

# Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation

Hui Cheng and Steve Goddard  
Department of Computer Science and Engineering  
University of Nebraska — Lincoln  
Lincoln, NE 68588-0115  
{hcheng, goddard}@cse.unl.edu

## Abstract

*The challenge in conserving energy in embedded real-time systems is to reduce power consumption while preserving temporal correctness. Previous research has focused on power conservation for either the processor or I/O devices alone. The system-wide energy conservation has received little attention. In this paper, we analyze the problem of system-wide energy-efficient scheduling for hard real-time systems based on the preemptive periodic task model with non-preemptive shared resources. We propose an online system-wide energy-efficient scheduling algorithm System-wide Energy-Aware EDF (SYS-EDF), which integrates Dynamic Power Management (DPM) for I/O devices and Dynamic Voltage Scaling (DVS) for the processor. An evaluation of SYS-EDF shows that it yields significant energy savings with respect to DVS alone or DPM alone techniques.*

## 1 Introduction

Embedded real-time systems often consist of a battery-operated microprocessor system with Input/Output (I/O) devices and a limited battery life. Energy conservation techniques are thus needed to extend their lifetimes. The need to prolong system lifetime has resulted in much work done in energy-efficient task scheduling for real-time systems.

In the last decade, much work has been done on processor-based power management techniques. Dynamic Voltage Scaling (DVS) is one of the most popular techniques to reduce the processor energy consumption. DVS-based real-time scheduling algorithms can effectively reduce the processor energy consumption by lowering the processor speed, while still guarantee that all jobs meet their deadlines. However, DVS-based algorithms reduce the dynamic power consumption of the processor at the cost of increased execution time, which in turn increases the I/O device standby energy consumption. It has been observed [4, 11] that aggressively lowering the processor speed may increase the overall

system energy consumption rather than decreasing it.

The energy consumption of I/O devices can be reduced by shutting down devices under certain conditions. This method is commonly known as Dynamic Power Management (DPM). There have been some efforts [8, 9] in developing energy-efficient device scheduling algorithms that minimize the I/O device energy consumption for real-time systems. However, none of them considered the energy consumption of processors. As with the DVS alone scheduling algorithms, DPM alone cannot guarantee the overall system energy consumption is minimized.

In this paper, we analyze the problem of system-wide energy conservation for hard real-time systems based on the preemptive periodic task model with non-preemptive shared resources. Here we define the system-wide energy consumption as the sum of the processor energy consumption and the I/O device (including memory<sup>1</sup>) energy consumption. We propose an online system-wide energy-efficient scheduling algorithm, System-wide Energy-Aware EDF (SYS-EDF), which integrates device scheduling and processor voltage scaling to reduce the overall system energy consumption.

The rest of this paper is organized as follows. Section 2 discusses related work. The problem of energy-aware I/O device scheduling is analyzed in Section 3. Section 4 describes the SYS-EDF algorithm. Section 5 describes how we evaluated our system and presents the results. Section 6 presents our conclusions and describes future work.

## 2 Related Work

Compared to the research of processor-based energy conservation techniques or I/O-based energy conservation techniques, the research on system-wide energy conservation has received little attention. Only a few papers [4, 11] address this issue. In these papers, the negative effect of lowering processor speed is considered. Optimal slowdown factors of

---

<sup>1</sup>Some modern DRAM chips can be put in a *power down* state in which only the self-refresh circuitry is active to prevent data loss.

the processor speed to minimize the overall system energy consumption are computed and used as the lower-bound of the processor speed. They both achieve significant energy savings compared to DVS alone algorithms. Our work differs from the previous work in following aspects:

1. Our work supports periodic task sets with non-preemptive shared resources. In the previous work, all tasks were assumed to be fully preemptive. In practice, non-preemptive shared resources are pervasive in real-world applications. For example, a job that performs an uninterruptible I/O operation can block the execution of all jobs with higher priorities. Thus the time for the uninterruptible I/O operation needs to be treated as a non-preemptive resource access. Other resources besides I/O devices include critical sections of code, reader/writer buffers, etc.
2. Our work considers the problem of energy-efficient device scheduling and proposes a device scheduling algorithm, *i.e.*, Conservative Energy-Aware EDF (CEA-EDF). [4] and [11] made simplified assumption for the device scheduling. For example, [4] assumed that there is no delay for device state transition. Therefore, an aggressive device scheduling algorithm which turns off devices whenever they are not in use was implied in this work. However, this aggressive device scheduling is not applicable to hard real-time systems if devices that have non-zero transition delays are used. Similarly, [11] did not propose DPM for I/O devices.

