Practical Voltage-Scaling for Fixed-Priority RT-Systems

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

Abstract

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 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 while guaranteeing the schedulability of the real-time tasks. In this paper, we study the effect of limited number of operating frequencies on the performance of voltage-scaling algorithms. The optimal frequency grid which minimizes the effect of discrete operating frequencies is also derived. We then propose four alternative voltage-scaling schemes, Sys-Clock, PM-Clock, Opt-Clock and DPM-Clock. Each scheme is suitable for different hardware configuration which may have high or low voltage-scaling overhead and different taskset characteristics. We have implemented our voltage-scaling schemes on CMU's real-time OS, Linux/RK, on the 3700 series Compaq iPAQ and a 733MHz XScale BRH board modified to support voltage-scaling.

1. Introduction

The field of dynamic voltage scaling is currently the focus of a great deal of research interest. This is due to the fact that the dynamic power consumption of CMOS circuits [13, 16] is given by $P = aC_L V_{DD}^2 f$, where P is the power, 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 voltage down is the most effective way to minimize energy. However, lowering the supply voltage can also adversely affect the system performance due to the increasing delay. The maximum operating speed is a direct consequence of the supply voltage given by $f = K\frac{(V_{DD}-V_{th})^\alpha}{V_{DD}}$, where K is a constant specific to a given technology, V_{th} is the threshold voltage and α is the velocity saturation index where $1 \leq \alpha \leq 2$.

A variety of dynamic voltage scaling techniques [2, 6, 12, 18, 20] has addressed the tradeoffs between system performance and energy efficiency. These techniques make use of operating system information about the current workload 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 is minimized whenever the supply voltage is scaled down and (2) the voltage-scaling overhead is negligible.

These assumptions may not always be correct in practice. In some practical processors, Akihiko et al. [11] 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, operating frequencies usually 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 neither available nor energy-efficient. This inevitably results in more energy consumption.

We must also explicitly consider the effect of the overhead of changing processor frequencies. When frequency changes, internal clocks on the processor and DRAM timings may need to be re-synchronized. Depending upon the implementation, these frequency- or voltage-scaling delays can be either small enough to be negligible or large enough that they can disrupt taskset timings significantly.

This paper proposes four alternative voltage-scaling algorithms. Each is suitable for different system characteristics and will be selected automatically by our reservation-based resource kernel [4, 14, 15].

- **System Clock Frequency Assignment (Sys-Clock)** is suitable for systems such as iPAQ [3] where the overhead of voltage- or frequency- scaling is too high to perform at every context switch. *One* clock frequency is determined at admission control and kept constant until the taskset changes.
- **Priority-Monotonic Clock Frequency Assignment (PM-Clock)** is suitable for systems with low voltage- or frequency-scaling overhead such as our modified

version of the XScale BRH board [1]. Each task is 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)** 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. Several pruning techniques are proposed to dramatically reduce the scheme's complexity for off-line usage.
- **Dynamic PM-Clock (DPM-Clock)** is suitable for systems where the average execution time of a task is considerably less than its worst case.

These algorithms assume the deadline-monotonic scheduling policy. However, they can be easily applied to other fixed-priority preemptive scheduling policies by computing the preemption time based on the assigned task priorities.

1.1. System Model and Terminology

We use a reservation-based resource kernel [4, 14, 15] as the system model. A task τ_i has its specification as $\{C_i, T_i, D_i\}$, where C_i is its worst-case required processor cycles, T_i is its period and D_i is its relative deadline from arrival time. The clock frequency we refer throughout the paper is a relative frequency normalized to f_{max}. We assume processors have a convex non-decreasing power-frequency relation which is expected to be true in practice. We use the term "frequency-scaling" which actually scales both frequency and voltage simultaneously if a platform is voltage-scaling enabled.

