Practical Voltage-Scaling for Fixed-Priority RT-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 equipments are power-constrained, while on-demand multimedia servers have thermal limits which constrain the amount of heat they can dissipate. In CMOS circuits, power consumption is proportional to the product of the frequency and the square of the supply voltage. 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 propose four alternative voltage-scaling algorithms, Sys-Clock, PM-Clock, Opt-Clock and DPM-Clock. Each scheme is suitable for different hardware which may have high or low voltage-scaling overhead and different taskset characteristics. We have implemented our scaling schemes on CMU's real-time OS, Linux/RK, on the 3700 series Compaq iPAQ PDA and a 733MHz XScale BRH board modified to support voltage-scaling. We also study the effect of limited number of operating frequencies in practical processors on the performance of voltage-scaling schemes. The optimal frequency grid which minimizes the effect of discrete operating frequencies is also derived.

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 dynamic power consumption of CMOS circuits [16, 19] is given by $P = aC_LV_{DD}^2f$, where P is the power consumption, a is the average activity factor, C_L is the average load capacitance, V_{DD} is the supply voltage and f is the operating frequency. 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 the increasing delay in CMOS circuits. The maximum speed a processor can operate is a direct consequence of the supply voltage and is given by $f = K\frac{(V_{DD}-V_{th})^{\alpha}}{V_{DD}}$, where K is a proportionality constant specific to a given technology, V_{th} is the threshold voltage and α is the velocity saturation index of CMOS circuits which has value within the range $1 \leq \alpha \leq 2$.

A variety of dynamic voltage scaling techniques [1, 6, 15, 20, 22] has addressed the tradeoffs between system performance and 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 tasks must be satisfied. The above techniques typically assume that

1. the energy consumption in processors is minimized whenever the supply voltage is scaled down.
2. the voltage-scaling overhead is negligible.

Hence, they focus on how to determine the lowest possible frequency in order to achieve an acceptable performance and meet the deadlines of all real-time tasks.

However, these assumptions are not always correct. In some practical processors, Akihiko et. al. [13] showed that there are some energy-inefficient operating frequencies in the sense that operating the same workload with a higher frequency will counter-intuitively consume less energy. Those energy-inefficient operating frequencies therefore must be avoided under all circumstances. In addition, practical 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 energy-efficient operating frequency if a desired frequency is not available or is energy-inefficient. This inevitably results in more energy consumption. We also explicitly consider the effect of the overhead of changing processor frequencies. When clock voltage/frequency changes, internal clocks on the processor and DRAM timings may need to be re-synchronized. Depending upon the implementation, these frequency- and voltage-scaling delays can be either small enough to be negligible or large enough that they can disrupt task set timings significantly.

This paper focuses on DVS algorithms for these practical processors which may have a limited number of operating frequencies, different voltage-scaling overheads and different taskset characteristics relative to worst-case execution times. We first study the effect of finite operating frequencies on the performance of the voltage-scaling algorithms. We also investigate the optimal operating frequency grid on these processors which minimizes loss of energy savings. Finally, we propose four alternative voltage-scaling algorithms, Sys-Clock, PM-Clock, Opt-Clock and DPM-Clock. Each is suitable for different hardware and taskset characteristics and will be selected automatically by our reservation-based resource kernel [3, 12, 14, 17, 18].

- **System Clock Frequency Assignment (Sys-Clock):** This scheme is suitable for systems such as Compaq iPAQ [2] where the overhead of voltage- or frequency- scaling is too high to do frequency-scaling at every context switch. One operating clock frequency for all tasks will be determined during admission control and kept constant until the taskset changes.
- **Priority-Monotonic Clock Frequency Assignment (PM-Clock):** This scheme is suitable for systems with low voltage- or frequency-scaling overhead such as our modified version of XScale BRH board [7] which enables the voltage-scaling feature. Each task will be assigned its own clock frequency. The scheme scales voltage and frequency at every context switch based on the frequency assigned to the next task to run.
- **Optimal Clock Frequency Assignment (Opt-Clock):** This scheme uses non-linear optimization techniques to determine the optimal clock frequency for each task in order to minimize the energy consumption. Opt-Clock has high complexity and is unsuitable for on-line usage. In this paper, we propose several pruning techniques which can dramatically reduce the computational complexity of Opt-Clock.
- **Dynamic Priority-Monotonic Clock Frequency Assignment (DPM-Clock):** This scheme is suitable for systems where the average execution times of tasks are considerably less than their worst-case execution times.

These algorithms assume the deadline-monotonic scheduling policy. However, they can be easily applied to other fixed-priority preemptive scheduling policies.

1.1 System Model and Terminology

We use a reservation-based resource kernel [3, 12, 14, 17, 18] as the system model. A task τ_i has its specification as $\{C_i, T_i, D_i\}$, where C_i is the worst-case required processor cycles, T_i is the period and D_i is the relative deadline from arrival time. The clock frequency we refer throughout the paper is a relative frequency normalized to f_{max}. We assume that all processors have a convex non-decreasing relation between power and frequency which is expected to be true in practice. Throughout this paper, we also use the term " frequency-scaling" which actually scales both frequency and voltage simultaneously if a platform is voltage-scaling enabled.

This paper is organized as follows. Section 2 discusses DVS algorithms proposed in the literature. Section 3 studies the influence of having a limited number of operating frequencies on the performance of DVS algorithms. Sections 4 and 5 describe our system- and task-oriented clock frequency assignment schemes, Sys-Clock, PM-Clock and Opt-Clock respectively. Section 6 describes our dynamic voltage scaling scheme, DPM-Clock. Section 7 presents simulation results and briefly describes our XScale and iPAQ platform implementations. Concluding remarks and comments on future research appear in Section 8.

2 Related Work

A large number of DVS algorithms have been proposed in the context of both general and real-time systems. Many real-time voltage-scaling algorithms have modified the deadline-monotonic (DM) and the earliest-deadline first (EDF) scheduling policies. While the schedulability test of voltage-scaling algorithms for EDF-based schemes is relatively straightforward, the schedulability test of DM-based schemes is not simple.

