Procrastination Scheduling in Fixed Priority Real-Time Systems

Ravindra Jejurikar Rajesh Gupta

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
1 (949) 824-8168

E-mail: jezz@cecs.uci.edu, gupta@cs.ucsd.edu

CECS Technical Report #04-09
April, 2004

Abstract

Procrastination scheduling has gained importance for energy efficiency due to the rapid increase in the leakage power consumption. Under procrastination scheduling, task executions are delayed to extend processor shutdown intervals, thereby reducing the idle energy consumption. We propose algorithms to compute the maximum procrastination intervals for tasks scheduled by either the fixed priority or the dual priority scheduling policy. We show that dual priority scheduling always guarantees longer shutdown intervals than fixed priority scheduling. We further combine procrastination scheduling with dynamic voltage scaling to minimize the total static and dynamic energy consumption of the system. Our simulation experiments show that the proposed algorithms can extend the sleep intervals up to 5 *times while meeting the timing requirements. The results show up to* 18% *energy gains over dynamic voltage scaling.*

Contents

List of Figures

List of Tables

1 Introduction

Embedded systems are pervasive in the consumer electronics, telecommunications, entertainment, industrial control and medical sectors. These systems are usually portable with limited battery life and power management is a crucial component in the operation of these systems. A processor is central to an embedded system and consumes a significant portion of the total energy. Total energy consumption consists of dynamic and static parts. The dynamic power consumption is due to the circuit switching activity, whereas static power consumption is present even when no logic operations are being performed. There are two primary ways to reduce power consumption of the processor: processor *shutdown* and processor *slowdown*. Slowdown based on Dynamic Voltage Scaling (DVS) has been shown to significantly reduce the dynamic energy consumption at the cost of increased execution time. Note that the longer execution time, while decreasing the dynamic power consumption, increases the static energy consumption. The primary components of static power consumption are the standby currents including the device leakage currents. With device scaling to sub-100 nm process technologies, leakage currents are increasingly a dominant component of the standby power consumption [23]. This implies that a straightforward slowdown while meeting timing requirements is no longer sufficient for reducing overall energy consumption. Instead, a balance between the amount of processor slowdown and processor shutdown is needed to minimize the overall energy consumption for a given set of tasks.

Most of the earlier works on energy aware scheduling have addressed the problem of minimizing the dynamic power consumption. Among the earliest works, Yao *et. al.* [36] presented an off-line algorithm to schedule a given set of jobs with arrival times and deadlines. The solution is optimal, based on the assumption of a continuous voltage range and the Earliest Deadline First (EDF) scheduling policy. Generalization of the problem, considering discrete voltage levels and fixed priority scheduling have been addressed in [17] [30] [37]. Low power scheduling for periodic real-time systems has also been widely studied. The extent of slowdown that the system can sustain is computed based on known feasibility results for periodic task-sets. Computation of slowdown factors for an independent task-set, based on dynamic (EDF) and fixed (rate monotonic) priority scheduling, is addressed in [29, 3, 34, 8]. Earlier work, including our own, have addressed extension of slowdown algorithms to handle task synchronization [13, 38] and aperiodic tasks through periodic servers [24]. Dynamic slowdown techniques in [28, 3, 16] show additional savings in energy by reclaiming run-time slack arising from variation in the task execution times. Such combined static and dynamic slowdown approaches have shown to result in significant energy savings. The problem of maximizing the system value for a given energy budget, as opposed to minimizing the total energy, is addressed in [33, 31, 32, 2].

Recently, leakage abatement has been an important focus of the work on power and energy minimization. Leakage is an increasing concern with a predicted five-fold increase in the leakage power with each technology generation [4]. Techniques such as input vector control [15] and power supply gating [26] have been proposed to minimize leakage. The exponential dependence of sub-threshold leakage current on the threshold voltage has led to Multi Threshold CMOS (MTCMOS) circuit techniques [5]. Scaling the threshold voltage by controlling the body bias voltage has also been proposed to minimize leakage [27, 23]. Scheduling techniques based on adaptive body biasing have shown to reduce the total static and dynamic power consumption [23, 18].

While many works have addressed leakage minimization, these are based on the premise that the energy savings are proportional to the extent of slowdown. This need not be true considering the increase in leakage current and the power consumption of other components such as memory and I/O [7]. This

motivates a combined slowdown and shutdown approach for energy minimization. Among the earliest works, Irani *et. al.* [12] consider the combined problem of DVS and shutdown to schedule a given set of tasks with deadlines. The authors propose a *3-competitive* off-line algorithm based on the assumption of a continuous voltage range and a convex power consumption function. Lee *et. al.* [20] address *procrastination* scheduling in periodic real-time systems and have proposed Leakage Control EDF (LC-EDF) and Leakage Control Dual Priority (LC-DP) scheduling algorithms. Procrastination by a component refers to its choice to enter or remain in a shutdown mode even when there are pending tasks.

Integration of procrastination and dynamic voltage scheduling is a promising way to reduce overall energy consumption. We have earlier addressed use of procrastination in EDF scheduling [14]. The results show up to 18% energy savings. In this paper, we address scheduling in fixed and dual priority task systems. We show that procrastination under LC-DP algorithm by Lee *et. al.* [20] can lead to deadline misses. We present a remedy to this problem with a procrastination algorithm that guarantees all task deadlines.

Our contributions are as follows: (1) we compute task procrastination intervals under the fixed priority scheduling policy as well as the dual priority scheduling policy; (2) we show that the procrastination intervals under dual priority scheduling can be greater than that of fixed priority scheduling which can further reduce the idle energy consumption of the system; (3) based on the leakage energy characteristics of the 70*nm* technology, we combine dynamic voltage scaling with procrastination to minimize the total energy consumption.

The rest of the paper is organized as follows: Section 2 formulates the problem with motivating examples. In Section 3, we present the procrastination algorithm for fixed priority and dual priority scheduling policies. Section 4 explains the integration of procrastination and dynamic voltage scaling. The leakage power model is discussed in Section 5 and the experimental results are given in Section 6. Finally, Section 7 concludes the paper with future directions.

2 Preliminaries

In this section, we introduce the necessary notation and formulate the problem. An example follows to illustrate how scheduling by LC-DP algorithm can result in tasks missing the deadline.

2.1 System Model

A task set of n periodic real time tasks is represented as $\Gamma = \{\tau_1, \ldots, \tau_n\}$. A task τ_i is a 3-tuple $\{T_i, D_i, C_i\}$, where T_i is the period of the task, D_i is the relative deadline and C_i is the worst case execution time (WCET) of the task at the maximum processor speed. The tasks are scheduled on a single processor system based on a preemptive scheduling policy. A task set is said to be *feasible* if all tasks meet the deadline. The processor utilization for the task set, $U = \sum_{i=1}^{n} C_i/T_i \leq 1$, is a necessary condition for the feasibility of any schedule [21]. In this work, we assume task deadlines are equal to the period ($D_i = T_i$) and tasks are scheduled by a fixed or dual priority scheduling policy [21]. All tasks are assumed to be independent and preemptive.

A wide range of processors like the Intel XScale [11], PowerPC 405LP [9] and Transmeta Crusoe [35] support variable voltage and frequency levels. Dynamic voltage scaling (DVS) based on slowdown factors has shown to result in significant energy savings. A *slowdown factor* (η_i) is defined as the normalized operating frequency, i.e., the ratio of the current frequency to the maximum frequency of

the processor. Since processors support discrete frequency levels, the slowdown factors are discrete points in the range [0,1]. Tasks are assigned slowdown factors based on the functional and performance requirements of the system to minimize the total energy consumption.

