2.3p

Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems

Ravindra Jejurikar
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697
jezz@cecs.uci.edu

Rajesh Gupta
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
gupta@cs.ucsd.edu

ABSTRACT

Traditionally, dynamic voltage scaling (DVS) techniques have focused on minimizing the processor energy consumption as opposed to the entire system energy consumption. The slowdown resulting from DVS can increase the energy consumption of components like memory and network interfaces. Furthermore, the leakage power consumption is increasing with the scaling device technology and must also be taken into account. In this work, we consider energy efficient slowdown in a real-time task system. 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. Our simulation experiments using randomly generated task sets show on an average 10% energy gains over traditional dynamic voltage scaling. We further combine slowdown with procrastination scheduling which increases the average energy savings to 15%. We show that our scheduling approach minimizes the total static and dynamic energy consumption of the systemwide resources.

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

General Terms: Algorithms.

Keywords: low power, DVS, resource standby energy, critical speed, real-time systems, EDF scheduling, procrastication.

1. INTRODUCTION

System-level power management is important for reliability, packaging costs as well as to extend the limited battery life of portable embedded systems. The two primary ways to reduce power consumption in computing systems are: (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 [6, 9, 20]. DVS is supported by recent processors and known to be more effective than DPM in reducing the processor energy consumption [5, 19]. 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. Thus longer execution time implies more leakage energy.
- 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.
- A minimum power consumption is associated with keeping the processor active. Some of the major contributors are the PLL circuitry, which drives up to $200mA$ current [8] and the I/O and analog components of the processor. Note that the power consumption of these components do not scale with DVS and longer execution time can increase the total energy consumption.

Components such as memory banks, flash drives, co-processors (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). Especially with compiler assisted DPM techniques [6], standby time increases proportionally to the task execution time. Thus DVS techniques need to consider the standby power consumption of the peripherals devices in the computation of slowdown factors to reduce the total energy consumption of the system.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

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, 19], whereas later works have focussed on leakage to minimize the total static and dynamic power consumption [17, 21, 12, 11]. Slowdown tradeoffs in the computation and communication subsystems are considered in [14, 16]. Recent works have also considered the combined processor and memory energy consumption. Fan *et. al.* [7] consider memory power consumption to show that excess slowdown can increase the total energy consumption.

While most of the work on DVS is focussed on minimizing the processor energy consumption, the resource standby energy is usually ignored. It is observed that devices like memory banks are in the active state $30\% - 90\%$ of the task execution time [6]. 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 consumption. Furthermore, we combine task slowdown with procrastination scheduling proposed in our earlier work [11, 10]. Procrastination is known to reduce the energy consumption by minimizing the number of processor on/off transitions as well as extending the sleep intervals within the performance requirements of the system.

The rest of the paper is organized as follows. In Section 2, we introduce the system model. Section 3 formulates the problem followed by an algorithm to compute task slowdown factors. The experimental results are discussed in Section 4 and Section 5 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 to be equal to the period ($D_i = T_i$) and the tasks are scheduled by the Earliest Deadline First (EDF) scheduling policy [15]. All tasks are assumed to be independent and preemptive. The tasks are 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.

Recent processors support variable voltage and frequency levels for energy efficient operation of the system. Let the available frequencies be $\{f_1, ..., f_s\}$ in increasing order of frequency and the corresponding voltage levels be $\{v_1, ..., v_s\}$. 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 $(\frac{f_1}{f_s}, \frac{f_2}{f_s}, ..., 1)$ in the range [0,1]. The slowdown factor assigned to task τ_i is represented by 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 slowdown factor 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, 19]. 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 be in the shutdown state (no leakage) or in the standby state (active + idle) where leakage power is dissipated.

In addition to the processor, the system has a set of m resources $\mathcal{R} = \{R_1, ..., R_m\}$ 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 $\mathcal{R}$, represented by $\mathcal{R}^{\tau_i}$. 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 policies. Though the standby time can potentially vary with slowdown under OS directed DPM policies, 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 state during the execution of τ_i. If a task does not use resource R_j, then $C_i^{R_j} = 0$. In this work, we consider task level slowdown factors as opposed to intra-task slowdown. We compute task slowdown factors that minimize the total system energy consumption including the resource standby energy contribution. Note that we are not proposing DPM policies, but considering the standby energy in computing static slowdown factors.