Our work builds on the foundational results of Pillai and Shin [15], who proposed a wide-ranging class of voltage-scaling algorithms for real-time systems. Their algorithms were also implemented and extensively evaluated on real platforms. We attempt to build on their work, trying to (a) obtain similar performance with lower complexity, (b) find optimal schemes for fixed-priority preemptive scheduling policies, e.g. DM scheme. In their static voltage scaling algorithm, instead of running tasks with different speeds, only one system frequency is determined and used. Besides, their operating frequency is determined by equally scaling all tasks in order to complete the lowest-priority task as late as possible. We will show later that this turns out to be pessimistic. Since the amount of preemption by high-priority tasks is not uniformly distributed when there are multiple task periods, a task can encounter less preemption relative to its own computation and save more energy if it completes earlier than its deadline.

Aydin et al. [1] proposed the optimal static voltage scaling algorithm using the solution in reward-based scheduling. Many of our algorithms adopt their approach of having jobs from the same task run at a constant operating frequency while each task can have a different operating frequency. We also make their assumption that voltage-scaling processors have a convex non-decreasing power-frequency relation. While Aydin et al. focus on the EDF scheduling policy, we focus on fixed-priority preemptive scheduling policies and search for not just the optimal but also sub-optimal, practical schemes for on-line usage. We also emulate their strive to adjust the operating frequency of some tasks to utilize slack from tasks which necessarily run at a higher but available energy-efficient operating frequency. In addition, our schemes take into account the limited number of operating frequencies that are expected to be available.

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 at run-time. Kim et al. [9] developed slack estimation algorithms for EDF policy by reducing the operating frequency of tasks whenever there is an early completion. Pillai et al. whose work was mentioned earlier 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.

Since the on-line slack computation of DM is much more complex ($O(mn^2)$) than that of EDF [10], we develop a simple dynamic voltage scaling scheme called DPM-Clock 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 $O(1)$ computational complexity of slack computation in DPM-Clock is much simpler, compared to the $O(n^2)$ complexity of cycle-conserving RT-DVS for DM proposed by Pillai et al. In addition, DPM-Clock 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 will be seen from the experiments, DPM-Clock in the average case performs comparable to cycle-conserving RT-DVS in terms of energy consumption. It must be noted that DPM-Clock initialization uses an operating frequency computation of PM-Clock which has a relatively high computational complexity of $O(mn^2)$. However, this is acceptable since it is performed only during admission control, and Linux/RK already performs a computation of similar complexity during admission control.

3 The Effect of A Finite Number of Operating Frequencies

Practical processors usually provide a finite operating frequency granularity. 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. We refer to such an energy loss due to finite operating frequencies as the *energy quantization error*.

(a) 2 operating points

(b) N operating points

Figure 1: Processors with a limited number of operating points

We first 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 1(a). A frequency of 0 represents the processor in its *sleep* state. We will verify that f_x should be located at $f_{max}/\sqrt{2}$ 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 eventually 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.

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 $P(f) \propto C_L V_{DD}^2 f \approx kf^3$. 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 and the practical processors respectively. Let P_{idle_f} be the idle power at the operating frequency f. Then, $\Delta E_1 = E_{f_x} - E_\epsilon$ where

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

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 and the practical processors for this case respectively. We have $\Delta E_2 = E_{f_{max}} - E_{f_x}$ and

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

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

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

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 (1) 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 1(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,

$$\begin{aligned} \Delta E_1 &= k\lambda f_1^2 \\ \Delta E_i &= k\lambda f_i^2 - k\lambda f_{i-1}^2 \ for \ \ 2 \le i \le N-2 \\ \Delta E_N &= k\lambda f_{max}^2 - k\lambda f_{N-1}^2 \end{aligned}$$

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

Solving $\Delta E_1 = \Delta E_2 = \ldots = \Delta E_N$, we obtain

$$f_i = \sqrt{\frac{i}{i+1}} f_{i+1} \ for \ \ 1 \leq i < N-1; \ f_{N-1} = \sqrt{\frac{N-1}{N}} f_{max} \tag{2}$$

Figure 2 gives an example of 10 operating frequency grid that minimizes energy-quantization error.

Operating Point	Normalized Freq. (%)	Freq. Range (%)
0	0	31.6
f_1	31.6	13.1
f_2	44.7	10.1
f_3	54.8	8.4
f_4	63.2	7.5
f_5	70.7	6.8
f_6	77.5	6.2
f_7	83.7	5.7
f_8	89.4	5.5
f_9	94.9	5.1
f_{max}	100	-

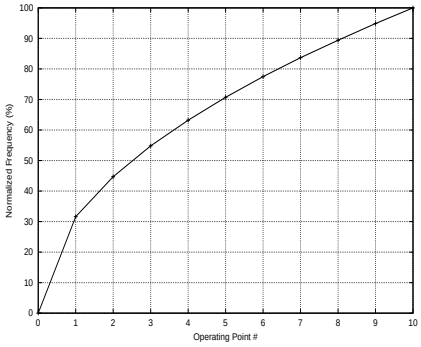


Figure 2: Optimal operating frequencies for a processor with 10 operating points

We now analyze how much the number of operating points affects energy loss. Let ΔE be the worst-case quantization error of a processor with N operating points whose frequency grid is defined by equation (2) in order to minimize the energy loss. Let E_{noDVS} be the energy consumed by the processor without any voltage-scaling, which is given by $E_{noDVS} = k\lambda f_{max}^2$. Since we equalize the energy quantization error at each level, we can just consider ΔE_N. Therefore,

$$\Delta E = \Delta E_N = k\lambda f_{max}^2 - k\lambda f_{N-1}^2 = \frac{k\lambda f_{max}^2}{N} \tag{3}$$

Compare equation (3) to E_{noDVS}. We get

$$\Delta E = \frac{E_{noDVS}}{N} \tag{4}$$

As can be seen in equation (4), the deviation is linear-inversely proportional to the number of operating frequencies. That is, if we want only 10% energy quantization loss compared to having an infinite number of operating points, a processor will need at least $1/0.1 = 10$ operating frequencies.

