Systemwide Energy Minimization in Real-Time Embedded 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 #04-14
May, 2004

Abstract

Traditionally, dynamic voltage scaling (DVS) techniques have focused on minimizing the processor power consumption as opposed to the entire system energy. However, the slowdown resulting from DVS can increase the energy consumption of components like memory and network interfaces. Furthermore, leakage power consumption, which is increasing with the scaling device technology, must also be considered. In this paper, we present an algorithm to compute task slowdown factors based on the contribution of the processor leakage and standby energy consumption of the resources in the system. We combine slowdown with procrastination of task executions to extend sleep intervals which significantly reduces the leakage energy consumption. We show that the scheduling approach minimizes the total static and dynamic energy consumption of the systemwide resources. In this work, we consider a real-time system model using the Earliest Deadline First (EDF) policy. Our simulation experiments using randomly generated task sets show on an average 10% *energy gains over traditional dynamic voltage scaling. Our procrastination scheme increases the average energy savings to* 15%*.*

Contents

1 Introduction **1**

2 System Model **2**

3 Low power scheduling **3**
3.1 Slowdown Algorithm 4
3.2 Procrastination Algorithm 4

4 Processor Power Model **5**
4.1 Shutdown Overhead 6

5 Experimental Setup **7**
5.1 Energy Consumption 9

6 Conclusions and Future Work **9**

List of Figures

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 7
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. 8
3 Energy consumption normalized to no-DVS 10

List of Tables

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. There are two primary ways to reduce power consumption in computing systems: (1) *resource shutdown*, commonly known as dynamic power management (DPM) and (2) *resource slowdown*, also known as dynamic voltage scaling (DVS). Resources such as memory banks, disk drives, displays and network interfaces possess shutdown capability and DPM techniques have been proposed to minimize the power consumption of these resources [7, 11, 25]. Recent processors support slowdown and DVS techniques are known to be more effective in reducing the processor energy consumption [5, 24]. DVS techniques exploit an energy-delay tradeoff that arises due to the quadratic relationship between voltage and power whereas a linear relationship between voltage and delay (frequency). Note that DVS decreases the energy consumption at the cost of increased execution time. The longer execution time while decreasing the dynamic power consumption of the processor, can increase the energy contribution of other components for the following reasons:

- The standby leakage currents are increasing with the advances of CMOS technology and a five fold increase in the leakage power is predicted with each technology generation [6]. Longer execution time implies more leakage power.

- A minimum power consumption is associated in keeping the processor active. Some of the major contributors are the PLL circuitry, which drives up to $200mA$ current [10, 27] and the I/O and analog components of the processor. Note that the I/O and analog components require higher voltage levels (2.5V to 3.3V) that are not scaled with DVS. Only the processor core voltage (V_{dd}) [10] is scaled under DVS.

- If components such as memory and other I/O interfaces need to be active (on state) along with the processor, slowdown can increase the total energy consumption of the system.

Components such as memory banks, flash drives, co-processor (DSP, FPU, codecs), FPGA components, analog interfaces and wired /wireless communication devices are pervasive in modern embedded systems. Most of these resources support multiple shutdown-states for energy minimization. Due to the energy and delay costs of state transitions, the shutdown decisions have to be judiciously made to meet the system requirements. This results in the device operating in the standby state (on-state but idle) where significant power is consumed. Memory modules are present in almost all computing systems with DRAMs and RDRAMs having standby current in the range of $30mA$ to $120mA$ [2, 3]. These devices have operating voltages in the range of $1.8V$ to $3.3V$, and can consume up to $0.36W$ of power. SRAM modules have still higher standby currents of the order of $150mA$ to $250mA$. The standby power consumption of devices such as flash drives and wireless interfaces is up to $0.5W$ [1] and $1.4W$ [4] respectively. Other components like FPGAs, co-processors and codecs also consume significant power based on their functionality. The resource standby time is related to the program execution and can increase with DVS (slowdown). With compiler assisted DPM techniques [7], standby time increases proportionally to the execution time. Thus DVS techniques need to consider the standby power consumption of the peripherals devices in the computation of slowdown factor to reduce the total power consumption of the system.

