Energy Reduction Techniques for Systems with Non-DVS Components*

Chuan-Yue Yang†, Jian-Jia Chen‡, Tei-Wei Kuo†, Lothar Thiele‡
†Department of Computer Science and Information Engineering
National Taiwan University, Taiwan
‡Computer Engineering and Networks Laboratory (TIK)
ETH Zürich, Switzerland
Email: †{r92032, ktw}@csie.ntu.edu.tw, ‡{chen, thiele}@tik.ee.ethz.ch

Abstract

Dynamic voltage scaling (DVS) has been widely adopted to reduce the energy consumption resulting from the dynamic power of modern processors. However, while the leakage power resulting from the leakage current becomes significant, how to aggregate the idle time to turn processors to the sleep or dormant modes is crucial in reducing the overall energy consumption. Moreover, for systems with non-DVS components, the execution order of tasks also affects the system-wide energy consumption. With the consideration of the dynamic and leakage power of processors as well as the power consumption resulting from non-DVS components, this paper summarizes our work on energy-efficient real-time task scheduling for both uniprocessor and multiprocessor platforms through procrastination of task executions, preemption control, and proper task assignment.

Keywords: Task procrastination, Leakage-aware scheduling, Preemption control, Heterogeneous multiprocessor, and Real-time systems.

1 Introduction

Performance boosting is used to be one of the most important design goals in system development. Because of the adoption of numerous gates in modern electric circuits, the power density and the power dissipation of modern processors have been increased dramatically. This trend makes low-power design become an active area in both academics and industry. From the aspect of micro-architectural techniques, dynamic voltage scaling (DVS) and dynamic power management (DPM) are two of the most promising techniques for energy efficiency. The DVS processors can adjust the supply voltage dynamically to reduce the power dissipation resulting from charging and discharging of gates, whereas some devices only support DPM and thus can only adjust the system mode dynamically for energy saving.

For real-time systems which have non-functional timing constraints to maintain the system stability, the objective of energy-efficient task scheduling is to minimize the energy consumption while completing all tasks without any violation of timing constraint. In the past decade, energy-efficient task scheduling algorithms for DVS processors with various processor/task models have been developed. In addition, since the power consumption resulting from leakage current has been getting comparable to the dynamic power dissipation recently, researchers have started to consider non-negligible leakage current.

In order to reduce the energy consumption from the leakage current, a processor might be turned to a dormant mode in which the power consumption of the processor can be significantly reduced. However, turning on the processor requires time and energy overheads, resulting from the wakeup/shutdown of the processor and data fetch in the register/cache. For example, the Transmeta processor with the 70nm technology has $483\mu J$ energy overhead and less than 2 millisecond timing overhead [12].

For systems with non-negligible leakage, Jejurikar et al. [12] and Lee et al. [15] proposed energy-efficient scheduling on a uniprocessor by procrastination scheduling to decide when to turn off the processor. Jejurikar and Gupta [11] then further considered real-time tasks that might complete earlier than its worst-case estimation by extending their previous work [12]. Fixed-priority scheduling was also considered by Jejurikar and Gupta [10] and Chen and Kuo [1]. For uniprocessor scheduling of aperiodic real-time tasks, Irani et al. [8] proposed a 3-approximation algorithm for the minimization of energy consumption with the considerations of leakage current on a uniprocessor DVS system with continuous available speeds. The basic idea behind the above results is to pro-

*This work was supported in part by the NSC under grant No. 95-2221-E-002-095-MY3, 95-2221-E-002-094-MY3, NSC-096-2917-I-564-121, and 97-2221-E-002-206-MY3, by the Excellent Research Projects of National Taiwan University under grant No. 97R0062-05, and by the Ministry of Economic Affairs under grant No. 96-EC-17-A-01-S1-034 in Taiwan. This research was also supported by the European Network of Excellence on Embedded System Design ARTIST-Design and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator (Grant 216008).

crastinate the execution of the real-time jobs as long as possible so that the idle interval is long enough to reduce the energy consumption. However, greedy procrastination at this moment might sacrifice the possibility to turn off the processor in the near future, and, hence, might consume more energy.

Energy-efficient task scheduling is complicated with the considerations of systems with non-DVS peripheral devices. The stretching of the execution time of a task might lead to more energy consumption on devices although some energy saving is obtained because of DVS scheduling over the processor. Energy-efficient task scheduling should consider the energy consumption of both the processor and devices! In this direction, Swaminathan et al. [20, 21] explored DPM of real-time tasks in shutting down system devices for energy efficiency without DVS considerations. Cheng and Goddard [3] explored on-line I/O subsystem schedules without DVS capability in the system. Kim et al. [13] developed an algorithm which minimizes the number of preemption in a dynamic schedule. Some other research focused on the reduction of stack or memory size in [4, 5] via preemption threshold scheduling. Mochocki et al. [18] provided algorithms which consider energy consumption minimization of the processor and an associated networking interface at the same time.