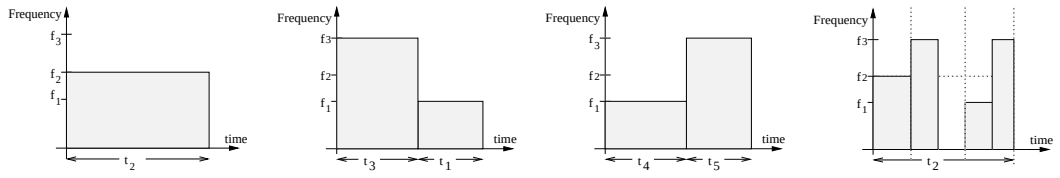
4 System Clock Frequency Assignment

As previously introduced, system clock frequency assignment is suitable for embedded systems where the cost of voltage- or frequency-switching is high. One example for these systems is Compaq iPAQ

[2] which needs 20 milliseconds to synchronize SDRAM timing after switching frequency[2]. In this section, we present how Sys-Clock determines the energy-minimizing clock frequency for the system.

4.1 Energy-Minimizing Clock Frequency

Sys-Clock applies Theorem 1 to calculate the energy-minimizing frequency for the entire taskset.



(a) Constant (b) Scale up/down (c) Scale down/up (d) Mixed

Figure 3: Possible speed settings of a processor

Theorem 1 *Consider processors with different frequency assignments which execute the same amount of workload within a given deadline. A processor that completes the workload exactly at the deadline with a constant speed will consume minimum energy.*

Proof.

Figure 3 shows four processors with different clock frequency settings. For example, the first processor operates at clock frequency f_2 constantly for t_2 time units. Both the scale up/down and the scale down/up processors of Figures 3(b) and 3(c) will consume the same amount of energy if $t_1 = t_4$ and $t_3 = t_5$ and hence only one analysis of either processor is sufficient. In addition, the comparison with all other frequency settings can be partitioned to form the first three cases as shown in Figure 3(d). Therefore, it is sufficient to just compare the first and second processors to draw a conclusion.

Since processors complete the same amount of work (the same number of processor cycles) within the same deadline, we have $(f_2 \times t_2) = (f_1 \times t_1) + (f_3 \times t_3)$ and $t_2 = t_1 + t_3$. Therefore, $t_1 = (\frac{f_3 - f_2}{f_3 - f_1})t_2$ and $t_3 = (\frac{f_2 - f_1}{f_3 - f_1})t_2$. Let $E_{constant}$ and $E_{scale\_up/down}$ be the energy usage of the first and second processors, 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. Under the assumption that the processor's power and frequency relation $P(f)$ is a non-decreasing convex

[2] It must note that iPAQ platform is supporting only frequency scaling but voltage scaling.

function, we get $m_{f_2 f_3} \geq m_{f_1 f_2}$. That is,

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

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

4.2 Sys-Clock Algorithm

Theorem 1 indicates that for a given workload with a deadline, a processor will consume minimum energy if it runs at a constant clock frequency and completes the workload exactly at the deadline. However, in a system using a fixed-priority preemptive scheduling policy, the workload needed to complete a task's request composes of the task's own execution and the preemption by higher priority tasks. When there are multiple task periods, the preemption is not uniformly distributed over the task's critical zone [11][3]. The workload hence varies in preempting processor cycles and depends on when the task completes. Sys-Clock determines the energy-minimizing clock frequency by finding the minimum constant clock frequency that allows every task to complete before its deadline. We will illustrate the algorithm by an example.

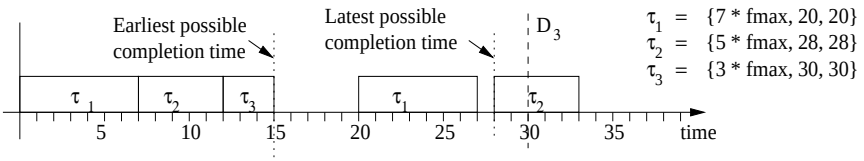


Figure 4: Varying workload vs. completion times

We assume that the deadline-monotonic scheduling policy is used, any higher priority task $\tau_i, D_i < D_j$ will preempt τ_j. Consider a taskset that has three tasks $\{\tau_1, \tau_2, \tau_3\}$ with the task specification $\{7 * f_{max}, 20, 20\}, \{5 * f_{max}, 28, 28\}, \{3 * f_{max}, 30, 30\}$, respectively. Figure 4 depicts the critical zone of task τ_3 at the maximum clock frequency f_{max}. As can be seen, the completion time of τ_3 is 15, which we refer to as *the earliest possible completion time*. If the processor operates at a lower clock frequency, not only the execution time needed by all tasks increases but also the number of preempting processor cycles. As can be seen in Figure 4, τ_3 can be preempted by τ_2 at time 28 to 33. Therefore, if τ_3 cannot complete its job before 28, it will definitely miss its deadline. We refer to this last time

[3] The critical zone assumes that the requests of all tasks arrive simultaneously. This leads to the worst-case scenario.

instant that a task can complete and meet its deadline as *the latest possible completion time*. Since the workload changes at the end of each idle period[4], Sys-Clock determines the constant clock frequencies which allow a task to complete execution exactly at the end of each idle period between the earliest and latest possible completion time. The energy-minimizing clock frequency of a task hence is the minimum clock frequency among these constant clock frequencies.

By convention, we denote the workload needed by a task and its total preemption at time t as β_i^t, the corresponding constant clock frequency as α_i^t and the energy-minimizing clock frequency as ϵ_i, where i represents a task τ_i and t represents the end of an idle period. For this example, the ends of idle periods to consider for τ_3 are 20 and 28. Hence,

$$\beta_3^{t=20} = (C_1 + C_2 + C_3)/f_{max} = 15, \quad \alpha_3^{t=20} = 15/20 = 0.75$$
$$\beta_3^{t=28} = (2C_1 + C_2 + C_3)/f_{max} = 22, \quad \alpha_3^{t=28} = 22/28 = 0.786$$

Therefore, the energy-minimizing of task τ_3, ϵ_3, is given by $min\{\alpha_3^{t=20}, \alpha_3^{t=28}\} = min\{0.75, 0.786\} = 0.75$. This clock frequency is a relative frequency normalized to f_{max}.