The method proposed in this paper provides a energy-efficient device scheduling algorithm, CEA-EDF, for periodic task sets with non-preemptive shared resources. The optimal processor speed is then analyzed based on the proposed device scheduling algorithm. Finally, the SYS-EDF algorithm is proposed to reduce the overall system energy consumption by integrating CEA-EDF and the processor voltage scaling. To the best of our knowledge, no previous publication has addressed the same problem.

### 3. Energy-aware device scheduling

I/O devices usually have fewer power states than processors. Throughout this paper, we assume that a device has two states: *active* and *idle*. In a real-time system, in order to guarantee that jobs will meet their deadlines, a device cannot be made idle without knowing when it will be requested by a job, but, the precise time at which an application requests the operating system for a device is usually not known. Even without knowing the exact time at which requests are made, we can safely assume that devices are requested within the time of execution of the job making the request. Therefore, our method is based on inter-task device scheduling rather than intra-task scheduling. That is, the scheduler does not put devices in sleep while tasks that require them are being executed, even though there is no I/O requests at that time.

As discussed before, the energy-aware device scheduling algorithm needs to support the preemptive scheduling of periodic tasks with non-preemptive shared resources. However, the only known published energy-aware device scheduling algorithm for preemptive schedules, Maximum Device Overlap (MDO) [9], does not address the issue of resource blocking. As an offline method, it is difficult to integrate a resource accessing policy into MDO because it is hard to predict exact points that jobs access resources at the offline phase. It is possible that a seemingly feasible offline job schedule causes jobs to miss their deadlines at runtime.

An obvious online approach is to aggressively shut down devices whenever they are not needed; and start them as soon as they are needed, which is called the Aggressive Shut Down (ASD) algorithm [2]. Unfortunately, ASD cannot be directly applied to hard real-time systems, because the power consumption and the delay of the device state transition is usually too large to be neglected.

In our previous study [2], some online device scheduling algorithms that support preemptive schedules with shared resources are proposed. Among them, CEA-EDF can be used together with a DVS-based scheduler without any modification. As we will see shortly, CEA-EDF is independent of processor speed change, which makes it ideal for easy integration with DVS.

#### 3.1 Device energy model

Associated with each device  $\lambda_i$  are the following parameters: the transition time from the *idle* state to the *active* state represented by  $t_{wu}(\lambda_i)$ ; the transition time from the *active* state to the *idle* state represented by  $t_{sd}(\lambda_i)$ ; the energy consumed per unit time in the *active* and *idle* states represented by  $P_a(\lambda_i)$  and  $P_i(\lambda_i)$  respectively; the energy consumed per transition from the *active* state to the *idle* state represented by  $E_{sd}(\lambda_i)$ ; and the energy consumed per transition from the *idle* state to the *active* state represented by  $E_{wu}(\lambda_i)$ . We assume that for any device, the state switch can only be performed when the device is in a stable state, *i.e.*, the idle state or the active state. Therefore, the total energy consumed by a device  $\lambda_i$  is given by,

$$E_{\lambda_i} = P_a \times T_a(\lambda_i) + P_i \times T_i + E_{sd} \times N_{sd} + E_{wu} \times N_{sw} \quad (1)$$

where,  $T_a$  is the time that  $\lambda_i$  is in the *active* state;  $T_i$  is the time that  $\lambda_i$  is in the *idle* state;  $N_{sd}$  is the number of the transition of the device from active to idle; and  $N_{wu}$  is the number of the transition of the device from idle to active.

#### 3.2. Energy-aware device scheduling

CEA-EDF is a simple, online energy-aware device scheduling algorithm for hard real-time systems. All devices that a job needs are active at or before the job is released. Thus devices are safely shut down without affecting the schedulability of tasks.

```

1  Device scheduling at time  $t$ :
2  If ( $t$ : instance when job  $J_{i,j}$  is completed)
3    If ( $\exists \lambda_k, \lambda_k = \text{active}$  and  $T_{req}(\lambda_k, t) - t > BE(\lambda_k)$ )
4       $\lambda_k \rightarrow \text{idle}$ ;
5       $Up(\lambda_k) \leftarrow T_{req}(\lambda_k, t) - t_{wu}(\lambda_k)$ ;
6    End
7  End
8  If ( $t: \exists \lambda_k, \lambda_k = \text{idle}$  and  $Up(\lambda_k) = t$ )
9     $\lambda_k \rightarrow \text{active}$ ;
10    $Up(\lambda_k) \leftarrow -1$ ; // Clear the power up timer for  $\lambda_k$ 
11  End

```

**Figure 1. The CEA-EDF algorithm.**  $Up(\lambda_k)$  is the power up time set to  $\lambda_k$ , at when the device will be powered up.