Procrastination scheduling has been shown to increase the processor sleep intervals, by delaying task executions when the processor is shutdown (sleep state). We say a task is *procrastinated* (or delayed) if on task arrival, the processor is in a shutdown state and continues to remain in the shutdown state, despite the task being ready for execution. The procrastination interval of a task is the time interval by which a task is procrastinated. Note that the processor is shutdown, only when the processor ready queue is empty.

2.2 Dual Priority Scheduling

We briefly describe the dual priority scheduling policy, since the task promotion times used in dual priority scheduling are used in computing the task procrastination intervals. Dual priority scheduling was proposed in [6] to improve the response time of aperiodic tasks while meeting the deadlines of all periodic tasks. Dual-priority scheduling uses three distinct priority queues in decreasing order of priority : upper, middle and lower. Each periodic task has two priorities, one when it is in the lower priority queue and the other in the upper priority queue. The middle priority queue is used by the aperiodic tasks that arrive in the system. Each task τ_i on arrival is added to the lower priority queue. After a fixed time called the *promotion time*, Y_i, the task is promoted to its upper priority queue. Tasks can be preempted by other higher priority tasks in the same priority queue. The promotion time of each task is computed based on the response time analysis of fixed priority scheduling. If R_i is the worst case response time of a task and D_i its deadline, then the promotion time, Y_i, of a task satisfies the following condition:

$$Y_i \leq D_i - R_i. \tag{1}$$

The detailed scheduling algorithm and the computation of the task promotion times are given in [6]. Our algorithms use task promotion times in the computation of maximum task procrastination intervals.

2.3 Procrastination under LC-DP Scheduling

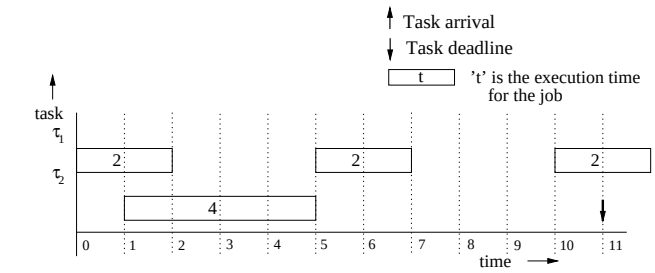
Lee *et. al.* have proposed the LC-DP algorithm to extend the idle intervals under fixed priority scheduling [20]. The procrastination intervals are based on task promotion times computed under the dual priority scheduling [6]. The scheduling rules of LC-DP as proposed by Lee *et. al.* are as follows:

1. If the processor is busy and a new task arrives, the task is directly added to the upper priority queue.

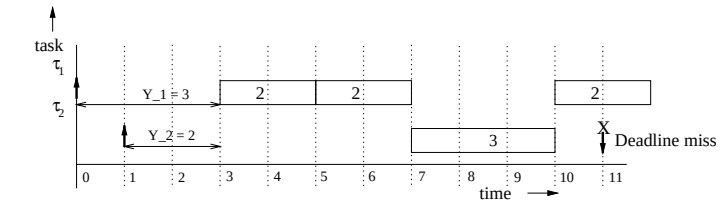
2. Whenever a task is promoted to the upper priority queue, all tasks in the lower priority queue are promoted to the upper priority queue.

3. When the processor is in the sleep state, tasks are added to the lower priority queue.

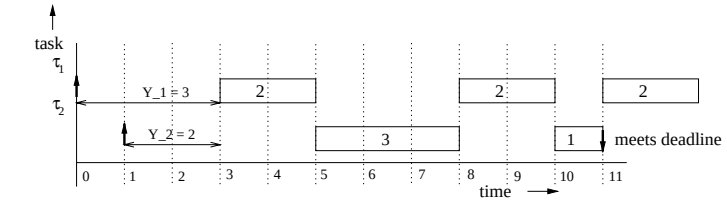
4. The procrastination interval is the minimum of the promotion times of all tasks in the lower priority queue.



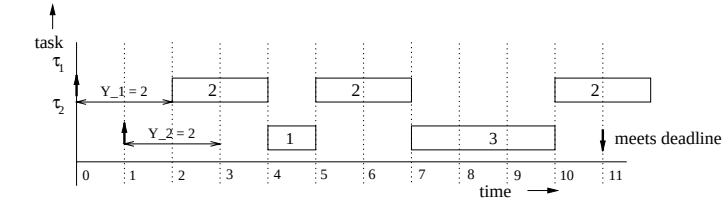
(a) Task set description: Task arrival times and WCET at maximum speeed



(b) Procrastination by the LC-DP algorithm and a deadline miss.



(c) Procrastination intervals under dual priority scheduling



(d) Procratination intervals under fixed priority scheduling

Figure 1. (a) Task set description with task arrival times, execution times and task deadlines. (b) Task schedule by the LC-DP algorithm and task τ_2 misses its deadlines (c) Procrastination intervals under dual priority scheduling that results in a feasible schedule (d) Procrastination intervals under fixed priority scheduling that results in a feasible schedule.

Note that Rules 1 and 2 are in contrast to the dual priority scheduling scheme where every task remains in the lower priority queue until its promotion time. We show that the above rules do not guarantee all task deadlines.

Consider a task set of two tasks with the following parameters where the tasks are executed at the maximum speed.

$$\tau_1 = \{2, 5, 5\} \textit{ and } \tau_2 = \{4, 10, 10\}$$

The promotion times for the tasks as computed by the dual priority algorithm are $Y_1 = 3$ and $Y_2 = 2$. We assume the processor is idle prior to time $t = 0$ and task τ_1 and τ_2 arrive at time $a_1 = 0$ and $a_2 = 1$ respectively, as shown in Figure 1(a).

The task schedule according to the LC-DP algorithm is shown in Figure 1(b). Based on the task arrival times, the promotion time of both tasks is $t = 3$. The LC-DP algorithm delays the task executions up to time $t = 3$. At $t = 3$, both tasks are promoted to the upper priority queue and task τ_1 executes up to time

$t = 5$, when the next instance arrives. Since tasks are immediately added to the upper priority queue when the processor is busy, the next instance of τ_1 has the highest priority and executes up to time $t = 7$. Task τ_2 begins execution at time $t = 7$ and at time $t = 10$ the third instance of τ_1 arrives and preempts task τ_2. Task τ_1 executes up to time $t = 12$ and task τ_2 which is not yet complete misses its deadline of $t = 11$. Thus we see that the LC-DP algorithm can result in an infeasible schedule.

Adding newly arrived tasks to the upper priority queue when the processor is busy, as opposed to adding the tasks to the lower priority queue, can increase the processor demand during an interval resulting in a deadline miss. We show later in this paper, that the task executions can be delayed by the task promotion times only under the dual-priority scheduling policy. The dual priority schedule of the task set is seen in Figure 1(c). At time $t = 3$, both tasks are promoted to the upper priority queue. Though instances of task τ_1 arrive at time $t = 5$ and $t = 10$, they arrive in the lower priority queue and are promoted to the upper priority queue only after residing in the lower priority queue up to their promotion time interval of 3 time units. Thus task τ_2 can execute for 4 time units before $t = 11$ to meet the deadline. The schedule is shown in Figure 1(c).