Most of the works on DVS consider the energy consumption of the processor in isolation. Earlier works have addressed minimizing the dynamic power consumption of the processor [5, 24], whereas

later works have focussed on leakage to minimize the total static and dynamic power consumption [21, 28, 14]. Slowdown tradeoffs in the computation and communication subsystems are considered in [17, 20]. Recent works have also considered the combined processor and memory energy consumption. Fan *et al.* [8] consider memory power consumption to show that excess slowdown can increase the total energy consumption. Miyoshi *et al.* [22] show that the slowdown decision can differ with different processor families. Various shutdown policies for resources such as memory, disks and wireless cards have been studied [7, 11, 25].

While most of the work on DVS is focussed on minimizing the processor power consumption, the standby power is usually ignored. It is observed that devices like memory banks are in the active state $30\% - 90\%$ of the task execution time [7]. With the steady increase in the amount of data which is often distributed, the network and disk activity increases and so is the standby time of these devices. We take into account the standby power consumption of the resources used by tasks to compute energy efficient task slowdown factors. Given the resource usage and the resource standby time for the tasks, we propose an algorithm to compute task slowdown factors to minimize the total energy. We enhance our technique with procrastination scheduling which further reduces the processor leakage contribution by extending the processor sleep intervals. Procrastination in real-time systems was first proposed by Lee *et al.* [16], however they assume all tasks are executed at the maximum speed. We have combined slowdown and procrastination scheduling in our earlier works [13]. We have proposed a better procrastination algorithm that guarantees all deadlines under the EDF scheduling policy. The algorithm has been extended to fixed priority scheduling in [12].

The rest of the paper is organized as follows: Section 2 introduces the system model and formulates the problem. In Section 3, we present the computation of slowdown factors and procrastination intervals under the EDF scheduling policy. The processor power model is discussed in Section 4 and the experimental results are discussed in Section 5. Finally, Section 6 concludes the paper with future directions.

2 System Model

A task set of n periodic real time tasks is represented as $\Gamma = \{\tau_1, ..., \tau_n\}$. A 3-tuple $\{T_i, D_i, C_i\}$ is used to represent each task τ_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. In this work, we assume task deadlines are equal to the period ($D_i = T_i$) and the tasks are scheduled by the EDF scheduling policy [18]. All tasks are assumed to be independent and preemptive. The tasks are to be scheduled on a single processor system based on a preemptive scheduling policy. We say a task is *procrastinated* (or delayed) if the processor remains idle despite the presence of the task in the processor ready queue. The procrastination interval of a task is the time interval by which a task is procrastinated.

Similar to recent processors such as Intel XScale [10] and Transmeta Crusoe [27], the processor support variable voltage and frequency levels. There are s available frequencies, $\{f_1, ..., f_s\}$ in increasing order of frequency and the corresponding voltage levels are $\{v_1, ..., v_s\}$. A *slowdown factor* (η_i) is defined as the normalized operating frequency i.e. the ratio of the current frequency to the maximum frequency, f_s, of the processor. The important point to note is when the frequency is changed to f_k, the voltage level is also proportionately set to v_k. The power consumption of the processor at a slowdown of η is represented as $P(CPU, \eta)$. Since processors support discrete frequency levels, the slowdown factors are discrete points $(\frac{f_1}{f_s}, \frac{f_s}{f_s}, ..., 1)$ in the range [0,1]. The processor supports shutdown to reduce

the leakage power consumption. The processor is said to be *idle* if it is not executing a task. In the idle state, the processor could in the shutdown state (no leakage) or in the standby state (active + idle) when it consumes leakage power. The slowdown factor assigned to task τ_i is represented as η_i. When task τ_i is assigned a slowdown factor $\frac{f_k}{f_s}$, the task slowdown factor is represented by η_i^k to make the assignment explicit when required. We assume that the overhead incurred in changing the processor speed is incorporated in the task execution time. This overhead, similar to the context switch overhead, is constant and can be incorporated in the worst case execution time of a task. We note that the same assumption is made in previous works [5, 24].

