Leakage-Aware Energy-Efficient Scheduling of Real-Time Tasks in Multiprocessor Systems*

Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106, ROC.
Emails:{r90079, b89108, ktw}@csie.ntu.edu.tw

Abstract

This work targets energy-efficient scheduling of periodic real-time tasks over multiple DVS processors with the considerations of power consumption due to leakage current. A polynomial-time algorithm with a 1.283 *approximation bound is proposed when the overheads in turning on/off a processor are negligible. When the overheads are non-negligible, we develop polynomial-time algorithms with a* 2 *approximation bound. A series of simulation experiments was done for the performance evaluation of the proposed algorithms. The simulation results show that the proposed algorithms could derive schedules very close to optimal solutions.*

Keywords: Leakage-aware scheduling, Real-time and embedded systems, and Task partitioning.

1 Introduction

With the advance technology of VLSI circuit designs, a modern processor might operate at different supply voltages. Technologies, such as Intel SpeedStep® and AMD PowerNOW!™, provide dynamic voltage scaling for laptops to prolong the battery lifetime. Different supply voltages lead to different execution speeds on a dynamic voltage scaling processor. Well-known example processors for embedded systems are Intel StrongARM SA1100 processor and the Intel XScale. The power consumption of processors in the dynamic voltage scaling part is a convex and increasing function of processor speeds. The lower the speed, the less the power consumption of the dynamic voltage scaling part is. Because a lower processor speed in executing a task might stretch its execution time, the energy consumption resulting from leakage current will increase.

In the past decades, energy-efficient task scheduling with various deadline constraints has received a lot of attention. Many studies have been done for uniprocessor scheduling [4, 6, 10, 11, 22]. Implementations of real-time systems with multiple processors are often more energy-efficient than those with a single processor [2], because of the convexity of power consumption functions. Various heuristics were proposed for energy consumption minimization under different task models in multiprocessor environments [1, 5, 7, 9, 17, 21, 23, 24]. In particular, several energy-efficient scheduling solutions are proposed based on list heuristics [9, 23] for real-time jobs with precedence constraints. There are also heuristics for periodic tasks in multiprocessor environments [1, 5]. Zhu, et al. [24] explored on-line task scheduling with reclamation of slacks from early completion of tasks during the run time. Mishra, et al. [17] explored energy-efficient scheduling issues with the considerations of the communication delay of tasks. In addition to the considerations of energy-efficient scheduling, Anderson and Sanjoy [2] explored the trade-off between the total energy consumption of task executions and the number of required processors, where all of the tasks in the proposed solutions run at the same speed. There are some approximation algorithms for scheduling frame-based tasks in the minimization of energy consumption, where tasks share the same power consumption function [7] or are allowed to have different power consumption functions [8]. Energy-efficient multiprocessor scheduling of frame-based task sets was also explored for chip-multiprocessor (CMP) architectures, in which cores, i.e., processors, on a chip must share the same processor speed at any given time moment [21].

Recently, researchers have started exploring energy-efficient scheduling with the considerations of the non-negligible power consumption of leakage current for current and future circuit manufacture process [12]. In such a direction, a processor might be turned off (or into a dormant mode) whenever needed. For uniprocessor scheduling of aperiodic real-time tasks, Irani, et al. [10] proposed a 2-approximation algorithm for the minimization of energy consumption with the considerations of leakage current, where an α-approximation algorithm guarantees to derive a solution with the energy consumption at most α times of an optimal

*Support in parts by research grants from ROC National Science Council NSC-94-2213-E-002-007 and NSC-94-2219-E-002-013.

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

solution. For periodic real-time tasks, Jejurikar, et al. [12] and Lee, et al. [13] proposed energy-efficient strategies on a uniprocessor by applying the procrastination scheduling to decide when to turn the processor into the dormant mode. Xu, et al. [20] considered multiprocessor leakage-aware scheduling for the determination of the number of activated processors.

This paper considers energy-efficient scheduling of periodic real-time tasks over homogeneous multiple processors. The power consumption function of the dynamic voltage scaling part of the processor is modeled as s^3 [6, 10, 22], where s is the execution speed. The power consumption resulting from the leakage current is assumed being a constant β [20], and the power consumption function $P()$ is modeled as $P(s) = s^3 + \beta$. We consider the possibility of turning each processor into the dormant mode, whenever needed, to save the energy consumption. Suppose that the activation of a processor from the dormant mode could be done instantly but might require additional energy consumption. A polynomial-time 1.283-approximation algorithm is proposed for the minimization of energy consumption and the satisfaction of task timing constraints, when the overheads in turning on/off a processor are negligible, and there is no upper bound on the processor speeds. When the overheads are non-negligible, optimal solutions might require more than one processor for energy minimization, and the minimum available speed is 0, a polynomial-time 1.667-approximation algorithm is proposed to partition tasks on processors in an off-line manner and to determine the activation/dormant time in an on-line fashion. When the minimum available speed is more than 0, the proposed algorithm has a 2 approximation bound. When there is an upper bound on processor speeds, we take an artificial-bound approach to minimize the energy consumption, which is motivated by the constraint violation study introduced in [14].

The rest of this paper is organized as follows: Section 2 defines the leakage-aware multiprocessor energy-efficient scheduling problem. Section 3 presents our approximation algorithm when the overheads in turning on/off a processor are negligible. When the overheads are non-negligible, approximation algorithms are then proposed in Section 4. Experimental results for the performance evaluation of the proposed algorithms are in Section 5. Section 6 is the conclusion.

2 System Models and Problem Definitions

Processor models We explore energy-efficient scheduling over M homogeneous dynamic voltage scaling (DVS) multiprocessors, where the power consumption function of each task remains the same for every processor. The dynamic power consumption function $P_d()$ of the dynamic voltage scaling part of the processor is a function of the adopted processor speed s [18]:

$$P_d(s) = C_{ef} V_{dd}^2 s, \tag{1}$$

where $s = \kappa \frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, and κ denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $\kappa > 0$, and $C_{ef} > 0$). $P_d()$ is a convex and increasing function of processor speeds.[1] When V_t is 0, the dynamic power consumption function $P_d(s)$ can be rephrased as a cubic function of the processor speed s. As discussed in the literature, e.g., [6, 10, 22], the dynamic power consumption function can be phrased as a function proportional to s^σ, where σ is a constant between 2 and 3. To simplify the presentation, we focus our discussions on the case that $P_d(s) \propto s^3 = \beta_1 s^3$, which was adopted in many previous research results, e.g., [17, 20, 21]. Note that the to-be-proposed algorithms in this paper could also be applied to systems with $P_d(s) \propto s^\sigma$ for any σ between 2 and 3.

The leakage power of each of the processor is a non-negative constant, denoted as β_2. The power consumption function is the summation of the dynamic power consumption and the leakage power. The power consumption function $P()$ of a processor in this paper is $(\beta_1 s^3 + \beta_2)$, which is as the same as the continuous power consumption function in [20]. The power consumption function is normalized as follows:

$$P(s) = s^3 + \beta, \tag{2}$$

where β is a non-negative constant. Each processor could be turned into the *dormant* mode (or turned off) independently. The power consumption of a processor is treated as 0 when it is in the dormant mode [10, 12]. We consider processors that could be turned on/off at instant. When needed, turning the processor into the dormant mode might further reduce the energy consumption. The energy consumption to turn off a processor is assumed being negligible, but it might require additional energy for the processor to be turned on for operations [10].[2] We denote E_{sw} as the energy of the *switching overheads* from the dormant mode to the *active* mode of a processor. For the rest of this paper, we say that a processor is *idle* at time instant t, if it does not perform any execution at time instant t.

In this study, we assume that each processor may operate at any speed in $[S_{\min}, S_{\max}]$, and the speed of each processor can be adjusted independently. The number of CPU cycles executed in a time interval is linear of the processor speed, and the energy consumed for a processor in the execution of a task at the processor speed s for t time units is $t \cdot P(s)$. Let the number of CPU cycles completed for a task running at a speed s for t time units be $s \cdot t$.

[1] A function $f()$ is convex if $f(\gamma x + (1-\gamma)y) \leq \gamma f(x) + (1-\gamma) f(y)$ for any x, y, and $\gamma x + (1-\gamma)y$ in the domain of $f()$ and $0 \leq \gamma \leq 1$.

[2] For processors with overheads to be turned off, the overheads could be treated as part of the overheads to turn on the processor. Our algorithms and their analysis also work well for systems with timing overheads in turn-on/off. The procedure could be slightly modified with the same analytical results. The detail is not included, due to the space limitation.

