequencies (in this example, τ_3's computation) to "fill" the idle time made available by the higher priority tasks. This "idle time" is denoted as θ_i, and is referred to as the *scalable duration* of task τ_i. The scalable duration of τ_i is determined by subtracting all execution times operating at inflated frequencies (in this case, the computation needed by τ_1 and τ_2), from its latest possible completion time. Therefore, the scalable duration of τ_3, θ_3, is given by $30 - (5/0.7) * 3 - (2/0.7) * 2 = 2.85$. The energy-saving clock frequency of τ_3 (to satisfy all task constraints not just τ_3's), α_3, then is the proportion of the execution time needed by the rest of the maximum aggregate demand at f_{max} divided by its scalable duration. Hence, $\alpha_3 = (C_i/f_{max})/\theta_3 = 1/2.85 = 0.35$. Figure 6(b) shows the final operating frequencies RK-SVS assigned to the processor assuming that operating frequencies of 0.7 and 0.35 are available.

Figure 7 summarizes the RK-SVS algorithm. The computation algorithm for the latest possible completion time δ_i and the busy time β_i of each task is summarized in Figure 8. It has a computational complexity of $O(mn^2)$ where $m < n$ and n is the number of tasks.

In summary, RK-SVS first tries to run every task at a constant speed. The additional frequency adjustment will be activated only when (1) some desired frequencies are not available, or (2) some tasks must operate at a higher frequency to satisfy the timing constraints of other tasks. Note that this frequency adjustment will only scale down the clock frequencies of some tasks while others will be maintained at the same clock frequencies. However, the energy consumption after the adjustment is always reduced.

```
/* RK_SVS (τ1, ..., τn):
/* Assume that there are n tasks and they are ordered by its deadline */
/* That is, D1 ≤ D2 ... ≤ Dn */
1.      for i = 1 to n: vi = 0
2.      for i = 1 to n: compute_alpha(τi)
3.      for i = 1 to n:
4.          /* lp is the set of tasks whose priority is same
5.                  or lower than τi's priority */
6.          vi =choose_operating_freq(max_{∀j∈lp(τi)} αj)
7.          if (i != 1) and (v_{i-1} > vi)
8.              /* reset the speed and adjust the slack */
9.              vi = 0
10.              for j = i to n: slack_adjust(τj)
11.              vi =choose_operating_freq(max_{∀j∈lp(τi)} αj)
12. and;
13.     compute_alpha(τi):
14.         /* Compute the busy time and the latest possible completion time of task τi*/
15.         compute_slack(τi, βi, δi)
16.         αi = βi/δi
17.         return αi
18.
19.     slack_adjust(τi):
20.         busy = 0
21.         θi = δi
22.         for j = 1 to n:
23.             if (Dj ≤ Di)
24.                 if (vj != 0)
25.                     /* Cannot adjust the operating frequency of these tasks */
26.                     θi -= ⌈δi/Tj⌉ * (Cj/(vj fmax))
27.                 else
28.                     /* The amount of scalable load */
29.                     busy += ⌈δi/Tj⌉ * Cj/fmax
30.         αi = busy/θi
31.         return αi
32.
33.     choose_opearting_frequency(x):
34.         use lowest f in operating_frequency_set such that f/fmax ≥ x
```