In addition to the processor, the system has a set of m resources $R = \{R_1, \ldots, R_m\}$ that model the peripheral devices. The resource is said to be in the *standby* state if it is on (active) but idle. The standby state power consumption of each resource R_i is given by $P(R_i)$ and the shutdown power of the resource is assumed to be zero. The power consumed in performing the resource functionality is independent of the task slowdown and not considered in our analysis. Each task τ_i uses a subset of the resources in R, represented by R^{τ_i}. Despite the use of DPM policies, the resources are in a standby state for a significant portion of time. We assume that the device standby time for each task is expressed in number of processor cycles. Since the device activities are related to the program execution the standby time is expected to be represented in terms of processor cycles. This is particularly true for compiler directed DPM. Though the standby time can potentially vary with slowdown under OS directed DPM, we assume that the number of cycles a resource is in standby state is independent of slowdown. Let $C_i^{R_j}$ be the number of cycles resource R_j is in the standby stand during the execution of τ_i. If a task does not use resource R_j, then $C_i^{R_j} = 0$. We compute the task slowdown factors that minimize the total system energy including the resource standby contribution. Note that we are not proposing DPM policies, but considering the standby energy in computing static slowdown factors. In our work, we consider task level slowdown factors as opposed to intra-task level slowdown.

3 Low power scheduling

Considering the contribution of the processor leakage power and the resource standby power, the slowest speed need not be the optimal slowdown factor. The energy consumption of a task τ_i at speed η is given by :

$$E_i(\eta) = \frac{C_i}{\eta} P(CPU, \eta) + \sum_{R_j \in R^{\tau_i}} \frac{C_i^{R_j}}{\eta} P(R_j) \tag{1}$$

The slowdown factor for a task that minimizes its total energy consumption is called the *critical speed* for the task. Based on the the EDF scheduling policy, a task-set of n independent periodic tasks is feasible at a slowdown factor of η_i for task τ_i if the utilization under slowdown is no more than unity. Thus we have an optimization problem:

$$minimize: \quad \sum_{i=1}^{n} \frac{1}{T_i} E_i(\eta_i) \tag{2}$$

$$subject\ to: \quad \sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i} \leq 1 \tag{3}$$

$$\forall_i : \eta_i \in \{\frac{f_k}{f_s} | k = 1, ..., s\} \quad (4)$$

3.1 Slowdown Algorithm

While we do not know the time complexity of problem to compute the optimal task slowdown factors, we present a heuristic algorithm to compute energy efficient slowdown factors. The proposed heuristic is motivated by the algorithm in [23]. The algorithm consists of two phases : (1) computing the critical speed for each task (2) Increasing the task slowdown factors if the task set is not feasible. We compute the energy consumption of each task at each discrete slowdown factors and the critical speed is the slowdown factor that minimizes the task energy. Due to different resource usages of task, the critical speed can differ for each task. In the second step, we present a heuristic to select a task whose speed is increased. The candidate tasks for speedup are the tasks that do not have the maximum speed. Given η_i^k is the current slowdown of a candidate task τ_i, the next higher slowdown factor is represented by η_i^{k+1}. Among all candidate tasks, we increase the slowdown of a task that results in the minimum energy increase per unit time. For each candidate task τ_i, we compute the increase in energy consumption, ΔE_i, and the time gained by the speedup, Δt_i, where $\Delta E_i = E_i(\eta_i^{k+1}) - E_i(\eta_i^k)$ and $\Delta t_i = C_i(\frac{1}{\eta_i^k} - \frac{1}{\eta_i^{k+1}})$. The slowdown factor (speed) of the candidate task with the minimum value of $\frac{\Delta E_i}{\Delta t_i}$ is increased. The same heuristic is used in [23] to increase the task slowdown factor. The pseudo-code for the algorithm is given in Figure 1.