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

Task models Tasks under discussions in this paper are periodic and independent in executions. A periodic task is an infinite sequence of task instances, referred to as *jobs*, where each job of a task comes in a regular period [15, 16]. Each task τ_i is associated with its initial arrival time (denoted as a_i), its computation requirement in CPU cycles (denoted as c_i), and its period (denoted as p_i). Note that c_i denotes the maximum number of CPU cycles required to complete the execution of any job of τ_i. Given a set $\mathbf{T}$ of tasks, the *hyper-period* of $\mathbf{T}$, denoted as L, is defined as the least common multiple (LCM) of the periods of tasks in $\mathbf{T}$. Let the relative deadline of each task τ_i be equal to its period p_i in this paper. That is, the arrival time and deadline of the j-th job of task τ_i are $a_i + (j-1) \cdot p_i$ and $a_i + j \cdot p_i$, respectively. Throughout this paper, we focus our discussions on the case in which all of the tasks arrive at time 0. The number of jobs in the hyper-period of task τ_i is $\frac{L}{p_i}$.

Problem Definition We define the problem explored in this paper as follows:

Definition 1 *Leakage-Aware Multiprocessor Energy-Efficient Scheduling (*LAMS *problem):*

Consider a set $\mathbf{T}$ *of independent tasks over* M *identical processors with a common power consumption function* $P(s) = s^3 + \beta$*, where* $\beta \geq 0$*, and all tasks in* $\mathbf{T}$ *are ready at time* 0*. Each periodic task* $\tau_i \in \mathbf{T}$ *is associated with a computation requirement in* c_i *CPU-cycles and a period* p_i*, where the relative deadline of* τ_i *is* p_i*. The energy of the switching overheads from the dormant mode to the active mode of a processor is* E_{sw}*, and a processor can operate at any speed in* $[S_{\min}, S_{\max}]$*. The problem is to minimize the energy consumption in the hyper-period* L *of tasks in* $\mathbf{T}$ *in the scheduling of tasks in* $\mathbf{T}$ *without missing the timing constraints, where each task is executed entirely on a processor.* □

A *schedule* of a task set is a mapping of the executions of the tasks in the set to processors in the system with an assignment of processor speeds for the corresponding time intervals of the tasks. A schedule is *feasible* if all processor speeds assigned for its time intervals are valid, no task misses its timing constraint, and each task is executed entirely on a processor. The energy consumption of a schedule SC is denoted as $\Phi(SC)$. A schedule is optimal if it is feasible, and its energy consumption is equal to the minimum energy consumption of all feasible schedules. The following statement shows the hardness of the LAMS problem.

Lemma 1 *The* LAMS *problem is strongly* $\mathcal{NP}$*-hard even when* $S_{\max}$ *is* ∞*, and* E_{sw} *and* β *are both* 0*.*

Proof. It comes directly from a special case when $S_{\max}$ is ∞, E_{sw} and β are both 0, and all of the tasks share a common arrival time and deadline [7]. □

Due to the $\mathcal{NP}$-hardness of the LAMS problem, we focus the study on approximation algorithms with a worst-case guarantee on the energy consumption. Based on [19], a polynomial-time α-approximation algorithm for the LAMS problem must have a polynomial-time complexity of the input size and could derive a feasible solution with the energy consumption at most α times of an optimal solution, for any input instance, in which α is also referred to as the *approximation ratio (bound)* of the approximation algorithm.

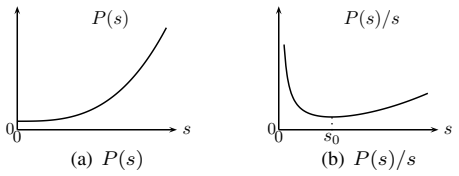

Figure 1. An illustrative example for $P(s)$ and $P(s)/s$

3 Approximation Algorithm for Negligible Switching Overheads

This section first considers the case in which there is no upper bound $S_{\max}$ on the available processor speeds for task executions, i.e., $S_{\max} = \infty$. An approximation algorithm is proposed with a guaranteed approximation ratio. When systems with a maximum bound on the available processor speeds are considered, i.e., $S_{\max} \neq \infty$, we could show that the proposed algorithm can bound the maximum processor speed by a constant factor.

3.1 Results for the Power Consumption Function

Critical speed Because $P(s)$ is a convex and increasing function of the processor speed s, an optimal solution for scheduling tasks in $\hat{\mathbf{T}}$ on a processor must execute at a common speed s [4]. The energy consumption in the hyper-period L to execute all of the tasks in $\hat{\mathbf{T}}$ at speed s is $P(s)(\sum_{\tau_i \in \hat{\mathbf{T}}} \frac{Lc_i}{p_i})/s$, which is proportional to $\frac{P(s)}{s}$. Although $P()$ is a convex and increasing function, $\frac{P(s)}{s}$ is merely a convex function. By solving $\frac{\mathrm{d}(P(s)/s)}{\mathrm{d}s} = 0$, we know that $P(s)/s$ is minimized when s is equal to $\sqrt[3]{\frac{\beta}{2}}$. As a result, no task would execute at any speed lower than $\max\{\sqrt[3]{\frac{\beta}{2}}, S_{\min}\}$, referred to as the *critical speed* s_0 for the rest of this paper. (We focus our study on the case $s_0 \leq S_{\max}$.) In other words, executing any task at any speed less than s_0 would either consume more energy than that at s_0 or violate the speed constraint. The function $P(s)$ and the function $P(s)/s$ are illustrated in Figure 1.

Let SC be a feasible schedule of $\mathbf{T}$ for the LAMS problem. SC_m is the partial schedule of SC on the m-th processor, and $\mathbf{T}_m$ denotes the set of tasks assigned to execute on the m-th processor. In other words, $\cup_{m=1}^{M} \mathbf{T}_m = \mathbf{T}$ and $\mathbf{T}_m \cap \mathbf{T}_n = \emptyset$ for any $m \neq n$. As shown in [15], the earliest-deadline-first (EDF) scheduling algorithm is an optimal uniprocessor

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

scheduling algorithm for independent real-time tasks. When the total utilization of a task set is no more than 100%, the task set is schedulable on a uniprocessor by applying the EDF scheduling algorithm, where the *utilization* of a task is defined as its execution time divided by its period. When E_{sw} is 0, there must exist an optimal schedule $\hat{SC}$ that partitions $\mathbf{T}$ into $\hat{\mathbf{T}}_1, \hat{\mathbf{T}}_2, \ldots, \hat{\mathbf{T}}_M$ disjoint subsets with the following two properties for any partial schedule $\hat{SC}_m$ of $\hat{SC}$ with $1 \leq m \leq M$:

Lemma 2 *(1) For every task τ_i in $\hat{\mathbf{T}}_m$, all of the jobs of τ_i are executed at one common processor speed. (2) The minimum energy consumption schedule would execute all of the tasks in $\hat{\mathbf{T}}_m$ at speed s_0 if $\sum_{\tau_i \in \hat{\mathbf{T}}_m} \frac{c_i}{p_i} \leq s_0$, or at speed $\sum_{\tau_i \in \hat{\mathbf{T}}_m} \frac{c_i}{p_i}$, otherwise.*

Proof. This lemma comes directly from the convexity of the power consumption function and the optimality of the energy consumption at the critical speed. The proof is similar to that in [3, 4]. □

Energy Consumption For brevity, let $\phi(\mathbf{T}_m)$ be the minimum energy consumption in the hyper-period L to complete all of the tasks in $\mathbf{T}_m$ in time on the m-th processor. Since E_{sw} is 0 in this section, an optimal schedule would turn a processor into the dormant mode if the processor is idle. By Lemma 2, if $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i} \leq s_0$, $\phi(\mathbf{T}_m)$ is $\frac{L}{s_0} \cdot (\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}) \cdot P(s_0)$; otherwise, $\phi(\mathbf{T}_m)$ is $L \cdot P(\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i})$. The energy consumption in completing a set of tasks $\mathbf{T}_m$ in time with *load* $\ell = \sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}$ could be calculated by a function $\psi(\ell)$, where

$$\psi(\ell) = \begin{cases} L(\ell^3 + \beta), & \text{if } \ell > s_0; \\ \frac{\ell}{s_0} L(s_0^3 + \beta), & \text{otherwise.} \end{cases} \quad (3)$$

The following lemmas resulting from the convexity of the power consumption function will be widely used for algorithmic analysis in this section:

Lemma 3 *$\psi(\gamma x + (1-\gamma)y) \leq \gamma\psi(x) + (1-\gamma)\psi(y)$, for any non-negative reals x, y and $0 \leq \gamma \leq 1$.*

Lemma 4 *Suppose that $\ell_m + \ell_n = \ell'_m + \ell'_n$ and $\ell_m \leq \ell'_m, \ell'_n \leq \ell_n$. Then, $\psi(\ell_m) + \psi(\ell_n) \geq \psi(\ell'_m) + \psi(\ell'_n)$.*

Proof. The proofs of the above lemmas are omitted due to the space limitation. □

Based on Lemma 4, executing each task τ_i on the i-th processor at speed $\max\{S_{\min}, \frac{c_i}{p_i}\}$ results in an optimal schedule when $|\mathbf{T}| \leq M$. For the rest of Section 3, we shall consider only non-trivial cases, i.e., those with $|\mathbf{T}| > M$.

