Energy-Efficient Real-Time Task Scheduling with Task Rejection *

Jian-Jia Chen, Tei-Wei Kuo, Chia-Lin Yang
Department of Computer Science and Information Engineering
National Taiwan University Taipei, Taiwan.
Email:{r90079, ktw, yangc}@csie.ntu.edu.tw

Ku-Jei King
xSeries Development
IBM Systems Technology Group (STG)
Email: kujei@tw.ibm.com

Abstract

In the past decade, energy-efficiency has been an important system design issue in both hardware and software managements. For mobile applications with critical missions, both energy consumption reduction and timing guarantee have to be provided by system engineers to extend operation duration and maintain system stability. This research explores real-time systems composed of homogeneous multiple processors with the capability of dynamic voltage scaling (DVS), in which a given task can be rejected with a specified value of rejection penalty. The objective is to minimize the summation of the total rejection penalty for the tasks that are not completed in time and the energy consumption of the system. This study provides analysis to show that there does not exist any polynomial-time approximation algorithm for the studied problem, unless $\mathcal{P} = \mathcal{NP}$. Moreover, we propose algorithms for systems with ideal and non-ideal DVS processors. The capability of the proposed algorithms is provided with extensive evaluations. The evaluation results reveal that our proposed algorithms could derive effective solutions of the energy-efficient scheduling problem with task rejection considerations.

Keywords: Energy-Efficient Scheduling, Task Rejection, Real-Time Task Scheduling.

1. Introduction

Along with the low-power demands in electronic circuit designs, a modern processor can now operate at different supply voltages to balance its power consumption and performance. Different supply voltages lead to different execution speeds on a dynamic voltage scaling (DVS) processor. Well-known DVS processors for embedded systems are Intel StrongARM SA1100 processor [17] and Intel XScale [18]. Moreover, technologies, such as Intel SpeedStep® and AMD PowerNOW!™, provide dynamic voltage scaling for laptops to prolong the battery lifetime.

In the past decade, energy-efficient designs have received a lot of attention in industry and academics. For systems with real-time demands, energy-efficient task scheduling has been studied to minimize the energy consumption with timing guarantee, especially for uniprocessor systems with DVS supports. Due to the convexity of the power consumption function, implementations in multiprocessor systems are often more energy-efficient [2]. Moreover, since many chip makers, such as Intel and AMD, are releasing multi-core chips, multiprocessor energy-efficient scheduling is becoming more and more important. Various heuristics were proposed for energy consumption minimization under different task models in multiprocessor environments, e.g., [1, 4–7, 15, 19] for independent real-time tasks and [9, 20] for real-time tasks with precedence constraints.

Due to the increase of leakage power consumption in technology, researchers have started exploring energy-efficient scheduling with the considerations of the non-negligible power consumption of leakage current [12]. For uniprocessor scheduling, Irani et al. [10] proposed approximation algorithms for aperiodic real-time tasks. For periodic real-time tasks in uniprocessor systems, Jejurikar et al. [12], Lee et al. [14], and Chen et al. [8] provided scheduling algorithms with task procrastination to decide when to turn the processor into a dormant mode. Moreover, Chen et al. [6] developed approximation algorithms for multiprocessor leakage-aware scheduling.

However, most studies for energy-efficient real-time task scheduling do not take task rejection into considerations. Most heuristics for multiprocessor energy-efficient scheduling cannot guarantee the schedulability of the derived schedules. Chen et al. [6] applied the constraint violation approach to augment the highest available speed with a $\frac{4}{3}$ factor. However, resource augmentation might not be possible since it is hardware-dependent. Hence, some tasks might be rejected to guarantee the schedulability of the selected tasks.

This research explores systems with the possibility to reject a task for execution with a specified cost (penalty). If a task is more important than another, its rejection penalty should be specified with a greater value. We consider a homogeneous multiprocessor system with continuously available speeds or discretely available speeds. The objective is to minimize the summation of the total rejection cost for the tasks that are not completed in time and the energy consumption of the system. The contribution of this paper is on two folds. Firstly, we show the $\mathcal{NP}$-hardness of the studied problem, and provide analysis on the non-existence of polynomial-time approximation algorithms, provided that $\mathcal{P} \neq \mathcal{NP}$. Secondly, we propose a branch-and-bound approach and heuristic algorithms. The proposed algorithms are evaluated by extensive experiments. The evaluation results reveal that our proposed algorithms could derive effective solutions of the energy-efficient scheduling problem with task rejection considerations.

The rest of this paper is organized as follows: Section 2 defines the energy-efficient task scheduling problem with task rejection and provides the hardness analysis. Section 3 presents our algorithms. Experimental results for the performance evaluation of the proposed algorithms are presented in Section 4. Section 5 is the conclusion.