We also consider task procrastination under the fixed priority scheduling policy. In Section 3, we prove that delaying a task execution by the minimum promotion time (Y_i) over all lower and equal priority tasks ensures all deadlines. Thus the execution of task τ_1 can be delayed by only 2 time units and the maximum procrastination intervals for the tasks are $Z_1 = 2$ and $Z_2 = 2$. A feasible schedule with task executions delayed up to time $t = 2$ is shown in Figure 1(d).

3 Procrastination Scheduling

In this section, we propose algorithms to compute maximum task procrastination intervals that guarantee feasibility of the task-set. The basis of the algorithm are the two main results presented next.

3.1 Dual Priority Scheduling

Dual priority scheduling was proposed to improve the response time of aperiodic tasks. As discussed in Section 2.2, a promotion time Y_i is associated with each task τ_i when it is promoted from the lower to upper priority queue. Delaying task executions by their promotion time ensures all task deadlines under the dual priority scheduling policy.

Theorem 1 *Given tasks are scheduled by the dual-priority scheduling policy, all task deadlines are guaranteed if the maximum procrastination interval, Z_i, of each task τ_i satisfies:*

$$Z_i \leq Y_i \tag{2}$$

where Y_i represents the promotion time of task τ_i.

Proof: Note that, task procrastination is analogous to executing aperiodic tasks in the system. Periodic tasks are delayed in both cases by either explicit procrastination or by servicing aperiodic tasks. A processor shutdown under procrastination scheduling can be considered as generating an aperiodic task with large execution time. Since the processor is shutdown when idle and task promotion times are used for procrastination, the duration of the sleep interval under procrastination scheduling is equal to the duration for which the aperiodic task would be serviced. If the aperiodic task is considered as deleted

when the processor wakes up under procrastination scheduling, then the schedule of the periodic tasks is identical under both procrastination scheduling and scheduling with aperiodic tasks. Thus the validity of the procrastination scheduling follows directly from the correctness of the dual-priority scheduling algorithm. ■

3.2 Fixed Priority Scheduling

We have shown that the task promotion times cannot be used as the maximum procrastination interval under fixed priority scheduling. On the other hand, task feasibility is guaranteed if the maximum procrastination interval of each task is bounded by the promotion times of all equal and lower priority tasks.

Theorem 2 *Let tasks be ordered in non-increasing order of their priority. Given tasks are scheduled under the fixed priority scheduling policy, all task deadlines are guaranteed if the maximum procrastination interval, Z_i, of each task τ_i satisfies:*

$$\forall_{j \geq i} \;\; Z_i \leq Y_j \tag{3}$$

where Y_i represents the promotion time of task τ_i based on dual priority scheduling.

Proof: We prove the claim by contradiction. Suppose the claim is false and let t be the earliest time when a task, say τ_i, misses its deadline. Let t' be the the latest time before t such that there are no pending jobs with arrival times before t' and a priority greater than or equal to task τ_i. Since no requests can arrive before system start time $(time = 0)$, t' is well defined. The only jobs that execute in the interval $[t', t]$ are the jobs released in that interval with a priority greater than or equal to that of task τ_i. Let $A \subseteq \{\tau_1, ...\tau_i\}$ be the set of jobs that execute in $[t', t]$, then the workload of the jobs in A is bounded by the response time, R_i, of task τ_i. However, the processor demand in an interval can increase due to procrastination scheduling. Tasks can be procrastinated if the processor is in a shutdown state at time instance t'. We show that the total procrastination interval is bounded by Y_i. By Theorem 2, the maximum procrastination interval of each task in A is bounded by Y_i. Since a task in A arrives at time t', it is true that tasks are not procrastinated beyond $t' + Y_i$. Once the processor resumes execution, there is no further procrastination up to time t as there are pending tasks at all time within the interval $[t', t]$. Since a task misses its deadline, it must be true that $R_i + Y_i > X$, where $X = t - t'$ is the length of the interval $[t', t]$. The release time and deadline of task τ_i lies in the interval $[t', t]$ and it is true that $X \geq D_i$. Thus it follows that

$$R_i + Y_i > D_i$$

which contradicts the definition of task promotion interval, given by Equation 1. Hence all tasks meet the deadline under procrastination scheduling. ■

3.3 Procrastination Algorithm

The procrastination algorithm is designed to ensure that no task τ_i is delayed by more than its maximum procrastination interval Z_i. Note that the procrastination algorithm is independent of the scheduling policy, however the computation of the maximum procrastination intervals is governed by the scheduling policy implemented in the system. The algorithm, describing how procrastination is handled in the

system, has been proposed in our earlier work [14]. Our earlier work has addressed procrastination in dynamic priority systems [14] and we consider fixed and dual priority scheduling in this work.

Task executions are procrastinated when the processor is in the sleep state and it is necessary that the *power manager* handling task procrastination be implemented as a separate controller. When the processor enters sleep state, it hands over the control to the power manager (controller), which handles all the interrupts and task arrivals while the processor is in sleep state. The controller has a timer to keep track of time and it wakes up the processor after a specified time period. When the processor is in sleep state and the first task τ_i arrives, the timer is set to Z_i. The timer counts down every clock cycle. If another task, τ_j arrives before the counter expires, the timer is updated to the minimum of the current timer value and Z_j. This ensures that no task in the system is procrastinated by more than its maximum procrastination interval. When the counter counts down to zero (expires), the processor is woken up and the scheduler dispatches the highest priority task in the system for execution. All tasks are scheduled at their assigned priority levels.

When no pending tasks are present in the processor ready queue, the processor can be shutdown. Note that a shutdown has its associated overhead and shutdown decisions need to be made wisely to result in energy savings. In making shutdown decisions with procrastination scheduling, it is important to know the minimum idle period guaranteed by the procrastination algorithm. The following results give the length of guaranteed idle period.

Corollary 1 *Given tasks are scheduled by the dual-priority scheduling policy, the minimum idle period, Z_{min}, guaranteed by the procrastination algorithm is given by*

$$Z_{min} = min_{1 \leq i \leq n} \quad Y_i \tag{4}$$

where Y_i represents the promotion time of task τ_i.

Corollary 2 *Given tasks are scheduled by the fixed priority scheduling policy, the minimum idle period, Z_{min}, guaranteed by the procrastination algorithm is given by*

$$Z_{min} = min_{1 \leq i \leq n} \quad Y_i \tag{5}$$

where Y_i represents the promotion time of task τ_i based on dual priority scheduling.

The claim follows immediately from the procrastination algorithm and the results given by Theorem 1 and Theorem 2. Though the minimum idle period under both scheduling policies is the same, the task procrastination intervals under dual priority scheduling are always greater than or equal to that by fixed priority scheduling. The following result proves the same.