3.2 An Approximation Algorithm when $S_{\max} = \infty$

In this subsection, we present a scheduling algorithm for the LAMS problem when the maximum available processor speed is infinite. Our proposed algorithm shown in Algorithm 1 (Algorithm LA+LTF) adopts the *Largest-Task-First* strategy by considering tasks in a non-increasing order of their *loads*, where the load of task τ_i is defined as c_i/p_i. For frame-based real-time tasks with the same arrival time and a common deadline, the *Largest-Task-First* strategy was proved being a 1.13-approximation algorithm when $P(s) \propto s^3$ and $S_{\min} = 0$ in [7]. In this paper, Algorithm LA+LTF is an enhancement for systems with leakage power consumption considerations.

Algorithm 1 : LA+LTF

Input: $(\mathbf{T}, M)$;

1: sort all tasks in $\mathbf{T}$ in a non-increasing order c_i/p_i for $\tau_i \in \mathbf{T}$;
2: derive the critical speed s_0;
3: set $\ell_1, \ell_2, \cdots, \ell_M$ to 0, and $\mathbf{T}_1, \mathbf{T}_2, \cdots, \mathbf{T}_M$ to $\emptyset$;
4: **for** $i = 1$ to $|\mathbf{T}|$ **do**
5: find the smallest ℓ_m; (break ties arbitrarily with $1 \leq m \leq M$)
6: $\mathbf{T}_m \leftarrow \mathbf{T}_m \cup \{\tau_i\}$ and $\ell_m \leftarrow \ell_m + \frac{c_i}{p_i}$;
7: return the schedule $SC_{\text{LA+LTF}}$ which turns a processor into the dormant mode instantly when it is idle, and executes all of the tasks in $\mathbf{T}_m$ $(1 \leq m \leq M)$ in an earliest-deadline-first order on the m-th processor at speed s_0 if $\ell_m \leq s_0$, or at speed ℓ_m, otherwise;

For the rest of this section, let task set $\mathbf{T}$ be a sorted set in a non-increasing order of the ratio of the computation requirement to the period of each task, i.e., $\frac{c_i}{p_i} \geq \frac{c_j}{p_j}$ if $i < j$. We assume that tasks are indexed in such an order for the simplicity of presentation (for the rest of this section). ℓ_m denotes the *load* on the m-th processor. The load of a processor is defined as the sum of the ratios of the computation requirement to the period of all of the tasks assigned to that processor. Let $\mathbf{T}_m$ denote the set of the tasks assigned to the m-th processor. Algorithm LA+LTF tries to partition $\mathbf{T}$ into $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$. Algorithm LA+LTF assigns a task to the processor with the smallest load among these M processors in the task order in $\mathbf{T}$. After all, we take the task partition $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$ as our solution. The resulting schedule $SC_{\text{LA+LTF}}$ executes all of the tasks in $\mathbf{T}_m$ $(1 \leq m \leq M)$ in an earliest-deadline-first order on the m-th processor at speed s_0 if $\ell_m \leq s_0$, or at speed ℓ_m, otherwise, where ℓ_m is defined as $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}$ for the rest of this section. Once a processor is idle, it is turned into the dormant mode instantly. The time complexity of Algorithm LA+LTF is $O(|\mathbf{T}| \log |\mathbf{T}|)$ since $|\mathbf{T}| > M$. Since the total utilization of the tasks assigned on each processor is at most 100%, the earliest deadline first schedule could complete all of the tasks in time.

We now show the approximation ratio of Algorithm LA+LTF. We first derive a lower bound of the minimum energy consumption. The approximation ratio can then be obtained by comparing $\Phi(SC_{\text{LA+LTF}})$ to the lower bound of optimal solutions for all input instances.

Let k^* be the largest index satisfying $M \leq k^* \leq 2M$ and $\frac{c_{i+M}}{p_{i+M}} \geq \frac{1}{2}\frac{c_{M-i+1}}{p_{M-i+1}}$ for all $1 \leq i \leq k^* - M$. $\mathbf{T}^f$ represents the set of the first k^* tasks of $\mathbf{T}$. Note that, if $|\mathbf{T}^f| < 2M$ and $\mathbf{T} \setminus \mathbf{T}^f \neq \phi$, we know $\frac{c_{|\mathbf{T}^f|+1}}{p_{|\mathbf{T}^f|+1}} < \frac{1}{2}\frac{c_{2M-|\mathbf{T}^f|}}{p_{2M-|\mathbf{T}^f|}}$. We relax

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

the constraint of the LAMS problem so that any task in $\mathbf{T} \setminus \mathbf{T}^f$ could be executed on more than one processor simultaneously. We denote such a relaxed problem as the SEMI-LAMS problem. Unlike the hardness of the LAMS problem, an optimal solution of the SEMI-LAMS problem could be derived efficiently.

Here let us show an optimal schedule of the SEMI-LAMS problem: First, we assign the tasks in $\mathbf{T}^f$ by applying Algorithm LA+LTF. Let ℓ'_m denote the load of the m-th processor after the task assignment is done. Let λ be the positive value that satisfies

$$\sum_{m=1}^{M} \max\{\lambda - \ell'_m, 0\} = \sum_{\tau_i \in \mathbf{T} \setminus \mathbf{T}^f} \frac{c_i}{p_i}. \quad (4)$$

Since task migration and simultaneous execution of a task on multiple processors are allowed for the tasks in $\mathbf{T} \setminus \mathbf{T}^f$, we can distribute the computation of these tasks among the processors. For $1 \leq m \leq M$, if $\lambda > \ell'_m$, we distribute load $\lambda - \ell'_m$ of the tasks in $\mathbf{T} \setminus \mathbf{T}^f$ to the m-th processor, and the resulting ℓ'_m is equal to λ. For the m-th processor, ℓ'_m is either greater than or equal to λ. Let SC^* denote the resulting schedule, where $\Phi(SC^*) = \sum_{m=1}^{M} \psi(\ell'_m)$. We now prove the optimality of SC^* and the relation of the loads in SC^* and $SC_{\text{LA+LTF}}$.

Lemma 5 *SC^* is optimal for the* SEMI-LAMS *problem.*

Proof. For any three tasks τ_i, τ_j, and τ_k in $\mathbf{T}^f$, the following equation holds when (1) k is greater than i, or (2) k is no less than $2M - k^* + 1$:

$$\frac{c_i}{p_i} + \frac{c_j}{p_j} \geq \frac{c_k}{p_k}. \quad (5)$$

As a result, in SC^*, task τ_i in $\mathbf{T}^f$ is assigned to the i-th processor when $i \leq M$ or the $(2M - i + 1)$-th processor when $i > M$. Let SC be any feasible schedule of the SEMI-LAMS problem. We prove this lemma by transforming SC into SC^* without increasing the energy consumption of $\Phi(SC)$. For brevity, let $\mathbf{T}^{f,1}$ be $\{\tau_i \mid 1 \leq i \leq 2M - k^*\}$ and $\mathbf{T}^{f,2}$ be $\{\tau_i \mid 2M - k^* + 1 \leq i \leq k^*\}$.

We first show that we could transform SC into another schedule which assigns at most two tasks in $\mathbf{T}^f$ on a processor. Let $\mathbf{T}^{sc}_m$ and ℓ^{sc}_m be the set of tasks in $\mathbf{T}^f$ and the load assigned on the m-th processor in SC, respectively. Suppose that there are 3 tasks in $\mathbf{T}^{sc}_m$ for some m-th processor. By the pigeon hole principle, there must be another task set $\mathbf{T}^{sc}_{m'}$ consisting of at most one task in $\mathbf{T}^{f,2}$, where $m \neq m'$.