Algorithm 1 Computing Slowdown Factors

```
1: Compute the critical speed for each task;
2: Initialize η_i to critical speed of τ_i;
3: while ( not feasible) do
4:     Let τ_m be task satisfying:
5:     (a) η_m is not the maximum speed;
6:     (b) ΔE_m/Δt_m is minimum;
7:     Increase speed of task τ_m;
8: end while
9: return slowdown factors η_i;
```

3.2 Procrastination Algorithm

Task procrastination [16] [13] has been shown to increase the sleep intervals and thereby reduce the energy consumption of idle intervals. Task procrastination is increasingly important as the device leakage currents is rapidly increasing. We use the procrastination algorithm proposed in our earlier work [13]. A maximum procrastination interval Z_i is computed for each task τ_i as given by Theorem 1. The procrastination algorithm ensures that no task is procrastinated more than its Z_i. Task executions are procrastinated only when the processor is in the sleep state. It is assumed that the *power manager* that handles task procrastination is implemented as a 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 wake the processor after a

specified time period. The procrastination algorithm is shown in Figure 2. 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 arrives before the timer expires, the timer is adjusted based on the new task arrival. When another task τ_j arrives, 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 timer 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 slowdown factor.

Algorithm 2 Procrastination Algorithm [13]

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: $timer \leftarrow Z_i$; {Initialize timer}
5: **else**
6: $timer \leftarrow min(timer, Z_i)$;
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}

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:*

$$\overset{\forall i}{i = 1, \ldots, n} \quad \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \tag{5}$$

$$\forall_{k<i} \; Z_k \leq Z_i \tag{6}$$

The details of the proof are given in the technical report [26].

4 Processor 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,

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

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance. The major contributors of leakage are the subthreshold leakage and the reverse bias junction current. We use the power model and the technology parameters described by Martin *et al.* [21].

The subthreshold current I_{subn}, as a function of the supply voltage V_{dd} and the body bias voltage V_{bs} is given below :

$$I_{subn} = K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} \tag{8}$$

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

$$P_{DC} = V_{dd} I_{subn} + |V_{bs}| I_j \tag{9}$$

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,

$$t_{inv} = \frac{L_d K_6}{(V_{dd} - V_{th})^{\alpha}} \tag{10}$$

The technology constants for the $70nm$ technology given in [21] are used. The value for C_{eff} based on the Transmeta Crusoe processor, scaled to $70nm$ are used. 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 1.

In addition to the gate level leakage, there is an inherent cost in keeping the processor on. 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 $P_{on} = 0.1W$.

We define the *processor critical speed* as the operating point that minimizes the energy consumption per cycle. Figure 2 shows the energy consumption per cycle for the $70nm$ technology. It is seen From the power model that the processor critical speed is at $V_{dd} = 0.70V$. From the voltage frequency relation described in Equation 10, $V_{dd} = 0.7V$ 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 $\eta_{crit} = 1.26/3.1 = 0.41$.

4.1 Shutdown Overhead

In previous work, 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. When switched to the deepest sleep mode, the processor loses its register and cache contents. Thus the cost of shutdown and wakeup includes additional costs as follows : (1) inherent energy delay cost of processor wakeup (2) saving registers to main memory (3) flushing dirty data cache lines to main memory (4) extra misses on a cold start (empty structures) in components such as data / instruction caches, translation look aside buffers (TLBs) and branch target buffers (BTBs), which are empty on wakeup. This results in extra memory accesses and thereby an additional energy overhead.