In addition, to conquer the dramatic increasing power density of electronic circuits, multiprocessor platforms have been adopted for the improvement of performance without incurring too much energy overhead. Energy-efficient task scheduling/partition in homogeneous multiprocessor systems has been studied extensively in the literature. However, only few results have been developed for energy-efficient task partition in heterogeneous multiprocessor systems. Specifically, Huang, Tsai, and Chu [6] developed a greedy algorithm based on affinity to assign frame-based real-time tasks with re-assignment in pseudo polynomial-time to minimize the energy consumption when any processing speed can be assigned for a processor. Luo and Jha [17] developed heuristics based on the list-scheduling for tasks with precedence constraints in heterogeneous distributed systems. The approaches by Hung, Chen, and Kuo [7] for platforms with one DVS processor and one non-DVS processor provide worst-case guarantees in energy consumption minimization or energy saving maximization.

This paper summarizes our recent work in energy-efficient scheduling with the consideration of leakage current and non-DVS components [2, 22, 23]. First, while considering the power consumption resulting from the leakage current, we propose a series of approaches in [2] that could avoid the disadvantage of greedy procrastination. For tasks which need to cooperate with some non-DVS devices, our preemption control mechanisms in [22] can be adopted to reduce energy consumption caused by some devices. When some non-DVS processing elements have the ability of executing tasks to reduce the workload of the general-purpose processing unit, a proper task partition among processing elements can reduce energy consumption efficiently. For such a scenario, our scheme in [23] can partition tasks properly and provide worst-case performance guarantee in certain task/processor models. With a series of simulations, our proposed algorithms can reduce the energy consumption significantly through procrastination of task executions, preemption control, and proper task assignment.

The rest of this paper is organized as follows: Section 2 shows the system models. Sections 3 and 4 present the proposed approaches and the performance evaluation, respectively. Section 5 concludes this paper.

2 System Models

2.1 Power consumption and execution models

In this work, we consider a set of M heterogeneous processors in which the power consumption of each processor m_j, denoted as $P_j(s)$, can be divided into two parts: $P_j^d(s)$ and P_j^{ind}, i.e., the power consumption which are dependent and independent on the operating speed s, respectively [24]. For a CMOS DVS processor, the speed-dependent power consumption $P_j^d(s)$ due to gate switching at speed s is

$$P_j^d(s) = h_j s^\kappa, \tag{1}$$

where h_j and $2 \leq \kappa \leq 3$ are two system-dependent constants that denote the effective switching capacitance and the dynamic power exponent, respectively. On the other hand, the speed-independent power consumption P_j^{ind} is a constant and is mainly caused by the leakage power resulting from the short-circuit power consumption.

Each processor in the processor set has two modes: *active mode* and *sleep mode*. While a processor can only perform computation in the active mode, the sleep mode is designed for energy savings. The DPM technique allows a processor to switch between the active and sleep modes dynamically to reduce the energy consumption resulting from the speed-independent power if possible. The power consumption of a processor when it is in the sleep mode can be treated as 0 by scaling the static power consumption [8]. However, switching between modes does introduce some time and energy overhead. Since the actions of turning the processor to and from the sleep mode are usually pairwise, we can assume the procedure to turn the processor to the sleep mode is done instantaneously with negligible energy overhead by treating the overhead as a part of the overhead to turn the processor to the active mode. We denote E_j^{sw} (t_j^{sw}, respectively) as the energy (time, respectively) of the *switching overhead* for processor m_j from its sleep mode to its active mode.

While a processor m_j is in the active mode, the number of CPU cycles executed in a time interval is assumed to be linear to its operating speed. That is, if $s_j(t)$ denotes the operating speed of the processor at the time in-

stance t on processor m_j, the cycles executed and the energy consumption of the processor during the time interval $[t_1, t_2)$ are $\int_{t_1}^{t_2} s_j(t)\mathrm{d}t$ and $\int_{t_1}^{t_2} P_j(s_j(t))\mathrm{d}t$, respectively. For simplicity of the presentation, we assume the processor can operate at any speed between the maximum available speed $s_j^{\max}$ and the minimum available speed $s_j^{\min}$. But, by executing tasks with two adjacent speeds of an unavailable speed [9, 14], it is possible to extend the results to processors with discrete speeds.

2.2 Device Models

In addition to the processors, some tasks will need to cooperate with some peripheral devices to complete its execution. In this work, we are interested in systems with O devices, which are denoted as $D_1, D_2, \ldots, D_O$, in which each device has the capability of DPM with three modes, i.e., *active, standby,* and *shutdown* modes. In this system, a device D_k is usually in the standby mode which consumes P_k^σ power, and is turned to the shutdown mode which is assumed to consume no power for energy saving when there is no task which might require this device. Moreover, when a device is accessed by a task, it is automatically switched from the standby mode to the active mode. The energy and timing consumption between mode transition of the active and standby mode of any device is assumed negligible, which is the same as that in [25]. However, each access of device D_k consumes P_k^a energy. Hence, once a task accesses D_k for r times within t time units, it will result in that device D_k consumes $P_m^\sigma \cdot t + P_m^a \cdot r$ energy.

For a *volatile* system device, such as Mobile RAM, the device must remain in the standby mode to maintain the data correctness for the required tasks between the starting time and the completion time of the task. For a *non-volatile* system device, such as flash memory, the device can be switched into the shutdown mode if it is no longer required by any task.