2. Problem Definition and Hardness Analysis

Processor models This paper explores energy-efficient scheduling on M homogeneous DVS multiprocessors, where the power consumption function of each task is the same on every processor. The power consumption function $P(s)$ of the adopted processor speed on a DVS processor can be divided into two parts $P_d(s)$ and P_{ind}, in which $P_d(s)$ is dependent (P_{ind} is independent, respectively) upon the processor speed s [21]. The speed-dependent power consumption function is mainly contributed by the dynamic power consumption resulting from the charging or discharging of CMOS gates and the short-circuit power consumption, while the leakage power consumption contributes the major of the speed-independent power consumption. The algorithms proposed in this paper can be adopted with many power consumption function formulations, such as those in

* Support in parts by research grants from ROC National Science Council NSC-95-2752-E-002-008-PAE, Aim for Top University Plan 95R0062-A100-07, and IBM Faculty Award.

[16, §5.5]. We consider systems with $P_d(s)$ as a convex and increasing function, e.g., $P_d(s) \propto s^\alpha$ for any $\alpha > 1$.

The number of CPU cycles executed in a time interval is linear of the processor speed. That is, the number of CPU cycles completed in time interval $(t_1, t_2]$ is $\int_{t_1}^{t_2} s(t)\mathrm{d}t$, where $s(t)$ is the processor speed at time t. The energy consumed in $(t_1, t_2]$ is $\int_{t_1}^{t_2} P(s(t))\mathrm{d}t$. We first target *ideal* processors, in which a processor may operate at any speed in $[S_{\min}, S_{\max}]$. We also show the extension to cope with *non-ideal* processors with discrete speeds. For non-ideal processors, there are H available speeds indexed by $s_1, s_2, \ldots, s_H$ in an increasing order. For non-ideal processors, for brevity, s_{H+1} and $P(s_{H+1})$ are both assumed ∞, $S_{\min}$ is s_1, and $S_{\max}$ is s_H.

When needed, turning the processor into a dormant mode (or turning the processor off) might further reduce the energy consumption. However, turning off or waking up a processor takes time and has energy overheads. For processors with non-negligible overheads to be turned off, the overheads could be treated as part of the overheads to turn on the processor [6, 10]. We denote E_{sw} (t_{sw}, respectively) as the energy (the time, respectively) requirement of the *switching overheads* for the whole process on turning off the processor and then turning on the processor.

Task models Tasks considered in this paper are periodic and independent in execution. 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. 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). The relative deadline of each task τ_i is equal to its period p_i. 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. We assume that all the tasks arrive at time 0, but extensions can be achieved easily for tasks with different arrival times. Given a task set $\mathbf{T}$, the *hyper-period* of $\mathbf{T}$, denoted by L, is defined as the minimum L so that L/p_i is an integer for any task τ_i in $\mathbf{T}$. For example, L is the least common multiple (LCM) of the periods of tasks in $\mathbf{T}$ when the periods of tasks are all integers. Without loss of generality, we only consider tasks τ_is with $\frac{c_i}{p_i} \leq S_{\max}$, since it is not possible to complete any task τ_j with $\frac{c_j}{p_j} > S_{\max}$ in time.

This research explores systems with the possibility to reject a task for execution with a specified cost (penalty) provided by system designers. If a task is more important than another, its rejection cost should be specified with a greater value. If a task instance of task τ_i is not completed in time, the system receives χ_i penalty, where $\chi_i > 0$. (If a task can be rejected without penalty, we can reject the task directly.) If a task is very important and cannot be rejected, its rejection cost should be specified as ∞. If the rejection costs of all the tasks are infinite, all the tasks are asked to be completed in time.

Problem definition This paper explores the problem on the minimization of the energy consumption of the system and the rejection cost at the same time. We pursue the objective on the linear combination of the energy consumption and the rejection cost, i.e., $(1-\alpha)E + \alpha\Pi$, where α is a non-negative factor no more than 1 specified by the system designer, E is the energy consumption of the system in the hyper-period, and Π is the total rejection penalty of the task instances missing their deadlines in the hyper-period. If energy consumption minimization is more important than task rejection penalty minimization, α should be specified as close to 0, and vice versa.

For notational brevity, we normalize the rejection penalty of task τ_i as $\alpha\chi_i$, the power consumption function $P()$ as $(1-\alpha)P()$, the energy switching overheads as $(1-\alpha)E_{sw}$. Hence, the objective of the linear combination can be treated as the summation of the (normalized) penalty and the (normalized) energy consumption.

The problem explored in this paper is defined as follows:

DEFINITION 1. *Energy-eFFicient schEduling with rejeCting Tasks (*EFFECT*):*

Consider a task set $\mathbf{T}$ of N independent tasks over M identical processors with a common power consumption function $P(s)$. Each periodic task $\tau_i \in \mathbf{T}$ arrives at time 0 and is associated with a computation requirement in c_i CPU-cycles, a rejection cost (penalty) χ_i, and a period p_i, where the relative deadline of task τ_i is p_i. The energy consumption and timing of the switching overheads are E_{sw} and t_{sw}, respectively. The problem is to derive a schedule of $\mathbf{T}$ to minimize the summation of the penalty (cost) of the task instances that miss their deadlines and the energy consumption of the system in the hyper-period L of tasks in $\mathbf{T}$, in which a job of task τ_i is executed entirely on a processor. □

For brevity, for the rest of this paper, the objective function of the EFFECT problem is called as *energy-penalty* (*EP* for abbreviation).