3. LOW POWER SCHEDULING

We want to compute task slowdown factors that minimize the energy consumption of the entire system. The total energy consumption when task τ_i is executed at a speed η is given by :

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

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. The number of executions of each task is inversely proportional to the task period and the optimization problem can be stated as:

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

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

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

Note that the slowest speed need not be the optimal slowdown factor when the contributions of the processor leakage power and the resource standby power are considered. The slowdown factor for a task that minimizes its total energy consumption, called the *critical speed* for the task, is important in energy minimization.

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 [18]. The algorithm consists of two phases : (1) computing the critical speed for each task; and (2) increasing the task slowdown factors if the task set is not feasible. We compute the energy consumption of each task at all possible discrete slowdown factors and the slowdown factor that minimizes the task energy is the critical speed. Due to different resource usages of task, the critical speed can differ with each task. If the task-set is infeasible, the second step increases the execution speed of tasks to achieve feasibility. A heuristic to select a task whose speed is increased is described next. 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 [18] to increase the task slowdown factor. The pseudo-code is presented in Algorithm 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;
```

4. EXPERIMENTAL SETUP

We implemented the different 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 are a common validation methodology in previous works [13, 5]. Based on real life task sets, 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 time (WCET) for each task at the maximum processor speed is set to $u_i \cdot T_i$. The execution time of each task is scaled to ensure a processor utilization less than one, thereby making the task set feasible. All tasks are assumed to execute up to their WCET.

The tasks are scheduled on a single processor system. We use the processor power model presented in our earlier work [11], which captures both dynamic and static power consumption. The processor is assumed to support discrete voltage levels in steps of $0.05V$ in the range $0.5V$ to $1.0V$. From the power model, it is seen that the operating point that minimizes the processor energy is at $V_{dd} = 0.7V$, which corresponds to a slowdown factor of $\eta = 0.41$. The idle power consumption is $0.24W$ and the energy cost of wakeup is assumed to be $483\mu J$, as discussed in [11]. This makes shutdown energy efficient only when the idle interval is greater than

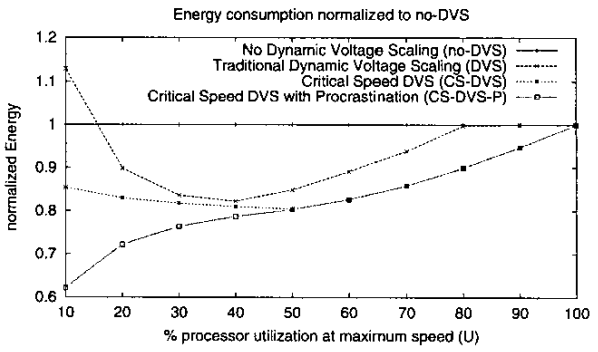

Figure 1: Energy consumption normalized to no-DVS

$2.01ms$, which is the threshold idle interval $t_{threshold}$ for shutdown. 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 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 standby power) is assigned to a task only if the task uses all resources.

Experiments were performed on various task sets and the average results are presented. We 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 by Algorithm 1, presented in Section 3.
- Critical Speed DVS with Procrastination (CS-DVS-P): This is the Critical Speed DVS (CS-DVS) slowdown along with the procrastination scheduling policy described in [11].

Under scheduling with no procrastination, the processor is shutdown if the processor is idle and the next task arrival is later than the threshold idle interval ($t_{threshold}$). With procrastination, the upcoming idle time and the minimum procrastination interval are considered to make a shutdown decision. Thus procrastination enables longer shutdown intervals and reduces the leakage energy consumption.

Figure 1 shows the energy consumption of the techniques normalized to no-DVS scheme. The processor utilization at maximum speed, U, is shown along the X-axis with the normalized 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. On the other hand, CS-DVS computes task slowdown factors considering the resource standby power consumption and saves on an average 10% energy over 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 a utilization of 30% and 40%. We see 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 shutdown overhead 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 DVS scheme.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a task slowdown algorithm that considers the contributions of resource standby energy as well as processor leakage to minimize the total energy consumption in a system. We show that detailed power models of the resources are important in computing energy efficient operating points. Incorporating the resource usage patterns and their power models is increasingly important as systems become 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 energy consumption. Such a scheduling framework which has the view of the entire system results in an energy efficient operation 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.

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.

6. 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] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.