2.3 Task Models

The tasks which are considered to be executed in the system are a set of independent periodic real-time tasks [16]. For a given periodic real-time task set $\mathbf{T}$, each task τ_i in $\mathbf{T}$ is associated with its initial arrival time (denoted by a_i), its period (denoted by p_i), and its relative deadline which is assumed being equal to its period in this paper. That is, the arrival time and the *absolute deadline* of the j-th task instance of task τ_i are $a_i + (j-1) \cdot p_i$ and $a_i + j \cdot p_i$, respectively. For brevity, we will use deadline of a task instance to stand for the absolute deadline of the task instance implicitly. Moreover, the *hyper-period* of a given task set $\mathbf{T}$, denoted by $\mathcal{H}$, is defined as the minimum positive number so that $\mathcal{H}/p_i$ is an integer for any task τ_i in $\mathbf{T}$. For example, if all periods of tasks in $\mathbf{T}$ are integers, we can let $\mathcal{H}$ as the least common multiple of the periods of tasks in $\mathbf{T}$.

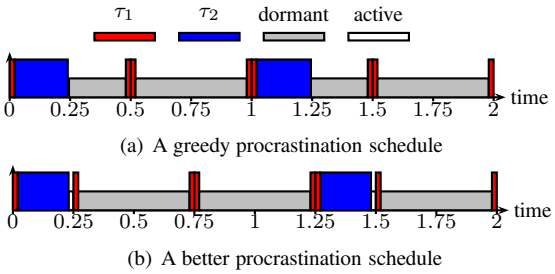
As we focus on a heterogeneous multiprocessor system, the worst-case execution time of a task τ_i depends on which processor the task is assigned to. That is, when task τ_i is executed on processor m_j at its maximum available speed $s_j^{\max}$, its worst-case execution time is $c_{i,j}$. When task τ_i requires not only the processor but also some devices to complete its execution. We use a set Φ_i to denote devices that τ_i will access during its execution. Let $v_{i,k}$ be the number of access of device D_k in Φ_i by τ_i within one of its task instances. Then the energy consumption due to its access of devices in Φ_i is $E_i^a = \sum_{D_k \in \Phi_i} P_k^a v_{i,k}$ which is a constant and is independent on the operating speed of the processor. Since the access pattern of devices is hard to be known in a priori, throughout this paper, all devices in Φ_i must be in the standby mode when τ_i is executed in the system. Hence, $\beta_i = \sum_{D_k \in \Phi_i} P_k^\sigma$ which is the summation of the standby power of devices in Φ_i when task τ_i is executed.

3 Energy-Efficient Scheduling Approaches

Given a periodic real-time task set $\mathbf{T}$ with O devices over M processors, we are interested in how to schedule the task set under its timing constraints with the minimized energy consumption. There are two categories of scheduling to schedule tasks over a multiprocessor environment: the global schedule and the partition schedule. While the global schedule dispatches task instances to different processors during runtime, the partition schedule allocates tasks to processors so that a task is only executed on one processor. For heterogeneous processors with different instruction sets, the global schedule suffers from considerable implementation cost, so we adopt the partition schedule in our study. Then the heterogeneous multiprocessor task scheduling problem can be divided into two parts: (1) how to partition tasks into processors, and (2) how to schedule tasks which have been allocated to a processor. When a task partition is determined, we will use $\mathbf{T}_j$ to denote the set of tasks allocated to processor m_j in the rest of this paper. Sections 3.1 and 3.2 present our approaches to minimize the energy consumption of tasks allocated in a processor, while the energy-efficient task partition approach is presented in Section 3.3.

3.1 Leakage-Aware Task Procrastination

When the total utilization of the task set is less than 100%, there will be some idle intervals which can be used to turn the processor to the sleep mode for energy saving with the DPM technique during runtime. However, turning the processor to the sleep mode is only worthwhile if the idle interval is longer than $E_j^{sw}/(P_j^d(s_j^{\min}) + P_j^{ind})$, which is referred to as the *break-even time*, denoted as bet, due to the non-negligible switching overhead E_j^{sw}. In order to make sure the energy saving resulting from turning the processor to the sleep mode can be maximized, procrastination algorithms [8, 12, 19] are proposed to aggregate the idle intervals to reduce the number of switches



(a) A greedy procrastination schedule

(b) A better procrastination schedule

Figure 1. An example for the disadvantage of greedy procrastination.

and thus the energy caused by the switching overhead.

However, most of existing procrastination algorithms greedily procrastinate the execution of tasks right after an idle interval and might have some shortcoming in some cases. Consider two tasks τ_1 and τ_2 arriving at time 0, in which $(p_1, p_2) = (0.25, 1)$, $(c_{1,j}, c_{2,j}) = (\epsilon, 0.25 - 2\epsilon)$ with a small and positive number ϵ, and $\Phi_1 = \Phi_2 = \emptyset$. The speed-dependent and speed-independent power of processor m_j are s^3 and 2, respectively. Moreover, its minimum available speed $s_j^{\min}$ is 0.5. And the energy and time of switching overhead E_j^{sw} and t_j^{sw} are 0.25 and 0.1, respectively. Then the break-even time of the processor is $0.25/(2+0.5^3) = 0.118$. We assume that the processor is in the active mode at time 0. To minimize the energy consumption for task execution, both of these two tasks would be executed at the critical speed, i.e., $\hat{s}_1 = \hat{s}_2 = 1$. By applying the greedy procrastination algorithms in [8, 12, 19], the processor is turned to the sleep mode at time $0.25 - \epsilon$, $0.5 + \epsilon$, $1.25 - \epsilon$, and $1.5 + \epsilon$ during $(0, 2]$. The derived schedule is shown in Figure 1(a). Its energy consumption while the processor is idle is 1. If we do not procrastinate the execution of coming task instances greedily while the processor is idle, we may have a better result as shown in Figure 1(b) in which the processor is only turned to the sleep mode at time $0.25 + \epsilon$, $0.75 + \epsilon$, and $1.5 + \epsilon$ during $(0, 2]$. The energy consumption while the processor is idle during $(0, 2]$ is $0.75 + 2\epsilon(2.125)$ which is much less than that of the schedule in Figure 1(a). Therefore, instead of greedy procrastination, we present a parametric procrastination algorithm to deal with the drawback suffered from the greedy procrastination algorithms.