Note that the energy-minimizing clock frequency of a task τ_i is the clock frequency needed by τ_i itself *and* its higher priority tasks to minimize the energy and meet τ_i's deadline. However, this frequency may not satisfy other tasks' timing constraints. Therefore, Sys-Clock will set the system clock frequency to the maximum frequency among energy-minimizing clock frequencies of *all* tasks to maintain schedulability. For this example, Sys-Clock assigns the system clock frequency to $max\{\epsilon_1, \epsilon_2, \epsilon_3\}$ $= max\{0.35, 0.60, 0.75\} = 0.75$. Figure 5 summarizes the Sys-Clock algorithm.

4.3 Sys-Clock Optimality

From an energy-minimizing perspective, the Sys-Clock frequency assignment scheme is optimal for fixed-priority preemptive scheduling policies that use a single clock frequency. The only way for other schemes to consume less energy than Sys-Clock is assigning a lower system clock frequency to a taskset. However, lowering the clock frequency than the Sys-Clock frequency which is an energy-minimizing clock frequency of a task will cause the task to miss its deadline. Sys-Clock therefore is optimal among system clock frequency assignment schemes in the sense that there is no other schemes that consume less energy and guarantee timing constraints of a taskset.

5 Task Clock Frequency Assignment

Task clock frequency assignment schemes determine one clock frequency for each task under the assumption that the voltage-scaling overhead at every context switch is acceptable. We will present two

[4] An idle period is a time period where the processor is idle.

During Admission Control:

For each task τ_i: ϵ_i = **Energy-Minimizing-Freq**(C_i, T_i, D_i) **End for**

Sys-Clock-Freq = lowest operating frequency f such that $(f/f_{max}) \geq max_{\forall i}\{\epsilon_i\}$

Energy-Minimizing-Freq(C_i, T_i, D_i)**:**

/* To compute the energy-minimizing clock frequency for task τ_i */

/* S = slack, I = idle duration, t is the end of an idle period */

/* β = workload for time t */

/* IN_BZP is set if the current ω is in busy period */

/* $hp(\tau_i)$ is a set of tasks with priorities higher than or equal to τ_i's. */

$S = I = \beta = \Delta = 0$, $\alpha = 1$, IN_BZP = TRUE

$\omega = C_i/f_{max}$, $\omega' = 0$

Do while $(\omega < D_i)$

  If (IN_BZP == TRUE) **then**

    $\Delta = D_i - \omega$;

    Do while $(\omega < D_i)$ && $(\Delta > 0)$

      $\omega' = (\sum_{j \in hp(\tau_i)} \frac{C_j}{f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)) + S$

      $\Delta = \omega' - \omega$, $\omega = \omega'$

    End while

    IN_BZP = FALSE

  Else

    $I = min_{j \in hp(\tau_i)}\{(T_j * \lceil \frac{\omega}{T_j} \rceil) - \omega, D_i - \omega\}$

    $S = S + I$, $\omega = \omega + I$, $t = \omega$, $\beta = \omega - S$

    If $(\frac{\beta}{t} < \alpha)$ **then**

      $\alpha = \frac{\beta}{t}$

    End if

    IN_BZP = TRUE

  End if

End while

return α

Figure 5: Sys-Clock algorithm

schemes. Opt-Clock is an optimal task clock frequency assignment scheme for fixed-priority preemptive scheduling policies. Unfortunately, Opt-Clock has a high degree of computational complexity. A second scheme, PM-Clock, is sub-optimal. However, it consumes energy very close to that of the optimal scheme but has much less complexity. We first present the simpler PM-Clock scheme.

5.1 Priority-Monotonic Clock Frequency Assignment Algorithm

PM-Clock is an algorithm adapted from Sys-Clock. First, it determines energy-minimizing clock frequencies of all tasks like Sys-Clock. With different timing constraints, it is possible that a high priority task τ_H may have a more stringent timing constraint and hence need a higher clock frequency than a low priority task τ_L. To meet τ_H's deadline, it is necessary to assign a higher clock frequency

than that for τ_L. Besides the timing constraints, in the case of processors with limited operating frequencies, it is also possible that the desired energy-minimizing clock frequency is not available, and therefore a higher frequency needs to be chosen. In either case, PM-Clock will save more energy by recalculating a lower clock frequency for tasks such as τ_L exploiting the additional slack from these higher frequency assignments used by tasks such as τ_H.

We illustrate the algorithm first using an example. Consider a set of three tasks: τ_1 $\{5 * f_{max}, 10, 10\}$, τ_2 $\{2 * f_{max}, 15, 15\}$ and τ_3 $\{1 * f_{max}, 30, 30\}$. Following the energy-minimizing clock frequency calculation described in Section 4.2, we get

$$\begin{aligned}
\epsilon_1 &= min\{\alpha_1^{t=10}\} = min\{5/10\} = 0.5 \\
\epsilon_2 &= min\{\alpha_2^{t=10}\} = min\{7/10\} = 0.7 \\
\epsilon_3 &= min\{\alpha_3^{t=10}, \alpha_3^{t=20}, \alpha_3^{t=30}\} = min\{8/10, 15/20, 20/30\} = 0.67
\end{aligned}$$

As can be seen, even though τ_3 only requires τ_1 and τ_2 to run at 0.67, both τ_1 and τ_2 must run at a *higher* operating frequency of 0.7 to satisfy τ_2's schedulability. We refer to this higher frequency as an *inflated frequency* which respects to τ_3. The use of an inflated frequency by a higher priority task leads to more available slack for a lower priority task such as τ_3. PM-Clock then recalculates the new energy-minimizing clock frequency of τ_3 by fixing the clock frequency of τ_1 and τ_2 to their inflated frequency. Figure 6 shows the critical zone of τ_3 before and after the effect of inflated frequency. The new slack for τ_3 becomes $30 - (3C_1 + 2C_2)/(0.7 * f_{max}) = 2.857$. Therefore, the clock frequency for τ_3 can be reduced to $1/2.857 = 0.35$ to fill up the additional slack. In the rest of this paper, we will denote the task clock frequency assigned by PM-Clock as v_i where i is the task index. Figure 7 summarizes the PM-Clock algorithm. The body of **Energy-Minimizing-Freq** function is the same as that shown in Figure 5.