Figure 7: The Resource Kernel Static Voltage Scaling, RK-SVS

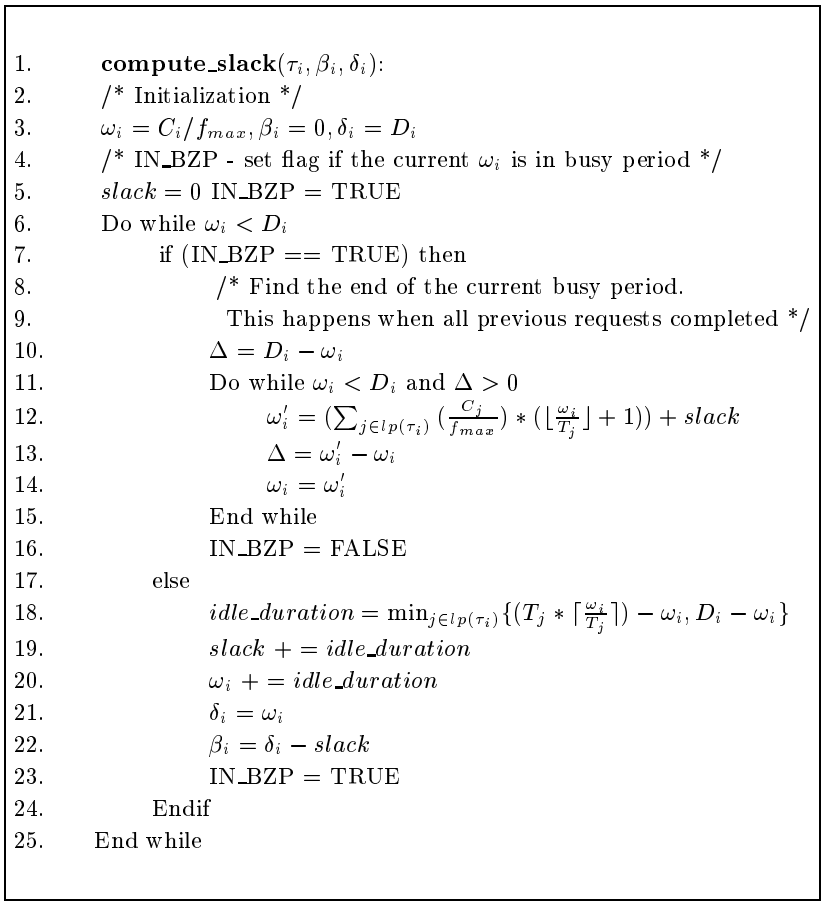


Figure 8: The latest possible completion time and busy period computation algorithm

5 Resource Kernel Dynamic Voltage Scaling (RK-DVS)

In this section, we present an online DVS scheme which monitors the actual execution times of tasks, and further reduces energy consumption when they are less than the worst-case execution times. Our Resource Kernel Dynamic Voltage Scaling (RK-DVS) detects the early completion task instances and makes use of the additional slack time by reducing the processor speed. This objective is the same as that of other DVS schemes, particularly those aimed at real-time systems. Instead of seeking an optimal operating frequency setting, the RK-DVS focuses on a sub-optimal solution which

1. does not violate the timeliness guarantees for all tasks,
2. offers computational simplicity so that it can be run efficiently online in real-time, but
3. yields good performance in terms of reduced energy consumption.

Even though the available slack from the early completion of a task instance can be shared among multiple tasks, RK-DVS instead devotes the slack only to the next job to be scheduled. The new operating frequency of this "lucky" task is computed relative to its current assigned operating frequency. Every new invocation of a task instance always starts with its original operating frequency and changes its operating frequency dynamically only if it gets the slack. We denote υ_i' as the current operating frequency of τ_i, and υ_i denotes the original operating frequency set by RK-SVS.

The detailed description of RK-DVS is as follows. Suppose a task τ_i has an original operating frequency of υ_i. Let $WCEC_i$ be the worst-case processor cycles needed by τ_i and EC_i be the processor cycles already executed in the current instance of τ_i. Let CEC_i be the processor cycles required by the current instance of τ_i and therefore $CEC_i = EC_i$ when a task completes its job. Assume that a task completes its current invocation early after it has executed for CEC_i processor cycles. The elapsed slack time made available by this task, is given by $(WCEC_i - CEC_i)/(\upsilon_i * f_{max})$. This slack time will be dedicated to the next ready task whose priority is lower than or equal to the early completed task. Let us assume that the next ready task is τ_j. Task τ_j can scale down its frequency and let the rest of its execution $(WCEC_j - EC_j)$ fill up the slack. This new operating frequency of τ_j is given by,

$$\upsilon_j' = \frac{(WCEC_j - EC_j)/\upsilon_j'}{(WCEC_j - EC_j)/\upsilon_j' + (WCEC_i - EC_i)/\upsilon_i} * \upsilon_j'$$