Hardness analysis Since most previous studies on multiprocessor energy-efficient scheduling did not take task rejection penalty into considerations, the schedulability of the derived schedules cannot be guaranteed, e.g., [4, 9]. As shown in [6], it is $\mathcal{NP}$-hard to derive a schedule with the minimum energy consumption to complete all the tasks in time without rejecting any real-time task. The following lemma shows that the EFFECT problem is still $\mathcal{NP}$-hard even if we have the flexibility to reject some tasks for execution.

LEMMA 1. *The* EFFECT *problem is $\mathcal{NP}$-hard in a strong sense even when E_{sw} is 0, and all the tasks have the same rejection penalty.*

Proof. It can be proved by a reduction from the leakage-aware multiprocessor energy-efficient rejection problem [6] with the same period p. The rejection cost of each task is a constant greater than $P(S_{\max}) \cdot p$. The detail is omitted due to space limitation. □

Due to the $\mathcal{NP}$-hardness of the EFFECT problem, polynomial-time approximation algorithms might be pursued for the provision of approximated solutions with worst-case guarantees. A polynomial-time β-approximation algorithm for the EFFECT problem must have polynomial-time complexity of the input size and could derive a solution with an objective value at most β times of an optimal solution, for any input instance. However, in addition to the $\mathcal{NP}$-hardness of the EFFECT problem, the following theorem shows the hardness on the approximability of polynomial-time algorithms.

THEOREM 1. *There does not exist any polynomial-time approximation algorithm for the* EFFECT *problem unless $\mathcal{P} = \mathcal{NP}$.*

Proof. This theorem can be proved by a *gap reduction* from the $\mathcal{NP}$-complete PARTITION problem: Given a set of N non-negative numbers, denoted by $o_1, o_2, \ldots, o_N$, the PARTITION problem is to answer whether there is a partition of these N numbers into two sets, so that the sum of the numbers in each set is the same. Suppose for contradiction that there is a polynomial-time $(1+\epsilon)$-approximation algorithm, denoted by Algorithm $\mathcal{A}$, with $\epsilon > 0$ for the EFFECT problem. We will show that we can use Algorithm $\mathcal{A}$ to answer the PARTITION problem in polynomial time, which contradicts the assumption on $\mathcal{P} \neq \mathcal{NP}$.

To solve the PARTITION problem by applying Algorithm $\mathcal{A}$, we have to create an input instance for the EFFECT problem. For each number o_i, a unique task τ_i is created with c_i as o_i, p_i as $\frac{\sum_{j=1}^{N} o_j}{2}$, and χ_i as $(1+\epsilon)(\sum_{j=1}^{N} o_j)$, where $P(s) = s^3$ and $E_{sw} = 0$. Moreover, $S_{\max}$ is 1, and $S_{\min}$ is no more than 1. If the input instance of the PARTITION problem admits a positive answer, the optimal solution for the constructed input instance is $\sum_{j=1}^{N} o_j$. By the construction, there exists no feasible solution with EP more than $\sum_{j=1}^{N} o_j$ and no more than $(1+\epsilon)\sum_{j=1}^{N} o_j$. Since Algorithm $\mathcal{A}$ is a $(1+\epsilon)$-approximation algorithm, Algorithm $\mathcal{A}$ guarantees to derive a solution whose EP is $\sum_{j=1}^{N} o_j$. If the input instance of the PARTITION problem does not admit a positive answer, the solution answered by Algorithm $\mathcal{A}$ must be greater than $\sum_{j=1}^{N} o_j$.

Since the construction of the input instance of the EFFECT problem takes $O(N)$ time, and Algorithm $\mathcal{A}$ is with polynomial-time complexity, we can determine whether an input instance of the PARTITION problem admits a positive answer in polynomial time by verifying the solution of Algorithm $\mathcal{A}$, which is a contradiction. □

3. Our Algorithms

By Theorem 1, it is impossible to derive optimal solutions or approximated solutions with worst-case guarantee for the EFFECT problem in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. This section provides a branch-and-bound approach and heuristics to derive solutions. We first partition tasks into $M + 1$ task sets, denoted by $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M, \mathbf{T}_{M+1}$, so that the tasks in task set $\mathbf{T}_m$ are executed on the m-th processor for $m \leq M$ and the tasks in $\mathbf{T}_{M+1}$ are rejected. The off-line derivation is obtained by assuming negligible switching overheads. Whether a rejected task instance determined in the off-line phase can be executed for performance improvement is done in an on-line fashion.

If a task has high computation requirement but low rejection penalty, it should be a good candidate to be rejected to reduce the EP, and vice versa. For the rest of this section, tasks are sorted non-increasingly according to $\frac{\chi_i}{c_i}$. We will consider the execution or rejection of tasks in the sorted order. Moreover, throughout this section, the earliest-deadline-first (EDF) schedule will be applied for task scheduling on each processor. By [3], a task set $\mathbf{T}_m$ is schedulable on a processor if and only if $\sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i} \leq S_{\max}$.