Let task τ_j be any task in $\mathbf{T}^{sc}_m$. Below, we show that the move of τ_j to the m'-th processor along with a load reassignment does not increase the energy consumption. Let $\mathbf{T}^{sc'}_m$ be $\mathbf{T}^{sc}_m \setminus \{\tau_j\}$ and $\mathbf{T}^{sc'}_{m'}$ be $\mathbf{T}^{sc}_{m'} \cup \{\tau_j\}$. By Equation (5), we know that both $\sum_{\tau_i \in \mathbf{T}^{sc'}_m} c_i/p_i$ and $\sum_{\tau_i \in \mathbf{T}^{sc'}_{m'}} c_i/p_i$ are between $\sum_{\tau_i \in \mathbf{T}^{sc}_{m'}} c_i/p_i$ and $\sum_{\tau_i \in \mathbf{T}^{sc}_m} c_i/p_i$. Let $\ell_{m+m'}$ be $\ell^{sc}_m + \ell^{sc}_{m'} - \sum_{\tau_i \in \mathbf{T}^{sc}_m \cup \mathbf{T}^{sc}_{m'}} \frac{c_i}{p_i}$. Let $\lambda'_{m+m'}$ be the value that satisfies $(\max\{\lambda'_{m+m'} - (\sum_{\tau_i \in \mathbf{T}^{sc'}_m} c_i/p_i), 0\} + \max\{\lambda'_{m+m'} - (\sum_{\tau_i \in \mathbf{T}^{sc'}_{m'}} c_i/p_i), 0\}) = \ell_{m+m'}$. $\ell^{sc'}_m$ and $\ell^{sc'}_{m'}$ are $\max\{\lambda'_{m+m'}, \sum_{\tau_i \in \mathbf{T}^{sc'}_m} c_i/p_i\}$ and $\max\{\lambda'_{m+m'}, \sum_{\tau_i \in \mathbf{T}^{sc'}_{m'}} c_i/p_i\}$, respectively. Both of $\ell^{sc'}_m$ and $\ell^{sc'}_{m'}$ are between ℓ^{sc}_m and $\ell^{sc}_{m'}$. As a result, by Lemma 4, $\psi(\ell^{sc}_m) + \psi(\ell^{sc}_{m'}) \geq \psi(\ell^{sc'}_m) + \psi(\ell^{sc'}_{m'})$. We could transform SC into another schedule which assigns at most two tasks in $\mathbf{T}^f$ on a processor without increasing the energy consumption.

By applying Lemma 4 and Equation (5), a similar argument could be made by showing that we could transform SC into SC^* without increasing the total energy consumption. We omit the detail proof due to the space limitation. □

Lemma 6 *For the m^*-th processor with $\ell'_{m^*} > \lambda$, ℓ_{m^*} is equal to ℓ'_{m^*}. For the m-th processor with the maximum load $\ell_{\hat{m}}$ among all of the $\hat{m}$-th processors with $\ell'_{\hat{m}} = \lambda$, ℓ_m is at most $\frac{3}{2}\ell_{\min}$, where $\ell_{\min}$ is the minimum load of the M processors derived from Algorithm* LA+LTF.

Proof. Once we consider task τ_i in $\mathbf{T} \setminus \mathbf{T}^f$ in the loop from Step 4 to Step 6 in Algorithm LA+LTF, there must be another processor with load less than the m^*-th processor by the pigeon-hole principle. Hence, Algorithm LA+LTF never assigns any task in $\mathbf{T} \setminus \mathbf{T}^f$ to the m^*-th processor. Namely, ℓ_{m^*} is equal to ℓ'_{m^*}.

We now prove the second statement. Two cases have to be considered: (1) $\ell'_m = \ell_m$ and (2) $\ell'_m < \ell_m$. Let the n-th processor be the processor with load equal to $\ell_{\min}$. For the first case, ℓ_n must be equal to ℓ_m, and the proof is done. For the second case, suppose that τ_k is the last task inserted into $\mathbf{T}_m$ when we execute Algorithm LA+LTF for $\mathbf{T}$. Since $\ell'_m = \lambda < \ell_m$, τ_k is in $\mathbf{T} \setminus \mathbf{T}^f$. Since τ_k is inserted into $\mathbf{T}_m$ instead of $\mathbf{T}_n$, we also know that $\ell_m - \frac{c_k}{p_k} \leq \ell_n$. If $\mathbf{T}_m \setminus \{\tau_k\}$ has only one task, $\frac{c_k}{p_k}$ is less than $\frac{1}{2}(\ell_m - \frac{c_k}{p_k})$ since τ_k is not in $\mathbf{T}^f$, and the index of the task in $\mathbf{T}_m$ is less than $2M - |\mathbf{T}^f| + 1$. If $\mathbf{T}_m \setminus \{\tau_k\}$ has more than one task, $\frac{c_k}{p_k}$ is no more than $\frac{1}{2}(\ell_m - \frac{c_k}{p_k})$ because of the largest task first strategy. Hence, $\frac{c_k}{p_k} \leq \frac{1}{2}\ell_n$. As a result, $\ell_m \leq \frac{3}{2}\ell_n = \frac{3}{2}\ell_{\min}$. □

The following lemma is required to show the approximation ratio of Algorithm LA+LTF.

Lemma 7 *Suppose $f(y) = \frac{\mu \cdot \psi(3y) + (\hat{M} - \mu)\psi(2y)}{\hat{M}\psi(\frac{\Lambda}{\hat{M}})}$ for positive numbers $\hat{M}$ and Λ, and a non-negative number μ, where $0 \leq y, 0 \leq \mu \leq \hat{M}$ and $\mu \cdot 3y + (\hat{M} - \mu) \cdot 2y = \Lambda$, then $f(y) \leq 1.283$.*

Proof. We have to consider three cases: (1) $3y < s_0$, (2) $s_0 < 2y$, and (3) $2y \leq s_0 \leq 3y$. For the first case, the numerator and the denominator of $f(y)$ are the same, and, hence, $f(y) = 1$. For the second case, $f(y) = \frac{\mu((3y)^3 + \beta) + (\hat{M} - \mu)((2y)^3 + \beta)}{\hat{M}((\frac{\Lambda}{\hat{M}})^3 + \beta)} \leq \frac{\mu(3y)^3 + (\hat{M} - \mu)(2y)^3}{\hat{M}(\frac{\Lambda}{\hat{M}})^3} \leq 1.13$, where the last inequality comes

from the proof in [7][Theorem 5]. For the rest of the proof, we focus our discussions on the cases that $2y \leq s_0 \leq 3y$ and $2y \leq \frac{\Lambda}{\hat{M}} \leq 3y$. We have two sub-cases: (1) $s_0 \leq \frac{\Lambda}{\hat{M}}$ and (2) $s_0 \geq \frac{\Lambda}{\hat{M}}$.

By the definition, μ is $\frac{\Lambda-2y\hat{M}}{y}$. For the first sub-case, $f(y) = \frac{\frac{\Lambda-2y\hat{M}}{y}((3y)^3+\beta)+\frac{3y\hat{M}-\Lambda}{y}\frac{2y}{s_0}(s_0^3+\beta)}{\hat{M}((\frac{\Lambda}{\hat{M}})^3+\beta)}$. Since $s_0 \geq 2y$, we know that $\frac{\Lambda-2y\hat{M}}{y}\beta + \frac{3y\hat{M}-\Lambda}{y}\frac{2y}{s_0}\beta \leq \hat{M}\beta$. As a result, $f(y) \leq \frac{\frac{\Lambda-2y\hat{M}}{y}(3y)^3+\frac{3y\hat{M}-\Lambda}{s_0}2s_0^3}{\hat{M}(\frac{\Lambda}{\hat{M}})^3}$. By rephrasing $\hat{y}$ as $\frac{y}{\Lambda/\hat{M}}$ and $\hat{s_0}$ as $\frac{s_0}{\Lambda/\hat{M}}$, we have $f(y) \leq \frac{\frac{\hat{M}-2\hat{y}\hat{M}}{\hat{y}}(3\hat{y})^3+2\frac{3\hat{y}\hat{M}-\hat{M}}{\hat{s_0}}\hat{s_0}^3}{\hat{M}} = g(\hat{y})$, and $2\hat{y} \leq 1 \leq 3\hat{y}$. By solving $g'(\hat{y}) = 0$ and showing $g''(\hat{y}) < 0$ when $2\hat{y} \leq 1 \leq 3\hat{y}$, we know that $g(\hat{y})$ is maximized when $\hat{y}$ is $\frac{3+\sqrt[2]{9+12\hat{s_0}^2}}{18}$. As a result, $f(y) \leq \frac{1}{2} + \frac{(3+4\hat{s_0}^2)\sqrt{9+12\hat{s_0}^2}}{18} - \hat{s_0}^2 \leq 1.283$ since $\hat{s_0} \leq 1$, where the last inequality comes from $\hat{s_0} = 1$. Similarly, we could have $f(y) \leq (\frac{1}{2} + \frac{(3+4\hat{s_0}^2)\sqrt{9+12\hat{s_0}^2}}{18} - \hat{s_0}^2)/\hat{s_0}^2 \leq 1.283$ for the second sub-case, where $1 \leq \hat{s_0} \leq 1.5$ and the last inequality comes from $\hat{s_0} = 1$. The proof is done. □

Theorem 1 *Algorithm* LA+LTF *is a* 1.283*-approximation algorithm of the* LAMS *problem with* $E_{sw} = 0$.