In the case that there is no ready task in the queue after the early completion of an invocation, RK-DVS decrements the available slack time when the processor remains idle. The next arriving task whose priority is lower than or equal to the early completed task would receive the updated slack time if the slack is still available (i.e. greater than 0). The additional overhead of RK-DVS to compute the new frequency has complexity $O(1)$, but must be performed at every context switch when an early

completion occurs. In addition, $WCEC_i$ and EC_i of each task must be tracked. Resource Kernels (RK) already perform such computations at context switches, and hence RK-DVS is a rather minor increment to RK OS routines. Figure 9 summarizes the algorithm. It is easy to show that the above

During admission control:
Set v_i and $WCEC_i$ based on RK-SVS
When the new request of task τ_i arrives:
Set $v_i' = v_i$
Set $EC_i = 0, CEC_i = 0$
In every context switch
If τ_i completes its job early and τ_j is the next ready task which has lower priority than τ_i
Update the execution of τ_i: $EC_i +=$ last_executed_processor_cycles
τ_i completes its job: $CEC_i = EC_i$
Set $v_j' = \frac{WCEC_j - EC_j}{(WCEC_j - EC_j) + (WCEC_i - EC_i)(v_j'/v_i)} * v_j'$
$v_j' =$ **choose_operating_frequency**(v_j')
If the τ_i completes its job early but no lower priority task is ready
Update the execution of τ_i: $EC_i +=$ last_executed_processor_cycles
τ_i completes its job: $CEC_i = EC_i$
$slack = (WCEC_i - EC_i)/(v_i * f_{max})$
If the processor was idle and τ_j is the next ready task which has lower priority than the slack owner.
$slack -=$ last_idle_duration
Set $v_j' = \frac{(WCEC_j - EC_j)/(v_j' * f_{max})}{\frac{(WCEC_j - EC_j)}{(v_j' * f_{max})} + slack} * v_j'$
$v_j' =$**choose_operating_frequency**(v_j')
If τ_i is just preempted
$EC_i +=$ last_executed_processor_cycles
Set the processor operating frequency based on the new task (v_j')

Figure 9: The resource kernel dynamic voltage scaling (RK-DVS) algorithm

method maintains schedulability. Assume that task τ_i completes early. Then the next ready task τ_j scales down its own operating frequency. Since there is no priority change, the frequency reduction of τ_j will delay only its lower priority tasks. We choose the next ready task to always have lower priority than the early completed task. However, since only the slack gained is filled by frequency reduction, the preemption time seen by other lower priority tasks will be the same. The run-time computational complexity of RK-DVS at each context switch is $O(1)$.

6 Experimental Results

In this section, we present the results of extensive simulations of our RK-SVS and RK-DVS algorithms. We compare these schemes with the static voltage scaling algorithm (SVS) and the cycle-conserving

RT-DVS (CYCLE) proposed in [8]. A real-time task set is generated randomly. Each task has a uniform probability of having a short (1-10 ms), medium (10-100 ms) or long (100-1000 ms) period as suggested in [8]. Task period is uniformly distributed in each range. Task computation is randomly selected based on

- **The utilization of the task set**: From this, we scale down the utilization of each task by the number of tasks in the system.
- **The ratio of best-case execution time (BCET) to worst-case execution time (WCET)**: A task requests a random number of processor cycles for each of its instances. The number of processor cycles is assigned randomly, uniformly distributed between BCET and WCET.