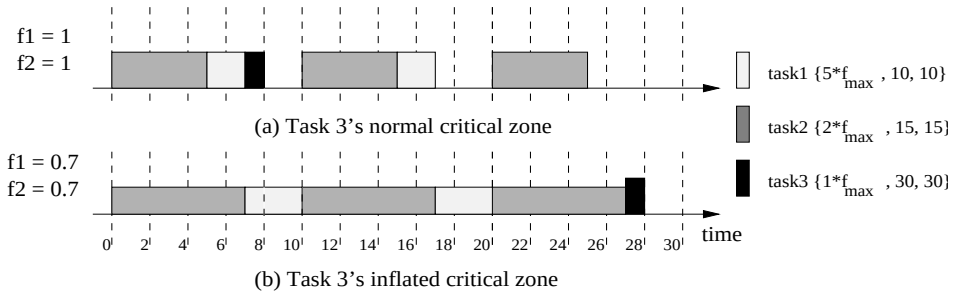


Figure 6: An Example of PM-Clock

Note: The term "priority-monotonic" comes from the fact that the clock frequency of a task assigned by PM-Clock will always be in priority order. In other words, a higher priority task will always be assigned a clock frequency higher than or equal to that of a lower priority task.

We now provide an example to show that PM-Clock is not optimal from an energy-saving per-

/* Assume that the taskset has n tasks and $D_1 \leq D_2 \leq D_3 \ldots \leq D_n$ */
During Admission Control:
For each task τ_i: $v_i = 0$, ϵ_i =**Energy-Minimizing-Freq**(C_i, T_i, D_i) **End for**
For $i = 1$ to n:
/* lp(τ_i) is a set of tasks whose priority is same or lower than τ_i's priority */
v_i = lowest operating frequency f such that $(f/f_{max}) \geq max_{\forall j \in lp(\tau_i)} \epsilon_j$
If (i != 1) and $(v_{i-1} > v_i)$ **then** /* Inflated frequency is detected */
$v_i = 0$
For $j = i$ to n: ϵ_j = **Inflated-Energy-Minimizing-Freq**(C_j, T_j, D_j) **End for**
End if
v_i = lowest operating frequency f such that $(f/f_{max}) \geq max_{\forall j \in lp(\tau_i)} \epsilon_j$
End for
Inflated-Energy-Minimizing-Freq(C_i, T_i, D_i):
/* S = slack, I = idle duration, t is the end of an idle period */
/* $inflated\_\beta$ and β are inflated and scalable workload for time t, respectively*/
/* δ = scalable time, IN_BZP is the busy period flag*/
/* $hp(\tau_i)$ is a set of tasks with priorities higher than or equal to τ_i's. */
$S = I = \beta = \Delta = \delta = 0$, $\alpha = 1$, IN_BZP = TRUE
$\omega = C_i/f_{max}$, $\omega' = 0$
Do while $(\omega < D_i)$
If (IN_BZP == TRUE) **then**
$\Delta = D_i - \omega$;
Do while $(\omega < D_i)$ && $(\Delta > 0)$
$inflated\_\beta = 0, \beta = 0$
For $j = 1$ to n:
If $(D_j \leq D_i)$ && $(v_j != 0))$ **then** /* Inflated frequency is required */
$inflated\_\beta = inflated\_\beta + \frac{C_j}{v_j * f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)$
End if
If $(D_j \leq D_i)$ && $(v_j == 0))$ **then** /* This task is still scalable */
$\beta = \beta + \frac{C_j}{f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)$
End if
End for
$\omega' = inflated\_\beta + \beta + S$
$\Delta = \omega' - \omega$, $\omega = \omega'$
End while
IN_BZP = FALSE
Else
$I = min_{j \in hp(\tau_i)} \{(T_j * \lceil \frac{\omega}{T_j} \rceil) - \omega, D_i - \omega\}$
$S = S + I$, $\omega = \omega + I$
$t = \omega$, $\delta = t - inflated\_\beta$
If $(\frac{\beta}{\delta} < \alpha)$ **then**
$\alpha = \frac{\beta}{\delta}$
End if
IN_BZP = TRUE
End if
End while
return α

Figure 7: The Priority-Monotonic Clock Frequency Assignment, PM-Clock

spective. Essentially, PM-Clock minimizes energy consumption in the Liu and Layland critical zone, but does not minimize energy in the hyperperiod of the taskset. The Opt-Clock scheme to be presented next minimizes energy consumption over a hyperperiod (which will be repeat). Consider the same taskset in Figure 6. Opt-Clock will use non-linear optimization programming to minimize energy consumption over a hyperperiod where the objective function is the energy given by $E = k(15 * f_1^2 + 4 * f_2^2 + f_3^2)$ and the constraint set is as follows.

$$5/f_1 \leq 10$$

$$5/f_1 + 2/f_2 + 1/f_3 \leq 10$$

$$15/f_1 + 4/f_2 + 1/f_3 \leq 30$$

Using the Opt-Clock scheme, we have $f_1 = 0.6783, f_2 = 0.7609, f_3 = 0.3805$, which consumes energy by $9.3617k$. Instead, PM-Clock scheme assigns $f_1 = 0.7, f_2 = 0.7, f_3 = 0.35$, which consumes energy by $9.4325k$.

5.2 Optimal Task Clock Frequency Assignment Algorithm

We will first show that finding the optimal clock frequency for each task to minimize energy is not a trivial problem but can be a large-scale non-linear optimization problem. Then we present pruning techniques which significant decrease the problem size.

Lemma 1 *Consider a voltage-scaling processor with a convex non-decreasing power-frequency function given by* $P = aC_L V_{DD}^2 f$ *where* $f = K\frac{(V_{DD} - V_{th})^\alpha}{V_{DD}}$, $1 \leq \alpha \leq 2$. *Assume that a fixed-priority preemptive scheduling policy is used. The optimal task clock frequency assignment algorithm is a minimization problem whose objective function is convex.*

Proof.