The paper is organized as follows. Section 2 discusses related work. Section 3 studies the influence of a limited number of operating frequencies on the performance of DVS algorithms. Sections 4 and 5 describe Sys-Clock, PM-Clock and Opt-Clock respectively. Section 6 describes our dynamic 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

Our work builds on the results of Pillai and Shin [12], 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. Their operating frequency is determined by equally scaling all tasks to complete the lowest-priority task as late as possible. 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.

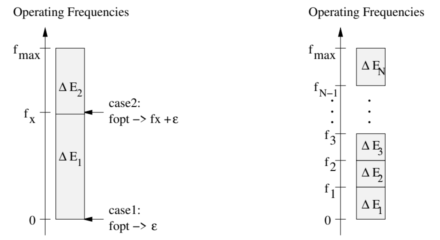
Aydin et al. [2] proposed the optimal static voltage scaling algorithm using the solution in reward-based scheduling. Many of our algorithms adopt their approach of having jobs from the same task run at a constant operating frequency while each task can have a different frequency. We also make their 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. In addition, our schemes take the existence of discrete and energy-inefficient operating frequencies into account.

Many researchers 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. [7] developed slack estimation algorithms for the EDF policy 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 for 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 [8], we develop a simple dynamic scheme called DPM-Clock which gives the slack generated by the early completion of a task only 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 while cycle-conserving scheme performs the computation at every arrival 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 savings. 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 Finite Frequency Granularity

Practical processors usually provide a finite number of operating frequencies. To sustain 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 as the *energy quantization error*.



(a) 2 operating points (b) N operating points

Figure 1. Processors with finite frequencies

We first investigate an operating frequency grid which minimizes the worst-case energy quantization error for a 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 show that f_x should be located at $f_{max}/\sqrt{2}$ or $0.707 f_{max}$.

Let f_{opt} be the desired operating frequency. 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. For the first case, the largest error occurs when $f_{opt} = \epsilon$, $\epsilon \to 0$ and the processor has to run at f_x. Similarly, for the latter case, the largest error occurs when $f_{opt} = f_x + \epsilon$.

If f_x is close to zero, ΔE_1 is definitely less than ΔE_2 and the worst-case energy quantization error, ΔE is given by $\Delta E = max\{\Delta E_1, \Delta E_2\} = \Delta E_2$. When we increase f_x, ΔE_2 is decreased, whereas ΔE_1 is increased. As long as $\Delta E_1 \leq \Delta E_2$, this increment of f_x will reduce ΔE. If we further increase f_x where $\Delta E_1 > \Delta E_2$, $\Delta E = max\{\Delta E_1, \Delta E_2\} = \Delta E_1$ will be increased. Therefore, we should place f_x where $\Delta E_1 = \Delta E_2$ 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_{I_f} be the idle power at the operating frequency f. Then, $\Delta E_1 = E_{f_x} - E_\epsilon$ where

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

Similarly, let E_{f_x} and $E_{f_{max}}$ be the energy needed by the ideal and practical processors when the desired frequency is given by $f_x + \epsilon$. Then, $\Delta E_2 = E_{f_{max}} - E_{f_x}$ and

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

Solving $\Delta E_1 = \Delta E_2$ where ϵ is negligible, we obtain

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

For large values of λ[1], the second term in the square root of equation (1) is negligible. Therefore, f_x should be placed at $f_{max}/\sqrt{2}$ to minimize the energy quantization error.

A similar approach can be employed for a processor with N operating points as shown in Figure 1(b). Solving $\Delta E_1 = \ldots = \Delta E_N$, we obtain

$$\begin{aligned} f_i &= \sqrt{\frac{i}{i+1}} f_{i+1},\ i \in [1, N-1); \\ f_{N-1} &= \sqrt{\frac{N-1}{N}} f_{max} \end{aligned} \quad (2)$$

We can now analyze how the number of operating points affects 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 obtain

$$\begin{aligned} \Delta E &= \Delta E_N = k\lambda f_{max}^2 - k\lambda f_{N-1}^2 = \frac{k\lambda f_{max}^2}{N} \\ \Delta E &= \frac{E_{noDVS}}{N} \end{aligned} \quad (3)$$

As can be seen in equation (3), the deviation is linear-inversely proportional to the number of operating frequencies. That is, if we want only 10% energy quantization loss from the ideal processor, a processor will need at least $1/0.1 = 10$ operating frequencies. Figure 2 gives an example of an optimal frequency grid for a processor with 10 operating frequencies.