Theorem 3 *Let Z_i^{FP} and Z_i^{DP} represent the maximum procrastination interval of task τ_i under fixed priority scheduling and dual priority scheduling respectively, then*

$$Z_i^{DP} \geq Z_i^{FP} \tag{6}$$

Proof: Our computation of task procrastination interval is based on the task promotion time, Y_i, under dual-priority scheduling. The procrastination interval of a task τ_i under dual priority scheduling, Z_i^{DP}, is bounded by Y_i (Theorem 1). The task procrastination interval under fixed priority scheduling, Z_i^{FP}, is also bounded by Y_i. In addition, Z_i^{FP} is also constraint to be no greater than the promotion times of all lower priority tasks (Theorem 2). These additional constraints can result in lowering Z_i^{FP} more than Y_i and hence it is true that Z_i^{DP} is greater than or equal to Z_i^{FP}. ■

4 Integrating Procrastination and Slowdown

As mentioned earlier, slowdown caused by DVS can increase the component of energy consumption due to static power. For a given technology choice, indeed, there is an optimum speed at which the processor should be clocked to reduce the overall energy consumption. This is indicated by the *critical speed* of the processor and is denoted by η_{crit}.

4.1 Critical Speed

Taking into account the leakage power and the power consumption of components such as memory and I/O that are not subject to DVS, the minimum voltage level at which the processor can run to meet the timing constraints need not correspond to a lower energy point. Fan *et. al.* [7] consider memory power consumption to show that slowdown beyond a point does not result in lowering the total energy. Miyoshi *et. al.* [25] show that the slowdown decision can differ with different processor families in minimizing the total energy. With the increasing leakage contribution in present and future CMOS technologies, it is important to compute the optimal point beyond which slowdown does not reduce the energy consumption. This optimal operating point at which the energy consumption is minimized is referred to as the *critical speed*.

4.2 Slowdown Factor Computation

Note that the task slowdown can be computed with any known dynamic voltage scaling algorithm. Since executing below the critical speed consumes more time and energy, the minimum value for the slowdown factor is set to the critical speed (η_{crit}). We update a task slowdown factor to the critical speed if it is smaller than η_{crit}. The computed slowdown factors are updated by the following procedure:

$$\overset{\forall i}{i = 1, ..., n} \quad if(\eta_i < \eta_{crit})\ \eta_i \leftarrow \eta_{crit} \tag{7}$$

Since we only increase slowdown factors of a feasible task set, the feasibility of the task set is maintained.

4.3 Combining Slowdown and Procrastination

Since we update the task slowdown factors based on the critical speed, the system can have inherent idle time while operating at the energy optimal point. If the computed slowdown factors do not utilize 100% of the processor, we can compute procrastination intervals which will further reduce the idle energy consumption. We use the results in Section 3 to compute maximum task procrastination intervals. Even though the results in Section 3 do not consider task slowdown, we can transform the task set to incorporated slowdown factors. Given a task set $\Gamma = \{\tau_1, ..., \tau_n\}$ with a slowdown factors η_i for task $\tau_i = \{T_i, D_i, C_i\}$, we transform the task set to $\Gamma' = \{\tau'_1, ..., \tau'_n\}$ where each transformed task is $\tau'_i = \{T_i, D_i, \frac{C_i}{\eta_i}\}$. The task promotion times and the procrastination intervals are computed based on this transformed task set. Since the executing time of each task under slowdown is bounded by C_i/η_i, the procrastination intervals for the transformed task set ensures meeting all task deadlines under slowdown.

Table 1. $70nm$ technology constants [23]

const	value	const	value	const	value
K_1	0.063	K_6	$5.26\,10^{-12}$	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	$4.8\,10^{-10}$
K_3	$5.38\,10^{-7}$	V_{dd0}	1	C_{eff}	$0.43\,10^{-9}$
K_4	1.83	V_{bs0}	0	L_d	37
K_5	4.19	α	1.5	L_g	$4\,10^6$

5 Power Model

In this section, we describe the power model used to compute the static and dynamic components of power consumption of CMOS circuits. The dynamic power consumption (P_{AC}) of CMOS circuits is given by,

$$P_{AC} = C_{eff} V_{dd}^2 f \tag{8}$$

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance. Among the different leakage sources [1], the major contributors of leakage are the sub-threshold leakage and the reverse bias junction current. We use the power model and the technology parameters described by Martin *et. al.* [23]. The sub-threshold current I_{subn}, as a function of the supply voltage V_{dd} and the body bias voltage V_{bs} is given below :

$$I_{subn} = K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} \tag{9}$$

where K_3, K_4 and K_5 are constant fitting parameters. The leakage power dissipation per device due to sub-threshold leakage (I_{subn}) and reverse bias junction current (I_j) is given by,

$$P_{DC} = V_{dd} I_{subn} + |V_{bs}| I_j \tag{10}$$

and the total leakage power consumption is $L_g \cdot P_{DC}$, where L_g is the number of devices in the circuit. The relation of threshold voltage V_{th} and V_{bs} is represented by, $V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs}$ where K_1, K_2 and V_{th1} are technology constants. The cycle time t_{inv} as a function of the V_{dd} and the threshold voltage V_{th} is given by,

$$t_{inv} = \frac{L_d K_6}{(V_{dd} - V_{th})^{\alpha}} \tag{11}$$

The technology constants for the $70nm$ technology are presented in Table 1 as given in [23]. The value for C_{eff} based on the Transmeta Crusoe processor, scaled to $70nm$ technology based on the technology scaling trends [4], is also given in the table. To reduce the leakage substantially, we use $V_{bs} = -0.7V$. The static and dynamic power consumption as the supply voltage is varied in the range of $0.5V$ and $1.0V$ is shown in Figure 2.

In addition to the gate level leakage, there is an inherent cost in keeping the processor on which must be taken into account in computing the optimal operating speed. Certain processor components consume power even when the processor is idle. Some of the major contributors are (1) the PLL circuitry, which drives up to $200mA$ current [10, 35] and (2) the I/O subsystem which is supplied a higher voltage (2.5V to 3.3V) that the processor core. This intrinsic cost (power) of keeping the system on is referred to as P_{on}.

The power consumption of these components will scale with technology and architectural improvement and we assume a conservative value of $P_{on} = 0.1W$. The total power consumption, P, of the processor is :

$$P = P_{AC} + P_{DC} + P_{on} \tag{12}$$

where P_{AC} and P_{DC} denote the dynamic and static power consumption respectively. The variation of the power consumption with supply voltage is shown in Figure 2.

5.1 Critical Speed

To evaluate the effectiveness of dynamic voltage scaling, we compute the energy consumption per cycle for different supply voltage values. Due to the decrease in the operating frequency with voltage, the leakage can adversely effect the total energy consumption with voltage scaling. We compute the energy per cycle to decide the aggressiveness of voltage scaling. The contribution of the dynamic energy E_{AC} and the leakage energy E_{DC} per cycle is given by,

$$E_{AC} = C_{eff} V_{dd}^2 \tag{13}$$

$$E_{DC} = f^{-1} \cdot L_g \cdot (I_{subn} V_{dd} + |V_{bs}| I_j) \tag{14}$$

where f^{-1} is the delay per cycle. The energy to keep the system on increases with lower frequencies and is given by, $E_{on} = f^{-1} P_{on}$. The total energy consumption per cycle, E_{cycle}, with varying supply voltage levels is given below and shown in Figure 3 .

$$E_{cycle} = E_{AC} + E_{DC} + E_{on} \tag{15}$$

We define the *critical speed* as the operating point that minimizes the energy consumption per cycle. Figure 3 shows the energy characteristics for the 70*nm* technology. From the figure, it is seen that the critical point is at $V_{dd} = 0.7V$. From the voltage frequency relation described in Equation 11, $V_{dd} = 0.7V$ corresponds to a frequency of 1.26 GHz. The maximum frequency at $V_{dd} = 1.0V$ is 3.1 GHz, resulting in a critical slowdown of $\eta_{crit} = 1.26/3.1 = 0.41$.