Because of the energy consumption associated with the device power state transition, it is not energy-efficient to frequently perform the power state transition. A *break-even time* is used to represent the minimum inactivity time required to compensate for the cost of entering and exiting the idle state. We let  $BE(\lambda_i)$  denote the break-even time of device  $\lambda_i$  hereafter. The computation of  $BE(\lambda_i)$  using our device energy model can be found in [2]. It is clear that if a device is idle for less than the break-even time, it is not worth performing the state switch. Therefore, CEA-EDF makes decisions of device state transition based on the break-even time rather than device state transition delay.

Next, we define the *next device request time* that is used in keeping track of the earliest time that a device is required.

**Definition 3.1. Next Device Request Time.** The next device request time is denoted by  $T_{req}(\lambda_k, t)$  and is the earliest time that a device  $\lambda_k$  is requested by any uncompleted job. Since a job can only use a device after the job is released, the next device request time of a device  $\lambda_k$  is given by

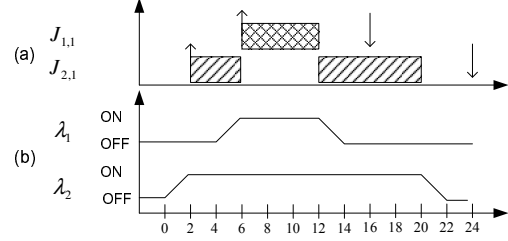
$$T_{req}(\lambda_k, t) = \text{Min}(R(J_{i,j})) \quad (2)$$

where  $J_{i,j}$  is any uncompleted job that requires device  $\lambda_k$  and  $R(J_{i,j})$  is the release time of job  $J_{i,j}$ .

With CEA-EDF, a device  $\lambda_i$  is switched to the low power state at time  $t$  when  $T_{req}(\lambda_i, t) - t > BE(\lambda_i)$ . CEA-EDF sets a power up time,  $Up(\lambda_i)$ , for device  $\lambda_i$  when  $\lambda_i$  is switched to the idle state. For any idle device, it is switched back to the active state if the power up time  $Up(\lambda_i)$  is equal to the current time  $t$ . The CEA-EDF scheduling algorithm then can be described as in Figure 1, and is invoked at *scheduling points* and when a power up time is reached. We define scheduling points as time instances at which jobs are released, completed, or exit critical sections. An example of CEA-EDF scheduling is illustrated in Figure 2.

#### 4. System-wide energy-efficient scheduling

In this section, we first provide a power model for a typical DVS processor. Then we present a system-wide energy-



**Figure 2. CEA-EDF scheduling example;** (a)  $J_{1,1}$  is released at 6 and uses device  $\lambda_1$ .  $J_{2,1}$  is released at 2 and uses device  $\lambda_2$ .  $J_{1,1}$  has a higher priority than  $J_{2,1}$ . (b) the device state transition with the CEA-EDF algorithm.

efficient task scheduling algorithm, SYS-EDF, which integrates CEA-EDF and processor voltage scaling.

#### 4.1 DVS processor energy model

In a CMOS circuit, the overall power consumption consists of dynamic power consumption and static power consumption. For a DVS processor, the dynamic power consumption can be given by,

$$P_{AC} = C_{eff} V_{dd}^2 f \quad (3)$$

where  $C_{eff}$  is the switched capacitance,  $V_{dd}$  is the supply voltage and  $f$  is the operating frequency. The relationship of  $f$  and  $V_{dd}$  is given by [6]

$$f = (L_d K_6)^{-1} ((1 + K_1) V_{dd} + K_2 V_{bs} - V_{th1})^\alpha \quad (4)$$

where  $V_{bs}$  is the body bias voltage and  $K_1, K_2, K_6, L_d, V_{th1}$  and  $\alpha$  are technology constant parameters.