3.1 Off-line derivation of task partitions with negligible switching overheads

Although the power consumption function $P(s)$ is a convex and increasing function, the energy consumption at speed s, which is $\frac{P(s)}{s}$, might be not. For example, if $P(s) = s^3 + \gamma$, $\frac{P(s)}{s}$ is a decreasing function for s in $(0, \sqrt[3]{\frac{\gamma}{2}}]$ and an increasing function for s in $(\sqrt[3]{\frac{\gamma}{2}}, S_{\max}]$. If the switching overheads are negligible, there is a lower-bounded execution speed for tasks, referred to as the *critical speed* s^* as in [6, 8, 12]. For ideal processors, the critical speed s^* can be derived by solving $\frac{\mathrm{d}(P(s^*)/s^*)}{\mathrm{d}s^*} = 0$ [6]. By the definition, if s^* is greater than $S_{\min}$, the critical speed s^* is revised as $S_{\min}$. If $s^* > S_{\max}$, s^* is $S_{\max}$. For non-ideal processors, the critical speed s^* is s_h with $P(s_{h+1})/s_{h+1} > P(s_h)/s_h$ and $P(s_{h-1})/s_{h-1} \geq P(s_h)/s_h$ for $h = 1, 2, \ldots, H$ by taking $P(s_0)/s_0$ and $P(s_{H+1})/s_{H+1}$ as ∞ for boundary checking.

For clarity, we first focus on systems with ideal processors. The extensions to systems with non-ideal processors will be shown by the end of this subsection. A task partition is said a *feasible* solution if all the selected tasks for execution can meet their deadlines.

3.1.1 A branch-and-bound approach for ideal processors

For a given task partition $(\mathbf{T}_1^*, \mathbf{T}_2^*, \ldots, \mathbf{T}_M^*, \mathbf{T}_{M+1}^*)$ with ℓ_m defined as $\sum_{\tau_i \in \mathbf{T}_m^*} \frac{c_i}{p_i}$. If $\ell_m \leq S_{\max}$ for all $m = 1, 2, \ldots, M$, the earliest-deadline-first (EDF) schedule on each processor by executing all the tasks in $\mathbf{T}_m$ at speed $\min\{s^*, \ell_m\}$ can make all the tasks in $\mathbf{T}_m^*$ complete in time with the minimum energy consumption for the task partition [3]. Therefore, we can apply the depth-first search in a search tree to obtain the task partition $(\mathbf{T}_1^*, \mathbf{T}_2^*, \ldots, \mathbf{T}_M^*, \mathbf{T}_{M+1}^*)$ with the minimum EP in $O((N + M)N^{M+1})$ time.

The branch-and-bound (BB) approach can be adopted to reduce the time complexity on exploration of the solution space. Since homogeneous multiprocessor systems are under considerations, we can restricted τ_1 to be executed on the first processor by symmetry or to be rejected. In our BB approach, we visit the search tree rooted from τ_1, and the k-th level represents the selection of task τ_k to a task set $\mathbf{T}_m$ with $m = 1, 2, \ldots, M, M + 1$.

Suppose that we are at the n-th level in the search tree. The basic pruning condition is on the schedulability test. If $\frac{c_n}{p_n} + \sum_{\tau_i \in \mathbf{T}_m} \frac{c_i}{p_i}$ is greater than $S_{\max}$, the BB approach can eliminate all subsets containing the infeasible subset. The lower-bounded elimination is

Algorithm 1 : LEP

Input: $\mathbf{T}^\dagger, \mathbf{T}^\sharp, n$;

1: $\mathbf{T}^\flat \leftarrow \{\tau_i \mid n < i \leq N\}$;
2: $y_i \leftarrow 0, \forall \tau_i \in \mathbf{T}^\flat, U^1 \leftarrow \sum_{\tau_i \in \mathbf{T}^\dagger} \frac{c_i}{p_i}$;
3: **for** $(i \leftarrow n + 1; i \leq N; i \leftarrow i + 1)$ **do**
4: Let y_i be the value between 0 and 1 which minimizes $P^*(\frac{\frac{c_i}{p_i} y_i + U_1}{M})M + (1 - y_i)\frac{\chi_i}{p_i}$ with $\frac{c_i}{p_i} y_i + U_1 \leq M \cdot S_{\max}$;
5: **if** $(y_i < 1)$ **then**
6: return $L \cdot (P^*(\frac{\frac{c_i}{p_i} y_i + U_1}{M})M + (1 - y_i)\frac{\chi_i}{p_i} + \sum_{\tau_j \in \mathbf{T}^\sharp} \frac{\chi_j}{p_j} + \sum_{j=i+1}^{N} \frac{\chi_j}{p_j})$;
7: **else**
8: $U_1 \leftarrow U_1 + \frac{c_i}{p_i}$;
9: return $L \cdot (P^*(\frac{U_1}{M})M + \sum_{\tau_j \in \mathbf{T}^\sharp} \frac{\chi_j}{p_j})$;

Algorithm 2 : BB

Procedure: DFSBB($n, \mathbb{X}$)