6.1 Static Voltage Scaling (SVS)and Cycle-Conserving RT-DVS (CYCLE) Overview

Pillai et al. propose SVS and CYCLE in [8] for deadline-monotonic and earliest-deadline-first scheduling policies. In the experiments, we will compare our schemes with the deadline-monotonic version of both their schemes. Instead of seeking an operating frequency of each single task like RK-SVS does, SVS only picks a system operating frequency that can guarantee timing constraints of all tasks. This system operating frequency will be fixed unless the task set is changed. CYCLE is a dynamic voltage scaling scheme that dynamically changes the processor clock frequency when there is slack available. This slack is computed by comparing the expected workload that should be done until the next earliest deadline by SVS and the current workload in the system. If the system has less workload, CYCLE scales the clock frequency down in order to complete the current workload exactly at the earliest deadline. In the next section, we will compare the performance of four schemes, SVS, CYCLE, RK-SVS and RK-DVS under the effect of task set characterization and the effect of processors with a different number of operating frequencies.

6.2 The Effect of Task Set Characterization

In this section, we study the effect of the following variables on energy consumption: the system utilization, BCET/WCET, the number of tasks. We compare the energy consumption of four voltage-scaling algorithms including the total energy consumption in a non-DVS algorithms. We model the processor using the data from the Crusoe processor [2], which is shown in Table 1.

6.2.1 The Effect of System Utilization

In this simulation, we determine the effect of varying the processor utilization of the task sets. Figure 10 shows the normalized energy consumption for the task sets under the four power-aware schemes normalized to the performance of a non-DVS scheme when the system utilization is 0.7. Each task set

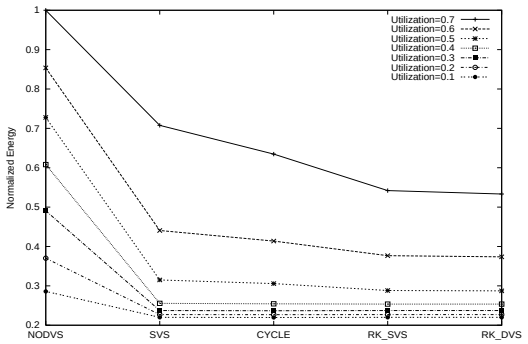
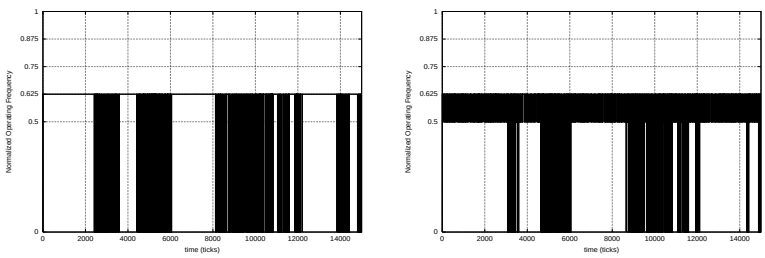


Figure 10: Normalized energy consumption with varying processor utilization with BCET/WCET = 0.5, $n = 10$

has 10 tasks and BCET/WCET = 0.5. As can be seen, RK-SVS outperforms both SVS and CYCLE when the processor utilization is high. This is due to the fact that RK-SVS computes a low frequency with respect to the latest possible completion time, and not the deadline as SVS does. When the processor utilization is high, the latest possible completion time can be much shorter than the deadline. To be fair, in all of our experiments, we exclude the energy-inefficient operating frequency 225 MHz from the set of operating points managed by all algorithms. Otherwise, our algorithms will perform relatively better when other algorithms choose to operate the system at those energy-inefficient operating points. Figure 11 gives dynamic snapshots of operating frequencies of the SVS and RK-SVS



(a) SVS (b) RK-SVS

Figure 11: Frequency scaling snapshots with $BCET/WCET = 0.5, n = 10, U = 0.6$

algorithms. They show that RK-SVS switches between three states (idle, $0.5f_{max}$ and $0.625f_{max}$) while SVS switches only between two states. This explains the higher energy benefits obtained by RK-SVS compared to SVS.