5.2 Shutdown Overhead

In previous works, the overhead of processor shutdown/wakeup has been neglected or considered only as the actual time and energy consumption incurred within the processor. However, a processor shutdown and wakeup has a higher overhead than the energy required to turn on the processor. For example, processors lose all register and cache contents when switched to the deepest sleep mode, leading to additional overhead. The various overheads associated with a processor shutdown and wakeup are enumerated below:

1. Prior to a shutdown, all registers must to be saved in main memory.
2. The dirty data cache lines need to be flushed to main memory before a shutdown.
3. The inherent energy and delay cost of processor wakeup, as specified in datasheets.

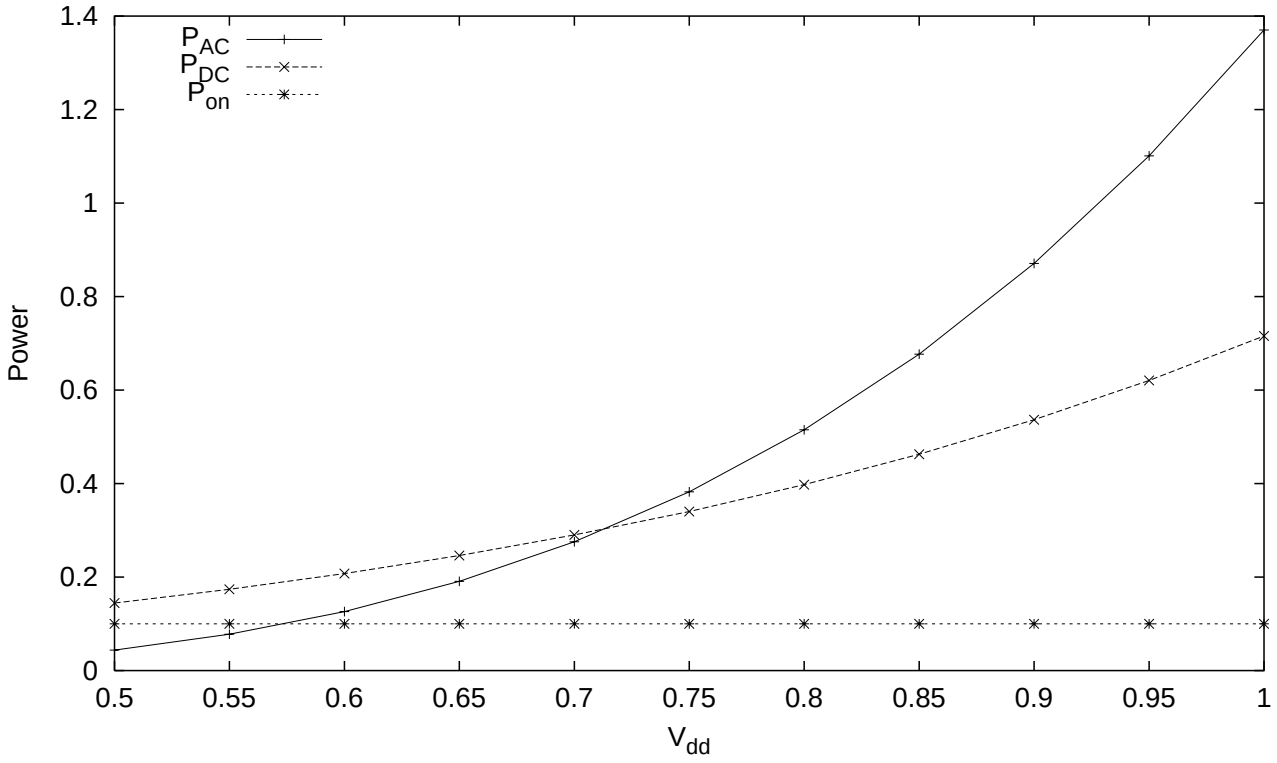


Figure 2. Power consumption of $70nm$ technology for Crusoe processor: P_{DC} is the leakage power, P_{AC} is the dynamic power and P_{on} is the intrinsic power consumption in on state.

4. On wakeup, components such as data and instruction caches, data and instruction translation look aside buffers (TLBs) and branch target buffers (BTBs) are empty and result in extra misses on a cold start (empty structures).

This results in extra memory accesses and hence energy overhead. This cost will vary, depending on the nature of the application and the processor architecture.

We perform an energy overhead estimation, similar to our earlier work [14], which is used in our simulations. With typical embedded processors having cache sizes between 32KB and 128KB, we conservatively assume a 32KB instruction and data cache. Assuming 20% lines of the data cache to be dirty before shutdown results in 6554 memory writes. With an energy cost of 13nJ [19] per memory write, the cost of flushing the data cache is computed as $85\mu J$. On wakeup, there is an additional cost due to cache miss. Note that a context switch occurs when a task resumes execution and has its own cache miss penalty. However, shutdown has its own additional cost than a regular context switch due to the fact that these structures are empty. We assume 10% additional misses rate in both the instruction and data cache. Therefore, the total overhead of bringing the processor to active mode is 6554 cache misses. A cost of 15nJ [19] per memory access, results in $98\mu J$ overhead. Adding the cache energy overhead to the energy of charging the circuit logic ($300\mu J$), the total cost is $85 + 98 + 300 = 483\mu J$.

Due to the cost of shutdown, the shutdown decision needs to be made wisely. An unforeseen shutdown can result in extra energy and/or missing task deadlines. Based on the idle power consumption, we can compute the minimum idle period, referred to as the *idle threshold* interval $t_{threshold}$, to break