We estimate the energy overhead due to shutdown which we use in our simulations. Typical embedded processors such as the Intel PXA family processors have typically cache sizes of 32KB. Assuming 20%

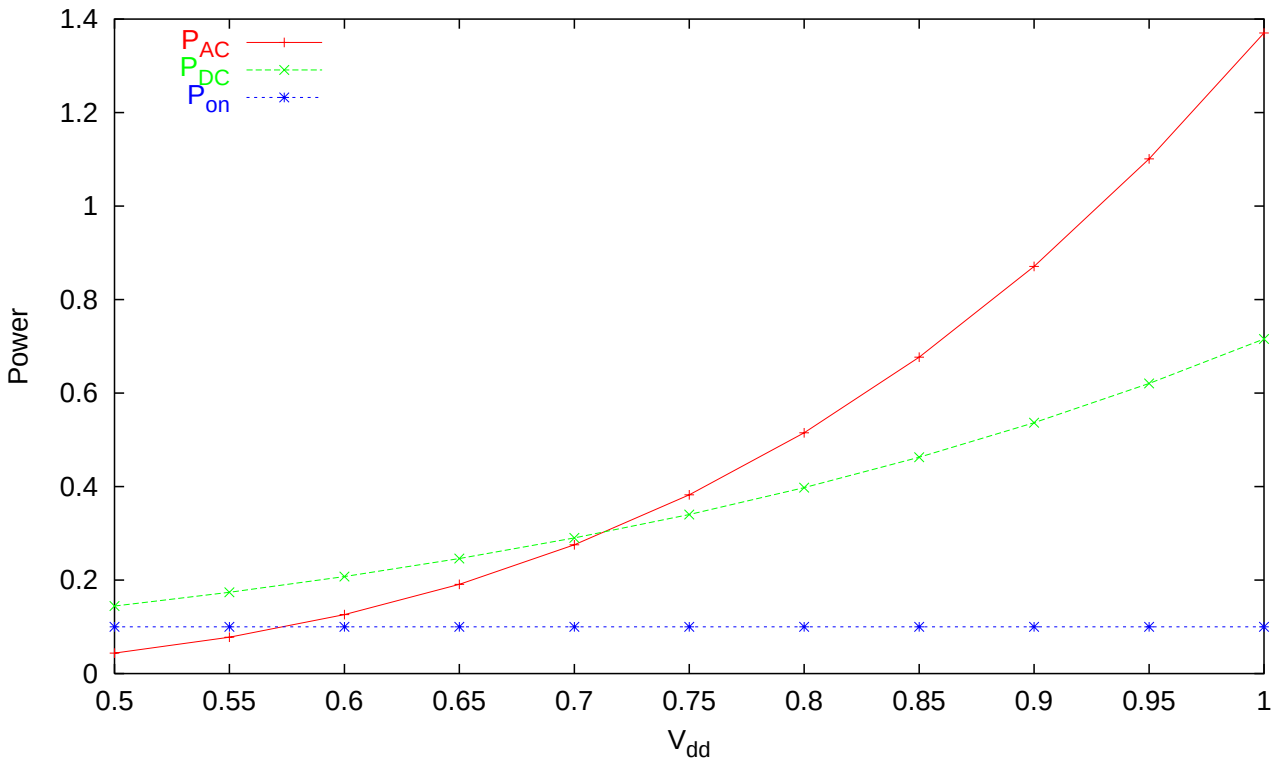

Figure 1. Power consumption of $70nm$ technology for Crusoe processor: P_{DC} is the leakage power, P_{AC} is the dynamic power and P_{on} is the intrinsic power consumption in on state

lines of the data cache to be dirty before shutdown results in 6554 memory writes. With an energy cost of 13nJ [15] 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 which 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 [15] 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, we have to make a decision whether to shutdown or not. 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. Since the idle power consumption in our power model 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.