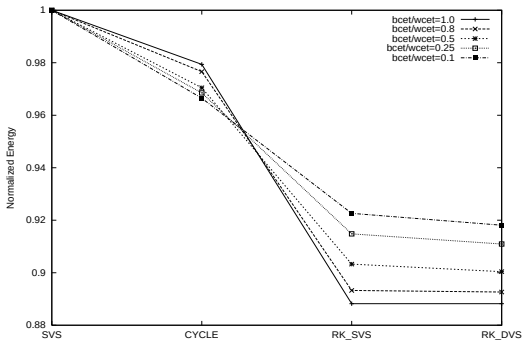


Figure 12: Normalized energy consumption with varying BCET/WCET ratio with $n = 10, U = 0.5$

6.2.2 The Effect of BCET/WCET Ratio

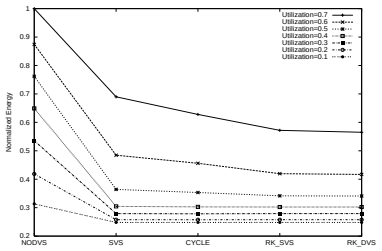
In this experiment, we study the effect of tasks with varying computation times on the performance of DVS algorithms. Figure 12 shows the result of task sets with the processor utilization uniformly distributed in the range $0.45 - 0.55$. Each task set has 10 tasks. We plot their energy consumption normalized to that of the SVS scheme. When tasks use the cycles much less than their worst-case computation times, CYCLE performs better than SVS. In addition, RK-SVS outperforms both CYCLE and SVS by as much as 7.2% and 9.5% for the case of BCET/WCET=0.5. RK-DVS performs about the same as RK-SVS because the slack obtained by RK-DVS's slack estimation algorithm is too small to benefit from the coarse frequency granularity of the Crusoe processor. This will be confirmed in the second set of experiments when we vary the number of available operating points in the processor.

6.2.3 The Effect of Task Set Size

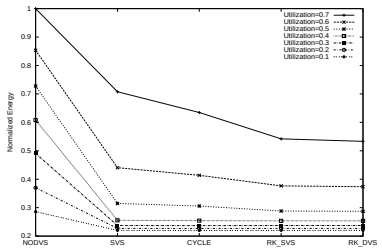
In this experiment, we vary the size of task sets with $U = 0.5$ and $BCET/WCET = 0.5$. Figure 13 shows that as the number of task increases, most DVS algorithms, especially RK-SVS, perform relatively better. This is due to the fact that while SVS picks a single system speed that can satisfy all tasks' constraints, RK-SVS picks one speed for each single task instead. Therefore, when the number of task increases, RK-SVS will switch between different operating frequencies more and consume less energy.

6.3 The Effect of Operating Frequency Granularity

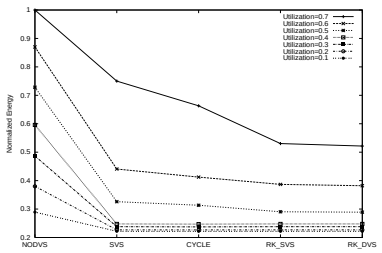
In this set of experiments, we vary the number of available operating frequencies (8, 16, 32 and 64 frequencies) assuming that the processor exhibits the perfect cubic power-frequency function. Task sets are randomly generated with a total system utilization uniformly distributed in the range $0.45 - 0.55$ and $BCET/WCET = 0.5$. Each task set has 10 tasks. Figure 14 shows the energy consumption



(a) 5 tasks



(b) 10 tasks



(c) 15 tasks

Figure 13: The effect of the size of the task sets on DVS algorithms with $n = 10, BCET/WCET = 0.5$

of four voltage-scaling algorithms normalized to the energy consumption in a non-DVS system. As previously discussed in subsection 6.2.2, the performance of RK-DVS improves when the frequency granularity increases since the algorithm can benefit from the slack gain even though the slack gain is small. This is also true for CYCLE for the same reason. In addition, SVS also performs better when the number of operating frequencies increases since the processor can operate close to its desired frequency. In contrast, the chance that RK-SVS benefits from frequency adjustment when the desired operating frequency is not available, is reduced. With this combination, when the number of operating points increases (≥ 32 points), CYCLE starts to perform better than RK-DVS.