4. Sys-Clock Algorithm

Sys-Clock applies a well-known property of a processor with a non-decreasing convex power-frequency function: the energy is minimized if the processor executes its

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

Point	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_{10}	100	-

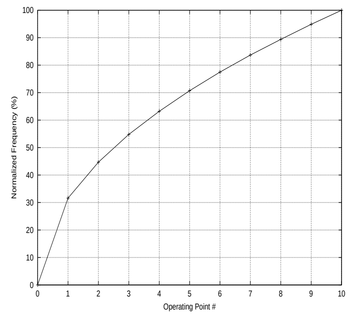


Figure 2. Optimal freq. grid for 10 frequencies

workload at the lowest possible *constant* speed. 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 [9][2]. 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 clock frequency that allows every task to complete before its deadline. We will illustrate the algorithm by an example.

We assume DM scheduling policy is used, and that 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 as shown in Figure 3. The timeline 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. Since τ_3 can be preempted by τ_2 at time 28 to 33, the task will definitely miss its deadline if it cannot complete before time 28. We refer to this last time 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[3], Sys-Clock determines the 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 is chosen to be the minimum clock frequency among these 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

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

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

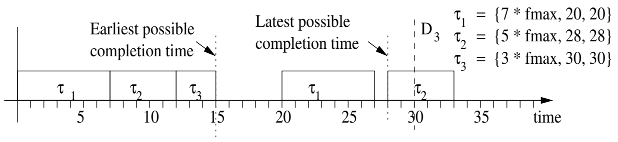


Figure 3. Workload vs. completion times

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.

$$\beta_3^{20} = \frac{(C_1 + C_2 + C_3)}{f_{max}} = 15, \quad \alpha_3^{20} = \frac{15}{20} = 0.75$$
$$\beta_3^{28} = \frac{(2C_1 + C_2 + C_3)}{f_{max}} = 22, \quad \alpha_3^{28} = \frac{22}{28} = 0.79$$

Hence, $\epsilon_3 = min\{\alpha_3^{20}, \alpha_3^{28}\} = min\{0.75, 0.79\} = 0.75$.

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. For schedulability, Sys-Clock will set the system clock frequency to the maximum frequency among energy-minimizing clock frequencies of *all* tasks. 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 4 summarizes the Sys-Clock algorithm.

During Admission Control:
  For each task τ_i:
    ϵ_i = **Energy-Min-Freq**(C_i, T_i, D_i)
  End for
  Sys-Clock-f = lowest f such that $f/f_{max} \geq max_{\forall i}\{\epsilon_i\}$
Energy-Min-Freq(C_i, T_i, D_i)**:**
  // S = slack, I = idle duration, t is the end of an idle period
  // β = workload for time t, IN_BZP is the busy period flag
  // $hp(\tau_i)$ are tasks with priorities $\geq \tau_i$'s priority
  $S = I = \beta = \Delta = 0, \alpha = 1$, 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 4. Sys-Clock algorithm

4.1. Sys-Clock Optimality

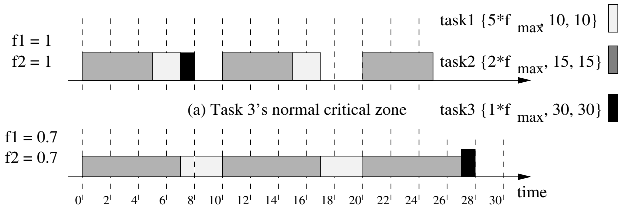
From an energy perspective, Sys-Clock is optimal for fixed-priority preemptive scheduling policies that use a single clock frequency. The only way for other schemes to consume less energy than Sys-Clock is assigning a lower system clock frequency to a taskset. However, lowering the clock frequency than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

5. Task Clock Frequency Assignment

Task clock frequency assignment determines one clock frequency for each task under the assumption that the voltage-scaling overhead at context switch is acceptable. Two schemes, PM-Clock and Opt-Clock, will be presented.

