Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems

Ravindra Jejurikar[§]
jezz@cecs.uci.edu

Cristiano Pereira[¶]
cpereira@cs.ucsd.edu

Rajesh Gupta[¶]
gupta@cs.ucsd.edu

[§] Center for Embedded Computer Systems, University of California at Irvine, Irvine CA 92697
[¶] Department of Computer Science, University of California at San Diego, La Jolla, CA 92093

ABSTRACT

A five-fold increase in leakage current is predicted with each technology generation. While Dynamic Voltage Scaling (DVS) is known to reduce dynamic power consumption, it also causes increased leakage energy drain by lengthening the interval over which a computation is carried out. Therefore, for minimization of the total energy, one needs to determine an operating point, called the *critical speed*. We compute processor slowdown factors based on the critical speed for energy minimization. Procrastination scheduling attempts to maximize the duration of idle intervals by keeping the processor in a sleep/shutdown state even if there are pending tasks, within the constraints imposed by performance requirements. Our simulation experiments show that the critical speed slowdown results in up to 5% energy gains over a leakage oblivious dynamic voltage scaling. Procrastination scheduling scheme extends the sleep intervals to up to 5 times, resulting in up to an additional 18% energy gains, while meeting all timing requirements.

Categories and Subject Descriptors: D.4.1 [Operating System]: Process Management – scheduling.

General Terms: Algorithms.

Keywords: leakage power, critical speed, low power scheduling, real-time systems, EDF scheduling, procrastication.

1. INTRODUCTION

Power management is of primary importance in the operation of embedded systems, which can be attributed to longer battery life, reliability and packaging costs. Power consumption of a device is broadly classified into (1) *Dynamic* power consumption which arises due to switching activity in a circuit and (2) *Static* power consumption which is present even when no logic operations are performed. CMOS has emerged as a dominant technology because of its low static power consumption. CMOS device scaling trend, driven by the need for faster devices and higher transistor densities, shows a 30% decrease in the device dimensions with each technology generation [5]. Constant electric field scaling allows a proportional reduction in supply voltage for smaller devices. As supply voltage is reduced, the threshold voltage (V_{th}) must be proportionately reduced to maintain the desired performance (gate delay) improvements. This reduction of threshold voltage results in an exponential increase in the subthreshold leakage current, leading to larger standby current [6].

Leakage current in CMOS circuits contribute to a significant portion of the total power consumption and has become a major concern. The subthreshold leakage current is $0.01\mu A/\mu m$ for the $130nm$ and is projected to be $3\mu A/\mu m$ for the $45nm$ technology [1]. A five fold increase in the leakage power is predicted with each technology generation [5]. The static power consumption is comparable to the dynamic power dissipation and projected to surpass it if measures are not taken to minimize leakage current [9].

To address the issue of leakage, efforts are being made at process, circuit design and micro-architecture level. The exponential relation of subthreshold leakage current to the threshold voltage has led to threshold voltage scaling. Scaling the threshold voltage by controlling the body bias voltage has been proposed to minimize leakage [24, 22]. Multi threshold CMOS (MTCMOS) [7] is a popular technique to reduce standby current. Other techniques such as input vector control [15] and power supply gating [23] have also been proposed. At higher levels of abstraction, recent works have focused on minimizing the leakage of components such as cache. Techniques like cache decay [11] and turning off cache lines [10] are effective in reducing cache leakage. Clock gating techniques are also used to control leakage in Systems on Chip (SoC). The IBM PowerPC 405LP [8] implements clock gating at the IP core and register level. The Intel PXA [12] family processors also support fine granularity clock gating to exploit the fact that not all system transistors are used at the same time. The chip aggressively shuts down elements of the processor which are idle by gating them off or disabling their input. Processors support various shutdown mode to minimize the idle power. For examples, the Transmeta Crusoe [28] processor support various sleep modes (normal, autohalt, quick start, deep sleep, off) for various types of workload. These power states can be used to reduce the processor power consumption when little or no CPU activity is needed.