The goal of the optimal task clock frequency assignment algorithm is to minimize the energy consumption of a taskset in one hyper-period[5]. Combining the power and frequency functions in the lemma statement, we get $P(f) = cf^x, x \geq 3$ where c is a constant value. Each instance of a task τ_i takes $C_i/(f_i * f_{max})$ time unit where C_i is the number of required processor cycles and f_i is the relative clock frequency normalized to f_{max}, $0 < f_i \leq 1$. The energy consumption of the task set, E, is as follows.

$$\begin{aligned} E &= \sum_{i=1}^{n} c * (H/T_i) * (C_i/(f_i * f_{max})) * (f_i * f_{max})^x, x \geq 3 \\ &= c * H \sum_{i=1}^{n} \mathcal{U}_i * f_{max}^y * (1/\delta_i)^y, y \geq 2 \end{aligned}$$

[5]Consider the system of n periodic tasks $\tau_1, \ldots, \tau_n$ with $\tau_i = \{C_i, T_i\}$, for $1 \leq i \leq n$. The taskset's hyper-period, denoted as H, is given by $LCM\{T_i | i = 1, \ldots, n\}$.

where (H/T_i) determines the number of τ_i's instances in one hyper-period, $\mathcal{U}_i$ is the task utilization given by C_i/T_i and δ_i is the reciprocal of f_i. From [4], if $f(x)$ has a second derivative in $[a, b]$ then a necessary and sufficient condition for it to be convex on that interval is that the second derivative $f''(x) > 0$ for all x in $[a, b]$. Applying this theorem and properties of convex functions that the combination and the scaling of convex functions are also convex, we can prove that this problem is a minimization problem (to minimize energy) whose objective function $E(\delta_1, \ldots, \delta_n)$ (the energy consumption over a hyperperiod) is convex. □

Theorem 2 *Consider the same voltage-scaling processor stated in Lemma 1. The optimal task clock frequency assignment algorithm is a convex non-linear minimization problem.*

Proof.
As described in Subsection 4.2, a task's workload[6] varies with its completion time, i.e. the clock frequency assigned to each task. Consequently, each task has multiple choices of constraints in order to complete the workload at any time before its deadline. For example, consider the taskset shown in Figure 4. The constraint of task τ_2 can be either one of the following conditions.

$$\begin{aligned} (20/T_1) * (C_1 * \delta_1/f_{max}) + (20/T_2) * (C_2 * \delta_2/f_{max}) &\leq 20 \\ (28/T_1) * (C_1 * \delta_1/f_{max}) + (28/T_2) * (C_2 * \delta_2/f_{max}) &\leq 28 \end{aligned}$$

where the first and second equations are necessary conditions to complete τ_2 before time 20 and 28, which are the end of idle periods. These constraints are linear and can be considered as convex. Therefore, the optimal task clock frequency assignment is a convex non-linear minimization problem since its non-linear objective function and constraint set are both convex [21]. □

As previously shown in Theorem 2, the optimal task clock frequency assignment needs to solve several non-linear optimization problems of one objective function with alternative constraint sets. The problem size grows dramatically not only with the number of tasks but also the number of constraint choices of each task. This latter number depends on the number of idle periods in the task's critical zone and can be very large if the ratio of the task's and the highest priority task's periods is large. For the rest of this section, we will present three pruning techniques which potentially decrease the problem size:

1. **Pruning of High Workload Constraints:**
   Consider a task's alternative constraints, a constraint is said to be a *high-workload constraint* if it requires a shorter completion time β_S and has higher average processor demands from all higher priority tasks than another constraint with a longer completion time β_L. This technique eliminates such redundant high workload constraints. Since the average demands from all tasks

[6] Recall that the task's workload composed of its own execution and its preemption by higher priority tasks

decrease if the task completes at β_L, using the optimal frequency set that satisfies the constraint for β_S will definitely generate some slack. Consequently, some task's clock frequencies can always be reduced to save more energy. Therefore, the solution from the constraint set for β_S will impossibly give the optimal solution and can be pruned.

2. **Pruning of Conflict Constraints:**
   From constraints left over from the pruning of high workload constraints, this technique eliminates unfeasible combinations of constraints where a lower priority task requires a longer completion time than a higher priority task, which does not schedulable by fixed-priority preemptive scheduling.

3. **Pruning of Inactive Constraints:** Unlike the first two techniques, this technique uses additional information from a solution of one problem to prune others. It first determines an inactive constraint[7] from the current solution. Apply the Karush-Kuhn-Tucker (KKT) theorem that the optimal solution subject to set of active and inactive constraints is the same as that subject to only active constraints. Since a better result is not obtained by adding more constraints to the problem, it is sufficient to eliminate other constraint sets which alter only the inactive constraints.

Our simulation experiments indicate that using our proposed pruning techniques, the number of constraint sets for 10 tasks can be reduced from $2 * 10^{13}$ to 1000 on the average, a significant drop. However, other experiments indicate that PM-Clock delivers almost as much energy savings on the average ($E_{PM-Clock}/E_{Opt-Clock} \approx 101\%$) but at significantly lower complexity.

6 Dynamic Clock Frequency Assignment

In this section, we present an on-line DVS scheme which can monitor actual execution times of tasks, and further minimize energy consumption when those execution times are less than the pre-reserved and guaranteed worst-case execution times. Our dynamic priority-monotonic clock frequency assignment (DPM-Clock) detects the early completion times and makes use of the additional slack time created at run-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 on optimal operating frequency setting, DPM-Clock 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 on-line in real-time, but

[7] An inequality constraint, $g(\delta) \leq D$, is said to be *inactive* at δ_x if $g(\delta_x) < D$. It is *active* at δ_x if $g(\delta_x) = D$.