5 Experimental Setup

We implemented the different scheduling techniques using a discrete event simulator. To evaluate the effectiveness of our scheduling techniques, we consider several task sets, each containing up to

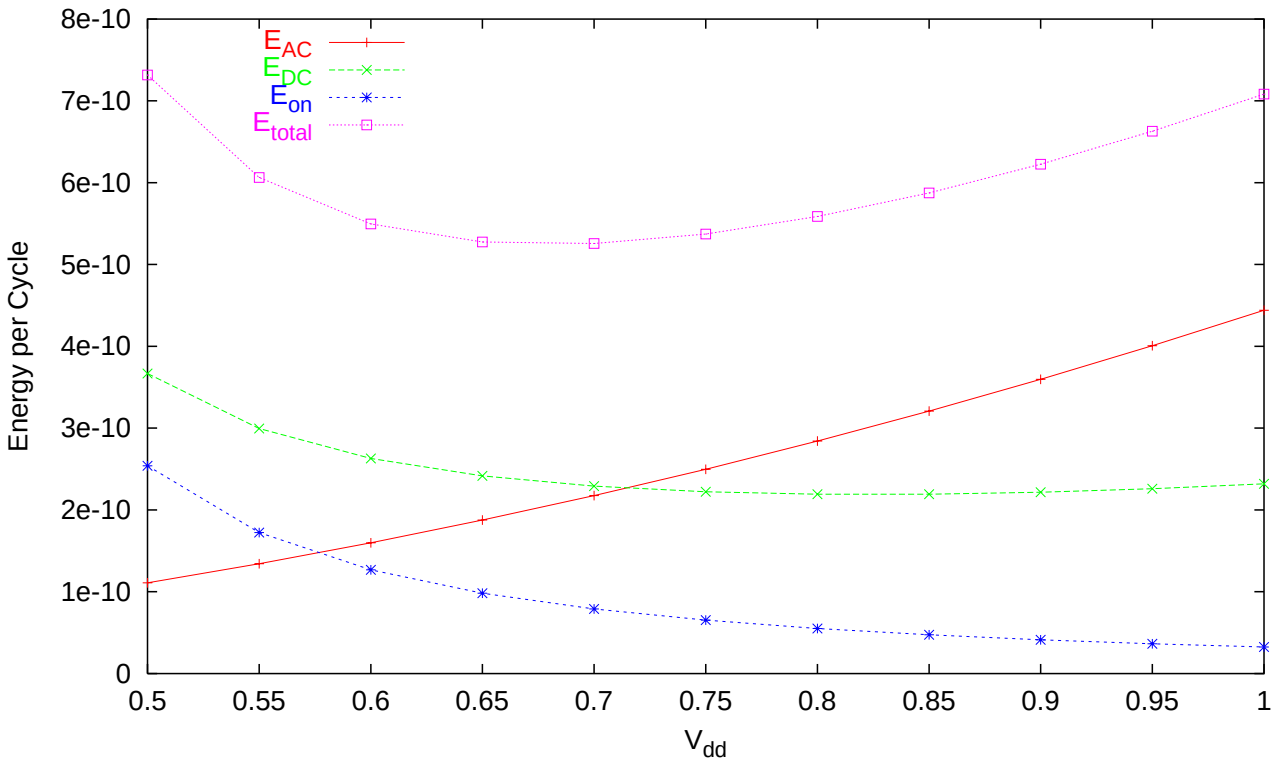

Figure 2. Energy per Cycle for $70nm$ technology for the Crusoe processor: E_{AC} is the switching energy, E_{DC} is the leakage energy and E_{on} is the intrinsic energy to keep the processor on.

20 randomly generated tasks. We note that such randomly generated tasks are a common validation methodology in previous works [16, 24, 9]. Based on real life task sets [19], tasks are assigned a random period in the range [10 ms,120 ms]. An initial utilization u_i of each task is uniformly assigned in the range [0.05, 0.5]. The Worst Case Execution Times (WCET) for each task is set to $u_i \cdot T_i$ at the maximum processor speed. The execution time of each task is scaled to ensure a processor utilization less than one, thereby making the task set feasible.

The tasks are scheduled on a single processor system. In addition to the processor, the system has three resources with standby power consumption of $0.2W$, $0.4W$ and $1.0W$. These are typical standby power consumption for memory, flash drives and 802.11 wireless interfaces and represent these resources. The typical standby time for the resources as a percentage of the task execution time is assumed to be in the range [20,60], [10,25] and [5,20] respectively. While the usage of network interfaces widely vary based on the applications, we assume conservative standby time. Note that our techniques will result in increased gains with larger resource standby intervals. Each task is assumed to use minimum one (memory) and maximum all resources and the standby time is uniformly assigned in the corresponding ranges. The wireless interface (1.0W) is assigned to a task only if the task uses all resources.