Several leakage sources contribute to the total static power consumption. According to [6], the leakage power dissipation is given by,

$$P_{DC} = L_g (V_{dd} I_{subn} + |V_{bs}| I_j) \quad (5)$$

where  $L_g$  is the number of devices in the circuit,  $I_{subn}$  is the subthreshold current, and  $I_j$  is the reverse bias junction current. The formal mathematical formulation and detailed explanations of related technical parameters can be found in [6]. The total power consumption of a processor is given by,

$$P_{cpu} = \begin{cases} P_{AC} + P_{DC} + P_{on} & \text{CPU is active} \\ 0 & \text{CPU is not active} \end{cases} \quad (6)$$

where  $P_{on}$  is an inherent power cost in keeping the processor on [3]. We assume that a processor does not consume energy when it is not in the active state.

Since the voltage transition delay of a processor is very short, we assume that the overhead incurred in changing the processor speed is negligible. The same assumption is made in previous works [7, 10, 11].

## 4.2. System-wide optimal processor speed

We let  $\nu$  denote the normalized processor speed. That is, the ratio of the current processor speed to the maximal processor speed. As with previous work [7, 10, 11], we assume that the processor speed is approximately proportional to the current operating frequency  $f$ . Thus  $\nu$  can be represented by  $f/f_{high}$ , where  $f_{high}$  is the maximum operating frequency. We assume that a DVS processor can provide  $m$  discrete operating frequency represented by  $\{f_1, f_2, \dots, f_m = f_{high}\}$ .

Because of the standby energy dissipation of I/O devices, the lowest processor speed is not necessarily the most energy-efficient speed as assumed in previous DVS-alone scheduling algorithms. The leakage power dissipation of the processor and the standby energy dissipation of I/O devices increase with the extended task lifetime. Let  $\Lambda(t)$  be the active device set that contains all devices that are in the active state at time  $t$ . Note that with the CEA-EDF device scheduling algorithm, devices not required by the current executing job may be kept in the active state to ensure the system schedulability. As shown in Figure 2, all devices in  $\Lambda(t)$  are kept in the active state until the current job is completed.

Suppose that a processor can complete 1 unit workload in 1 unit time with the highest operating frequency, then the processor will take  $1/\nu$  time units to complete 1 unit workload with a processor speed of  $\nu$ . Next, we introduce *energy efficiency scale* to compare the overall system energy efficiency to complete 1 unit workload with different processor speeds. The energy efficiency scale is denoted by  $ES(\nu, \Lambda)$  and is modelled by,

$$ES(\nu, \Lambda(t)) = \frac{P_{cpu}(\nu)}{\nu} + \frac{1}{\nu} \sum_{\lambda_k \in \Lambda(t)} (P_a(\lambda_k) - P_i(\lambda_k)) \quad (7)$$

where  $P_{cpu}(\nu)$  is the processor energy consumption rate with a given processor speed  $\nu$ , which can be acquired from the processor energy model presented in Section 4.1.  $P_a(\lambda_k) - P_i(\lambda_k)$  is the difference of the energy consumption rate of device  $\lambda_k$  in the active state and the idle state. We use this difference rather than  $P_a(\lambda_k)$  alone to evaluate how much energy can be saved by putting  $\lambda_k$  in the idle state.

The processor speed that can minimize the energy efficiency scale is the system-wide optimal processor speed. We let  $\nu_{opt}(\Lambda(t))$  denote the optimal processor speed for a given active device set  $\Lambda(t)$ . Figure 3 shows the energy efficiency scale for three different active device sets. The CPU is based on Transmeta Crusoe processor with 70nm technology<sup>2</sup> [3, 6]. The technical parameters for these devices can be found in Table 1. It can be seen from Figure 3 that the energy efficiency scale varies with different active device sets, and so does the optimal processor speed. For example, the most energy efficient processor speed is 0.4 when only the Mobile RAM is in the active state, while the optimal processor speed is 0.9 when both the Mobile RAM and the

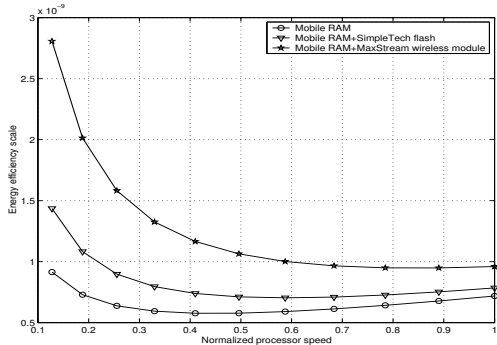


Figure 3.  $ES(\nu, \Lambda(t))$  of different normalized processor speeds for three active device sets.