Input: $n, \mathbb{X}$, where X_i is an integer between 1 and $M + 1$ for $i < n$;

1: **for** $m \leftarrow 1; m \leq M + 1; m \leftarrow m + 1$ **do**
2: **if** $m \leq M$ and $\frac{c_n}{p_n} + \sum_{i:1 \leq i \leq n-1}$ and X_i is m $\frac{c_i}{p_i} > S_{\max}$ **then**
3: continue;
4: $X_n \leftarrow m$;
5: **if** n is equal to N **then**
6: evaluate the EP by executing τ_i at the X_i-th processor with $X_i \leq M$ and rejecting task τ_is with $X_i = M + 1$;
7: save this task partition if the EP is better than the best solution so far;
8: **else**
9: $\mathbf{T}^\dagger \leftarrow \{\tau_i \mid 1 \leq i \leq n$ and $X_i \leq M\}$;
10: $\mathbf{T}^\sharp \leftarrow \{\tau_i \mid 1 \leq i \leq n$ and $\tau_i \notin \mathbf{T}^\dagger\}$;
11: $EP_m \leftarrow$ LEP$(\mathbf{T}^\dagger, \mathbf{T}^\sharp, n)$;
12: **if** EP_m is greater than the best solution so far **then**
13: continue;
14: **else**
15: call DFSBB$(n + 1, \mathbb{X})$

Procedure: BB()

1: sort tasks in $\mathbf{T}$ non-increasingly according to $\frac{\chi_i}{c_i}$;
2: initialize $\mathbb{X}$ with $X_i \leftarrow M + 1$, for $i = 1, 2, \ldots, N$;
3: call DFSBB($1, \mathbb{X}$) to obtain the task partition;

applied by verifying whether the lower bound of the EP of the feasible solutions for the subsets of solutions rooted at the n-th level is lower than the best solution derived so far. If the lower bound is greater than the best solution derived so far, we can prune all the subsets rooted at the n-th level. For a specified partition of set $\{\tau_i \mid 1 \leq i \leq n\}$ into two disjoint sets $\mathbf{T}^\dagger$ and $\mathbf{T}^\sharp$ by rejecting all the tasks in $\mathbf{T}^\sharp$ and executing all the tasks in $\mathbf{T}^\dagger$, Algorithm LEP, shown in Algorithm 1, can be applied to calculate a lower bound of the EP of feasible solutions, where $P^*(s)$ in Steps 4, 6, and 9 is

$$P^*(s) = \begin{cases} P(s), & \text{when } s > s^*, \text{ and} \\ \frac{s}{s^*} P(s^*), & \text{otherwise.} \end{cases} \quad (1)$$

The proof for the correctness on the provision of the lower-bounded EP of Algorithm LEP is omitted due to space limitation.

The branch-and-bound approach is presented in Procedure DFSBB in Algorithm 2, in which the search space is pruned with the feasibility test in Step 2 and Step 3 and the lower-bounded elimination between Step 9 and Step 13. The solution in this phase is obtained by calling DFSBB($1, \mathbb{X}$) with initialization shown in Procedure BB in Algorithm 2.

3.1.2 Polynomial-time algorithms for ideal processors

This section presents efficient algorithms, i.e., in polynomial time, for the determination of the task partition. The rationale behind the proposed algorithms is to select tasks with higher $\frac{\chi_i}{c_i}$ for execution

Algorithm 3 : SGA

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

1: sort tasks in $\mathbf{T}$ non-increasingly according to $\frac{\chi_i}{c_i}$;
2: let y_i^* be the value of y_i of task τ_i after calling LEP($\emptyset, \emptyset, 0$);
3: $\mathbf{T}^\dagger \leftarrow \{\tau_i \mid y_i^* = 1\}$, $\mathbf{T}^\sharp \leftarrow \mathbf{T} \setminus \mathbf{T}^\dagger$;
4: let $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger, \ldots, \mathbf{T}_M^\dagger)$ be the task partition of $\mathbf{T}^\dagger$ on M processors derived from Algorithm LA+LTF in [6];
5: **for** $m \leftarrow 1; m \leq M; m \leftarrow m + 1$ **do**
6:     **while** $\sum_{\tau_i \in \mathbf{T}_m^\dagger} \frac{c_i}{p_i} > S_{\max}$ **do**
7:         let τ_j be the task with the minimum $\frac{\chi_j}{p_j}$ in $\mathbf{T}_m^\dagger$;
8:         $\mathbf{T}_m^\dagger \leftarrow \mathbf{T}_m^\dagger \setminus \{\tau_j\}$, $\mathbf{T}^\sharp \leftarrow \mathbf{T}^\sharp \cup \{\tau_j\}$;
9: return $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger, \ldots, \mathbf{T}_M^\dagger, \mathbf{T}^\sharp)$ as the task partition;

and tasks with lower $\frac{\chi_i}{c_i}$ for rejection. Let $\mathbf{T}^\dagger$ be the set of tasks decided to be executed on these M processors. Initially, $\mathbf{T}^\dagger$ is $\emptyset$.