All tasks are assumed to execute up to their WCET. Experiments were performed on various task sets and the average results are presented. We use the processor power model described in Section 4 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.
- Critical Speed DVS (CS-DVS): where task slowdown factors are computed based on the algorithm presented in Section 3.
- Critical Speed DVS with Procrastination (CS-DVS-P): This is the Critical Speed DVS (CS-DVS) slowdown with the procrastination technique under EDF scheduling policy.

5.1 Energy Consumption

Figure 3 shows the energy consumption of the techniques normalized to no-DVS scheme. We refer the processor utilization at maximum speed as U and is shown along the X-axis with the energy consumption along the Y-axis. With the resource standby time in the specified range, the resources consume around 10% of the total energy in our experimental setup. Traditional DVS scheme does not consider the resource standby time and no-DVS and DVS schemes have similar energy consumption at higher values of U (80% to 100%). With the processor consuming the majority of the energy, DVS leads to energy gains at U drops below 80%. At lower utilization however, traditional DVS scheme results in increased processor leakage as well as longer resource standby time and consumes more energy. As U drops below 40%, the energy consumed by DVS increases and even surpasses no-DVS at very low values of U. CS-DVS computes task slowdown factors considering the resource standby power consumption and saves at least 5% to 10% over the traditional DVS. The CS-DVS technique executes each task no slower than its critical speed and shuts down the system to minimize energy consumption. However if the idle intervals are not sufficient to shutdown, significant energy savings cannot be achieved (over DVS) as seen at utilization of 30% and 40%. It is seen that the procrastination scheme results in more energy saving from this point. As the utilization lowers, executing tasks by the CS-DVS scheme results in idle intervals in the system and the idle energy consumption contributes to a significant portion of the total energy. The procrastination scheme (CS-DVS-P) clusters task executions thereby increasing the sleep intervals and achieves more energy savings. CS-DVS-P minimizes the idle energy consumption to result on an average 15% energy savings over the CS-DVS scheme.

6 Conclusions and Future Work

In this paper, we have presented a scheduling algorithm that consider the contributions of processor leakage and resource standby energy to minimize the total energy consumption in a system. We show that executing at the maximum or minimum processor speed need not be the optimal operating point. Detailed power models of the resources are important in computing the optimal operating point. Incorporating the resource usage patterns and their power models is increasingly important as systems are getting diverse with more resources contributing to the total energy consumption. Our experimental results show that computing the critical execution speeds for tasks results on an average 10% energy savings. The procrastination scheme increases the average energy savings to 15% by extending the sleep intervals thereby controlling leakage power consumption. Such a scheduling framework will result in an energy efficient operation of the system while meeting all timing requirements. We plan to extend these techniques to scheduling multiple resources with DVS capability and their effects on system wide DPM policies.