Processor slowdown (through dynamic frequency/voltage scaling) and shutdown (through clock gating and other means) are two primary ways to reduce power consumption. Between slowdown and shutdown, generally slowdown is preferred due to the quadratic dependence of power on voltage level, thus making shutdown a secondary strategy (e.g., shutdown periods are sought after applying applicable slowdown strategies). However, energy savings based on DVS come at the cost of increased execution time, which implies greater leakage energy consumption. With the steep increase in device leakage current with each technology generation, it is not obvious whether to perform DVS or to execute the system at maxi-

DAC'04, June 7–11, 2004, San Diego, California, USA.

mum speed and shutdown. Thus we have to judiciously balance the extent of slowdown and shutdown to minimize the total energy consumption. As shown later, operating at the maximum or minimum possible voltage (frequency) need not be the optimal point. Furthermore, the additional time and energy cost of shutdown makes the problem harder.

Previous works have addressed Dynamic Voltage Scaling (DVS) based on performance requirements to minimize the dynamic power consumption [26, 25, 27, 3, 4, 16]. While techniques to optimize the total static and dynamic power consumption have been proposed [17, 22], they are still based on the premise that the energy savings are proportional to the extent of slowdown. Irani *et. al.* [13] consider the combined problem of DVS and shutdown and propose a *3-competitive* off-line algorithm. Their result is based on the assumption of a continuous voltage range and a convex power consumption function. Lee *et. al.* propose an Leakage Control EDF (LC-EDF) [19] scheduling algorithm to minimize the leakage energy consumption in real-time systems. The algorithm computes the time interval by which task executions can be procrastinated, to extend the idle intervals. Note that their algorithm is based on the assumption that all tasks are executed at the maximum speed, which may not be energy efficient. In this work, we combine dynamic voltage scaling with procrastination to minimize the total energy consumption. Our work differs from [19], in that we seek to minimize total energy through use of procrastination intervals and task slowdown factors. Further, our algorithm runs in constant time and is simpler to implement than the linear time LC-EDF algorithm. We also prove that the minimum idle period guaranteed by our algorithm is always greater than or equal to that achieved by LC-EDF, making our technique superior. Our contributions are as follows: (1) based on the leakage characteristics of the 70*nm* technology, we compute the *critical speed* for the system; (2) we combine dynamic voltage scaling and procrastination to minimize the total energy consumption; (3) we propose a novel algorithm to compute maximum task procrastination intervals for a dynamic priority system.

The rest of the paper is organized as follows: Section 2 and 3 discuss the leakage power model and the computation of the critical speed. In Section 4, we present the critical speed slowdown followed by a procrastination scheduling algorithm under the EDF scheduling policy. The experimental results are presented in Section 5 and Section 6 concludes the paper with future directions.

2. 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,

$$P_{AC} = C_{eff} V_{dd}^2 f \tag{1}$$

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance. Dynamic voltage scaling reduces the dynamic power consumption due to its quadratic dependence on voltage.

Different leakage sources contribute to the total static power consumption in a device [2]. The major contributors of leakage are the subthreshold leakage and the reverse bias junction current which can increase significantly with adaptive body biasing [22]. We use the power model and the technology parameters described by Martin *et. al.* [22]. The threshold voltage V_{th}, subthreshold current I_{subn}, and cycle time t_{inv} as a function of the supply voltage V_{dd} and the body bias voltage V_{bs} are given below :

Table 1: 70*nm* **technology constants [22]**

Const	Value	Const	Value	Const	Value
K_1	0.063	K_6	$5.26x10^{-12}$	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	$4.8x10^{-10}$
K_3	$5.38x10^{-7}$	V_{dd0}	1	C_{eff}	$0.43x10^{-9}$
K_4	1.83	V_{bs0}	0	L_d	37
K_5	4.19	α	1.5	L_g	$4x10^6$

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \tag{2}$$

where K_1, K_2 and V_{th1} are technology constants.

$$I_{subn} = K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} \tag{3}$$

where K_3, K_4 and K_5 are constant fitting parameters.

$$t_{inv} = \frac{L_d K_6}{(V_{dd} - V_{th})^{\alpha}} \tag{4}$$

The leakage power dissipation due to subthreshold leakage (I_{subn}) and reverse bias junction current (I_j) is given by,

$$P_{DC} = V_{dd} I_{subn} + |V_{bs}| I_j \tag{5}$$

Equation 5 gives the leakage per device and the total leakage power consumption is $L_g \cdot P_{DC}$, where L_g is the number of devices in the circuit. The technology constants for the 70*nm* technology are presented in Table 1, as given in [22]. The value for C_{eff} based on the Transmeta Crusoe processor, scaled to 70*nm* technology based on the technology scaling trends [5], 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 $0.5V$ to $1.0V$, is shown in Figure 1.