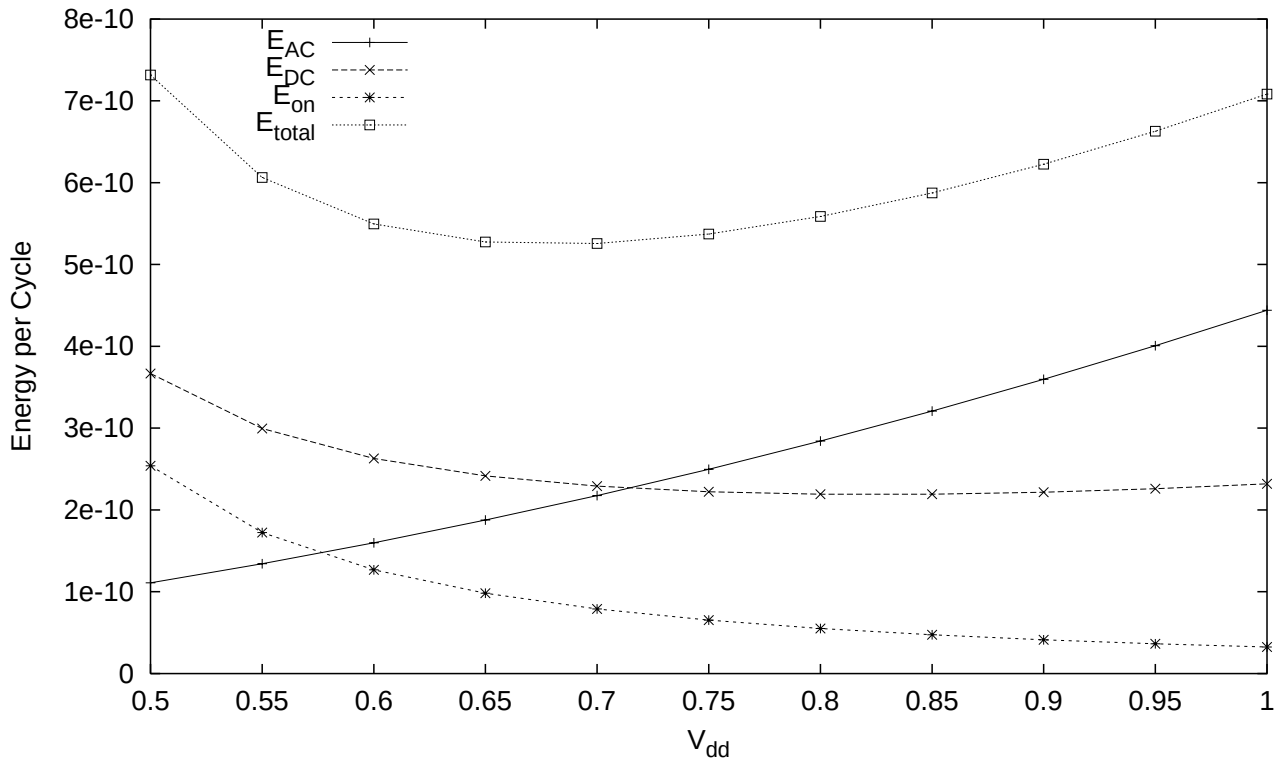


Figure 3. Energy per Cycle for $70nm$ technology for the Crusoe processor: E_{AC} is the switching energy, E_{DC} is the leakage energy and E_{on} is the intrinsic energy to keep the processor on.

even with the wakeup energy overhead. Since the idle power consumption at the lowest operating voltage/frequency is $240mW$, and the shutdown energy overhead is $483\mu J$, $t_{threshold}$ is $2.01ms$. We assume a sleep state power of $50\mu W$, which can account for the power consumption in the sleep state and that of the controller.

6 Experimental Setup

We implemented the proposed scheduling techniques in a discrete event simulator. To evaluate the effectiveness of our scheduling techniques, we consider several task sets, each containing up to 20 randomly generated tasks. We note that such randomly generated tasks is a common validation methodology in previous works [3, 20, 34]. Based on real life task sets [22], tasks were assigned a random period and WCET in the range [10 ms,125 ms] and [0.5 ms, 10 ms] respectively. All tasks are assumed to execute up to their WCET. We use the processor power model described in Section 5 to compare the energy consumption of the following techniques :

- No DVS (no-DVS): where all tasks are executed at maximum processor speed.
- Traditional Dynamic Voltage Scaling (DVS) : where tasks are assigned the minimum possible slowdown factor.

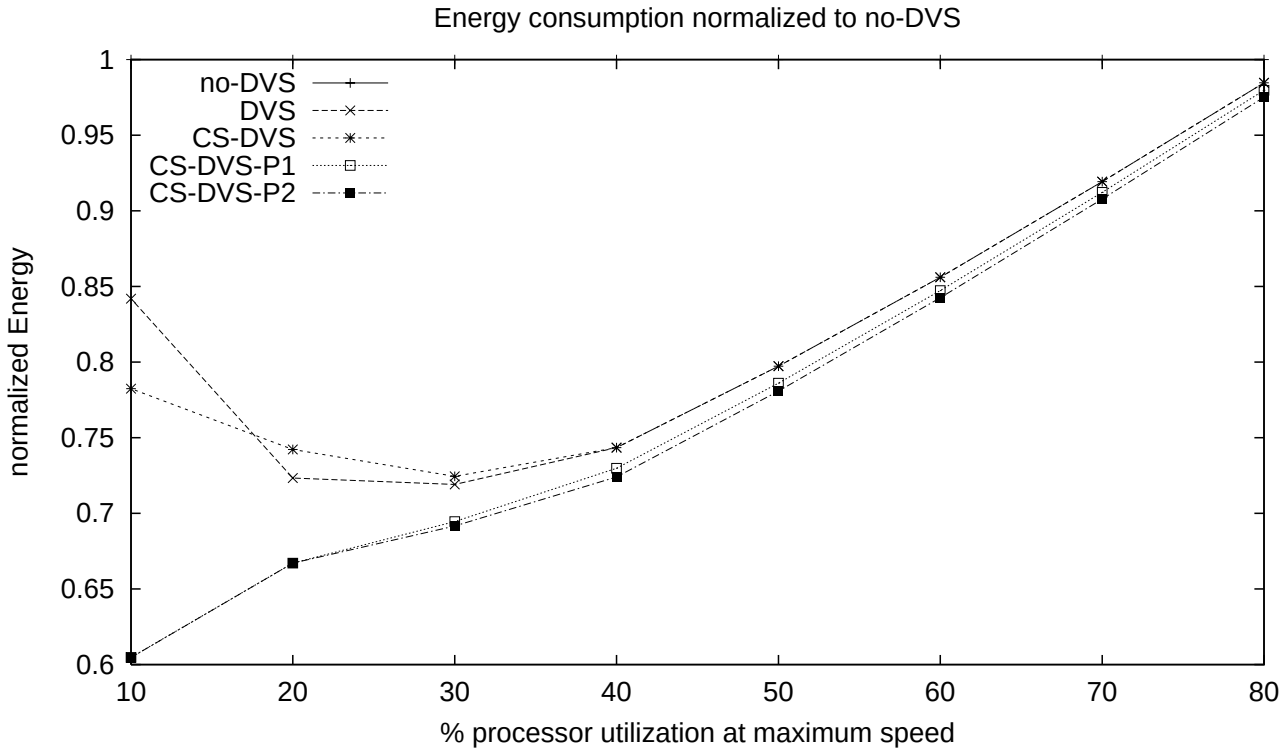


Figure 4. Energy consumption normalized to no-DVS, based on fixed and dual priority scheduling policies.

- Critical Speed DVS (CS-DVS): where all tasks are assigned a slowdown greater than or equal to the critical speed (η_{crit}).
- Procrastination with fixed priority scheduling (CS-DVS-P1): This is the Critical Speed DVS (CS-DVS) slowdown with the procrastination technique under fixed priority scheduling.
- Procrastination with dual priority scheduling (CS-DVS-P2): This is the Critical Speed DVS (CS-DVS) slowdown with the procrastination technique under dual-priority scheduling.

We assume that the tasks have a rate monotonic priority ordering and the task slowdown factors are computed by the algorithm given in [34]. We assume that the processor supports discrete voltage levels in steps of $0.05V$ in the range $0.5V$ to $1.0V$. These voltage levels correspond to discrete slowdown factors and each computed slowdown factor is mapped to the smallest discrete level greater than or equal to it. When procrastination is not implemented, the processor wakes up on the arrival of a task in the system and the idle interval is the time period before the next task arrival. The procrastination schemes add the minimum guaranteed procrastination interval to the next task arrival time to compute the minimum idle interval. The processor is shutdown if the upcoming idle period is greater than $t_{threshold}$, the minimum idle period to result in energy gains. Note that the same shutdown policy is implemented under all scheduling algorithms discussed in the paper.