For scheduling the selected tasks on these M processors in polynomial time, we apply Algorithm LA+LTF (Leakage-Aware Largest-Task-First) in [6] to partition these tasks into M disjoint sets. Algorithm LA+LTF sorts these selected tasks in a non-increasing order of their *loads*, in which the load of a task τ_i is defined by its computation requirement divided by its period, i.e., $\frac{c_i}{p_i}$. Then, Algorithm LA+LTF assigns tasks according to the sorted order to the processor with the least load so far.

The first algorithm is Algorithm SGA, stands for Standard Greedy Algorithm. For each iteration, we consider the selection of task τ_i according to the non-increasing order of $\frac{\chi_j}{c_j}$ for tasks τ_j in $\mathbf{T}$. Algorithm SGA applies Algorithm LEP for the determination. Let $(y_1^*, y_2^*, \ldots, y_N^*)$ be the vector of y_is of tasks τ_is after calling LEP($\emptyset, \emptyset, 0$). Algorithm SGA then first attempts to execute all the tasks in $\mathbf{T}^\dagger \leftarrow \{\tau_i \mid y_i^* = 1\}$ on these M processors. By applying Algorithm LA+LTF to assign tasks in $\mathbf{T}^\dagger$ to M processors, we can have a task partition $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger, \ldots, \mathbf{T}_M^\dagger)$. However, $\sum_{\tau_i \in \mathbf{T}_m^\dagger} \frac{c_i}{p_i}$ might be greater than $S_{\max}$, and, hence, we must reject some tasks in $\mathbf{T}^\dagger$. Algorithm SGA then repeatedly evicts the task with the minimum $\frac{\chi_j}{p_j}$ from $\mathbf{T}_m^\dagger$ until the schedulability is guaranteed on the m-th processor. Algorithm SGA is summarized in Algorithm 3. The time complexity is $O((N + M) \log(N + M))$.

Algorithm EGA, stands for Enhanced Greedy Algorithm, is an enhancement of Algorithm SGA. The difference is on the derivation of $(y_1^*, y_2^*, \ldots, y_N^*)$ in Algorithm LEP. Instead of returning the result when $y_i < 0$ in Step 6 in Algorithm 1, the revised Algorithm LEP continues the loop by setting y_i to 0. The time complexity of Algorithm EGA is the same as that of Algorithm SGA.

Algorithm ES+EGA (Enhanced Greedy Algorithm with Estimated Schedule) applies Algorithm LA+LTF on the fly to verify whether the execution of task τ_i can reduce the EP by evaluating the EP of the derived schedule.[1]

Both Algorithms SGA and EGA evict those tasks τ_is with $y_i^* < 1$, and Algorithm ES+EGA evicts a task τ_i if executing τ_i and the selected tasks has greater EP. However, execution of some of these tasks with eviction on some selected tasks might reduce the EP. Algorithm TE+EGA (Enhanced Greedy Algorithm with Task Eviction) is the revision of Algorithm ES+EGA with the possibility of evictions of tasks already in $\mathbf{T}^\dagger$. If applying Algorithm LA+LTF to execute $\mathbf{T}^\dagger \cup \{\tau_i\}$ is not a feasible solution or with greater EP than that to execute $\mathbf{T}^\dagger$, Algorithm TE+EGA first finds the index m', in which $\mathbf{T}_{m'}^\dagger$ is the task set $\mathbf{T}_m^\dagger$ of the task partition of $\mathbf{T}^\dagger$ derived from Algorithm LA+LTF with the smallest $\sum_{\tau_j \in \mathbf{T}_m^\dagger} \frac{\chi_j}{p_j} - P^*(\sum_{\tau_j \in \mathbf{T}_m^\dagger} \frac{c_j}{p_j})$. That is, m' is the index, in which evicting all the tasks in $\mathbf{T}_{m'}^\dagger$ increases no greater EP than any other index. Then, if Algorithm LA+LTF can

[1] The pseudo-code of Algorithm ES+EGA is to eliminate the steps between Step 6 and Step 10 in Algorithm 4.