3. CRITICAL SPEED

In addition to the static and dynamic power consumption per device, there is an inherent power cost in keeping the processor on, which is denoted by P_{on}. Similar to device leakage power, 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 200*mA* current [12, 28] and (2) the I/O subsystem with a supply voltage, V_{IO}, (2.5V to 3.3V) higher than the processor core voltage and peak currents of 400*mA* during I/O. Though the current is comparatively lower when there is no I/O, the power consumption adds to a significant portion of the idle power consumption. The power consumption of these components will scale with technology and architectural improvements and we assume a conservative value of $P_{on} = 0.1W$. Considering the static (P_{DC}), dynamic (P_{AC}) and P_{on} components, the total processor power consumption, P, is given by:

$$P = P_{AC} + P_{DC} + P_{on} \tag{6}$$

The variation of the power consumption with supply voltage is shown in Figure 1. The linear dependence of static power consumption on voltage and the quadratic dependence of dynamic power on voltage is seen in the figure. Though the total power consumption decreases as V_{dd} is scaled, it does not imply energy savings. The decrease in the operating frequency with voltage scaling increase the static energy consumption, which can surpass the gains of dynamic voltage scaling. Thus the aggressiveness of voltage scaling has to be based on leakage power characteristics. To evaluate the effectiveness of dynamic voltage scaling, we compute the energy

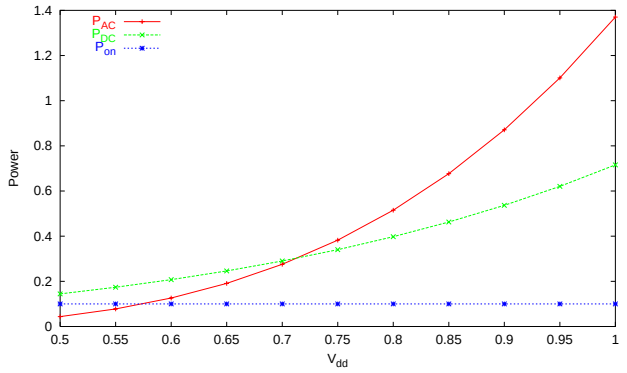

Figure 1: Power consumption of 70*nm* technology for Crusoe processor: P_{DC} is the leakage power, P_{AC} is the dynamic power and P_{on} is the intrinsic power consumption in on state

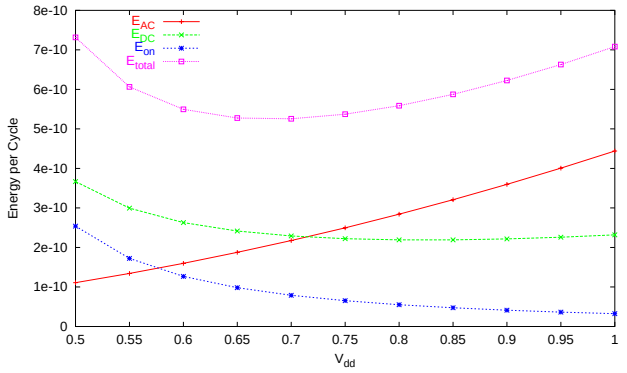

Figure 2: Energy per Cycle for 70*nm* technology for the Crusoe processor: E_{AC} is the switching energy, E_{DC} is the leakage energy and E_{on} is the intrinsic energy to keep the processor on.

consumption per cycle, for different supply voltage values. The contribution of the dynamic energy per cycle, E_{AC}, is

$$E_{AC} = C_{eff} V_{dd}^2 \tag{7}$$

The leakage power per device is given by Equation 5. Since the cycle time increases as voltage decreases, the leakage energy per cycle, E_{DC}, is given by

$$E_{DC} = f^{-1} \cdot L_g \cdot (I_{subn} V_{dd} + |V_{bs}| I_j) \tag{8}$$

where f^{-1} is the delay per cycle. The energy per cycle to keep the system on is $E_{on} = f^{-1} P_{on}$ and increases with lower frequencies. The total static and dynamic energy consumption per cycle, E_{cycle}, with varying supply voltage levels is as follows:

$$E_{cycle} = E_{AC} + E_{DC} + E_{on} \tag{9}$$

Figure 2 shows the components of the energy consumption per cycle for the 70*nm* technology. We see that E_{cycle} decreases as V_{dd} is scaled up to $0.7V$, beyond which static energy consumption dominates. The total energy consumption increases with further slowdown and it is not energy efficient to slowdown beyond $V_{dd} = 0.7V$. Executing at $V_{dd} = 0.7V$ and shutting down the system is more energy efficient than executing at lower voltages levels. The operating point that minimizes the energy consumption per cycle is called the *critical speed*. From the figure, it is seen that the critical speed of operation is $V_{dd} = 0.7V$. Note that the critical speed can be computed by evaluating the gradient of the energy function with respect to V_{dd}. From the voltage frequency relation described in Equation 4, $V_{dd} = 0.7V$ 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 $\eta_{crit} = 1.26/3.0 = 0.41$.

4. REAL TIME SCHEDULING

In this section, we enhance real-time scheduling techniques with the knowledge of critical speed η_{crit}, to minimize the total energy consumption of the system. We begin with the description of the system model.

4.1 System Model

In a classical real-time system model, tasks arrive periodically and have deadlines. A task set of n periodic real-time tasks is represented as $\Gamma = \{\tau_1, ..., \tau_n\}$. 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) for the task at maximum speed. The tasks are scheduled on a single processor system based on a preemptive scheduling policy. 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 [20]. In this work, we assume task deadlines are equal to the period ($D_i = T_i$) and the tasks are scheduled by the Earliest Deadline First (EDF) scheduling policy [20]. All tasks are assumed to be independent and preemptive.

Recent processor [28, 12] support Dynamic voltage scaling (DVS) to minimize the dynamic power consumption of a processor. A task slowdown factor is the extent of slowdown that can be applied while meeting specified performance requirements. A *slowdown factor* (η_i) can be viewed as the normalized operating frequency and lies in the range [0,1]. At a given instance, it is the ratio of the assigned frequency to the maximum processor frequency.

4.2 Slowdown and Critical Speed

Note that the task slowdown can be computed with any known dynamic voltage scaling algorithm. We update the computed task slowdown factors based on the processor critical speed, η_{crit}. Since executing below the critical speed consumes more time and energy, we set the minimum value for the slowdown factor as the critical speed. We update a task slowdown factor to the critical speed if it is smaller than η_{crit}. The algorithm is as follows:

$$\overset{\forall i}{i = 1, ..., n} \qquad if(\eta_i < \eta_{crit})\ \ \eta_i \leftarrow \eta_{crit} \tag{10}$$

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

4.3 Shutdown Overhead

In previous works, the overhead of processor shutdown/wakeup has been either neglected or only the actual time and energy overhead incurred within the processor is considered. However, a processor shutdown and wakeup has a higher overhead than the inherent energy/delay cost of turning on the processor, as specified in datasheets. The processor loses the register and cache contents, when switched to the deepest sleep mode. Thus prior to shutdown, all registers must be saved and the dirty data cache lines must be flushed to main memory, resulting in an additional overhead. On wakeup, components such as data and instruction caches, data and instruction translation look aside buffers (TLBs) and branch target buffers (BTBs) have to be initialized, resulting in cold start misses in case of caches and TLBs, and branch mispredictions in case of the BTBs. This results in extra memory accesses and hence addi-

tional energy consumption. The exact cost will vary based on the processor architecture and the application.

Due to the cost of shutdown, a shutdown decision should 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 even with the wakeup energy overhead. It is not energy efficient to shutdown when the idle intervals are shorter than $t_{threshold}$. The threshold interval depends on the idle state power consumption and the shutdown overhead. Let P_{idle} be the power consumption in the idle state, in addition to the power consumption in the shutdown state. Given $t_{shutdown}$ and $E_{shutdown}$ are the time and energy overhead incurred due to shutdown (the overhead of shutdown as well as wakeup), then $t_{threshold}$ is given by:

$$P_{idle} \cdot t_{threshold} = E_{shutdown}$$

Thus longer idle interval are required to increase the changes of shutdown and reduce the idle energy consumption. We present a procrastination scheduling scheme to achieve this goal.