Parametric Procrastination At time point t in which the processor becomes idle and we have to determine whether the processor should be turned to the sleep mode or not, procrastination algorithms first find out the latest time point W_t in which the processor should go back to the active mode to prevent task instances from missing deadlines. Let $\mathcal{R}_t$ be the earliest arrival time of task instances arriving after the instant t. We refer the interval $(t, \mathcal{R}_t]$ as *residual interval*. Since the processor must be idle during the residual interval, we will always turn the processor to the sleep mode if $\mathcal{R}_t - t$ is greater than the break-even time bet. When the length of the residual interval is not greater than bet, it is possible to combine the residual interval with the *procrastination interval*, i.e., $(\mathcal{R}_t, W_t]$ to turn the processor to the sleep mode. We introduce a user-specified parameter α to control the portion of the procrastination interval that will be adopted while determine whether the processor should be turned to the sleep mode or not, where $0 \leq \alpha \leq 1$. Then the processor is turned to the sleep mode at time t if

$$\mathcal{R}_t - t + \alpha(W_t - \mathcal{R}_t) \geq bet. \tag{2}$$

We denote this approach as Algorithm P-Procrastination.

3.2 Dynamic Preemption Control

Consider a scheduling point t_θ at which a high-priority task instance J_H preempts a low-priority task instance J_L, the standby power of devices in $\Phi_L \setminus \Phi_H$ will induce some energy overhead during the execution of J_H, since they should be kept in the standby mode to preserve the correctness of their temporal data. Thus, if the preemption can be eliminated, the energy consumption of task executions might be reduced. Let J_H and J_L be the task instances of tasks τ_H and τ_L, respectively. In the following presentation, we suppose the amount of the remaining worst-case execution time for the low-priority task instance J_L at time t_θ is w_L. Let $r_{i,\theta}$ be the earliest arrival time of a task instance of task τ_i no earlier than t_θ, i.e., $r_{i,\theta} = a_i + \lceil \frac{t_\theta - a_i}{p_i} \rceil p_i$. The absolute deadline of the task instance of task τ_i arriving at time $r_{i,\theta}$ is denoted by $d_{i,\theta}$, where $d_{i,\theta}$ is $r_{i,\theta} + p_i$. Then, we define Δ_θ as $\min_{\tau_i \in \mathbf{T}_j}\{d_{i,\theta}\}$, which is the minimum absolute deadline of the task instances of tasks in $\mathbf{T}_j$ arriving no earlier than t_θ. In the rest of this section, let $\hat{s}_i$ denote the execution speed of task τ_i.

Linear-Time Determination Let $\mathbf{J}_\theta$ be the set of task instances arriving in time interval $[t_\theta, \Delta_\theta)$ with higher priorities than J_L does. That is,

$$\mathbf{J}_\theta = \{J_{i,\theta}, \forall \tau_i \in \mathbf{T}_j \mid r_{i,\theta} < \Delta_\theta \text{ and } d_{i,\theta} < d_L\}, \tag{3}$$

where $J_{i,\theta}$ is the task instance of task τ_i arriving at $r_{i,\theta}$[1]. In the original EDF schedule, since only task instances in $\mathbf{J}_\theta$ are possible to preempt J_L and execute during $[t_\theta, \Delta_\theta)$, we can allow J_L to block high-priority task instances if all task instances in $\mathbf{J}_\theta$ and J_L itself can be completed by the minimum absolute deadline Δ_θ. In other words, if

$$w_L \frac{s_j^{\max}}{\hat{s}_L} + \sum_{J_{i,\theta} \in \mathbf{J}_\theta} c_{i,j} \frac{s_j^{\max}}{\hat{s}_i} \leq \Delta_\theta - t_\theta, \tag{4}$$

then J_L can continue its execution at time instant t_θ by blocking all of the high-priority task instances before the

[1]We assume tasks are scheduled under the earliest-deadline-first (EDF) scheduling policy.

completion of J_L; otherwise, J_H preempts J_L to start its execution. This approach is denoted as Algorithm A-EDF. Its time complexity at each scheduling point t_θ is dominated by the determination of $\mathbf{J}_\theta$ which consists of at most $|\mathbf{T}_j|$ task instances and thus is $O(|\mathbf{T}_j|)$.

