Energy-Efficient Optimal Real-Time Scheduling on Multiprocessors*

Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki
Graduate School of Science and Technology
Keio University, Yokohama, Japan
{funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract

Optimal real-time scheduling is effective to not only schedulability improvement but also energy efficiency for real-time systems. In this paper, we propose real-time static voltage and frequency scaling (RT-SVFS) techniques based on an optimal real-time scheduling algorithm for multiprocessors. The techniques are theoretically optimal when the voltage and frequency can be controlled both uniformly and independently among processors. Simulation results show that the independent RT-SVFS technique closely approaches the lower bound on energy consumption if the voltage and frequency can be controlled minutely.

1. Introduction

Multiprocessor architectures such as Simultaneous Multithreading (SMT) and Chip Multiprocessing (CMP) are becoming more attractive for intelligent embedded systems. It is important for embedded systems that real-time tasks such as robot controls and image processing meet their real-time constraints. Therefore powerful processors are desirable for these systems. On the other hand, the trade-off between system performance and energy efficiency is critically important for battery-based embedded systems. Real-time operating systems must go together with both requirements.

Real-time voltage and frequency scaling techniques have been introduced to solve the problem. The processors of most recent computer systems are based on CMOS logic. Maximum processor frequency f depends on supply voltage V (i.e., $V = g(f)$), and energy consumption P is proportional to processor frequency and square of supply voltage (i.e., $P \propto fV^2$) [5]. Real-time voltage and frequency scaling techniques can potentially save energy at a cubic order, while they meet real-time constraints. Real-time voltage and frequency scaling is based on the essential characteristic of real-time tasks; namely the tasks can be executed slowly as long as all deadlines are met.

Real-time voltage and frequency scaling techniques are constructed on real-time scheduling theories to meet real-time constraints. For single-processor systems, EDF [13] is an optimal real-time scheduling algorithm. On the other hand, EDF-FF and EDF-US, which are the extensions for multiprocessors, are not optimal [14, 4]. Approximately 50% processor time is wasted to meet real-time constraints on the algorithms at the worst-case. In other words, the algorithms theoretically require twice as many processors or powerful processors as optimal algorithms do. Accordingly the systems which leverage the algorithms expend more energy than ideal. Fortunately three optimal real-time scheduling algorithms for multiprocessors are presented (i.e., PD^2 [1], EKG [2], and LNREF [7, 8]). PD^2 incurs significant run-time overhead due to its quantum-based scheduling approach. EKG concentrates workloads on some processors due to the approach similar to partitioned scheduling. From the viewpoint of energy efficiency, energy consumption is minimized when the workloads are balanced among processors [3]. LNREF is an efficient algorithm on the balance as compared to the other optimal algorithms. Therefore we construct real-time voltage and frequency scaling techniques based on LNREF.

There are two approaches "static and dynamic" for real-time voltage and frequency scaling. Changing voltage and frequency takes some time to ensure system memory access due to physical limitations. Intel Pentium M [10] processors require 10-15μs per voltage and frequency scaling. It is possible to ignore the overhead in some cases. However next-generation real-time systems such as humanoid robots are controlled in the 200μs or faster control loop. For these systems, frequent voltage and frequency scaling incurs significant overhead. Furthermore the optimal real-time scheduling algorithms for multiprocessors cause more frequent context switches than the other algorithms do. Static voltage and frequency scaling is a good solution for these systems. In this paper, static techniques targeting the systems which can not ignore the overhead are presented. Additionally static approaches are desirable for

*This research is supported by CREST, JST.

978-0-7695-3132-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ISORC.2008.19

industrial systems because the system analysis is simple.

Optimal real-time static voltage and frequency scaling (RT-SVFS) on multiprocessors is a NP-hard partition problem since selectable processor frequency is discontinuous on practical systems. It is sufficient to completely solve the problem at the beginning if the system never changes indefinitely. However some real-time systems would like to vary their configurations without system standstill (e.g., system updates, physical reconfigurations). Additionally aperiodic tasks, which are executed by the server approach [17], are deemed to be the system reconfiguration if the arrivals are infrequent. Therefore real-time voltage and frequency scaling techniques must accommodate to dynamic environments even if the techniques are static approaches. Exhaustive algorithms for the NP-hard problem at every reconfiguration incur significant overhead. Consequently we assume that processor frequency can be controlled *continuously* at first. Then we prove that our techniques are theoretically optimal under the assumption. Finally the effectiveness of the technique is shown in the simulation on practical environments (i.e., discontinuous frequency).

The remainder of this paper is organized as follows. The next section discusses the related work. In section 3, we show the system model. Section 4 explains T-N Plane Abstraction, which is the basis of our RT-SVFS techniques. In section 5, we present new RT-SVFS techniques for optimal real-time scheduling on multiprocessors. Section 6 evaluates the technique on practical environments. Finally we conclude with a summary and future work in section 7.