7 Related Work

A large number of DVS algorithms has been proposed in the context of both general and real-time systems. Most of them assume that the power consumption of the processor is quadratically dependent on the supply voltage. Therefore, most of them focus on how to determine the lowest frequency that can produce an acceptable performance. However, Miyoshi et al. [7] find that in some processors, there are some operating frequencies which are energy-inefficient in the sense that they consume more energy than some higher frequencies. However, in our paper, a mechanism to efficiently determine

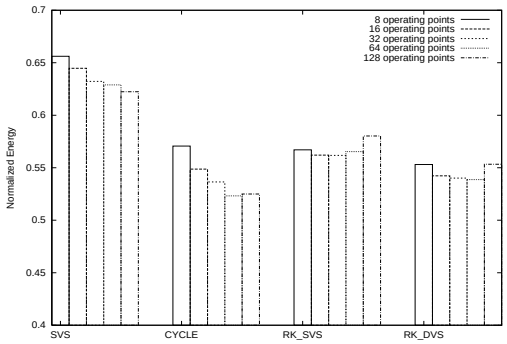


Figure 14: The effect of the number of available operating frequencies with $n = 10, BCET/WCET = 0.5, U = 0.6$

such energy-inefficient operating points has been proposed, especially for processors with a convex non-decreasing power-frequency relation.

Many real-time voltage scaling algorithms have modified the deadline-monotonic scheduling and the earliest-deadline first scheduling policies. While the schedulability test of the voltage-scaling earliest-deadline-first algorithm is rather straightforward, the schedulability test for modified deadline-monotonic is not easy. Our work focuses on this fixed priority approach. We now discuss the related work of static voltage scaling algorithms. We then move on to dynamic voltage scaling.

Aydin et al. [1] proposed the optimal static voltage scaling algorithm using the solution in reward-based scheduling to find the optimal operating frequency of each task. We share the same methodology that jobs from the same task should run at a constant operating frequency while each task can have different operating frequency and the processor has a convex non-decreasing power-frequency function. We find that processors with a convex non-decreasing power still have possibilities to be energy-inefficient while operating at some frequencies. In our scheme, we also take into account the limited number of operating frequencies in the processor. We thus adjust the operating frequency of some task to utilize some slack from tasks which must necessarily run at a higher but available operating frequency.

Pillai et al. [8] also proposed a static voltage scaling algorithm. Instead of running tasks with different speeds, only one system frequency will be determined and used. Besides, their operating frequency is determined by scaling the busy time of the task by its deadline to fill the slack and make sure that the scaled computation meets the task's deadline. This turns to be pessimistic since some preemption by higher priority tasks after the latest possible completion time can be scaled but the task can still meet the deadline.

Many research groups have developed dynamic voltage scheduling algorithms which share a similar methodology to ours, reducing the operating frequencies of tasks whenever there is slack in the system. Kim et al. [4] developed slack estimation algorithms for earliest deadline first (EDF) by reducing the

operating frequency of tasks whenever there is an early completion. Pillai et al. also developed the cycle-conserving RT-DVS and look-ahead DVS algorithms. Both schemes apply to EDF schedulers. Aydin et al. also proposed a dynamic reclaiming algorithm for EDF schedulers.

In our paper, we focus on the deadline-monotonic (DM) scheduling policy. Since the online slack computation of the deadline-monotonic (DM) is much more complex ($O(mn^2)$) than that of EDF [5], we develop a simple dynamic voltage scaling scheme which gives the slack generated by the early completion of a task to the next ready task in the system. This may not be an optimal solution but the computational complexity of slack computation in RK-DVS is very low ($O(1)$)compared to $O(n^2)$ of cycle-conserving RT-DVS for DM proposed by Pillai et al. In addition, RK-DVS performs this slack computation only when there is an early completion of a task while cycle-conserving RT-DVS performs the computation at every instance of a task. As can be seen from the experiments, RK-DVS in the average case performs better than cycle-conserving RT-DVS in terms of energy consumption. It must be noted that RK-DVS initialization uses an energy-saving and schedulable operating frequency computation of RK-SVS which has a relatively high computational complexity ($O(mn^2)$). However, this is acceptable since it is performed only in admission control.