4.4 Procrastination Algorithm

We pre-compute a maximum procrastination interval, Z_i, for each task τ_i in the system. The computation of Z_i, the time interval by which task τ_i can be delayed while guaranteeing all task deadlines, is given by Theorem 1. The procrastination algorithm ensures that no task τ_i is procrastinated by more than Z_i time units. The procrastination scheme is described in Algorithm 1. It is assumed that the *power manager* which handles task procrastination is implemented as a controller. When the processor enters the sleep/shutdown state, it hands over the control to the power manager (controller), which handles all the interrupts and task arrivals while the processor is shutdown. The controller has a timer to keep track of time and wake the processor after a specified time period. When the processor enters the 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 arrives before the counter expires, the counter is adjusted based on the new task arrival. Let the arrived task be τ_j, then the timer is updated to the minimum of the current timer value and Z_j. This ensures that no task τ_k in the system is procrastinated by more than Z_k time units after its arrival. When the counter counts down to zero (expires), the processor is woken up and the scheduler dispatches the highest priority task in the system. All tasks are scheduled at their assigned slowdown factor based on their priority.

THEOREM 1. *Given tasks are ordered in non-decreasing order of their period, the procrastination algorithm guarantees all task deadlines if the procrastination interval Z_i of each task τ_i satisfies:*

$$\overset{\forall i}{i = 1,\ldots,n} \qquad \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \tag{11}$$

$$\forall_{k<i} \;\; Z_k \leq Z_i \tag{12}$$

PROOF. The details of the proof are present in [14]. □

We also compute the minimum idle period guaranteed by the procrastination algorithm. This idle period helps make better shutdown decisions.

COROLLARY 1. *The minimum idle period guaranteed by the procrastination algorithm is is given as,*

$$Z_{min} = min_{1 \leq i \leq n} \qquad \left\{ Z_i = (1 - \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k}) T_i \right\} \tag{13}$$

Algorithm 1 Procrastination Algorithm

```
1: On arrival of a new job J_i:
2: if (processor is in sleep state) then
3:    if (Timer is not active) then
4:       timer ← Z_i; {Initialize timer}
5:    else
6:       timer ← min(timer, Z_i);
7:    end if
8: end if

9: On expiration of Timer (timer = 0):
10: Wakeup Processor;
11: Scheduler schedules highest priority task;
12: Deactivate timer;

13: Timer Operation:
14: timer − −; {Counts down every clock cycle}
```

We also analytically compare our algorithm to the LC-EDF [19] algorithm. Since the LC-EDF algorithm assumes all tasks execute at maximum speed, the result is based on the assumption that all tasks are executed at the maximum speed. Note that the results also extend to task procrastination with slowdown. We prove that our proposed algorithm guarantees more procrastination than LC-EDF if tasks are executed at maximum speed.

THEOREM 2. *Given, tasks are executed at maximum speed, the minimum delay interval guaranteed by the procrastination algorithm is greater than or equal to that guaranteed by LC-EDF.*

PROOF. The proof of the result is presented in [14]. □

5. EXPERIMENTAL SETUP

We have 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 [4, 19, 27]. Based on real life task sets [21], 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. Each task is assigned a slowdown factor equal to the utilization at maximum speed, which is the optimal slowdown under EDF scheduling policy, to minimize the dynamic energy consumption [4]. We compared 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.
- Critical Speed DVS (CS-DVS): where all tasks are assigned a slowdown greater than or equal to the processor critical speed.
- Critical Speed DVS with Procrastination (CS-DVS-P): This is CS-DVS with the procrastination scheduling scheme.

We use the processor power model described in Section 2, where the processor critical speed is $\eta_{crit} = 0.41$. 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. In all the scheduling schemes except CS-DVS-P, the processor wakes up on the arrival of a task in the system. The idle interval in these techniques is assumed to be the time period before the next task arrival in the system. CS-DVS-P adds the minimum guaranteed procrastination interval to estimate the minimum idle interval. The processor is shutdown if the idle period is greater than $t_{threshold}$, the minimum idle period to result in energy savings.