2. Related Work

Many real-time voltage and frequency scaling techniques have been proposed in many aspects for single processor systems. Pillai and Shin [15] show two RT-SVFS techniques based on EDF and RM [13]. EDF is an optimal real-time scheduling algorithm for single processors. Our uniform RT-SVFS on multiprocessors is analogous to EDF-based RT-SVFS. Real-time dynamic voltage and frequency scaling (RT-DVFS) techniques are also proposed for hard real-time systems [15], soft real-time systems [18], and dynamic real-time systems [12] to achieve more energy efficiency. On the other hand, previous works [19, 6, 16] for multiprocessors are based on partitioned scheduling or non-optimal global scheduling. As mentioned above, the algorithms require twice as many processors or powerful processors as optimal algorithms at the worst case. Consequently no optimal RT-SVFS technique is presented heretofore.

Our previous work [9] minimizes not total energy consumption but total processor frequency. This paper is the first work that realizes both optimal real-time scheduling and theoretically optimal RT-SVFS on multiprocessors.

3. System Model

In this paper, we present the problem of scheduling a set of hard periodic tasks with voltage and frequency scaling on a multiprocessor system. The system is modeled as a taskset $\mathbf{T} = \{T_1, \ldots, T_N\}$, which is a set of N periodic tasks to be executed on M processors $\mathbf{P} = \{P_1, \ldots, P_M\}$. Each processor P_k is characterized by *continuous* normalized processor frequency α_k ($0 \leq \alpha_k \leq 1$). Each processor can execute at most one task simultaneously. Each task can not be executed in parallel among processors. Each task T_i is characterized by two parameters, worst-case execution time c_i and period p_i. A task T_i executed on a processor P_k requires c_i/α_k processor time at every p_i interval. The relative deadline d_i is equal to its period p_i. All tasks must complete the execution by the deadlines. The ratio c_i/p_i, denoted u_i ($0 < u_i \leq 1$), is called task utilization. $U = \sum_{T_i \in \mathbf{T}} u_i$ denotes taskset utilization. Maximum task utilization is defined as $U_{\max} = \max\{u_i | T_i \in \mathbf{T}\}$. We assume that all tasks may be preempted and migrated among processors at any time, and are independent (i.e., they do not share resources and do not have any precedence).

In this paragraph, the differences between the system model and practical environments are discussed. (1) In practical environments, operable processor frequencies are discontinuous. The set of operable frequencies is defined as $\mathbf{f} = \{f_1, \ldots, f_m | f_1 < \cdots < f_m\}$. The lowest frequency $f_i \in \mathbf{f}$ such that $\alpha_k \leq f_i/f_m$ will be selected to bridge the gap between theory and practicality. (2) Processor throughput is not proportional to processor frequency in many cases as opposed to the system model described above. In practical systems, the frequency which can achieve the corresponding system throughput will be selected. (3) The system model assumes that no overhead occurs at run-time. In practical environments, the scaled frequency interferes with the scheduling and the resource control even if the frequency is not changed dynamically. The worst-case overhead must be included in the worst-case execution time c_i.

4. T-N Plane Abstraction

T-N Plane Abstraction [7, 8] is an abstraction technique of real-time scheduling. T-N Plane Abstraction is based on the fluid scheduling model [11]. In fluid scheduling, each task is executed at a constant rate at all times. Figure 1 illustrates the difference between the fluid schedule and a practical schedule. The figure represents time on horizontal axis and task's remaining execution time on vertical axis. In the fluid scheduling model, each task T_i is executed along its fluid schedule path, the dotted line from (r_i, c_i) to $(r_i + p_i, 0)$, where r_i is task's release time. It is impossible for the fluid scheduling model to realize optimal schedule on practical systems since one processor must execute some

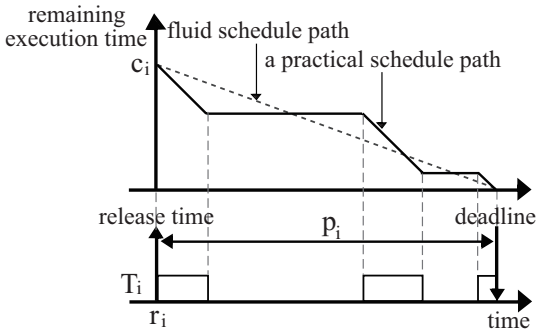

Figure 1. Fluid schedule and a practical schedule.