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, DPM-Clock instead devotes the slack to only the current job of the next ready task which has lower or same priority. 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 assigned by PM-Clock 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 its original operating frequency. In detail, DPM-Clock works as follows. Since τ_i has an original operating frequency of υ_i, it would take at most $WCET_i = \frac{C_i}{\upsilon_i f_{max}}$ units of execution time to complete its job. Assume that the task completes its current invocation early after it has executed for CEC_i units of processor cycles. The elapsed slack time denoted as S is given by $WCET_i - CET_i$ where $CET_i = CEC_i/(\upsilon_i * f_{max})$. This slack time will be dedicated to the next ready task which has lower or same priority. Let us assume that the next ready task is τ_j. Since τ_j has more slack time, it can scale down its frequency and let the rest of its execution fill up the slack. This new operating frequency of τ_j is given by $\upsilon_j' = (EC_j/f_{max})/((EC_j/(\upsilon_j * f_{max})) + S)$ where υ_j is the current operating frequency for the current invocation of task τ_j. In the case that there is no ready task in the queue after the early completion of an invocation, DPM-Clock 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 (ie. greater than 0). The DPM-Clock computation of the new frequency has complexity $O(1)$, and is easy to perform at every context switch when an early completion occurs. In addition, $WCET_i$ and ET_i of each task must be tracked. Resource Kernels (RK), such as CMU's Linux/RK [14], already perform very similar computations at context switches, and hence DPM-Clock is a minor increment to RK OS routines.

It is easy to show that the above 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 always choose the next ready task to have the same or lower priority than the task which completed early. Since only the slack gained is filled by frequency reduction, the preemption time seen by other lower priority tasks will be the same.

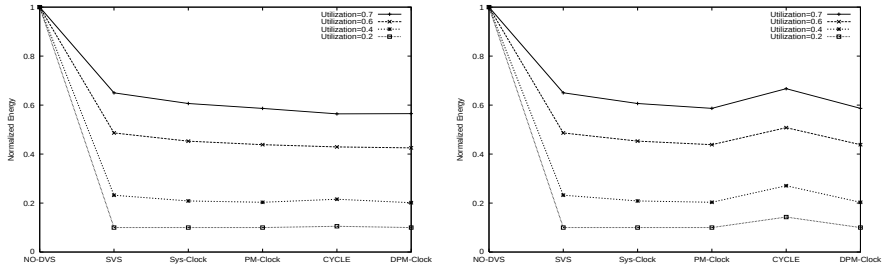
7 Experimental Results

In this section, we present the results of extensive simulations of our voltage-scaling algorithms. We compare them with foundation algorithms called the static voltage scaling algorithm (SVS) and the cycle-conserving RT-DVS (CYCLE) proposed in [15]. The processor is modeled based on $P = kf^3$ power-frequency relation, zero idle energy and 10 available operating points. A real-time taskset is

generated randomly. Each task has a uniform probability of having a short (0.1-1 ms), medium (1-10 ms) or long (10-100 ms) period. Task period is uniformly distributed in each range. Task computation is randomly selected and then adjusted based on:

- **The system utilization**: All tasks in the random taskset are scaled down equally by the ratio of their total utilization and the desired utilization.
- **The ratio of Best-Case (BCET) to Worst-Case Execution Time (WCET)**: For each instance, a task requests a random number of processor cycles which is assigned randomly and uniformly distributed between BCET and WCET.

We compare the energy consumption of six voltage-scaling algorithms and study the effect of the system utilization, BCET/WCET, the number of available operating points and the number of tasks.



(a) BCET/WCET = 0.5 (b) BCET/WCET = 1.0

Figure 8: Normalized energy consumption with varying system utilization

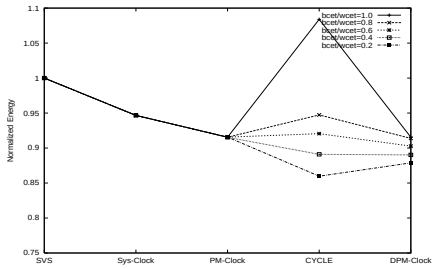
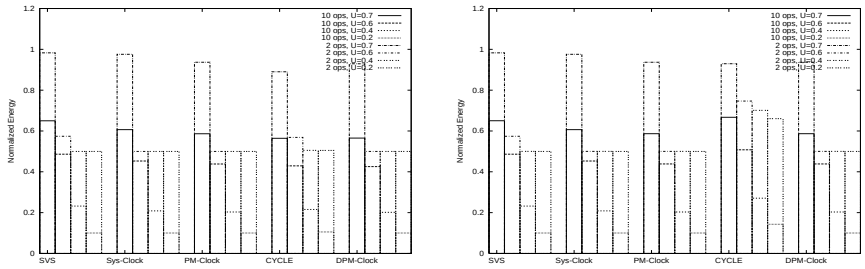


Figure 9: Normalized energy with varying BCET/WCET ratio at $U = 0.5$

In the first experiment, we determine the effect of varying system utilization. Figure 8 shows the average energy consumption normalized to energy of no-DVS system. Each taskset has 10 tasks and BCET/WCET = 0.5 and 1.0. In the second experiment, we determine the effect of BCET/WCET

ratio at system utilization 0.5 as shown in Figure 9 where the energy shown is normalized to energy of SVS algorithm. As can be seen from both experiments, Sys-Clock and PM-Clock always outperform SVS. This is due to the fact that both algorithms determine the frequency needed to complete the task at the end of the idle period which has minimum workload, not the deadline as SVS does. PM-Clock performs better than Sys-Clock as expected with the tradeoff of additional voltage-scaling overhead during context switch. Figure 9 shows that the cycle-conserving algorithm performs very well when the BCET/WCET is low. At BCET/WCET=1.0, the cycle-conserving algorithm performs somewhat poorly because it always executes any task at low speed with the hope of saving energy in the future if the task uses less resource. DPM-Clock performs very close to the cycle-conserving scheme when BCET/WCET is low, and better otherwise. It also does so with much less complexity.

In the third experiment, we do the same analysis on a processor with only 2 operating points and compare the energy consumption normalized to that of no-DVS system with the previous results. As can be seen in Figure 10, all voltage-scaling algorithms perform worse on the processor with the less number of available operating points. In Section 3, we showed that the minimum energy loss due to finite operating frequencies is given by E_{noDVS}/N where N is the number of available frequencies.



(a) BCET/WCET = 0.5 (b) BCET/WCET = 1.0

Figure 10: Performance comparison among processors with different number of operating points

In the fourth experiment, we vary the size of tasksets and fix system utilization and the BCET/WCET ratio. Simulation results show that there is not much effect of the size of the taskset on the performance of voltage-scaling schemes.