5.1 Shutdown Overhead

In this section, we estimate the energy overhead of shutdown, taking into account the on-chip cache. Typical embedded processors have a cache size between $32KB$ and $128KB$ and we assume that the processor has a $32KB$ I-cache and a $32KB$ D-cache. We assume 20% lines of the data cache to be dirty before shutdown which results in 6554 memory writes. With an energy cost of 13nJ [18] per memory write, the cost of flushing the data cache is computed as $85\mu J$. We assume the energy and latency of saving the registers to be negligible. On wakeup, there is an additional cost due to cache miss. Note that the locality of reference changes when a task resumes execution which has its own cache miss penalty. However, resuming execution after shutdown has an additional overhead due to the fact that these structures are empty after a shutdown. We assume the additional cache miss rate to be 10% of the cache size, in both I-cache and D-cache. We ignore the overhead resulting from TLBs and BTBs. Thus, the total overhead of bringing the processor to active mode is 6554 cache misses. A cost of 15nJ [18] per memory access, results in $98\mu J$ energy overhead. Adding the cache energy overhead to the actual charging of circuit logic, which we assume to be $300\mu J$, the total cost is $85+98+300=483\mu J$. Since the idle power consumption is $240mW$, the threshold idle interval, $t_{threshold}$ is $2ms$. 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 power manager (controller).

5.2 Experimental Results

The comparison of the energy consumption of the techniques is shown in Figure 3, 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 savings. no-DVS consumes the maximum energy and the energy consumption of all techniques is normalized to no-DVS. It is seen that all the techniques perform almost identical up to the critical speed. When the task slowdown factors fall below the critical speed, DVS technique starts consuming more energy due to the dominance of leakage. At lower speeds the energy consumed by DVS approaches close to that of no-DVS. 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, it can consume more energy that the DVS technique, as seen at utilization of 20% and 30%. CS-DVS leads to up to 5% energy gains over DVS and an average of 20% energy savings compared to no-DVS. The CS-DVS-P minimizes idle energy by maximizing the sleep intervals and reducing the shutdown overhead. We see that the CS-DVS-P results up to an additional 18% gains over CS-DVS at 10% utilization.

Figure 4 compares CS-DVS-P to CS-DVS. Figure 4(a) shows the number of wakeups and the idle energy comparison of CS-DVS-P 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

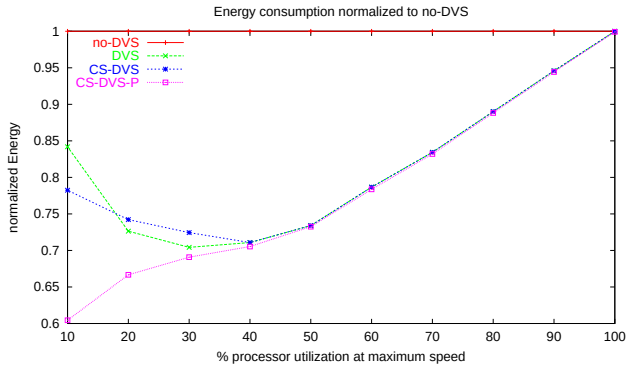

Figure 3: Energy consumption normalized to no-DVS, based on the EDF scheduling policy.

in dynamic reclamation [4] for more energy gains. However, we use these idle intervals for processor shutdown to compare the benefits of our procrastination scheme. At higher utilization, the inherent idle intervals are short and there is less chance for a shutdown. However, procrastination extends the idle periods and results in energy savings through shutdown. As seen in Figure 4(a), the number of shutdowns are higher at higher utilization which results in reduced idle energy consumption. At lower utilization, the relative number of wake-ups compared to CS-DVS decrease considerably. Note that the idle energy consumption of CS-DVS-P is always lower than that of CS-DVS. The number of wakeups are reduced to as much as 25% percent thereby reducing the shutdown overhead. We see from the figure that the idle energy consumption also reduces proportionately.