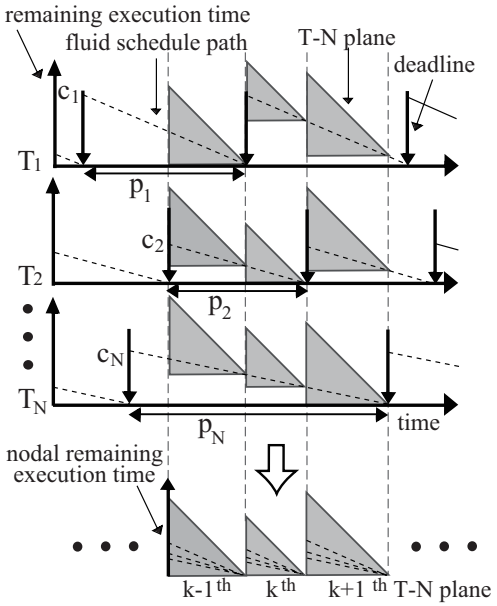

Figure 2. T-N Plane Abstraction.

tasks simultaneously. Note that the deadline is the only time at which we must track the fluid schedule path.

Figure 2 shows how real-time scheduling is abstracted. Deadlines divide time as the vertical dotted lines. The right isosceles triangles called T-N planes (Time and Nodal remaining execution time domain planes) are placed between every two consecutive deadlines. We make the rightmost vertex of the T-N plane coincide with the intersection of the fluid schedule path and the right side of the divided time-span. Since T-N planes in the same time-span are congruent, we have only to keep in mind an overlapped T-N plane shown in the lower of the figure at a time. The T-N plane represents time on horizontal axis and task's *nodal remaining execution time*, denoted l_i for each task T_i, on vertical axis. If the nodal remaining execution time becomes zero at

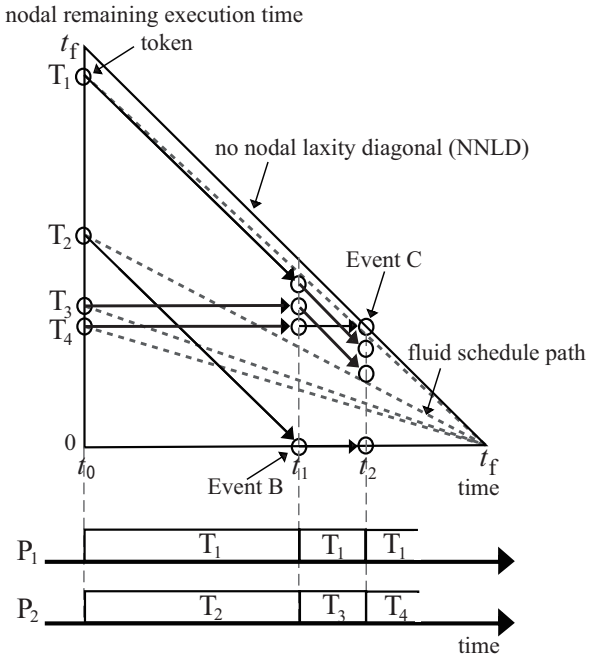

Figure 3. LNREF scheduling algorithm.

the rightmost vertex of the T-N plane, the execution accedes to the fluid schedule path for each deadline.

Figure 3 shows an overlapped T-N plane, where tokens representing tasks move from t_0 to t_f. The tokens are on their fluid schedule paths at the beginning of the T-N plane as shown in the following theorem, where $r_{i,j} = l_{i,j}/(t_f - t_j)$ denotes the *nodal utilization* of T_i at time t_j.

Theorem 1 (Cho et al.) *The initial nodal utilization value $r_{i,0} = u_i$ for all task T_i.* ■

A token moves diagonally down if the task is executed; otherwise it moves horizontally. If all tokens arrive at the rightmost vertex, all tasks meet their deadlines. We call the successful arrival to the rightmost vertex, *nodally feasible*. For nodal feasibility, new events at which the scheduling decision is made again in the T-N plane are laid on. Event C and event B occur when tokens hit the oblique side (NNLD) and the bottom side of the T-N plane, respectively. We assume that the jth event occurs at time t_j. M tokens which have the Largest Nodal Remaining Execution time are selected First (LNREF) on M processors for each event. LNREF is an optimal real-time scheduling algorithm for multiprocessors as shown in the following theorem.

Theorem 2 (Cho et al.) *Any periodic taskset **T** with utilization $U \leq M$ will be scheduled to meet all deadlines on M processors by LNREF.* ■

For example, there are four tasks (T_1, T_2, T_3, T_4) and two processors (P_1, P_2) as shown in Figure 3. Since there

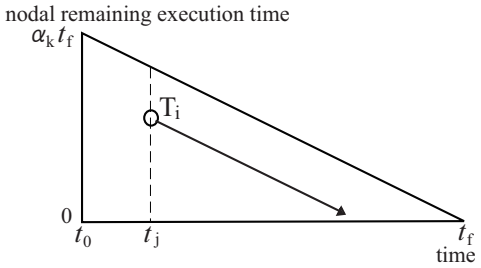

Figure 4. T-N Plane Transformation $(\alpha_k = 0.5)$**.**

are two processors, two tasks can be executed simultaneously. At time t_0, T_1 and T_2 are executed on P_1 and P_2 in the LNREF order. Event B occurs at time t_1 since T_2 hits the bottom side of the T-N plane. Then two tasks T_1 and T_3 are selected again. Event C occurs at time t_2 since T_4 hits the oblique side (NNLD) of the T-N plane. We ingeminate the rescheduling for each event.