$O(n \log n)$-Time Determination Since task instances in $\mathbf{J}_\theta$ are not all ready at t_θ, the actual execution time of task instances in $\mathbf{J}_\theta$ in the original EDF schedule may be much less than the summation of the execution times of task instances in $\mathbf{J}_\theta$. Therefore, instead of accumulating all execution times of task instances in $\mathbf{J}_\theta$, we simulate the original EDF schedule during $[t_\theta, \Delta_\theta)$ to determine whether J_L should be preempted by J_H or not. If the total length of the idle intervals in the simulated schedule is no less than $w_L \frac{s_j^{\max}}{\hat{s}_L}$, J_L can block all high-priority task instances before its completion without any deadline miss; otherwise, J_H has to preempt J_L and starts its execution. The approach is denoted as Algorithm S-EDF. Its running time at scheduling point t_θ is $O(|\mathbf{T}_j| \log |\mathbf{T}_j|)$.

Constant-Time Determination If we can determine a *tolerable-blocking time* R_i for each task τ_i in $\mathbf{T}_j$ according to their speed assignment off-line, we can just compare the remaining execution time of J_L with the tolerable-blocking time of J_H to determine whether the preemption should be taken or not during runtime. The definition of the tolerable-blocking time is that task τ_i can still meet its deadline even when it is blocked by low-priority task instances for at most R_i time units. Thus, if $w_L \frac{s_j^{\max}}{\hat{s}_L}$ is no greater than R_H, J_L could continue its execution by blocking J_H; otherwise, J_L should be preempted and the highest-priority task instance in the ready queue (it might be different from J_H) starts its execution. The determination of R_i can be done by indexing task in $\mathbf{T}_j$ such that $p_i \leq p_j$ if $i < j$ and according to the following equation:

$$R_i = p_i(1 - \sum_{j=1}^{i} \frac{c_j}{p_j \hat{s}_j}). \tag{5}$$

This approach is denoted as Algorithm R-EDF. Compared to the original EDF scheduling algorithm, its additional runtime overhead at each scheduling point is $O(1)$.

Figure 2 gives an illustration of the proposed preemption control algorithms for three tasks with periods $(p_1, p_2, p_3) = (2, 2.9, 6)$ and execution times $(t_{1,j}, t_{2,j}, t_{3,j}) = (0.2, 1.1, 3)$, where $R_1 = 1.8$, $R_2 = 1.51$, and $R_3 = 0$. When $t_\theta = 2.9$, since $w_L = 1.6$ is greater than $R_2 = 1.51$, Algorithm R-EDF preempts the execution of J_L. However, as $\Delta_\theta = 5.8$ and $\sum_{J_{i,\theta} \in \mathbf{J}_\theta} \frac{c_i}{\hat{s}_i} = 1.3$, Algorithm A-EDF lets J_L continue its execution at $t_\theta = 2.9$. The numbers of preemption for schedules in Figure 2(b) and Figure 2(c) in time interval $[0, 18)$ are 6 and 3, respectively, compared to that in the original EDF schedule is 8. For this example, the schedule derived by Algorithm S-EDF is the same as that shown in Figure 2(c) in time interval $[0, 18)$.

Hybrid Approaches If the high-priority task instance J_H is always blocked for R_H time units, it can be seen as arriving at $r_{i,\theta} + R_i$ with absolute deadline $r_{i,\theta} + p_i$. With this idea, we can incorporate Algorithm A-EDF or S-EDF with Algorithm R-EDF to derive a hybrid approach. More specifically, we can revise the definition of $\mathbf{J}_\theta$ as follows:

$$\mathbf{J}_\theta = \{J_{i,\theta}, \forall \tau_i \in \mathbf{T}_j \mid r_{i,\theta} + R_i < \Delta_\theta \text{ and } d_{i,\theta} < d_L\}. \tag{6}$$

The revision of Algorithms A-EDF and S-EDF by applying the definition of $\mathbf{J}_\theta$ in Equation (6) are denoted as Algorithms HA-EDF and HS-EDF, respectively. This revision does not increase the complexity at each scheduling point.

Algorithms A-EDF, S-EDF, R-EDF, HA-EDF, HS-EDF do derive feasible schedule. The details of the proof for their feasibility can be found in [22].

Slack Reclamation Policy When tasks complete earlier than their worst-case estimations, the unused execution time, also called slack, can be reclaimed to scale down the operating speeds of some incomplete task instances for energy saving. Although slack reclamation by slowing down can reduce the energy resulting from the dynamic power of the processor, the execution time of some tasks might be prolonged and the non-DVS components might, hence, consume more energy. Therefore, while non-DVS components are considered, slack should be first used for preemption control and only used for scaling down the operating speeds when there is no preempted task instance.

3.3 Energy-Efficient Task Partition

Conventionally, the energy consumption function is usually an increasing function of the workload of tasks allocated to the processor. Thus, it is reasonable to use the workload of tasks allocated to a processor to model the energy consumption of executing this set of tasks in the processor during the hyper-period $\mathcal{H}$. With this idea, we present two approaches to partition tasks over a heterogeneous multiprocessor environment in this section.