Algorithm 4 : TE+EGA

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

1: sort tasks in $\mathbf{T}$ non-increasingly according to $\frac{\chi_i}{c_i}$;
2: $\mathbf{T}^\dagger \leftarrow \emptyset$, $\mathbf{T}^\sharp \leftarrow \mathbf{T}$;
3: **for** $i \leftarrow 1; i \leq N; i \leftarrow i + 1$ **do**
4:     **if** applying Algorithm LA+LTF to execute $\mathbf{T}^\dagger \cup \{\tau_i\}$ has a feasible solution with less EP than the EP to execute $\mathbf{T}^\dagger$ by applying Algorithm LA+LTF **then**
5:         $\mathbf{T}^\dagger \leftarrow \mathbf{T}^\dagger \cup \{\tau_i\}$, $\mathbf{T}^\sharp \leftarrow \mathbf{T}^\sharp \setminus \{\tau_i\}$;
6:     **else**
7:         let $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger, \ldots, \mathbf{T}_M^\dagger)$ be the task partition of $\mathbf{T}^\dagger$ on M processors derived from Algorithm LA+LTF;
8:         let m' be the index m with the smallest $\sum_{\tau_j \in \mathbf{T}_m^\dagger} \frac{\chi_j}{p_j} - P^*(\sum_{\tau_j \in \mathbf{T}_m^\dagger} \frac{c_j}{p_j})$;
9:         **if** Algorithm LA+LTF can have a feasible task partition for task set $\mathbf{T}^\dagger \setminus \mathbf{T}_{m'}^\dagger \cup \{\tau_i\}$ with less EP than the EP by applying Algorithm LA+LTF to $\mathbf{T}^\dagger$ **then**
10:             $\mathbf{T}^\dagger \leftarrow \mathbf{T}^\dagger \setminus \mathbf{T}_{m'}^\dagger \cup \{\tau_i\}$, $\mathbf{T}^\sharp \leftarrow \mathbf{T}^\sharp \setminus \{\tau_i\} \cup \mathbf{T}_{m'}^\dagger$;
11: return $(\mathbf{T}_1^\dagger, \mathbf{T}_2^\dagger, \ldots, \mathbf{T}_M^\dagger, \mathbf{T}^\sharp)$, where $\mathbf{T}_m^\dagger$ is the task set on the m-th processor by applying Algorithm LA+LTF for $\mathbf{T}^\dagger$;

have a feasible task partition for task set $\mathbf{T}^\dagger \setminus \mathbf{T}_{m'}^\dagger \cup \{\tau_i\}$ with less EP than the EP by applying Algorithm LA+LTF to $\mathbf{T}^\dagger$, we update $\mathbf{T}^\dagger$ as $\mathbf{T}^\dagger \setminus \mathbf{T}_{m'}^\dagger \cup \{\tau_i\}$. The detail procedure is shown in Algorithm 4. Algorithm TE+EGA has the same time complexity as Algorithm ES+EGA, which is $O(N(N + M) \log(N + M))$.

3.1.3 Extensions to non-ideal processors

Algorithms in Sections 3.1.1 and 3.1.2 are designed for ideal processors. With slight modifications, they can be applied to systems with discretely available speeds. As shown in [11, 13], if a task is going to execute for t time units to complete C cycles, we can execute the task at two speeds s_h and s_{h+1}, in which $s_h < \frac{C}{t} \leq s_{h+1}$, for t_h and t_{h+1} time units so that $t_h + t_{h+1}$ is t and $t_h s_h + t_{h+1} s_{h+1}$ is C. Therefore, what we have to do is to re-define the power consumption function P^* in Equation (1) as follows:

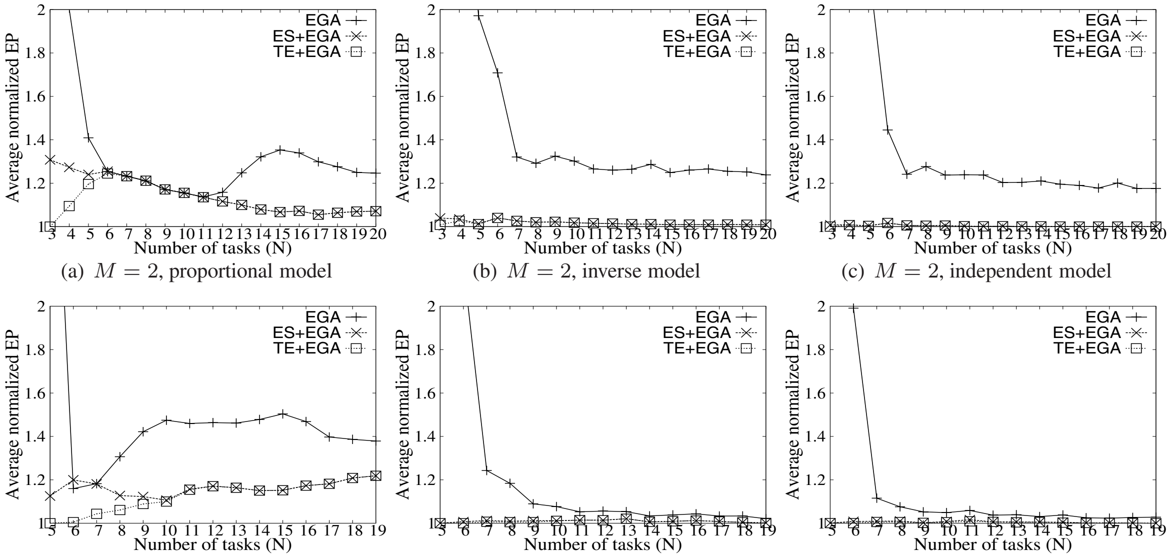
$$P^*(s) = \begin{cases} \left(\begin{array}{l} \frac{s_{h+1} - s}{s_{h+1} - s_h} P(s_h) + \\ \frac{s - s_h}{s_{h+1} - s_h} P(s_{h+1}) \end{array} \right), & \text{when } s_h < s < s_{h+1}, \\ P(s), & \text{when } s = s_h, \text{ for some } h \\ \frac{s}{s^*} P(s^*), & \text{otherwise.} \end{cases} \tag{2}$$

All the algorithms in Sections 3.1.1 and 3.1.2 can be applied to non-ideal processors according to the revision of $P^*(s)$ in Equation (2).