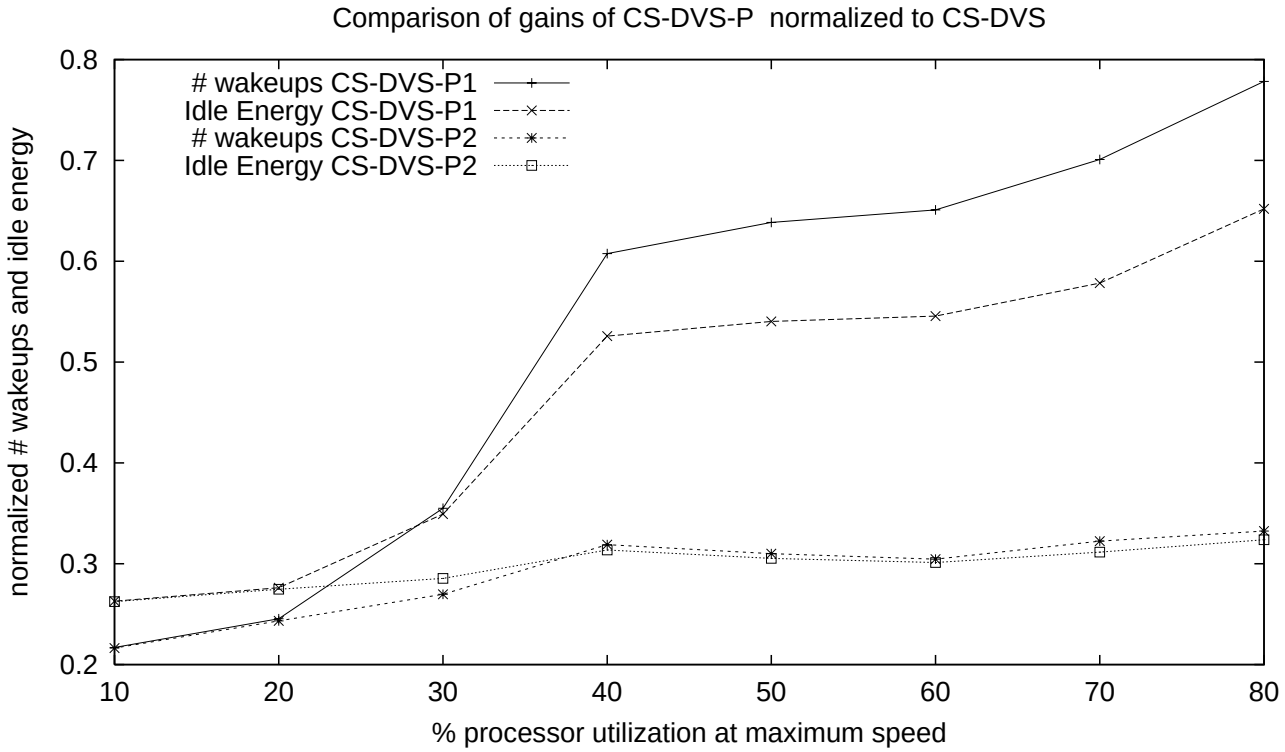


Figure 5. Comparison of # wakeups and idle energy of CS-DVS-P1 and CS-DVS-P2 normalized to CS-DVS.

Experimental Results

The energy consumption of all the techniques are shown in Figure 4 as a function of the processor utilization at maximum speed. When the processor is maximally stressed for computation, there are no opportunities for energy reduction. As the processor utilization decreases, slowdown results in energy reduction. The no-DVS scheme consumes the maximum energy and the energy consumption of the techniques is normalized to no-DVS. It is seen from the figure that all techniques perform almost identical up to the critical speed which maps to 40% utilization. When the task slowdown factors fall below the critical speed, DVS technique starts consuming more energy due to the dominance of leakage. The CS-DVS technique executes at the critical speed and shuts down the system to minimize energy. However, if the idle intervals are not sufficient to shutdown, CS-DVS can consume more energy than the DVS technique as seen at utilization of 20% and 30%. CS-DVS leads to as much as 22% energy gains over no-DVS and 6.5% gains over DVS. Procrastination schemes further minimize the idle energy by stretching idle intervals and increasing the opportunity of shutdown. Comparing the total energy consumption, the energy consumption of CS-DVS-P1 and CS-DVS-P2 are close and both schemes results up to an additional 18% gains over CS-DVS. The comparison of the relative gains of CS-DVS-P1 and CS-DVS-P2 over CS-DVS is presented next.

Figure 5 compares the number of wakeups (shutdowns) and the idle energy consumption of the procrastination schemes normalized to CS-DVS. Note that, since the slowdown factors are mapped to discrete voltage/frequency levels, there are idle intervals at higher utilization as well. These idle period can be used in dynamic reclamation [3] for more energy savings. However, we use these idle inter-

vals to shutdown the processor to compare the benefits of the procrastination schemes. Procrastination clusters the task executions and enables having longer and fewer sleep time intervals thereby decreasing the idle energy consumption. Compared to CS-DVS, procrastination always lowers the idle energy consumption with the idle energy consumption decreasing with a decrease in the processor utilization (at maximum speed). Procrastination under dual priority scheduling (CS-DVS-P2) performs better than procrastination under fixed priority scheduling (CS-DVS-P1). Since the task procrastination intervals under CS-DVS-P2 are always greater than or equal to that under CS-DVS-P1, the number of shutdowns and the idle energy consumption of CS-DVS-P2 are smaller than that under the CS-DVS-P1 scheme. The task procrastination intervals under CS-DVS-P1 and CS-DVS-P2 are identical at lower utilization (20% and 10%) which results in a similar idle energy consumption and number of shutdowns for both procrastination schemes. The idle energy consumption is reduced to as much as 25% percent by procrastination. It is seen from the figure that the idle energy consumption and the number of shutdowns follow the same trend.

Figure 6 compares the relative sleep time intervals of both procrastination schemes normalized to CS-DVS. The average sleep interval is seen to increase with a decrease in processor utilization. It is seen that CS-DVS-P1 increases the average sleep interval by 2 to 5 times. Since tasks have larger procrastination intervals under CS-DVS-P2, it further extends the sleep intervals with the average sleep intervals being 4 to 5 time that of CS-DVS. This extended and fewer sleep intervals are beneficial in minimizing the total system energy as it can allow a further shutdown of components such as memory banks and other I/O devices, which have larger shutdown overheads. The figure also compares the average idle interval (time intervals when no task is executing i.e. an active idle or sleep state). It is seen that the average idle interval increases by up to 3 to 7 times under CS-DVS-P1 and up to 5 to 7 times under CS-DVS-P2. This suggests that CS-DVS has many smaller idle intervals which result either in more shutdown overhead or additional leakage energy consumption if not shutdown. With procrastination, the power manager also has fewer shutdown decisions to make.

7 Conclusions and Future Work

In this paper, we see that scheduling policies that combine task slowdown with procrastination are important for energy efficiency. As leakage is significantly contributing to the total power consumption, it is important to compute the optimal operating speed and maximize the sleep intervals. Procrastination significantly reduces the number of wakeups while stretching the sleep intervals. The extended sleep periods result in an energy efficient operation of the system while meeting all timing requirements. We have proposed algorithms to computed maximum task procrastination intervals under both fixed priority and dual priority scheduling policies. The dual priority scheduling guarantees more energy savings than fixed priority scheduling through extended procrastination. We plan to extend these techniques to schedule the system wide resources for minimizing the total energy consumption.