5.1. PM-Clock Algorithm

PM-Clock is 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. In addition, it is also possible that the desired frequency is neither available nor energy-efficient, and hence 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 slack from these higher frequency assignments for tasks such as τ_H.



(b) Task 3's inflated critical zone

Figure 5. An example of PM-Clock

Consider a set of three tasks as shown in Figure 5. Following the energy-minimizing clock frequency calculation described in Section 4, we obtain $\epsilon_1 = 0.5, \epsilon_2 = 0.7$ and $\epsilon_3 = 0.67$. 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* with respect 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 at their inflated frequency. Figure 5 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 $1/2.857 = 0.35$.

Note: The term "priority-monotonic" comes from the fact that PM-Clock always assigns task clock frequencies in priority order. In other words, a high priority task will always be assigned a higher or same clock frequency compared to a low priority task.

// Assume a taskset has n tasks and $D_1 \leq D_2 \ldots \leq D_n$
// v_i is the task clock frequency assigned by PM-Clock
During Admission Control:
  For each task τ_i:
    $v_i = 0$, ϵ_i =**Energy-Min-Freq**(C_i, T_i, D_i)
  End for
  For $i = 1$ to n:
    // lp(τ_i) are tasks with priorities $\leq \tau_i$'s priority
    v_i = lowest 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**
      $v_i = 0$
      For $j = i$ to n: ϵ_j = **Inflated-f**(C_j, T_j, D_j) **End for**
    End if
    v_i = lowest f such that $(f/f_{max}) \geq max_{\forall j \in lp(\tau_i)} \epsilon_j$
  End for
Inflated-f(C_i, T_i, D_i):
  // i_β and β are inflated and scalable workload for time t
  // δ = scalable time, IN_BZP is the busy period flag
  // $hp(\tau_i)$ are tasks with priorities $\geq \tau_i$'s priority
  $S = I = \beta = \Delta = \delta = 0, \alpha = 1$, 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 ($\omega < D_i$) && ($\Delta > 0$)
        $i\_\beta = 0, \beta = 0$
        For $j = 1$ to n:
          If ($D_j \leq D_i$) && ($v_j != 0$)) **then**
            $i\_\beta = i\_\beta + \frac{C_j}{v_j * f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)$
          Else 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
        $\omega' = i\_\beta + \beta + S, \Delta = \omega' - \omega, \omega = \omega'$
      End while
      IN_BZP = FALSE
    Else
      $I = min_{j \in hp(\tau_i)} \{(T_j * \lceil \frac{\omega}{T_j} \rceil) - \omega, D_i - \omega\}$
      $S = S + I, \omega = \omega + I, t = \omega, \delta = t - i\_\beta$
      If ($\frac{\beta}{\delta} < \alpha$) **then** $\alpha = \frac{\beta}{\delta}$ **End if**
      IN_BZP = TRUE
    End if
  End while
  return α

Figure 6. PM-Clock algorithm

As previously noted, PM-Clock is not optimal from an energy-saving perspective. Essentially, PM-Clock minimizes energy consumption in the critical zone, but does not minimize energy in the hyperperiod of the taskset. The Opt-Clock scheme to be presented next uses non-linear optimization techniques to minimize energy consumption over a hyperperiod (which will be repeated). Consider the same taskset in Figure 5. Using the Opt-Clock scheme, we have $f_1 = 0.6783, f_2 = 0.7609, f_3 = 0.3805$, which consumes $9.3617k$ energy units. Instead, the PM-Clock scheme assigns $f_1 = 0.7, f_2 = 0.7, f_3 = 0.35$, which consumes $9.4325k$ energy units.