8 Concluding Remarks

Many embedded real-time systems exhibit power and thermal constraints that limit the peak and/or average power that can be consumed. Research interest in such "power-aware" real-time systems has therefore increased in recent years. Energy consumption can be reduced by scaling down the frequency and voltage at which a processor operates, which in turn forces a task to run longer. In this paper, we studied the general scheduling problem of saving energy consumption while still satisfying the real-time constraints of tasks. We study this problem from the practical perspective of supporting new power-aware scheduling policies which can be run on commercially available processors and with acceptable overhead on reservation-oriented modern operating systems. First, we identify that not all operating frequencies on a processor can lead to energy minimization. Such energy-inefficient frequencies must be avoided by all power-aware policies, and we provide an efficient approach to determine such frequencies. Secondly, we study the worst-case quantitative impact of having only a finite number of operating frequencies on a processor. We derive the grid of operating frequencies that minimizes the worst-case *energy-quantization error*, and find that the latter is linear-inversely proportional to the number of available operating frequencies.

Finally, we develop two algorithms, RK-SVS and RK-DVS, which exhibit acceptable complexity levels for saving energy-consumption in reservation-based resource kernels. RK-SVS is a static voltage scaling scheme that picks at admission control time the processor frequency at which each task runs. The uniqueness of RK-SVS is its awareness of energy-inefficient frequencies and the limited number of

operating frequencies. RK-DVS is a dynamic voltage-scaling scheme for resource kernels, and it can further reduce energy consumption by exploiting the slack detected at run-time when tasks execute for less than their worst-case execution times. Our RK-DVS scheme has only a complexity of O(1). Simulation results show that on Transmeta processors, our schemes reduce the energy consumption by 57% relative to not using any voltage-scaling schemes. Comparisons with other known policies show that our schemes provide better or similar performance but with much simpler schemes when the number of operating frequencies is less than 32. The improvements provided by RK-DVS over RK-SVS are relatively minor ($< 5\%$), perhaps arguing for the use of simple but very efficient power-aware scheduling policies.

Our future work will look at optimal solutions for both static and dynamic voltage algorithms. We also plan to implement and evaluate all policies presented in this paper and the optimal algorithms in actual resource kernels (like CMU Linux/RK), and extending this approach to other resources besides the processor.

References

[1] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Mejia-Alvarez. Dynamic and agressive scheduling techniques for power-aware real-time systems. In *Proceedings of the IEEE Real Time Systems Symposium*, London, UK, December 2001.

[2] Marc Fleischmann. Longrun power management-dynamic power management for cruesoe processors, 2001.

[3] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of the 36th Design Automation Conference, DAC 1998*, June 1998.

[4] Woonseok Kim, Jihong Kim, and Sang Lyul Min. A dyanmic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of Design Automation and Test in Europe*, 2002.

[5] J. Lehoczky and S. Ramos-Thuel. Scheduling periodic and aperiodic tasks using the slack stealing algorithm. In *Advances in Real-Time Systems, Sang H. Son, Ed., chapter 8, pp. 175–198. Prentice-Hall, Inc.*, 1995.

[6] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[7] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. Technical report, Department of Electrical and Computer Engineering, Carnegie Mellon University, October 2001.

[8] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[9] T. Sakurai and A. Newton. Alpha-power law mosfet model and its application to cmos inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, April 1990.

[10] Robert Sedgewick. *Algorithms in C.* Number ISBN 0-201-51425-7. Addison Wesley, 1990.

[11] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, June 2001.

[12] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Foundations of Computer Science*, October 1995.