Proof. $\Phi(SC_{\text{LA+LTF}})$ is equal to $\sum_{m=1}^{M}\psi(\ell_m) = \sum_{m=1}^{M}(\psi(\ell_m)\delta_{\ell'_m>\lambda} + \psi(\ell_m)\delta_{\ell'_m=\lambda})$, where δ_b is 1 if b is true; otherwise, it is 0. $\Phi(SC^*)$ is $\sum_{m=1}^{M}\psi(\ell'_m) = \sum_{m=1}^{M}(\psi(\ell'_m)\delta_{\ell'_m>\lambda} + \psi(\ell'_m)\delta_{\ell'_m=\lambda})$. Based on Lemma 6 that ℓ_m is equal to ℓ'_m when $\ell'_m > \lambda$ and the fact that SC^* is a lower bound of the optimal solution of the LAMS problem, we know that the approximation ratio $\mathcal{A}$ is:

$$\begin{aligned}\mathcal{A} &\leq \frac{\sum_{m=1}^{M}(\psi(\ell_m)\delta_{\ell'_m>\lambda} + \psi(\ell_m)\delta_{\ell'_m=\lambda})}{\sum_{m=1}^{M}(\psi(\ell'_m)\delta_{\ell'_m>\lambda} + \psi(\ell'_m)\delta_{\ell'_m=\lambda})} \\ &\leq \frac{\sum_{m=1}^{M}\psi(\ell_m)\delta_{\ell'_m=\lambda}}{\sum_{m=1}^{M}\psi(\ell'_m)\delta_{\ell'_m=\lambda}}.\end{aligned}$$

Let Λ be the value of $\sum_{m=1}^{M}\ell'_m\delta_{\ell'_m=\lambda}$. It could be shown that $\sum_{m=1}^{M}\ell_m\delta_{\ell'_m=\lambda}$ is equal to Λ. Let $\hat{M}$ be $\sum_{m=1}^{M}\delta_{\ell'_m=\lambda}$, which indicates the number of processors with loads equal to λ in SC^*. Moreover, $\sum_{m=1}^{M}\psi(\ell'_m)\delta_{\ell'_m=\lambda}$ is equal to $\hat{M}\psi(\frac{\Lambda}{\hat{M}})$. Let $\ell_{\min}$ be the minimum load of some processor derived from Algorithm LA+LTF. With Lemma 6, we know that $\ell_{\min} \leq \ell_m \leq \frac{3}{2}\ell_{\min}$ for any m-th processor with $\ell'_m = \lambda$. Let γ_m be the value that satisfies $\gamma_m\ell_{\min} + (1-\gamma_m)\frac{3}{2}\ell_{min} = \ell_m$. For any m-th processor with $\ell'_m = \lambda$, since $\ell_{\min} \leq \ell_m \leq \frac{3}{2}\ell_{\min}$, γ_m must be a real number between 0 and 1, and $\psi(\ell_m) \leq \gamma_m\psi(\ell_{\min}) + (1-\gamma_m)\psi(\frac{3}{2}\ell_{\min})$ by Lemma 3. As a result, we know that $\sum_{m=1}^{M}\psi(\ell_m)\delta_{\ell'_m=\lambda} \leq \hat{\gamma}\psi(\ell_{\min}) + (\hat{M}-\hat{\gamma})\psi(\frac{3}{2}\ell_{\min})$ with $\hat{\gamma}\ell_{\min} + (\hat{M}-\hat{\gamma})\frac{3}{2}\ell_{\min} = \Lambda$. With Lemma 7 by taking $\frac{\ell_{\min}}{2}$ as y, we know that the approximation ratio of Algorithm LA+LTF is 1.283. □

Corollary 1 *Algorithm* LA+LTF *is a* 1.13*-approximation algorithm of the* LAMS *problem when* β *and* $S_{\min}$ *are both* 0.

3.3 Scheduling when $S_{\max} \neq \infty$

We now consider the LAMS problem with a constraint on the maximum processor speed. For such a case, the derivation of a feasible schedule is $\mathcal{NP}$-complete even when β is 0 [7]. Similar to the approach in [7], we show that Algorithm LA+LTF is an artificial-bounded approach by adopting the constraint-violation approach [14]. For such a case, we first set an artificial upper bound on the processor speed and then derive feasible schedules in the minimization of energy consumption.

Theorem 2 *Algorithm* LA+LTF *would not derive any schedule which executes tasks at higher speed than* $(\frac{4}{3} - \frac{1}{3M})S_{\max}$ *for any input instance with a feasible schedule for the* LAMS *problem.*

Proof. The proof is omitted due to the space limitation. □

4 A Two-Phase Scheduling Algorithm for Non-Negligible Switching Overheads

In this section, we consider systems that have non-negligible switching overheads in turning on/off a processor, i.e., $E_{sw} \neq 0$. The partition method proposed in Section 3 is adopted to assign tasks in an off-line manner with a task re-assignment procedure. Each processor then individually decides the moment to turn on/off the processor on the fly. When a processor is idle, we have to decide whether we should turn the processor into the dormant mode to reduce energy resulting from the leakage current or not. If E_{sw} is 0, we could turn off a processor instantly when the processor is idle, as shown in Section 3. However, when E_{sw} is not 0, we have to make the decision carefully. If the idle period is not long enough, the energy consumption resulting from the switching overheads might be greater than that saved from the leakage current. When $S_{\max}$ is not ∞, the artificial bound is as the same as that in Section 3.3. We do not consider $S_{\max}$ in this section.

For any input instance $\mathbf{T}$ with $\sum_{\tau_i\in\mathbf{T}}\frac{c_i}{p_i} \leq s_0$, it is not difficult to see that the optimal schedule will use at most one processor for the executions of $\mathbf{T}$, and the LAMS problem is reduced to a uniprocessor problem. On such a special case, the 2-approximation algorithm in [10] could be adopted. For the rest of this paper, we only consider input instances with $\sum_{\tau_i\in\mathbf{T}}\frac{c_i}{p_i} > s_0$. Our proposed algorithm consists of two phases. In the first phase, Algorithm LA+LTF is adopted for task assignment along with better re-assignment. For the second phase, an EDF-based scheduling policy is adopted for each processor to schedule its tasks.