Figure 4(b) compares the relative increase of the sleep time intervals of CS-DVS-P over CS-DVS. We see that the average sleep interval is increased by 4 to 5 times. This extended sleep interval is beneficial as it allows for a shutdown of other peripheral devices that are idle. I/O devices such as memory have a time overhead of $10ms$ to wake up from deep sleep states. Procrastination increases the opportunity to shutdown more devices to minimize the total system energy. The figure also compares the average idle interval (intervals when no task is executing i.e. an idle or sleep state). We see that the average idle interval increases up to 7 times. This suggests that CS-DVS has relatively more idle intervals where it does not shutdown the processor resulting in leakage energy consumption. CS-DVS-P increasing the opportunity to shutdown by clustering task executions, thereby saving energy.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented scheduling techniques that consider leakage energy contribution to minimize the total energy consumption in a system. We show that executing tasks at the maximum or minimum processor speed need not be the optimal operating point. While operating at the minimum speed can increase the leakage energy contribution, we show that executing tasks at the critical speed and shutting down the processor is more energy efficient. This results in up to 5% energy gains. Furthermore, extending the sleep intervals by our procrastination scheme increases the energy savings to up to 18%. We show that a combined approach of slowdown and procrastination is important for an energy efficient operation of the system. The techniques are simple, energy efficient and can be easily implemented. We plan to extend these techniques for energy efficient scheduling of all system resources.

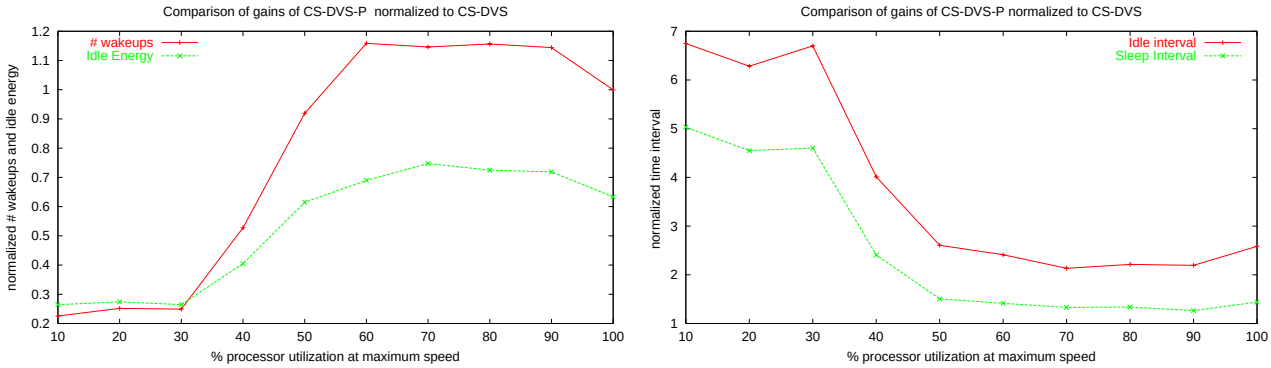

Figure 4: (a) Comparison of # wakeups and idle energy of CS-DVS-P normalized to CS-DVS (b) Comparison of average sleep and idle interval of CS-DVS-P normalized to CS-DVS

7. REFERENCES

[1] International Technology Roadmap for Semiconductors 2002 http://public.itrs.net.

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

[3] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of EuroMicro Conference on Real-Time Systems*, 2001.

[4] 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*, December 2001.

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

[6] J. A. Butts and G. S. Sohi. A static power model for architects. In *Intl. Symposium on Microarchitecture*, 2000.

[7] 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, 2003.

[8] G. Carpenter. Low power soc for ibm's powerpc information appliance platform. In *http://www.research.ibm.com/arl.*

[9] D. Duarte, N. Vijaykrishnan, M. J. Irwin, and Y.-F. Tsai. Impact of technology scaling and packaging on dynamic voltage scaling techniques. In *15th Annual IEEE International ASIC/SOC Conference*, September 2002.

[10] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of International Symposium on Computer Architecture*, June 2002.

[11] Z. Hu, S. Kaxiras, and M. Martonosi. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. In *ACM Transactions on Computer Systems*, May 2002.

[12] Intel PXA250/210 Processor. Intel Inc. *(www.intel.com).*

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

[14] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *CECS Technical Report #03-35, UC Irvine*, Dec. 2003.

[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] C. M. Krishna and Y. H. Lee. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings of Real-Time Technology and Applications Symposium*, May 2000.

[17] 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.

[18] H. G. Lee and N. Chang. Energy-aware memory allocation in heterogeneous non-volatile memory systems. In *ISLPED*, pages 420–423, 2003.

[19] 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.

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

[21] 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.

[22] 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 *Proc. of Intl. Conference on Computer Aided Design*, 2002.

[23] 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.

[24] 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*, 2003.

[25] 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.

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

[27] 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, 2000.

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