MaxStream wireless module are in the active state.

The system-wide optimal processor speed  $\nu_{opt}$  is computed offline and retrieved at runtime. For a given active device set  $\Lambda(t)$ ,  $\nu_{opt}(\Lambda(t))$  can be acquired by computing  $ES(\nu, \Lambda(t))$  for all possible  $\nu$  values. The speed that minimizes  $ES(\nu, \Lambda(t))$  is selected to be  $\nu_{opt}(\Lambda(t))$ . Since modern DVS processors provide finite discrete operating frequencies, this computation can be done in  $O(m)$  time complexity for each given  $\Lambda(t)$ , where  $m$  is the number of operating frequencies that the processor can provide. Let  $K$  denote the number of devices in the system, then there are at most  $2^K$  possible sets for  $\Lambda(t)$ . Therefore, the computational complexity of computing  $\nu_{opt}$  for all possible  $\Lambda(t)$  is  $O(m \times 2^K)$ . With all pre-computed  $\nu_{opt}(\Lambda(t))$  stored in memory, retrieving  $\nu_{opt}$  for any  $\Lambda(t)$  at runtime can be done in  $O(1)$  time.

## 4.3. SYS-EDF

The processor voltage scaling in SYS-EDF is based on the Dual Speed (DS) and the Dual Speed Dynamic Reclaiming (DSDR) algorithms proposed by Zhang *et al.*, [10]. The DS algorithm aims to minimize the dynamic energy consumption of the processor for real-time periodic tasks with non-preemptive blocking sections. The DSDR algorithm extends the DS algorithm by dynamically collecting unused run time for further slow down.

However, DS and DSDR considered only the dynamic energy dissipation of the processor. Based on the previous analysis, we develop the SYS-EDF algorithm, which improves DS and DSDR to reduce the overall system energy consumption. For the space limitation, we only discuss the basic improvement done to the DS algorithm in this paper. The basic idea is : the SYS-EDF algorithm keeps track of the active device set and computes the corresponding optimal processor speed. SYS-EDF uses the DS algorithm to

<sup>2</sup>This is a processor model based on the technology trends [6].

```

1 Initialize:
2  $\nu \leftarrow \max(L, \nu_{opt}(\Lambda(t))); \text{END\_H} \leftarrow -1;$ 
3  $H$  and  $L$  are pre-computed processor speeds [10];
4 Scheduling at time  $t$ :
5 If ( $t$ : instance when job  $J_{i,j}$  is completed)
6   update  $\Lambda(t)$ ;
7   If (there is no pending job)  $\nu \leftarrow 0$ ;
8   Else  $\nu \leftarrow \max(\nu, \nu_{opt}(\Lambda(t)))$ ;
9   End
10 End
11 If ( $t$ : instance when job  $J_{i,j}$  is released)
12   update  $\Lambda(t)$ ;
13    $\nu \leftarrow \max(\nu, \nu_{opt}(\Lambda(t)))$ ;
14   If ( $Prio(J_{i,j}) > Prio(J_{curr})$  and  $J_{i,j}$  is blocked by  $J_{curr}$ )
15      $\nu \leftarrow \max(H, \nu_{opt}(\Lambda(t)))$ ;
16      $\text{End\_H} \leftarrow \max(\text{End\_H}, \text{Deadline}(J_{curr}))$ ;
17   End
18 End
19 If ( $t$ : instance when  $t = \text{End\_H}$ )
20    $\text{End\_H} \leftarrow -1$ ;
21    $\nu \leftarrow \max(L, \nu_{opt}(\Lambda(t)))$ ;
22 End
23 Schedule devices by CEA-EDF ;
24 Schedule jobs by EDF(SRP);

```

**Figure 4. The simplified SYS-EDF algorithm.**

adjust the processor speed with only one limitation: the processor speed is never set below the optimal processor speed. The improvement to the DSDR algorithm follows a similar approach, but uses a different dynamic reclaiming algorithm because more than two processor speeds are utilized in SYS-EDF.

The SYS-EDF algorithm is presented in Figure 4. With the proposed device scheduling algorithm, *i.e.*, CEA-EDF, the active device set changes only at the time instances when a job is completed or a new job is released (line 6,12). As with [10], a pre-computed high speed  $H$  and a pre-computed low speed  $L$  are used in SYS-EDF. Because of the space limitation, we do not present the computation of  $H$  and  $L$  in this paper. We refer the reader to [10] for the detailed explanation and computation. Since  $H$ ,  $L$  and  $\nu_{opt}(\Lambda(t))$  are pre-computed, the overhead of performing SYS-EDF is very low.