5.2. Opt-Clock Algorithm

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

Theorem 1 *Consider a voltage-scaling processor with a convex non-decreasing power-frequency function given by* $P(f) = cf^x, x \geq 3$ *where c is a constant value*[4]. *Assume that a fixed-priority preemptive scheduling policy is used. The optimal task clock frequency assignment algorithm is a convex non-linear minimization problem.*

Proof.
The goal of Opt-Clock scheme is to minimize the energy consumption of a taskset in one hyper-period[5]. Each instance of a task τ_i takes $C_i/(f_i * f_{max})$ time unit where f_i is its assigned clock frequency. The energy consumed over one hyper-period, E, is as follows.

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

where (H/T_i) determines the number of τ_i's instances in a hyper-period, $\mathcal{U}_i$ is its utilization given by C_i/T_i and δ_i is the reciprocal of f_i. From [5], 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 $[a, b]$. Combine this theorem with a property of convex functions that the combination and the scaling of convex functions are also convex. It can be shown that this problem has a convex objective function.

As described in Section 4, a task's workload 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 3. The constraint of task τ_2 can be either one of the following conditions.

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

The first and second equations are necessary conditions to complete τ_2 before time 20 and 28, respectively. These constraints are linear and can be considered as convex. Hence, the optimal task clock frequency assignment is a convex non-linear minimization problem since its non-linear objective function and constraint set are convex [19]. □

Theorem 1 shows that 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. We now present three pruning techniques which can potentially decrease the problem size:

1. **Pruning of High Workload 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 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 previous step, this technique eliminates infeasible combinations of constraints where a lower priority task requires a shorter completion time than a higher priority task, which is 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[6] from the current solution. Apply the Karush-Kuhn-Tucker (KKT) theorem that the optimal solution subject to a 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

[4]This equation is derived from equations introduced in Section 1.

[5]Consider n periodic tasks with $\tau_i = \{C_i, T_i\}, i \in [1, n]$. The hyperperiod, denoted as H, is given by $LCM\{T_i | i = 1, \ldots, n\}$.

[6]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 $g(\delta_x) = D$.

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 ($E_{PM-Clock}/E_{Opt-Clock} \approx 101\%$) at significantly lower complexity.

6. DPM-Clock Algorithm

DPM-Clock is 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. It detects the early completion times and makes use of the additional slack time created at run-time by reducing the processor speed. Even though the available slack from the early completion of a task instance can be shared among multiple tasks, DPM-Clock instead devotes all slack to 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_j and υ'_j as τ_j's PM-Clock and dynamic clock frequency. S_i denotes the slack from τ_i's early completion. DPM-Clock expands the execution time of left cycles to fill all the slack. Therefore, $\upsilon'_j = \{(LEC_j/\upsilon'_j)/((LEC_j/\upsilon'_j) + S_i)\} * \upsilon'_j$ and $S_i = (WCEC_i - CEC_i)/\upsilon_i$ where LEC, WEC and CEC denote the unexecuted, the worst-case desired and the actual desired processor cycles of the current task instance.

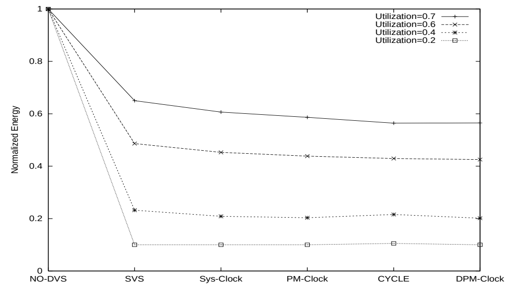
In the case that there is no ready task in the queue after the early completion of an invocation, DPM-Clock decrements the available slack time when the processor remains idle. The next arriving task whose priority is lower than or equal to the early completed task would receive the updated slack time if the slack is still available (ie. greater than 0).

It is easy to show that this method maintains schedulability. Since there is no priority change, the frequency reduction of τ_j will delay only tasks with same or lower priority. Hence, the preemption time seen by tasks is the same.