7.1 Voltage-Scaling Implementations in Linux/RK

We have implemented these voltage-scaling schemes on CMU's Linux/RK, extensions of which are commercially available from TimeSys Corporation. We refer to the resulting kernel "Power-Aware Linux/RK". We currently have two different hardware targets for Power-Aware Linux/RK. The first of these targets is the 206MHz Compaq iPAQ H3700 personal digital assistant[2], a popular and easily

accessible target not unlike the ubiquitous PC. On this target, frequency scaling is possible but not voltage-scaling. Also, frequency scaling requires re-synchronization between the CPU and SDRAM timing, which in turn takes a relatively astronomical delay of 20ms. In other words, once frequency scaling is initiated, the processor becomes unavailable for 20ms! Such a delay is unacceptable for many real-time systems. Linux/RK therefore uses the Sys-Clock algorithm to scale the frequency based on the taskset workload only at admission-control time or on task deletion[8]. The changes to the Linux/RK kernel are therefore confined to the admission control and task exit modules. These changes constitute less than 100 lines of code. Another quirk of the iPaq hardware is that the entire range of operating frequencies (about 70MHz - 206MHz) is *not* usable: the processor can operate within only one of two non-overlapping ranges (70MHz-140Hz or 155MHz-206MHz). The frequency cannot be scaled from a value within one range to a value in another range. Multimedia applications including a music player have been ported to demonstrate the functionality of our kernel. Actual measurements of energy savings are currently underway. Demonstrations of this technology will be given at RTAS 2003. This kernel with its support for frequency-scaling can be downloaded from `http://www.cs.cmu.edu/~rtml`.

As already known, more significant energy savings can be obtained by voltage-scaling. We have successfully modified the XScale BRH single-board computer, which boasts a 733MHz XScale processor an 128MB memory [7] to support voltage scaling. The Maxim 1855 evaluation kit [8], which is a high-power dynamically adjustable notebook CPU power supply application circuit, was used to provide the programmable supply voltage for the XScale processor card. The kit provides a digitally adjustable voltage from 0.6 to 1.75 V. Our hardware modifications are similar to those made previously in [5] to enable voltage scaling on an 80200 evaluation board. A total of 11 voltage settings (1.0 to 1.5 V with a step of 0.5 V)[9] is available for the OS. A particular value is chosen by simply writing a 4-bit value to a memory-mapped address. Voltage and resulting frequency changes take a negligible amount of time (of the order of a few microseconds). We have ported Power-Aware Linux/RK with Sys-Clock, PM-Clock and DPM-Clock algorithms to this modified BRH board. The Linux kernel has successfully been loaded and tested. With the low voltage-scaling overhead, the modified XScale BRH board can efficiently save more energy using the PM-Clock and DPM-Clock schemes. We will report on our experiments with this board later.

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

[8] We would like to thank Gaurav Bhatia for help with this implementation.

[9] The 80200 XScale processor can operate with the supply voltage from 0.95 to 1.55 V.

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 minimizing 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. 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 that the latter is linear-inversely proportional to the number of available operating frequencies.

Four alternative voltage-scaling algorithms, Sys-Clock, PM-Clock, Opt-Clock and DPM-Clock, have been proposed. Sys-Clock, PM-Clock and DPM-Clock schemes exhibit very acceptable complexity levels and deliver a significant amount of energy savings (50% at 50% system utilization) for on-line usage. The Opt-Clock is an optimal scheme for fixed-priority preemptive scheduling systems, but can only be used off-line despite the dramatic drop in computational complexity brought about by the use of three pruning techniques. Our simulation results also show that on the average for a taskset with 10 tasks, PM-Clock performs very close to Opt-Clock ($E_{PM-Clock}/E_{Opt-Clock} \approx 101\%$) even though it has much less complexity. We have proved that Sys-Clock is optimal for fixed-priority preemptive scheduling policies that use a single clock frequency. In addition, our dynamic voltage scaling scheme, DPM-Clock, has an additional complexity of $O(1)$ and its improvements over PM-Clock are relatively minor ($< 5\%$), perhaps arguing for the use of simple but very efficient power-aware scheduling policies.

We have implemented Sys-Clock algorithm on the Compaq iPAQ. We also have implemented Sys-Clock, PM-Clock and DPM-Clock on our the XScale BRH board modified to support voltage scaling. Our future work will measure the system-wide energy savings on those platforms. We also plan to investigate energy minimization techniques in cache, memory, disk, wired and wireless network, and other peripherals.

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] Compaq Computer Corporation. Compaq ipaq h3600 hardware design specification - version 0.2f. http://www.handhelds.org/Compaq/iPAQH3600/iPAQ_H3600.html.

[3] Dionisio de Niz, Luca Abeni, Saowanee Saewong, and Ragunathan (Raj) Rajkumar. Resource sharing in reservation-based systems. In *Proceedings of the IEEE Real Time Systems Symposium*, London, UK, December 2001.

[4] I. S. Gradshteyn and I. M. Ryzhik. *Tables of Integrals, Series, and Products, 6th ed.* Academic Press, San Diego, CA, 2000.

[5] Shalabh Gupta. Voltage scaling hardware for intel xscale 80200 evaluation platform. http://www.ee.ucla.edu/~shalabh/pads/vscaling.html.

[6] 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.

[7] ADI Engineering Inc. Brh reference platform. http://www.adiengineering.com/productsBRH.html.

[8] Maxim Integrated Products Inc. Maxim 1855 evaluation kit reference. http://pdfserv.maxim-ic.com/arpdf/MAX1716EVKIT-MAX1855EVKIT.pdf.

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

[10] 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.

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

[12] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems Computing System*, May 1994.

[13] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, June 2002.

[14] Shui Oikawa and Raj Rajkumar. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, Vancouver, June 1999.

[15] 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.

[16] B. M. Gordon R. Gonzalez and M. A. Horowitz. Supply and threshold voltage scaling for low power cmos. *IEEE Journal of Solid-State Circuits*, 32(8), Auguest 1997.

[17] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[18] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, Vienna, Austria, June 2002.

[19] 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.

[20] 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.

[21] Rangarajan K. Sundaram. *A First Course in Optimization Theory*. Cambridge University Press, 1996.

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