5. T-N Plane Transformation

We propose "T-N Plane Transformation," which is a technique to apply processor frequency scaling to LNREF scheduling. Figure 4 shows a T-N plane with frequency $\alpha_k = 0.5$. Selected tokens move diagonally down along the NNLD of the transformed T-N plane. We do not consider the case where the taskset is not feasible; therefore we assume that $U \leq M$. If α_k is given to the processor P_k, the voltage V_k is uniquely defined (i.e., $V_k = g(\alpha_k)$). Therefore voltage and frequency scaling is equivalent to a frequency decision. It is difficult to formulate V_k generally since V_k depends on the system architecture. Aydin and Yang [3] show the following theorem.

Theorem 3 (Aydin and Yang) *A task assignment that evenly divides the total utilization U among all the processors will minimize the total energy consumption.* ■

Consequently we have only to keep in mind the frequency.

In the following sections, we present two RT-SVFS techniques for different types of systems. SMT processors share resources among threads; therefore we can only control the voltage and frequency uniformly among threads. On the other hand, we can control the voltage and frequency independently among processors in most of CMP processors and Symmetric Multiprocessing (SMP) processors. We first show a uniform RT-SVFS technique targeting for SMT processors. Then an independent RT-SVFS technique targeting for CMP processors and SMP processors is constructed upon the uniform RT-SVFS technique.

Algorithm: DecideUniformFrequency
1: **foreach** 1...M **as** k
2: $\alpha_k = \max\{U_{\max}, U/M\}$
3: **end foreach**

Figure 5. Uniform frequency scaling.

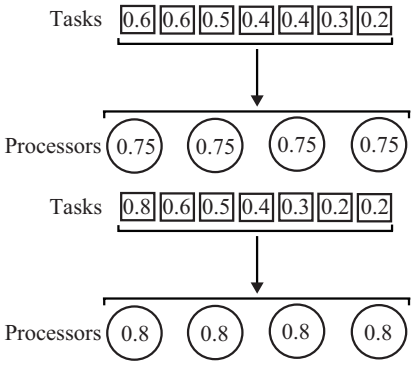

Figure 6. Examples of uniform frequency scaling.

5.1. Uniform RT-SVFS

We assume that all processors have the same frequency $\alpha(= \alpha_1 = \ldots = \alpha_M)$. Corollary 4 shows the condition where all tasks are feasible on the restriction.

Corollary 4 *Any periodic taskset **T** with utilization $U \leq \alpha M$ and $U_{\max} \leq \alpha$ will be scheduled to meet all deadlines on M processors with frequency α by LNREF.*
Proof *All tokens are on the T-N plane at time t_0, based on Theorem 1 since $U_{\max} \leq \alpha$. The subsequent proof is in a similar fashion as Theorem 2 [7]. See our previous work [9] for more detail.* ■

Figure 5 shows the uniform frequency scaling algorithm. All tokens are feasible on the transformed T-N plane, based on Corollary 4. The algorithm is theoretically optimal as a static approach if the voltage and frequency can be controlled only uniformly among threads or processors.

Corollary 5 *DecideUniformFrequency is an optimal real-time static voltage and frequency scaling algorithm if the frequency can be controlled only uniformly among processors.*
Proof *If $U > \alpha M$ or $U_{\max} > \alpha$, tasks miss their deadlines. Therefore DecideUniformFrequency is optimal.* ■

Additionally DecideUniformFrequency differs from the independent frequency scaling algorithm described in the next section in the sense that DecideUniformFrequency is also optimal on practical systems in obvious, which can not control processor frequency continuously.

Algorithm: DecideIndependentFrequency

Require: $u_1 \geq u_2 \geq \ldots \geq u_N$
$\mathbf{T}^{\text{heavy}} = \phi$
$\mathbf{T}^{\text{light}} = \mathbf{T}$
foreach $1...M$ **as** i
  if $L < M$ **and** $U^{\text{light}}_{\max} > U^{\text{light}}/(M-L)$ **then**
    $\mathbf{T}^{\text{heavy}} = \mathbf{T}^{\text{heavy}} \cup \{T_i\}$
    $\mathbf{T}^{\text{light}} = \mathbf{T}^{\text{light}} \backslash \{T_i\}$
  else
    break
  end if
end foreach
foreach 1...M **as** k
  if P_k executes a heavy task T_k **then**
    $\alpha_k = u_k$
  else
    $\alpha_k = U^{\text{light}}/(M-L)$
  end if
end foreach

Figure 7. Independent frequency scaling.