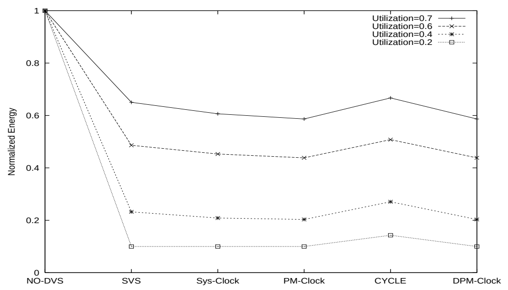
7. Experimental Results

We compare the energy consumption of our voltage-scaling algorithms with that of well-known algorithms called the static voltage scaling algorithm (SVS) and the cycle-conserving RT-DVS (CYCLE) proposed in [12]. 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)**: Each task instance requests a random number of processor cycles which is uniformly distributed between BCET and WCET[7].



(a) BCET/WCET = 0.5



(b) BCET/WCET = 1.0

Figure 7. Energy vs. system utilization

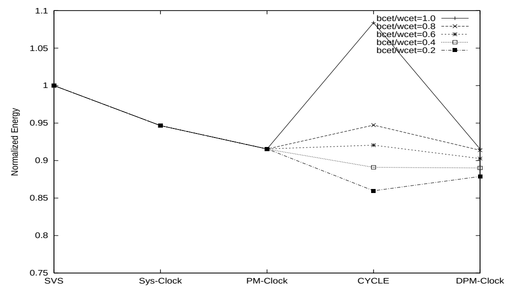


Figure 8. Energy vs. BCET/WCET at $U = 0.5$

In the first experiment, we determine the effect of varying system utilization. Figure 7 shows the average energy consumption normalized to energy of no-DVS system.

In the second experiment, we determine the effect of BCET/WCET ratio at system utilization 0.5 as shown in Figure 8 where the energy shown is normalized to energy of SVS algorithm. As can be seen from both experiments,

[7] These execution times are subjected to f_{max}

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 each context switch. Figure 8 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.

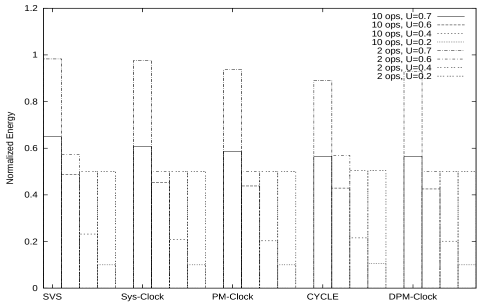


Figure 9. Energy vs. frequencies at $U = 0.5$

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 9, all voltage-scaling algorithms perform worse on the processor with a 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.

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 our 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[3], 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. This kernel with its support for frequency-scaling can be downloaded from `http://www.cs.cmu.edu/~rtml`.

As is 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 [1] to support voltage scaling. The Maxim 1855 evaluation kit [10], 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 [17] 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.

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. Energy consumption can be reduced

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

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

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. The worst-case quantitative impact of finite number of operating frequencies on a processor is also studied. We also derive the operating frequency grid that minimizes the worst-case *energy-quantization error* which has linear-inversely proportional relation to the number of operating frequencies.

We proposed four alternative voltage-scaling algorithms, Sys-Clock, PM-Clock, Opt-Clock and DPM-Clock. 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. 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 iPAQ. We also have implemented Sys-Clock, PM-Clock and DPM-Clock on our 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] ADI Engineering Inc. BRH reference platform. http://www.adiengineering.com/productsBRH.html.

[2] H. Aydin, R. Melhem, D. Mosse, and P. 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.

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

[4] D. de Niz, L. Abeni, S. Saewong, and R. R. Rajkumar. Resource sharing in reservation-based systems. In *Proceedings of the IEEE Real Time Systems Symposium*, London, UK, December 2001.

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

[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] W. Kim, J. Kim, and S. L. 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.

[8] 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.

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

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

[11] 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.

[12] P. Pillai and K. 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.

[13] B. M. G. 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, June 2001.

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

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Foundations of Computer Science*, October 1995.