A Dynamic Programming The basic idea of the dynamic programming is to enumerate all possible task partitions of the given task set $\mathbf{T}$. Then we choose a best one from all of the task partitions as the solution. More specifically, we use a tuple $\theta = (U_1^\theta, U_2^\theta, \ldots, U_M^\theta)$ to represent a state Θ standing for a (partial) task partition in our dynamic programming, where $U_j^\theta = \sum_{\tau_i \in \mathbf{T}_j^\Theta} \frac{c_{i,j}}{p_i}$ and $\mathbf{T}_j^\Theta$ is the workload in terms of the utilization at speed $s_j^{\max}$ and the set of tasks, respectively, allocated to processor m_j in the (partial) task partition. Initially, when all tasks

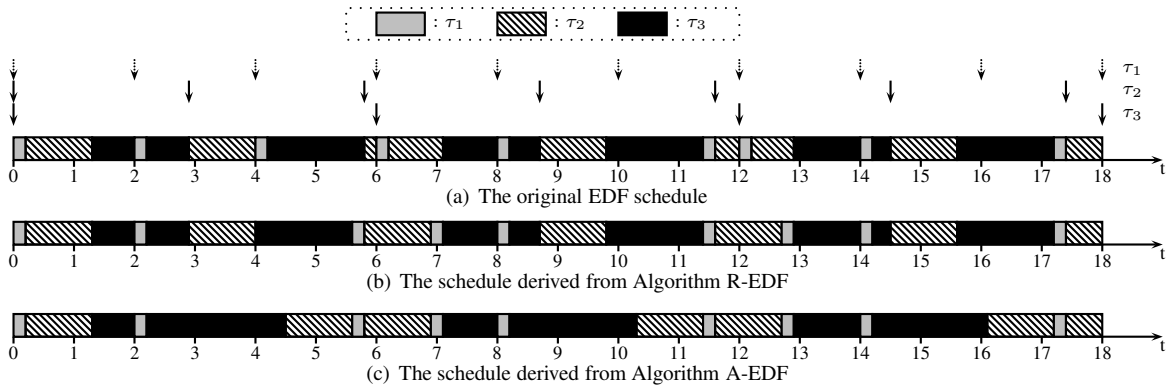


Figure 2. Illustrative examples of Algorithm R-EDF and Algorithm A-EDF.

have not been considered yet, the initial set of states $\mathcal{S}_0$ only contain one state, i.e., $\theta = (0, 0, \ldots, 0)$. As we consider tasks in $\mathbf{T}$ one by one, we will construct a set of new states $\mathcal{S}_i$ based on the existing ones in $\mathcal{S}_{i-1}$ when task τ_i is considered. That is, we construct M new states $(U_1^\theta + \frac{c_{i,1}}{p_i}, U_2^\theta, \ldots, U_M^\theta)$, $(U_1^\theta, U_2^\theta + \frac{c_{i,2}}{p_i}, \ldots, U_M^\theta)$, …, $(U_1^\theta, U_2^\theta, \ldots, U_M^\theta + \frac{c_{i,M}}{p_i})$ and adds them into set $\mathcal{S}_i$ for each state Θ, i.e., $\theta = (U_1^\theta, U_2^\theta, \ldots, U_M^\theta)$, in $\mathcal{S}_{i-1}$. After all tasks are considered, the set $\mathcal{S}_{|\mathbf{T}|}$ consists of all task partitions and can be used to choose the final solution. However, this approach takes exponential time and space, i.e., $O(M^{|\mathbf{T}|})$, in the worst case. It encourages us to develop a more efficient approach to derive near-optimal solutions while the task set $\mathbf{T}$ is too large to obtain the optimal solution.

An Approximation Scheme Since there are too many states in the sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{|\mathbf{T}|}$, we can improve the efficiency of the dynamic programming by only keeping some representative states and discarding the others. If the state pruning can be done carefully, the degradation for the optimality of the final result can be bounded. Given a user-specified parameter ϵ, we will derive a parameter, denoted by δ_j, for each processor m_j which is used for state pruning accordingly. According to the energy consumption function of workload allocated in processor m_j, denoted as $E_j(U_j)$, and the user-specified parameter ϵ, we will find a value for δ_j such that $E_j((1+\delta_j)U_j) \leq (1+\epsilon)E_j(U_j)$. That is, the increase in the energy consumption will be bounded if the increase in the allocated workload is bounded in $(1+\delta_j)$ times of the original one. Once we can find such a δ_j for each processor m_j, we can guarantee the performance of the derived solutions.

During the construction of states, we perform state pruning after each set of states $\mathcal{S}_i$ is constructed. The basic idea of our pruning process is to round down the workload U_j^θ for all processors m_j except processor m_1 for each state Θ in $\mathcal{S}_i$. More specifically, in order to round the workload of processor m_j, we first sort states in $\mathcal{S}_i$ according to U_j^θ in a non-decreasing order. We then choose the minimum U_j^θ as the base, denoted by U_j^B. After that, all U_j^θ's which are no greater than $(1+\delta_j)U_j^B$ are rounded down to U_j^B. While we encounter a U_j^θ which is greater than $(1+\delta_j)U_j^B$, the U_j^θ is set as the new base U_j^B. After the update of U_j^B, the same procedure is applied to the remaining states in $\mathcal{S}_i$. Once the rounding process is done for workloads of processors m_2 to m_M, we can start to prune states in $\mathcal{S}_i$. For a set of states which have the same workloads for processors m_2 to m_M, we use the one with the minimum U_1^θ to represent the set of states and remove all other states from $\mathcal{S}_i$.