Figure 6 shows examples of DecideUniformFrequency. There are seven tasks and four processors for each example. Since $U = 3$ and $M = 4$ in both examples, the ideal frequency for each processor is $U/M = 0.75$ as shown in the upper of the figure. However there is the bottleneck task with utilization $U_{\max} = 0.8$ in the lower of the figure. If the processor frequency can be controlled independently among processors, we can overcome the problem.

5.2. Independent RT-SVFS

We can overcome the bottleneck shown in the previous section if each processor frequency α_k can be controlled independently among processors. The strategy for independent RT-SVFS is analogous to EKG [2]. We classify tasks into two types of categories (i.e., *heavy* and *light*). Each heavy task T_i is exclusively executed on one processor P_k with frequency $\alpha_k = u_i$. All heavy tasks meet their deadlines in obvious. All light tasks are executed on the other processors by LNREF with DecideUniformFrequency.

We show the definitions for *heavy* and *light*. $\mathbf{T}^{\text{heavy}}$ and $\mathbf{T}^{\text{light}}$ denote the sets of heavy tasks and light tasks, respectively. The light taskset utilization is defined as $U^{\text{light}} = \sum_{T_i \in \mathbf{T}^{\text{light}}} u_i$. $U^{\text{light}}_{\max} = \max\{u_i | T_i \in \mathbf{T}^{\text{light}}\}$ denotes the maximum utilization of the light taskset. The number of heavy tasks is represented as $L = |\mathbf{T}^{\text{heavy}}|$. We assume that heavy tasks are executed on the processors $(P_1, \ldots, P_L)$. The number of processors for light tasks is $M - L$. α^{light} denotes the processor frequency for the light taskset.

All tasks are classified into *heavy* or *light* as shown in Figure 7. We first sort tasks in decreasing utilization and assume that all tasks are *light*. In the assumption, all tasks are feasible by DecideUniformFrequency. Then we decide whether a light task T_i with utilization $u_i = U^{\text{light}}_{\max}$ can be classified into *heavy* without missing any deadlines. Lemma 6 shows the condition where the light task T_i with utilization $u_i = U^{\text{light}}_{\max}$ can be classified into *heavy*. We define that $Z(x)$ represents the Z's value at the time when $L = x$, where Z is an arbitrary symbol.

Lemma 6 *If $U^{light}_{\max}(x) > U^{light}(x)/(M-x)$, the light task T_i with utilization $u_i = U^{light}_{\max}(x)$ can be classified into heavy without missing any deadlines.*

Proof *The proof is shown by the inductive method. Since all tasks are light at first, they are feasible by LNREF with DecideUniformFrequency; namely $\alpha^{light}(0) \leq 1$. We assume that all tasks are feasible at the time when $L = x$; then we show that they are also feasible at the time when $L = x+1$ if $U^{light}_{\max}(x) > U^{light}(x)/(M-x)$. Since all heavy tasks are feasible in obvious, we have only to keep in mind whether all light tasks are feasible. If $\alpha^{light}(x+1) \leq 1$, all light tasks are feasible by LNREF with DecideUniformFrequency after the light task T_i with utilization $u_i = U^{light}_{\max}(x)$ is classified into heavy. Therefore we prove that $\alpha^{light}(x+1) \leq 1$. $\alpha^{light}(x)$ and $\alpha^{light}(x+1)$ are calculated by DecideUniformFrequency as follows.*

$$\alpha^{light}(x) = \max\{U^{light}_{\max}(x), \frac{U^{light}(x)}{M-x}\}$$
$$\alpha^{light}(x+1) = \max\{U^{light}_{\max}(x+1), \frac{U^{light}(x) - U^{light}_{\max}(x)}{M-(x+1)}\}$$

There is the strong evidence that

$$U^{light}_{\max}(x) \geq U^{light}_{\max}(x+1). \quad (1)$$

Based on the assumption $U^{light}_{\max}(x) > U^{light}(x)/(M-x)$,

$$\frac{U^{light}(x)}{M-x} - \frac{U^{light}(x) - U^{light}_{\max}(x)}{M-(x+1)}$$
$$= \frac{U^{light}_{\max}(x)\,(M-x) - U^{light}(x)}{(M-x)\,(M-(x+1))} > 0$$
$$\Rightarrow \frac{U^{light}(x)}{M-x} > \frac{U^{light}(x) - U^{light}_{\max}(x)}{M-(x+1)}. \quad (2)$$

From the inequalities (1) and (2), we obtain $\alpha^{light}(x) \geq \alpha^{light}(x+1)$. At the beginning, we assumed that all tasks are feasible when $L = x$ (i.e., $\alpha^{light}(x) \leq 1$). Consequently $\alpha^{light}(x+1) \leq 1$. ■

Lemma 6 and the proof imply that (a) α_k is monotonically decreasing for all k (i.e., $\alpha_k(x) \geq \alpha_k(x+1)$) at the time when $U^{\text{light}}_{\max}(x) > U^{\text{light}}(x)/(M-x)$, and (b) the repetition of the classification leads the condition to $U^{\text{light}}_{\max}(x) \leq U^{\text{light}}(x)/(M-x)$. After the condition $U^{\text{light}}_{\max}(x) \leq U^{\text{light}}(x)/(M-x)$ is satisfied, we may be able

to classify the light task T_i with utilization $u_i = U^{\text{light}}_{\max}(x)$ into *heavy*. The problem is when we stop the classification. We define the energy consumption as $E'(x)$ at the time when $L = x$. The minimum $E'(x)$ based on the classification are shown in the following theorem.