#### 4.4. Schedulability

**Theorem 4.1.** *Suppose  $n$  periodic tasks are sorted by their periods. They are schedulable by SYS-EDF if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^k \frac{E(T_i)}{P(T_i)} + \frac{B(T_k)}{P(T_k)} \leq 1, \quad (8)$$

where  $E(T_k)$  and  $P(T_k)$  are the execution time and period of task  $T_k$  respectively; and  $B(T_k)$  is the maximal length that a job in  $T_k$  can be blocked.

**Proof:** The SYS-EDF algorithm consists of a energy-efficient device scheduling algorithm (CEA-EDF) and a processor voltage scaling algorithm. With the CEA-EDF algorithm, a device  $\lambda_k$  is guaranteed to be in the active state when

Device	$P_a$ (W)	$P_i$ (W)	$E_{wu}, E_{sd}$ (mJ) <sup>3</sup>
Realtek Ethernet Chip	0.187	0.085	1.25
MaxStream Wireless module	0.75	0.005	4
IBM Microdrive	1.3	0.1	6
Fujitsu MHL2300AT Hard disk	2.3	1.0	3
SimpleTech Flash Card	0.225	0.02	0.2
Mobile-RAM	0.075	0.00175	$\approx 0$

**Table 1. Device Specifications.**

any jobs requiring  $\lambda_k$  are released. Therefore, CEA-EDF does not affect the schedulability of any systems.

With the processor voltage scaling algorithm presented in Figure 4, the processor speed is set to the higher speed of the optimal processor speed and the speed when scheduled with the DS scheduling algorithm (line 2,8,15,21). In other words, the SYS-EDF algorithm keeps the processor at a speed no less than the speed when scheduled with the DS algorithm. Since Theorem 4.1 has been proved true for the DS scheduling algorithm in [10], Theorem 4.1 is also true for the SYS-EDF algorithm.  $\square$

## 5 Evaluation

This section presents evaluation results for the SYS-EDF algorithm. Section 5.1 describes the evaluation methodology used in this study. Section 5.2 describes the evaluation of SYS-EDF with various system utilizations.

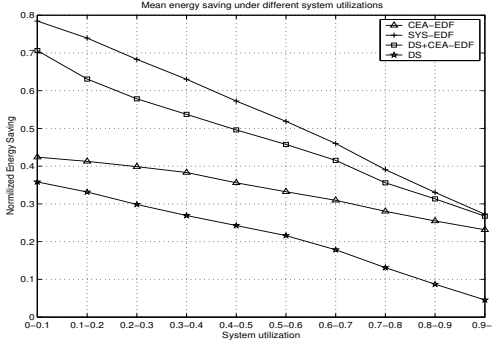
### 5.1. Methodology

We evaluated the SYS-EDF algorithm using an event-driven simulator. This approach is consistent with evaluation approaches adopted by other researches for energy-aware scheduling [8, 4, 11].

The power requirements and state switching times for devices were obtained from data sheets provided by the manufacturer. The devices used in experiments are listed in Table 1. The DVS processor we simulated is based on Transmeta Crusoe processor with 70nm technology [3, 6]. We assume that the processor supports discrete voltage from 0.5V to 1.0V in steps of 0.05V. The *normalized energy saving* is used to evaluate the energy savings of the algorithms. The normalized energy saving is the ratio of energy saving under a energy-conservation algorithm to the energy consumption when no energy-conservation technique is used, wherein all devices remain in the active state over the entire simulation.

In all experiments, we used randomly generated task sets to evaluate the performance of all algorithms. All task sets are pretested to satisfy the schedulability condition shown in Equation (8). Each generated task set contained 1 ~ 10 tasks. Periods of tasks are chosen from [100,1000]. Each

<sup>3</sup>Most vendors report only a single switching energy consumption. Thus we used this data for both  $E_{wu}$  and  $E_{sd}$ . The sources of these data can be found in [2].



**Figure 5. Mean normalized energy savings of different system utilization settings.**

task in a task set required the RAM module and additional 0 ~ 2 other devices from Table 1. Critical sections of all jobs were randomly generated. We repeated each experiment 500 times and present the mean value. During the whole experiment, we assume that the actual execution time of a task is equal to the WCET.