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

Algorithm 2 : FF

Input: $(\mathbf{T}, M, \mathbf{T}^\dagger, \mathbf{M}^\dagger)$;

1: mark each of the processors in $\mathbf{M}^\dagger$ as "unused";
2: **for** each unassigned task τ_i in $\mathbf{T}^\dagger$ **do**
3: **if** there exists a processor m in $\mathbf{M}^\dagger$ marked as "used" with $\ell_m + \frac{c_i}{p_i} \leq s_0$ **then**
4: $\ell_m \leftarrow \ell_m + \frac{c_i}{p_i}$, $\mathbf{T}_m \leftarrow \mathbf{T}_m \cup \{\tau_i\}$;
5: **else if** there exists a processor m in $\mathbf{M}^\dagger$ marked as "unused" **then**
6: mark processor m as "used", $\ell_m \leftarrow \frac{c_i}{p_i}$, and $\mathbf{T}_m \leftarrow \{\tau_i\}$;
7: **else**
8: return the task assignment of $\mathbf{T}$ resulting from LA+LTF$(\mathbf{T}, M)$;
9: return the resulting task assignment of $\mathbf{T}^\dagger$ on these $|\mathbf{M}^\dagger|$ processors and the task assignment of $\mathbf{T} \setminus \mathbf{T}^\dagger$ on the other $M - |\mathbf{M}^\dagger|$ processors in $SC_{\text{LA+LTF}}$;

4.1 No dormant-mode consideration

After applying Algorithm LA+LTF for task assignment, we re-assign tasks on all of the m-th processors with $\ell_m < s_0$ to reduce the number of activated processors in the system. Suppose that $\mathbf{M}^\dagger$ is the set of such processors with $\ell_m < s_0$. Let $\mathbf{T}^\dagger$ be $\cup_{m \in \mathbf{M}^\dagger} \mathbf{T}_m$. Tasks in $\mathbf{T}^\dagger$ are assigned onto these $|\mathbf{M}^\dagger|$ processors by applying the *first-fit* algorithm (denoted by Algorithm FF as shown in Algorithm 2): First of all, all of the processors in $\mathbf{M}^\dagger$ are marked as *unused*. In each iteration, we assign an un-assigned task τ_i in $\mathbf{T}^\dagger$ to a processor marked as *used* if the resulting load of the tasks assigned on the processor is no more than s_0. If no such a processor marked as "used" exists, and all of the processors in $\mathbf{M}^\dagger$ are marked as "used", then Algorithm FF returns the task assignment derived from Algorithm LA+LTF; otherwise, we mark an "unused" processor in $\mathbf{M}^\dagger$ as "used" and assign τ_i to the processor. The time complexity of the above task re-assignment is $O(|\mathbf{T}^\dagger||\mathbf{M}^\dagger|)$. Again, let $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$ be the resulting task partition of $\mathbf{T}$ after applying Algorithms LA+LTF and FF, and ℓ_m be $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}$.

For the second phase, an EDF-based scheduling policy could be adopted in each processor to schedule tasks assigned onto it. For the m-th processor with $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i} \geq s_0$, the execution of task τ_i in $\mathbf{T}_m$ at speed $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}$ in an earliest-deadline-first order would result in a feasible uniprocessor schedule of $\mathbf{T}_m$ under 100% utilization with the minimum energy consumption because of Lemma 2. Let $SC_{\text{LA+LTF+FF}}$ be the schedule by applying the following three rules: (1) turn off the m-th processor with $\mathbf{T}_m = \emptyset$ at time 0, (2) execute all of the tasks in $\mathbf{T}_m$ $(1 \leq m \leq M)$ in an earliest-deadline-first order on the m-th processor at speed $\max\{s_0, \sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}\}$, and (3) set the speed of the m-th processor to $S_{\min}$ when the processor is idle and $\mathbf{T}_m \neq \emptyset$.

Let the *total execution length* of $SC_{\text{LA+LTF+FF}}$ be the sum of the execution times of all of the tasks in $\mathbf{T}$ in the hyperperiod L. The *total idle length* of $SC_{\text{LA+LTF+FF}}$ is $M'L$ minus the total execution length of $SC_{\text{LA+LTF+FF}}$, where M' is the number of processors assigned to execute some tasks of $\mathbf{T}$ in $SC_{\text{LA+LTF+FF}}$. The following two lemmas show the relation between the total execution length and the total idle length of $SC_{\text{LA+LTF+FF}}$.

Lemma 8 *When $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} > s_0$, the total idle length is no longer than the total execution length in schedule $SC_{\text{LA+LTF+FF}}$.*

Proof. If the loads of all of the M' processors are no less than s_0, then each of the M' processors in $SC_{\text{LA+LTF+FF}}$ would have 100% utilization, and the total idle length is 0. We consider another case in which the load of some of the M' processors is less than s_0. If there is only one processor with load strictly between 0 and s_0, there must be another processor with load no less than s_0 in schedule $SC_{\text{LA+LTF+FF}}$, since $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} > s_0$. The total execution length must be at least L. Hence, the idle length in such a case is less than L.

We now consider the case in which there are at least two processors with loads less than s_0, i.e., $\sum_{\tau_i \in \mathbf{T}^\dagger} \frac{c_i}{p_i} > s_0$. $|\mathbf{T}^\dagger|$ must be at least 2 in such a case. We claim that $\sum_{\tau_i \in \mathbf{T}^\dagger} \frac{c_i}{p_i s_0}$ is at least $0.5(M' - (M - |\mathbf{M}^\dagger|))$. Hence, the total idle length is no longer than the total execution length in schedule $SC_{\text{LA+LTF+FF}}$. The claim could be proved as follows: Let $\mathbf{T}_i^\dagger$ be the subset of tasks in $\mathbf{T}^\dagger$ that consists of task τ_i and those tasks assigned before τ_i is considered in Algorithm FF. We show that if there are M_i' processors marked as used with $M_i' \geq 2$ when task τ_i is considered for assigning in Algorithm FF, $\sum_{\tau_j \in \mathbf{T}_i^\dagger} \frac{c_j}{p_j}$ is at least $\frac{1}{2}M_i'$. Clearly, there must be at most one of the M_i' processors with load less than $\frac{1}{2}s_0$. Otherwise, the first-fit strategy is contracted. Let $\hat{\ell}$ be the minimum load of these M_i' processors. The load of each of the other $M_i' - 1$ "used" processors must be no less than $\max\{s_0/2, s_0 - \hat{\ell}\}$. Otherwise, the solution contradicts with Algorithm FF. We know that $(\sum_{\tau_j \in \mathbf{T}_i^\dagger} \frac{c_j}{p_j s_0} \geq \hat{\ell} + (M_i' - 1)(\max\{s_0/2, s_0 - \hat{\ell}\}) \geq M_i'/2)$ since $M_i' \geq 2$. As a result, $M_i' \leq 2 \sum_{\tau_j \in \mathbf{T}_i^\dagger} \frac{c_j}{p_j s_0}$. We conclude that $\sum_{\tau_i \in \mathbf{T}^\dagger} \frac{c_i}{p_i s_0}$ is at least $0.5(M' - (M - |\mathbf{M}^\dagger|))$. This lemma is proved. □

Lemma 9 *When $\max_{m=1,2,\ldots,M} \ell_m \delta_{(\ell'_m = \lambda)} > s_0$, the total idle length is no longer than the half of the total execution length in schedule $SC_{\text{LA+LTF+FF}}$, in which λ is defined in Equation (4). δ_b is 1 if b is true. It is 0, otherwise.*

Proof. For brevity, let $\hat{\ell_1}, \hat{\ell_2}, \ldots, \hat{\ell_M}$ be the loads on the M processors right after applying Algorithm LA+LTF. Since $\max_{m=1,2,\ldots,M} \ell_m \delta_{\ell'_m = \lambda} > s_0$, we know that $\max_{m=1,2,\ldots,M} \hat{\ell_m} \delta_{\ell'_m = \lambda} = \max_{m=1,2,\ldots,M} \ell_m \delta_{\ell'_m = \lambda} > s_0$. Based on Lemma 6, we know that $\min_{m=1,2,\ldots,M} \hat{\ell_m} \geq \frac{2}{3} \max_{m=1,2,\ldots,M} \hat{\ell_m} \delta_{\ell'_m = \lambda} > \frac{2}{3} s_0$. Hence, $\sum_{\tau_i \in \mathbf{T}^\dagger} \frac{c_i}{p_i s_0} > \frac{2}{3}|\mathbf{M}^\dagger|$. As a result, the total idle length is no longer than the half of the total execution length in schedule $SC_{\text{LA+LTF+FF}}$. □

Again, $\Phi(SC^*)$ is a lower bound of the LAMS problem with non-negligible switching overheads. The following theorem shows the performance guarantee of schedule $SC_{\text{LA+LTF+FF}}$.

Theorem 3 *The energy consumption of schedule $SC_{\text{LA+LTF+FF}}$ is at most twice of the optimal energy consumption of the* LAMS *problem for any input instance* $\mathbf{T}$ *and* M *with* $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} > s_0$.