Theorem 7 *We assume that the condition of $U^{light}_{\max}(w) \leq U^{light}(w)/(M-w)$ is satisfied for the first time when $L = w$. The minimum energy consumption is $E'(w)$.*

Proof *The proof of Lemma 6 implies that α_k is monotonically decreasing for all k (i.e., $\alpha_k(x) \geq \alpha_k(x+1)$) at the time when $x \leq w$. Therefore the minimum energy consumption is $E'(w)$ when $x \leq w$. Meanwhile, when $x \geq w$, the sum of the processor frequency is unchanged from U. However the processor frequency α_{x+1} decreases from*

$$\alpha_{x+1}(x) = U^{light}/(M-x)$$

to

$$\alpha_{x+1}(x+1) = U^{light}_{\max}(x)$$

when $x \geq w$, since the condition where $U^{light}(x)/(M-x) \geq U^{light}_{\max}(x)$ is already satisfied. On the other hand, the other processor frequency $\alpha_{x+2}, \ldots, \alpha_M$ for light tasks increases from

$$\alpha_y(x) = U^{light}/(M-x)$$

to

$$\alpha_y(x+1) = \frac{U^{light}(x) - U^{light}_{\max}(x)}{M-(x+1)},$$

where $x+2 \leq y \leq M$ for all y. From the inverse of the inequality (2), we have $\alpha_y(x) \leq \alpha_y(x+1)$, where $x+2 \leq y \leq M$ for all y. It results in differences between α_{x+1} and $(\alpha_{x+2}, \ldots, \alpha_M)$ after the classification, while α_{x+1} equal to $(\alpha_{x+2}, \ldots, \alpha_M)$ before the classification. Based on Theorem 3, the energy consumption is monotonically increasing when $x \geq w$. Consequently the minimum energy consumption is $E'(w)$. ■

The algorithm is theoretically optimal as follows.

Theorem 8 *DecideIndependentFrequency is an optimal real-time static voltage and frequency scaling algorithm if the frequency can be controlled independently among processors.*

Proof *The proof is shown by contraposition. We enumerate all the possibilities with focusing attention on the task T_m with utilization $u_m = U_{\max}$. Because the task T_m has maximum utilization $U_{\max}$ in the system, T_m must be executed on the processor P_k with frequency $\alpha_k = u_m$ as shown in Theorem 3. Therefore T_m is executed on either a processor or more than one processor. (1) If T_m is executed on a processor, the algorithm is the same as DecideIndependentFrequency. (2) If T_m is executed on more than one processor, it produces the same results that T_m is scheduled by partitioned scheduling. Namely a processor and T_m can be removed from the global scheduled group, and T_m is executed on the processor. (1) and (2) show that partitioned T_m's scheduling leads us to the optimal voltage and frequency scaling. Furthermore we take it from the top with removing T_m and the processor executing T_m. The technique is DecideIndependentFrequency. Consequently no static algorithm achieves lower energy consumption than DecideIndependentFrequency.* ■

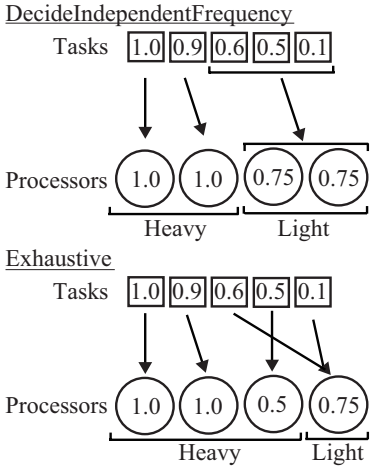

Figure 8. Examples of independent frequency scaling on a practical system.

DecideIndependentFrequency is not optimal on practical environments such as System1 on Table 1 as shown in Figure 8. The numbers show task utilization and processor frequency. When $w = 2$, the condition of $U^{\text{light}}(2)/(M-2) \geq U^{\text{light}}_{\max}(2)$ is satisfied; therefore two tasks with utilization 1.0 and 0.9 are classified into *heavy* in DecideIndependentFrequency. On the other hand, Exhaustive shown in the next section achieves less energy consumption than DecideIndependentFrequency. However Exhaustive can not accommodate to dynamic environments because it is NP-hard.