We did not measure scheduling overhead in a real system since all algorithms were evaluated with simulations. Instead, we compared the scheduling overhead of SYS-EDF with respect to EDF(SRP) in our simulations. We used *relative scheduling overhead* to evaluate the scheduling overhead of SYS-EDF. Let  $\rho$  denote the *relative scheduling overhead*, which is given by

$$\rho = \frac{\text{scheduling overhead with SYS-EDF}}{\text{scheduling overhead with EDF(SRP)}} - 1$$

The mean relative scheduling overhead of SYS-EDF is 3.2%, verifying that the overhead of SYS-EDF is low.

## 5.2. Average energy savings

To better evaluate the SYS-EDF algorithm, we compare SYS-EDF with three other algorithms for each simulation: (1) CEA-EDF is the algorithm that only performs DPM for devices; (2) DS is the DVS-alone algorithm proposed in [10], which considers only the dynamic energy conservation for processors; and (3) DS +CEA-EDF is the straightforward integration of (1) and (2), without considering the system-wide energy-efficient speed. Since [4] and [11] do not address the problem of resource blocking and the negative effect of device transition delays on system schedulability, we did not compare with them in this evaluation.

Figure 5 shows simulation results of the mean normalized energy saving for the SYS-EDF and other algorithms under different system utilizations. It can be seen that SYS-EDF saves more energy than the other algorithms. SYS-EDF can reduce the system energy consumption by up to 10% over DS +CEA-EDF. In most cases, as the system utilization increases, the normalized energy savings decreases. The ra-

tionale for this is that as tasks execute more, the amount of time devices can be kept in *idle* mode decreases and the processor voltage needs to be kept at a high value. As the system utilization approaches 100%, SYS-EDF, CEA-EDF and DVS+DPM perform comparable to each other, because there is not much space for processor energy saving and all of them merely perform DPM for devices.

## 6 Conclusion

This paper presents a system-wide energy-efficient scheduling algorithm, SYS-EDF, which supports the preemptive scheduling of periodic tasks with non-preemptive shared resources. SYS-EDF consists of a practical DPM algorithm for I/O devices and a corresponding processor voltage scheduling algorithm. The SYS-EDF algorithm provides remarkable power savings by wisely setting the processor speed to balance the energy consumption of all components in the system. The evaluation of SYS-EDF shows that it yields significant energy savings with respect to DVS alone or DPM alone techniques or the straightforward integration of DVS and DPM.

## References

- [1] Baker, T.P., "Stack-Based Scheduling of Real-Time Processes," *Real-Time Systems*, 3(1):67-99, March 1991.
- [2] Cheng, H. and Goddard, S., "Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems with Shared Resources", Technical Report, 2005, [http://csce.unl.edu/~hcheng/TR\\_CEAEDF\\_EASD.pdf](http://csce.unl.edu/~hcheng/TR_CEAEDF_EASD.pdf)
- [3] Jejurikar, R., Pereira, C., Gupta, R., "Leakage aware dynamic voltage scaling for real-time embedded systems", *41st annual conference on Design automation*, 2004.
- [4] Jejurikar, R., Gupta, R., "Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems", *Symp. on Low power electronics and design*, 2004.
- [5] Liu and Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM*, 20(1), January, 1973.
- [6] Martin, S., Flautner, K., Mudge, T., Blaauw, D., "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads", *IEEE/ACM intl. conf. on Computer-aided design*, 2002.
- [7] Qadi, A., Goddard, S., Farritor, S., "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks", *IEEE Real-Time Systems Symp.*, 2003.
- [8] Swaminathan, V., Chakrabarty, K., and Iyengar, S.S., "Dynamic I/O Power Management for Hard Real-time Systems" *9th Intl. Symp. on Hardware/Software Codesign*, 2001.
- [9] Swaminathan, V., and Chakrabarty, K., "Pruning-based, Energy-optimal, Deterministic I/O Device Scheduling for Hard Real-Time Systems", *ACM Transactions on Embedded Computing Systems*, 4(1):141-167, February 2005.
- [10] Zhang, F., Chanson, S., "Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections", *IEEE Real-Time Systems Symp.*, 2002.
- [11] Zhuo, J., Chakrabarti, C., "System-Level Energy-Efficient Dynamic Task Scheduling", *42nd annual conf. on Design automation*, 2004.