Proof. We prove this theorem by showing that $\Phi(SC_{\text{LA+LTF+FF}})$ is at most twice of $\Phi(SC^*)$. Let the total idle length of $SC_{\text{LA+LTF+FF}}$ be I. Since the energy consumption resulting from the executions of tasks in $SC_{\text{LA+LTF+FF}}$ is equal to that in $SC_{\text{LA+LTF}}$, we know that $\Phi(SC_{\text{LA+LTF+FF}})$ is equal to $I(S^3_{\min} + \beta) + \Phi(SC_{\text{LA+LTF}})$, in which E_{sw} is 0 in $SC_{\text{LA+LTF}}$. The approximation ratio $\mathcal{A}$ is $\frac{I(S^3_{\min}+\beta)+\Phi(SC_{\text{LA+LTF}})}{\Phi(SC^*)}$. Let $\ell_{\max}$ be $\max_{m=1,2,\ldots,M'} \ell_m \delta_{\ell'_m=\lambda}$, and $\ell_{\min}$ be $\min_{m=1,2,\ldots,M'} \ell_m \delta_{\ell'_m=\lambda}$. There are three cases to consider: (1) $\ell_{\min} > s_0$, (2) $\ell_{\max} \leq s_0$, and (3) $\ell_{\min} \leq s_0 < \ell_{\max}$.

For the first case, $\mathbf{T}^\dagger$ is an empty set, and each of the M processors executes some task at any time instant during the hyper-period L. As a result, I is 0, and $\mathcal{A}$ is at most 1.283 by Theorem 1 in such a case.

For the second case, $\Phi(SC_{\text{LA+LTF}})$ is equal to $\Phi(SC^*)$. By applying Lemma 8, I is at most $\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} \frac{c_i}{s_0}$, since the total execution length in $SC_{\text{LA+LTF+FF}}$ in such a case is at most $\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} \frac{c_i}{s_0}$. As a result, we have

$$\begin{aligned} \mathcal{A} &\leq \frac{\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} \frac{c_i}{s_0}(S^3_{\min} + \beta) + \Phi(SC_{\text{LA+LTF}})}{\Phi(SC^*)} \\ &\leq \frac{\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} \frac{c_i}{s_0}(S^3_{\min} + \beta)}{\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} \frac{c_i}{s_0}(s^3_0 + \beta)} + 1 \leq 2, \end{aligned} \tag{6}$$

where the last inequality comes from that $S_{\min} \leq s_0$.

For the last case, since $\ell_{\max}$ is greater than s_0, with Lemma 9, the total idle length I is at most $\frac{1}{3}|\mathbf{M}^\dagger|L$. Hence, we know that

$$\begin{aligned} \mathcal{A} &\leq \frac{\frac{1}{3}|\mathbf{M}^\dagger|L(S^3_{\min} + \beta) + \Phi(SC_{\text{LA+LTF}})}{\Phi(SC^*)} \\ &\leq^1 0.5 + \frac{\Phi(SC_{\text{LA+LTF}})}{\Phi(SC^*)} \leq 1.783. \end{aligned} \tag{7}$$

The second inequality, i.e., $\leq^1$, in Equation (7) comes from that the total execution length is at least $\frac{2}{3}|\mathbf{M}^\dagger|L$, and, hence, $\Phi(SC^*)$ is at least $\frac{2}{3}|\mathbf{M}^\dagger|L(s^3_0 + \beta) \geq \frac{2}{3}|\mathbf{M}^\dagger|L(S^3_{\min} + \beta)$, while the last inequality in Equation (7) comes from Theorem 1 directly.

With the considerations of the three cases above, we know that $\mathcal{A} \leq 2$, which is dominated by the second case. □

We could have the following corollary with a similar proof in Theorem 3.

Corollary 2 *The energy consumption of schedule $SC_{\text{LA+LTF+FF}}$ is at most* 1.667 *times of the optimal energy consumption of the* LAMS *problem for any input instance* $\mathbf{T}$ *and* M *with* $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} > s_0$ *when* $S_{\min}$ *is* 0.

Proof. The proof could be done by taking $S_{\min} = 0$ and $\beta = 2s^3_0$ into the calculation of $\mathcal{A}$, in which $\mathcal{A} \leq 5/3$ in Equation (6) and $\mathcal{A} \leq 1/3 + 1.283 < 1.617$ in Equation (7). □

4.2 Considerations of the dormant mode

We could derive schedules which consume less energy than the energy consumption of schedule $SC_{\text{LA+LTF+FF}}$ by turning the processor into the dormant mode on the fly in the second phase. The idea behind the scheduling on the fly is to lengthen and aggregate the idle period so that the idle time is long enough to turn off the processor. When a processor is active and is idle at a moment, the speed of the processor is set to $S_{\min}$. The power consumption in such a mode is $S^3_{\min} + \beta$. Hence, if the idle period is greater than $\frac{E_{sw}}{\beta + S^3_{\min}}$, it is worth of turning off the processor. The longer the idle period is, the less the energy consumption. Let t_θ be the threshold idle period $\frac{E_{sw}}{\beta + S^3_{\min}}$ for the rest of this paper.

The determination of idle periods could be done by procrastinating the arrival time of the next job to the system, such as the study in [10, 12]. Our approach for $\mathbf{T}_m$ with $0 < \ell_m < s_0$ is presented as follows: Let Z_i be the length of time that the arrival time of task τ_i could be procrastinated on the m-th processor, in which Z_i is $(1 - \sum_{\tau_j \in \mathbf{T}_m} \frac{c_j}{p_j s_0})p_i$. By definitions, $\frac{Z_i}{p_i} + \sum_{\tau_j \in \mathbf{T}_m} \frac{c_j}{p_j s_0} = 1$ for any task $\tau_i \in \mathbf{T}_m$. We schedule tasks assigned onto the m-th processor in an earliest-deadline-first order at speed s_0. If a job completes at time instant t, and no other job is ready for executions, we have to decide whether the processor should be turned into the dormant mode or idle. Let r_i be the arrival time of the next job of τ_i after time instant t for any τ_i in $\mathbf{T}_m$. If $(\min_{\tau_i \in \mathbf{T}_m}(r_i + Z_i)) - t < t_\theta$, the processor remains on the active mode and is idle at speed $S_{\min}$. Otherwise, we turn the processor into the dormant mode at time instant t, turn on the processor at time instant $(\min_{\tau_i \in \mathbf{T}_m}(r_i + Z_i))$, and procrastinate the arrival time of the next job of τ_j to $(\min_{\tau_i \in \mathbf{T}_m}(r_i + Z_i))$ for any task $\tau_j \in \mathbf{T}_m$ with $r_j \leq (\min_{\tau_i \in \mathbf{T}_m}(r_i + Z_i))$. When a job arrives at any time instant t, it is assigned into a ready queue. All of the jobs in the ready queue are executed in an earliest-deadline-first order at speed s_0. The on-line algorithm is denoted as Algorithm PROCRASTINATION. The time complexity to determine whether the processor should be turned into the dormant mode and to calculate the time to procrastinate the arrival time of the next jobs of some tasks is $O(|\mathbf{T}_m|)$. The following lemma shows the feasibility of Algorithm PROCRASTINATION.

Lemma 10 *Applying Algorithm* PROCRASTINATION *on the fly always derives a schedule that completes all of the tasks in* $\mathbf{T}_m$ *in time if* $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i} < s_0$.

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

Proof. The proof could be done with a similar argument in [12]. □

Let $SC_{\text{LA+LTF+FF+PROC}}$ be the schedule by applying Algorithms LA+LTF, FF, and PROCRASTINATION accordingly for the LAMS problem. Clearly, the energy consumption of $SC_{\text{LA+LTF+FF+PROC}}$ is no more than that of $SC_{\text{LA+LTF+FF}}$. We conclude this section with the following theorem.

Theorem 4 *For any input instance* $\mathbf{T}$ *and* M *with* $\sum_{\tau_i \in \mathbf{T}} \frac{c_i}{p_i} > s_0$*, applying Algorithms* LA+LTF, FF, *and* PROCRASTINATION *accordingly leads to a* 2*-approximation algorithm for the* LAMS *problem when* $S_{\min} > 0$*, and a* $\frac{5}{3}$*-approximation algorithm when* $S_{\min} = 0$.

Proof. This comes directly from Theorem 3, Corollary 2, and Lemma 10. □

5 Performance Evaluation

This section provides performance evaluation on the energy consumption of the proposed algorithms. Algorithm LA+LTF+FF+PROC denotes that applying Algorithms LA+LTF, FF, and PROC accordingly. Algorithm LA+LTF+PROC and Algorithm LA+LTF+FF are defined in a similar way. We compared the performance of Algorithm LA+LTF and Algorithm LA+RAND. The difference between Algorithm LA+RAND and Algorithm LA+LTF is that tasks under Algorithm LA+RAND are not sorted before the assignment procedure.