5.3. An Exhaustive Algorithm

Exhaustive is a non-optimal RT-SVFS algorithm on practical environments; however Overcoming Exhaustive will be complex. Exhaustive achieves less energy consumption than DecideIndependentFrequency since the voltage and frequency setting generated by DecideIndependentFrequency is certainly scanned by Exhaustive. Exhaustive is NP-hard since it leverages the notion of "partition of a set." $\mathbf{X}$ represents the set of all partitions of $\mathbf{P}$. For each partition of $\mathbf{X}$, each part of the partition is called *virtual processor*. If a virtual processor includes only one processor, the tasks

Table 1. Systems for simulation.

System 1		System 2		System 3	
α	V	α	V	α	V
0.5	3	0.5	3	0.36	1.4
0.75	4	0.75	4	0.55	1.5
1.0	5	0.83	4.5	0.64	1.6
		1.0	5	0.73	1.7
				0.82	1.8
				0.91	1.9
				1.0	2.0

executed on the virtual processor are scheduled by EDF and RT-SVFS for EDF [15]. Otherwise the tasks executed on the virtual processor are scheduled by LNREF and DecideUniformFrequency. For each partition of **P**, the number of parts is defined as V, and **Y** represents the set of all partition of **T**, where the number of parts must be equal to V. For each partition of **Y**, each part of the partition is called *co-scheduled tasks*. All combinations of virtual processor and co-scheduled tasks are scanned to find a good combination.

6. Simulation

We evaluate DecideIndependentFrequency on practical environments shown in Table 1. Each system has the operable sets of frequency α and voltage V as shown in the table. The normalized energy consumption is

$$Power = \frac{\sum_{P_k \in \mathbf{P}} \alpha_k V_k^2}{M V_{\max}^2},$$

where $V_{\max}$ is the maximum voltage in the system.

Three algorithms are compared. **DecideFrequency** represents DecideIndependentFrequency proposed in this paper. **Exhaustive** solves the NP-hard partition problem. **NoVFScaling** does not control voltage and frequency at all. **bound** represents the theoretical lower bound which reflects taskset utilization only and does not consider real-time constraints. The other RT-SVFS algorithms proposed in previous papers can not be compared since they can not guarantee the schedulability in higher utilization.

6.1. Simulation Setup

We evaluate the average *Power* of 1000 tasksets for each taskset's utilization in the range [0.5,4.0] at intervals of 0.25 on four processors. Each taskset with the target utilization U is generated as follows. We first assume that the taskset is empty. Then we push a task into the taskset until U is satisfied. Each task T_i is generated with the period p_i in the integer range $[1, 100]$ and the execution time c_i in the integer range $[1, p_i]$. Consequently the tasks have identically-distributed utilization. If the utilization of the taskset becomes over U, then we discard the last task and generate the new task which satisfies U. Finally *Power* is calculated.

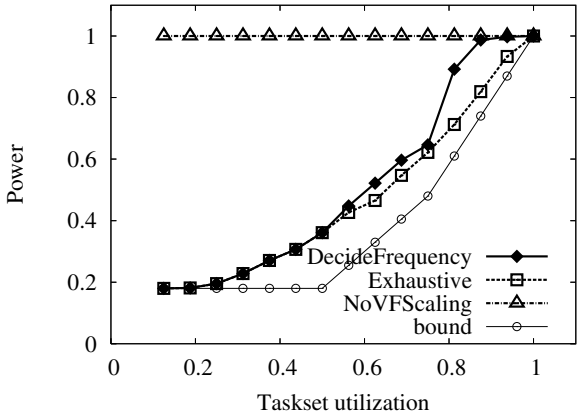

Figure 9. Power on System 1.

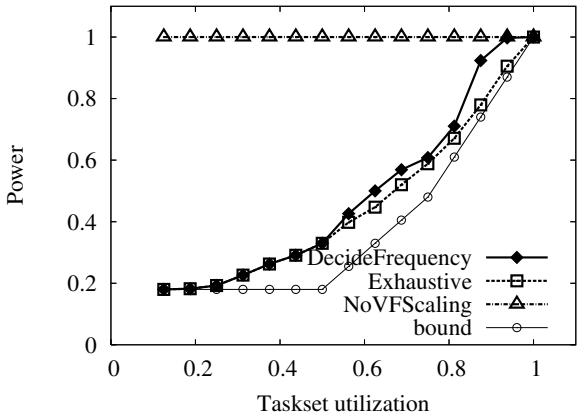

Figure 10. Power on System 2.

6.2. Simulation Results

Figures 9, 10, and 11 show the results for each system. The figures represent system utilization U/M on the horizontal axis and *Power* on the vertical axis. The maximum differences between DecideFrequency and Exhaustive are 0.172 at system utilization 0.8125 on System 1, 0.165 at system utilization 0.875 on System 2, and 0.043 at system utilization 0.9375 on System3. Namely DecideFrequency closely approaches the lower bound on energy consumption since DecideFrequency is optimal theoretically.