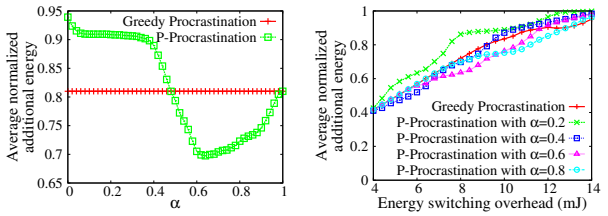
With the above process, the complexity of the dynamic programming can be reduced significantly. While the energy consumption function of the workload on each processor m_j is a non-decreasing and the increase of the energy consumption is bounded if the increase of the workload is also bounded, this approach can also provide certain worst-case performance guarantees if the pruning parameter δ_j of each processor m_j is set according to the user-specified parameter properly. The details of rationale for the setting of the parameters and the corresponding proof can be found in [23].

4 Performance Evaluation

4.1 Uniprocessor Energy-Efficient Techniques

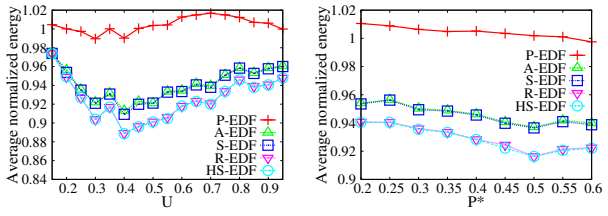
We evaluate algorithms presented in Sections 3.1 and 3.2 for Intel XScale. Its power consumption function can be modeled approximately as $P_j^d(s) + P_j^{ind} = 1.52s^3 + 0.08$ Watt while the range of its available speeds is from $s_j^{\min} = 0.15$GHz to $s_j^{\max} = 1$GHz. The periods of the synthetic tasks adopted in our simulations are chosen randomly in the scale of millisecond. By associating each task τ_i with a random variable μ_i^* in $(0, 1]$, its execution time is set as $\frac{\mu_i^*}{\sum_{\tau_l \in \mathbf{T}_j} \mu_l^*} U p_i$ under a specified total utilization U of a set of tasks $\mathbf{T}_j$.

Procrastination Algorithms When the number of tasks is 20, Figure 3 shows the evaluation results for the *normalized additional energy* of Algorithms P-Procrastination and Greedy Procrastination, where the normalized additional energy of an algorithm is the energy consumption



(a) $U = 0.3$ (b) $U = 0.5$

Figure 3. Evaluation results of Algorithm P-Procrastination.



(a) $P^* = 0.2$ (b) $U = 0.9$

Figure 4. Simulation results of Preemption Control Algorithms.

of the idle intervals in its derived schedule divided by that of the original EDF schedule. As shown in Figure 3(a), Algorithm P-Procrastination can improve the greedy procrastination by 12% with a proper setting of α when $E_j^{sw} = 10$mJ. Figure 3(b) shows the results of Algorithm P-Procrastination by varying E_j^{sw} from 4mJ to 14mJ. In general, when the energy of switching overhead increases, the value of α should increase accordingly to achieve better results.

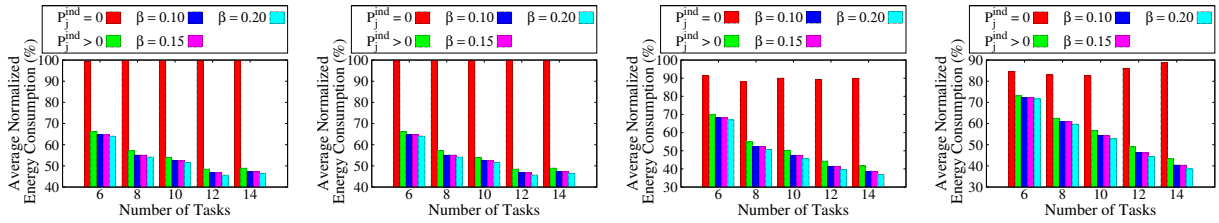
Preemption Control Algorithms In the simulations for preemption control algorithms, the number of devices required for the synthetic tasks is an integral random variable in $[2, 5]$, where the standby power of each device is uniformly distributed in $[0.05, P^*]$ Watt, where P^* is a given parameter. The probability that a task requires a particular device for its execution is set as 0.5. The actual execution time of each task is uniformly distributed from 80% to 100% of its worst-case execution time to evaluate the integration of preemption control algorithms and the slack reclamation policy. In addition to our proposed algorithms, the delayed-preemption control in [13], denoted by Algorithm P-EDF, is also implemented for comparison. All the evaluated algorithms execute tasks according to the speed assignment derived from Algorithm STATIC in [22]. Figure 4 shows the *normalized energy consumption* of the evaluated algorithms, where the normalized energy consumption is defined by the system energy consumption of the evaluated algorithm in the hyper-period divided by that of the original EDF schedule. Figure 4(a) shows the simulation results when $P^* = 0.2$ by varying U from 0.15 to 0.95. As the promotion of the execution speed of a high priority task instance significantly increased the dynamic energy consumption significantly, the energy consumption of the schedules derived from Algorithm P-EDF is close to and might be worse than the original EDF schedules. We can also find out that Algorithm HS-EDF has the best performance and can reduce the energy consumption by more than 10%. From Figure 4(b) which shows the simulation results when $U = 0.9$ by varying P^* from 0.2 to 0.6, we can see that the proposed algorithms had steady performance regardless the value of P^*.