Acknowledgments

The authors acknowledge support from National Science Foundation (Award CCR-0098335) and from Semiconductor Research Corporation (Contract 2001-HJ-899). We would like to thank the reviewers for their useful comments.

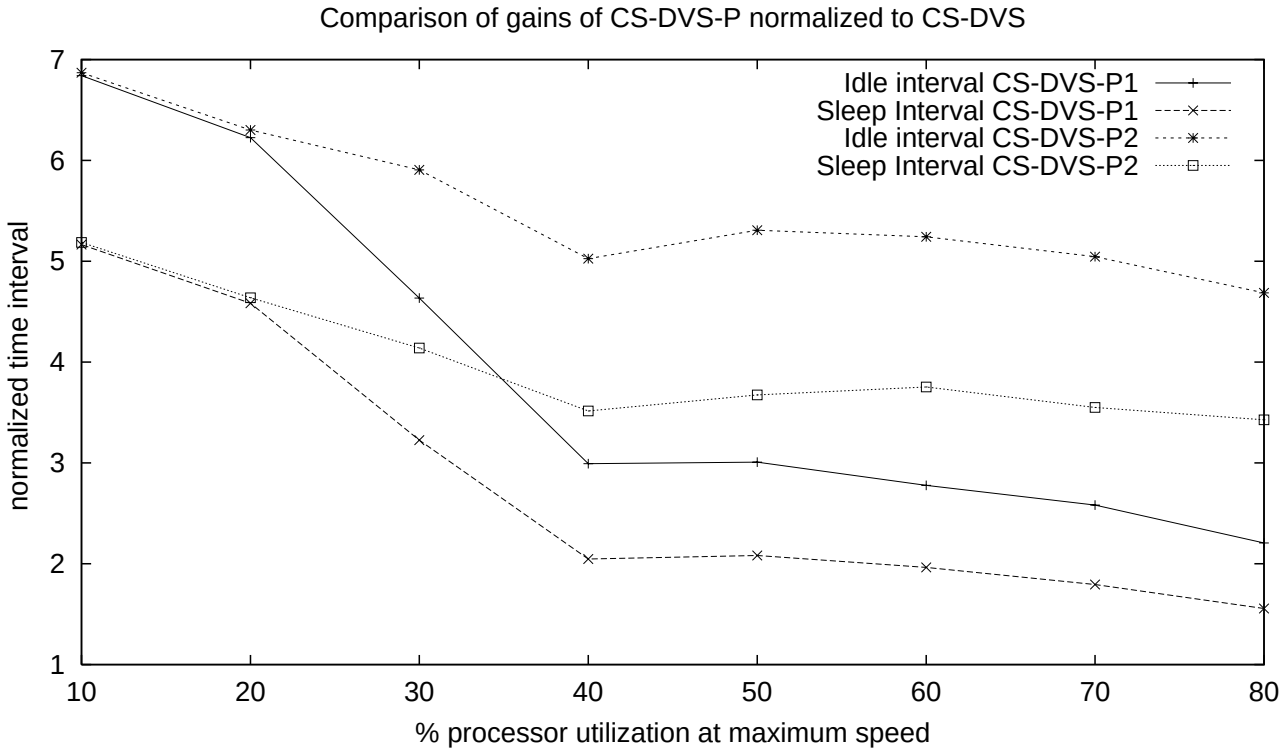


Figure 6. Comparison of average sleep and idle interval of CS-DVS-P1 and CS-DVS-P2 normalized to CS-DVS.

References

[1] Berkeley Predictive Technology Models and BSIM4 http://www-device.eeecs.berkeley.edu/research.html.

[2] T. A. AlEnawy and H. Aydin. Energy-constrained performance optimizations for real-time operating systems. In *Workshop on Compilers and Operating System for Low Power*, Sept. 2003.

[3] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2001.

[4] S. Borkar. Design challenges of technology scaling. In *IEEE Micro*, pages 23–29, Aug 1999.

[5] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in mtcmos. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 104–109, Aug. 2003.

[6] R. Davis and A. Wellings. Dual priority scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 100–109, Dec. 1995.

[7] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage. In *Workshop on Power-Aware Computing Systems*, Dec. 2003.

[8] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 46–51, Aug. 2001.

[9] IBM 405LP Processor. IBM Inc. *(http://www-3.ibm.com/chips/products/powerpc/cores.*

[10] Intel PXA250/PXA210 Processor. Intel Inc. *(http://www.intel.com)*.

[11] Intel XScale Processor. Intel Inc. *(http://developer.intel.com/design/intelxscale)*.

[12] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of Symposium on Discrete Algorithms*, Jan. 2003.

[13] R. Jejurikar and R. Gupta. Dual mode algorithm for energy aware fixed priority scheduling with task synchronization. In *Workshop on Compilers and Operating System for Low Power*, Sept. 2003.

[14] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, Jun. 2004.

[15] M. Johnson, D. Somasekhar, and K. Roy. Models and algorithms for bounds on leakage in cmos circuits. In *IEEE Transactions on CAD*, pages 714–725, 1999.

[16] 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*, Mar. 2002.

[17] W. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the Design Automation Conference*, pages 125–130, 2003.

[18] N. K. J. L. Yan, J. Luo. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. In *Proceedings of International Conference on Computer Aided Design*, Nov. 2003.

[19] H. G. Lee and N. Chang. Energy-aware memory allocation in heterogeneous non-volatile memory systems. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 420–423, Aug. 2003.

[20] Y. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *EcuroMicro Conf. on Real Time Systems*, 2003.

[21] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[22] C. Locke, D. Vogel, and T. Mesler. Building a predictable avionics platform in ada: a case study. In *Proceedings IEEE Real-Time Systems Symposium*, 1991.

[23] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of International Conference on Computer Aided Design*, Nov. 2002.

[24] P. Mejia-Alvarez, E. Levner, and D. Mosse. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 2(4), Nov. 2003.

[25] 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 International Conference on Supercomputing*, Jun. 2002.

[26] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-v power supply highspeed digital circuit technology with multithreshold- voltage cmos. In *IEEE Journal of Solid-State Circuits*, pages 847– 854, 1995.

[27] C. Neau and K. Roy. Optimal body bias selection for leakage improvement and process compensation over different technology generations. In *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 2003.

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

[29] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 28–33, 2001.

[30] G. Quan and X. Hu. Minimum energy fixed-priority scheduling for variable voltage processors. In *Proceedings of Design Automation and Test in Europe*, Mar. 2002.

[31] C. Rusu, R. Melhem, and D. Mosse. Multi-version scheduling in rechargeable energy-aware real-time systems. In *Proceedings of EuroMicro Conference on Real-Time Systems*, 2003.

[32] C. Rusu, R. Melhem, and D. Mosse. Maximizing rewards for real-time applications with energy constraints. In *ACM Transactions on Embedded Computer Systems*, accepted.

[33] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2002.

[34] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of International Conference on Computer Aided Design*, pages 365–368, Nov. 2000.

[35] Transmeta Crusoe Processor. Transmeta Inc. *(http://www.transmeta.com/technology)*.

[36] F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[37] H. Yun and J. Kim. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *Trans. on Embedded Computing Sys.*, 2(3):393–430, 2003.

[38] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2002.