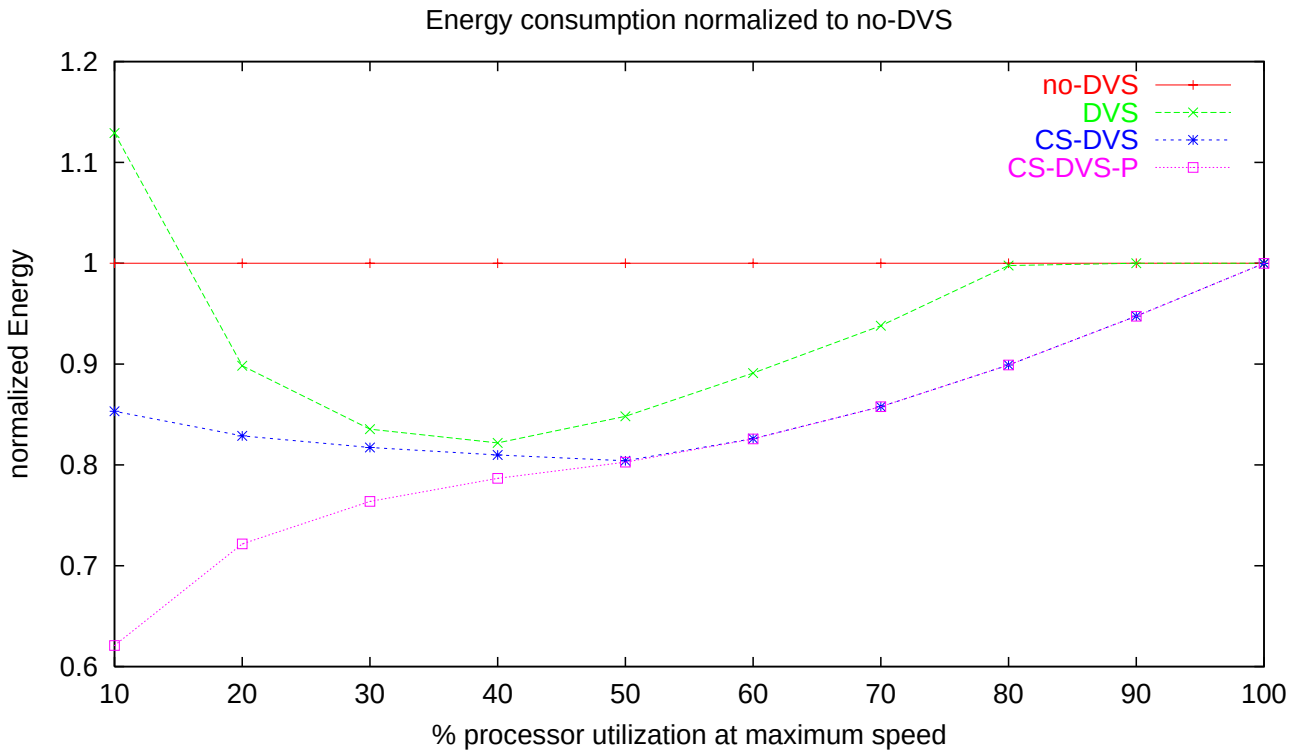

Figure 3. Energy consumption normalized to no-DVS

References

[1] Memtech SSD Corporation. *http://www.memtech.com*.

[2] Micron Technology, Inc. *http://www.micron.com*.

[3] Rambus Inc. *http://www.rambus.com*.

[4] Atheros Communications. Power consumption and energy efficiency comparisons of wlan products. In *Atheros White Papers (http://www.atheros.com/pt/papers.html)*, May 2003.

[5] 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*, Jun. 2001.

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

[7] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. Irwin. Hardware and software techniques for controlling dram power modes. *IEEE Transactions on Computers*, 50(11):1154–1173, 2001.

[8] 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.

[9] P. Gai, G. Lipari, and M. di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of IEEE Real-Time Systems Symposium*, Dec. 2001.

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

[11] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *Trans. on Embedded Computing Sys.*, 2(3):325–346, 2003.

[12] R. Jejurikar and R. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of Language Compilers and Tools for Embedded Systems*, Jun. 2004.

[13] 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.

[14] 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.

[15] 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.

[16] 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*, Jun. 2003.

[17] J. Liu, P. H. Chou, and N. Bagherzadeh. Communication speed selection for embedded systems with networked voltage-scalable processors. In *Proceedings pf International Symposium on Hardware/Software Codesign*, Nov. 2002.

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

[19] 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.

[20] J. Luo, N. Jha, and L. S. Peh. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Proceedings of Design Automation and Test in Europe*, Mar. 2003.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19, 2000.

[26] Skipped for Blind Review. Apr. 2004.

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

[28] W. Zhang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and V. De. Compiler support for reducing leakage energy consumption. In *Proceedings of Design Automation and Test in Europe*, Mar. 2003.