4.2 Task Partition for Heterogeneous Multiprocessor Systems

The proposed task partition algorithm is evaluated with a series of simulations based on the setup in [6]. The worst-case execution time $c_{i,j}$ of task τ_i on processor m_j is set randomly from $1000us$ to $3000us$ at speed s_j^{max}. We randomly choose M processors from nearly 30 types of processors, including general-purpose embedded processors, such as ARM9 and ARM11, and digital signal processors, such as TMS320C and TMS320D. Figure 5 shows the average of the *Normalized Energy Consumption* which is the energy consumption of the task partition derived from our approach divided by that of the task partition derived from the greedy-based algorithm in [6]. Moreover, $P_j^{ind} = 0$ stands for the results of model which is the same as that in [6], $P_j^{ind} > 0$ stands for the results of energy consumption models with non-negligible speed-independent power, and $\beta = 0.10, \beta = 0.15$, and $\beta = 0.20$ stand for the results of energy consumption models with different ranges of switching overheads by varying E_j^{sw} from $0.05 \cdot \mathcal{H}P_j(s_j^{crit})$ mJ to $\beta \cdot \mathcal{H}P_j(s_j^{crit})$ mJ. Although the performance of our proposed algorithm is similar to that of the greedy-based algorithm in [6] for systems whose the speed-independent power and switching overhead are negligible, our algorithm can achieve significant improvement when the speed-independent power or the switching overhead is non-negligible with a proper choice of the user-specified parameter ϵ for state pruning.

5 Conclusion

In order to cope with the increasing functional demands of modern computing systems, systems usually have more than one processing element. Thus, energy-efficient real-time task scheduling is a very complicated issue in system-level design. This paper summarizes (some of) our results to address parts of this challenging problem. The proposed procrastination algorithms [2] and preemption control algorithms [22] outperform existing algorithms and can reduce the energy caused by the leakage power of processors and the standby power of devices, respectively. We also discuss how to integrate the slack reclamation with our preemption control algo-



(a) $M = 2, \epsilon = 0.5$ (b) $M = 2, \epsilon = 1$ (c) $M = 4, \epsilon = 1$ (d) $M = 6, \epsilon = 1$

Figure 5. The average normalized energy consumption of the proposed task partition approach.

rithms. The proposed task partition approach [23] partitions tasks over heterogeneous multiprocessor environments energy-efficiently and can provide certain worst-case performance guarantees under some conditions.

There are still many open problems for energy-efficient scheduling for systems with DVS and non-DVS components. For example, as the leakage power might depend on the temperature, procrastination algorithms with the consideration of non-constant speed-independent power might be challenging. In addition, preemption control among multiprocessor environments for tasks with resource competition or precedence constraints is also open.

References

[1] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 153–162, 2006.

[2] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, pages 289–294, 2007.

[3] H. Cheng and S. Goddard. Online energy-aware i/o device scheduling for hard real-time systems. In *Design, Automation and Test in Europe*, pages 1055–1060, 2006.

[4] R. Ghattas and A. Dean. Preemption threshold scheduling: Stack optimality, enhancements, and analysis. In *Proceedings of the 13th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 147–157, 2007.

[5] R. Ghattas, G. Parsons, and A. Dean. Optimal unified data allocation and task scheduling for real-time multi-tasking systems. In *Proceedings of the 13th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 168–179, 2007.

[6] T.-Y. Huang, Y.-C. Tsai, and E. T.-H. Chu. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem. In *21th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.

[7] C.-M. Hung, J.-J. Chen, and T.-W. Kuo. Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element. In *the 27th IEEE Real-Time Systems Symposium (RTSS)*, pages 303–312, 2006.

[8] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46, 2003.

[9] T. Ishihara and H. Yasuura. Voltage scheduling problems for dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 197–202, 1998.

[10] R. Jejurikar and R. K. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 57–66, 2004.

[11] R. Jejurikar and R. K. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *DAC*, pages 111–116, 2005.

[12] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, pages 275–280, 2004.

[13] W. Kim, J. Kim, and S. Min. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 393–398, 2004.

[14] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the 40th Design Automation Conference*, pages 125–130, 2003.

[15] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[17] J. Luo and N. Jha. Static and dynamic variable voltage scheduling algorithms for realtime heterogeneous distributed embedded systems. In *the 15th International Conference on VLSI Design (VLSID'02)*, pages 719–726, 2002.

[18] B. Mochocki, D. Rajan, X. S. Hu, C. Poellabauer, K. Otten, and T. Chantem. Network-aware dynamic voltage and frequency scaling. In *Proceedings of the 13th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 215–224, 2007.

[19] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 140–148, 2004.

[20] V. Swaminathan and C. Chakrabarti. Energy-conscious, deterministic I/O device scheduling in hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7):847–858, 2003.

[21] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems. *ACM Trans. Embedded Comput. Syst.*, 4(1):141–167, 2005.

[22] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. Preemption control for energy-efficient task scheduling in systems with a DVS processor and Non-DVS devices. In *the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2007.

[23] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In *DATE*, pages 694–699, 2009.

[24] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, pages 397–407, 2006.

[25] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *ACM/IEEE Design Automation Conference(DAC)*, pages 628–631, 2005.