There exist some points at which the difference is smaller than at circumjacent points such as at system utilization 0.75 on System 1, and at system utilization 0.75 and 0.8125 on System 2 because we can select the frequency close to α

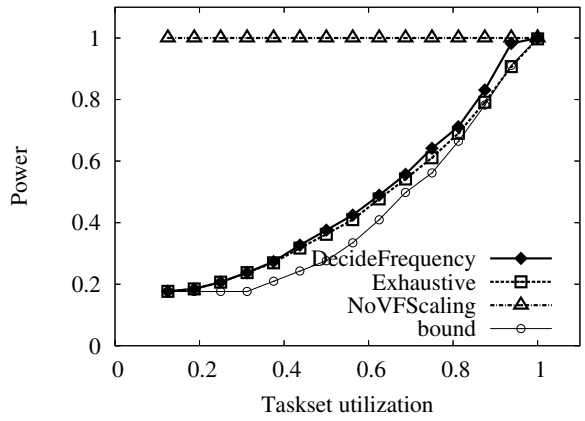

Figure 11. Power on System 3.

such as 0.75 on System 1, and, 0.75 and 0.83 on System 2, where α represents selectable one of frequencies shown in Table 1. Namely it is highly possible that energy consumption is minimized if the system utilization U/M is designed to be close to α.

7. Conclusions and Future Work

In this paper, we present two real-time static voltage and frequency scaling (RT-SVFS) techniques for multiprocessors. Uniform one is optimal both theoretically and practically. On the other hand, independent one is optimal only theoretically; however it can solve the problem in polynomial time, while the exhaustive algorithm is NP-hard. Real-time dynamic voltage and frequency scaling (RT-DVFS) is a topic for the future work for more energy efficiency.

References

[1] J. H. Anderson and A. Srinivasan. Early-Release Fair Scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[2] B. Andersson and E. Tovar. Multiprocessor Scheduling with Few Preemptions. In *Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, Aug. 2006.

[3] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Proc. of the 17th IEEE International Parallel and Distributed Processing Symposium*, pages 22–26, Sept. 2003.

[4] T. P. Baker. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768, Aug. 2005.

[5] T. D. Burd and R. W. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proc. of the 28th Annual Hawaii International Conference on System Sciences*, pages 288–297, Jan. 1995.

[6] J.-J. Chen and T.-W. Kuo. Allocation Cost Minimization for Periodic Hard Real-Time Tasks in Energy-Constrained DVS Systems. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, pages 255–260, Nov. 2006.

[7] H. Cho, B. Ravindran, and E. D. Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proc. of the 27th IEEE Real-Time Systems Symposium*, pages 101–110, Dec. 2006.

[8] H. Cho, B. Ravindran, and E. D. Jensen. Synchronization for an Optimal Real-Time Scheduling Algorithm on Multiprocessors. In *Proc. of the 2nd IEEE International Symposium on Industrial Embedded Systems*, pages 9–16, 2007.

[9] K. Funaoka, S. Kato, and N. Yamasaki. Real-Time Static Voltage Scaling on Multiprocessors. In *Proc. of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 142–149, Nov. 2007.

[10] S. Gochman, R. Ronen, I. Anati, A. Berkovis, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R. C. Valentine. The Intel Pentium M Processor: Microarchitecture and Performance. *Intel Technology Journal*, 7(2):21–36, May 2003.

[11] P. Holman and J. H. Anderson. Adapting Pfair Scheduling for Symmetric Multiprocessors. *Journal of Embedded Computing*, 1(4):543–564, May 2005.

[12] C. H. Lee and K. G. Shin. On-Line Dynamic Voltage Scaling for Hard Real-Time Systems Using the EDF Algorithm. In *Proc. of the 25th IEEE Real-Time Systems Symposium*, pages 319–335, Dec. 2004.

[13] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, pages 46–61, Jan. 1973.

[14] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 25–33, June 2000.

[15] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[16] D. Shu, R. Melhem, and B. R. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):686–700, July 2003.

[17] A. Srinvasan, P. Holman, and J. Anderson. Integrating Aperiodic and Recurrent Tasks on Fair-scheduled Multiprocessors. In *Proc. of the 14th Euromicro Conference on Real-Time Systems*, pages 19–28, 2002.

[18] J. A. Stankovic, C. Lu, and S. H. Son. The Case for Feedback Control Real-Time Scheduling. In *Proc. of the 11th Euromicro Conference on Real-Time Systems*, pages 11–20, June 1999.

[19] C. Xian, Y.-H. Lu, and Z. Li. Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time. In *Proc. of the 44th ACM/IEEE Design Automation Conrefence*, pages 664–669, 2007.