5.1 Workload Parameters and Performance Metrics

The power consumption function of the processor speed s was set as $P(s) = s^3 + \beta$, where β was a variable between 0 and 5. $S_{\min}$ was set as 0. L was 1. For any given task τ_i, ρ_i was an integral variable in $[1, 6]$, which represented the number of jobs of τ_i within L. Hence, p_i is L/ρ_i. c_i was a random variable in $[0, p_i]$.

We simulated two cases for different numbers of processors with different numbers of tasks to evaluate Algorithm LA+LTF when $\beta = 2$ and $E_{sw} = 0$. For the first case, we evaluated algorithms for the effects on the ratio of the number of tasks to the number of processors. For a given ratio η of the number of tasks to the number of processors, the number of processors M was an integral random variable between 10 and 30, and the number of tasks was set as the floor function value of the multiplication of η and M, i.e., $\lfloor \eta \cdot M \rfloor$. For the second case, the number of processors ranged from 4 to 16, and the task-set size ranged from 18 to 60. To evaluate the performance of Algorithm LA+LTF+FF and Algorithm LA+LTF+FF+PROC, we performed simulations for different task sets on $4, 6, 8$, and 10 processors by setting β as 2 and E_{sw} as 0.1 and 0.3.

The *normalized energy* was adopted as the performance metric in the experiments. The normalized energy of an algorithm for an input instance was defined as the ratio of the energy consumption of the derived schedule to that of the optimal schedule SC^* for the SEMI-LAMS problem derived in Section 3. Since the LAMS problem is $\mathcal{NP}$-hard, the performance metric on the normalized energy aimed at the providing of an approximated index. Experimental results were conducted with 128 independent runs for each configuration.

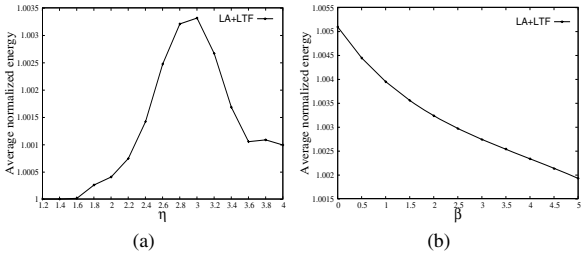

Figure 2. Simulation results when $E_{sw} = 0$ and (a) β was 2, and η ranged from 1.2 to 4, stepped by 0.2, and (b) η was 3, and β ranged from 0 to 5, stepped by 0.5

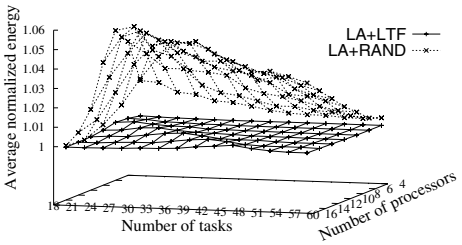

Figure 3. Simulation results when $\beta = 2$ and $E_{sw} = 0$, M was in $[4, 16]$, and N was in $[18, 60]$

5.2 Experimental Results

Figure 2 shows the average normalized energy of Algorithm LA+LTF by varying β and η when E_{sw} was 0. In Figure 2(a), β was 2, and η ranged from 1.2 to 4, stepped by 0.2. The average normalized energy was maximum when η was 3.

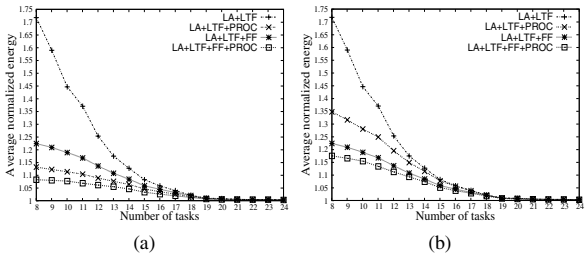

Figure 4. Simulation results when $M = 8$ and $\beta = 2$: (a) $E_{sw} = 0.1$, and (b) $E_{sw} = 0.3$

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE

When η was small, in most cases, most processors were assigned with only one task, and the assignment was almost as the same as the optimal schedule. However, when the value of η increased, the load of each processor was heavy enough even in the optimal schedule SC^*. Hence, the average normalized energy decreased when the ratio η increased with $\eta > 3$. In the Figure 2(b), β ranged from 0 to 5, stepped by 0.5, by restricting η as 3. When β increased, the average normalized energy decreased since the proportion of the energy consumption resulting from leakage current to the total energy consumption increased.

Figure 3 shows the average normalized energy when β was 2, E_{sw} was 0, the number of processors ranged from 4 to 16, and the task-set size ranged from 18 to 60. Algorithm LA+LTF derived solutions close to optimal ones in Figure 3. Similar to Figure 2(a), the average normalized energy was maximized when the ratio of the number of tasks to that of processors was close to 3.

The average normalized energy for Algorithms LA+LTF, LA+LTF+PROC, LA+LTF+FF, and LA+LTF+FF+PROC was shown in Figure 4(a)(/ 4(b)) by setting E_{sw} as $0.1(/0.3)$, M as 8, and β as 2. In both Figure 4(a) and Figure 4(b), Algorithm LA+LTF+FF+PROC always outperformed the other evaluated algorithms, where the maximum average normalized energy was less than 1.175. Algorithm LA+LTF+PROC outperformed LA+LTF+FF in Figure 4(a), and vice versa in Figure 4(b). Since the value of t_θ when $E_{sw} = 0.1$ was less than that when $E_{sw} = 0.3$, Algorithm PROCRASTINATION could save more energy when $E_{sw} = 0.1$ by turning a processor into the dormant mode. Similar observations were done when $M = 4, 6, 10$. We omit the detail due to the space limitation.

6 Conclusion

This paper explores energy-efficient scheduling of periodic real-time tasks in homogeneous multiprocessor environments in which the power consumption resulting from leakage current is non-negligible. A processor might be turned into the dormant mode to reduce energy consumption, whenever needed, and the power consumption function $P()$ is modeled as $P(s) = s^3 + \beta$, where s and β are the processor speed and the power consumption resulting from leakage current, respectively. When there is no upper bound on the processor speeds, a 1.13-approximation algorithm is proposed for the case in which β and the minimum available speed are both 0, and a 1.283-approximation algorithm is proposed where the overheads in turning on/off a processor are negligible. 2-approximation algorithms are also proposed when the overheads in turning on/off a processor are non-negligible. When there is an upper bound on the available processor speeds, we take an artificial-bound approach to minimize the energy consumption. Compared to the previous work, this paper is one of the pioneering studies that provide approximation bounds for energy-efficient scheduling in multiprocessor systems with leakage current considerations. The proposed algorithms could also be applied to systems with $P(s) = s^\sigma + \beta$ for any σ between 2 and 3. Evaluations show that the proposed algorithms could derive schedules close to optimal solutions.

For future research, we will explore energy-efficiency in heterogeneous multiprocessor environments with leakage considerations and tasks with resource competition.

References

[1] T. A. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'05)*, pages 213–223, 2005.

[2] J. H. Anderson and S. K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 428–435, 2004.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the IEEE EuroMicro Conference on Real-Time Systems*, pages 225–232, 2001.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 95–105, 2001.

[5] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 113 – 121, 2003.

[6] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 520–529, 2004.

[7] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *EuroMicro Conference on Real-Time Systems (ECRTS'04)*, pages 101–108, 2004.

[8] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks. In *International Conference on Parallel Processing (ICPP)*, pages 13–20, 2005.

[9] F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low energy systems using variable supply voltage processors. In *Proceedings of Asia South Pacific Design Automation Conference*, pages 449–455, 2001.

[10] 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. Society for Industrial and Applied Mathematics, 2003.

[11] 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.

[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] 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.

[14] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Symposium on Theory of Computing*, pages 771–782. ACM Press, 1992.

[15] 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.

[16] J. W. Liu. *Real-Time Systems*. Prentice Hall, Englewood, Cliffs, NJ., 2000.

[17] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem. Energy aware scheduling for distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, page 21, 2003.

[18] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.

[19] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[20] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems(LCTES)*, 2005.

[21] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the 8th Conference of Design, Automation, and Test in Europe (DATE)*, pages 468–473, 2005.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382. IEEE, 1995.

[23] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Annual ACM IEEE Design Automation Conference*, pages 183–188, 2002.

[24] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *Proceedings of IEEE 22th Real-Time System Symposium*, pages 84–94, 2001.

Proceedings of the Twelfth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)
0-7695-2516-4/06 $20.00 © 2006 IEEE