3.2 Systems with non-negligible switching overheads

For systems with non-negligible switching overheads, we first apply the first-fit strategy to re-assigned the tasks selected for execution to reduce the number of processors executed at the critical speed [6]. Then, each processor determines its schedule independently by applying the procrastination algorithm in [12]. Due to space limitation, we only sketch the ideas here.

Suppose that at time instant t, there is no task instance in the ready queue on a processor. By the procrastination algorithm [6, 12], the processor is either turned off or idle at the lowest available speed. The determination of the switching can be done by verifying whether the idle interval is longer than $\max\{t_{sw}, E_{sw}/P(S_{\min})\}$. If the processor is turned off, the scheduler has to decide when to turn on the processor, and the energy consumption in the idle interval is E_{sw}. Suppose that the procrastination schedule decides to turn off the processor at time instant t, and turn on the processor at time instant t^* by applying the procrastination algorithm [12]. We then evaluate whether there is a task instance which is decided to be rejected in the off-line phase and be done before the time instant t^*. If such a task instance exists and the EP obtained in the estimated



(a) $M = 2$, proportional model (b) $M = 2$, inverse model (c) $M = 2$, independent model

(d) $M = 4$, proportional model (e) $M = 4$, inverse model (f) $M = 4$, independent model

Figure 1. Average normalized energy-penalty (EP) for the evaluated algorithms under different models.

schedule is less than that by turning off the processor before t^*, we can execute the task instance instead of turning off the processor. On the other hand, we can also have a similar approach when the processor is determined to be idle before the next task instance assigned on the processor arrives.

4. Performance Evaluations

This section provides evaluation results of the proposed algorithms. Algorithms under simulations are Algorithm SGA, Algorithm EGA, Algorithm ES+EGA, and Algorithm TE+EGA. Due to space limitation, we only present the evaluation results for ideal processors. The results for non-ideal processors are similar.

Environment Setup We perform evaluations for systems with multiple Intel XScale processors. There are five available speeds $(0.15, 0.4, 0.6, 0.8, 1)$ GHz with corresponding power consumption $(80, 170, 400, 900, 1600)$ mW [18] in Intel XScale. For ideal processors, we approximate the power consumption of processor speed s on XScale as $P(s) = 0.08 + 1.52s^3$ W with $S_{\min}$ as 0.15 and $S_{\max}$ as 1. The energy E_{sw} of switching overheads is 483μJ [12].

For each task τ_i, the number of jobs arriving in the hyper-period is determined by an integral variable b_i in the range of $[1, 20]$, where the period of task τ_i is $\frac{L}{b_i}$ for any specified positive real number L. Each task τ_i has two weights $\mu_{i,1}$ and $\mu_{i,2}$ to determine the amount of CPU cycles of tasks on the DVS processors and the rejection penalty. For input instances with N tasks on M processors, the execution cycles c_i on the processor of task τ_i is set as $\frac{\mu_{i,1}}{\sum_{j=1}^{N}\mu_{j,1}}Mp_i$, and rejection penalty of τ_i is $\frac{\mu_{i,2}}{\sum_{j=1}^{N}\mu_{j,2}}3Mp_i$. The linear combination in the objective of the EFFECT problem is $0.2E + 0.8\Pi$, where E is the energy consumption of the system in the hyper-period, and Π is the total rejection penalty of the task instances missing their deadlines in the hyper-period. The value of $\mu_{i,1}$ is a random variable in $(0, 1]$. We explore different types of distribution of $\mu_{i,2}$ depending on the relationships to $\mu_{i,1}$. In the *independent* model, $\mu_{i,2}$ is a random variable in $(0, 1]$; in the *inverse* model, $\mu_{i,2}$ is a random variable in $(0, \frac{1}{\mu_{i,1}}]$; in the *proportional* model, $\mu_{i,2}$ is a random variable in $(\mu_{i,1}, \mu_{i,1} + 0.1]$.

The *normalized energy-penalty (EP)* for an algorithm of an input instance is the energy-penalty of the derived solution divided by the optimal solution of the input instance. For greater numbers of tasks and processors, instead of normalizing to the optimal solution, the relaxed normalized energy-penalty is defined as the energy-penalty of the derived solution divided by the lower bound derived from LEP$(\emptyset, \emptyset, 0)$. We perform independent tests for each configuration, and their average values are reported.

Evaluation Results The average normalized energy-penalty (EP) for the evaluated algorithms when $M = 2$ ($M = 4$, respectively) is shown in Figures 1(a), 1(b), and 1(c) (Figures 1(d), 1(e), and 1(f), respectively) for the proportional, inverse, and independent models. Since Algorithm EGA always outperforms Algorithm SGA, the results for Algorithm SGA are omitted for clarity. We only plot results whose normalized EP is no more than 2 in Figure 1 for clearance. When the number of tasks is quite close to the number of processors, i.e., $N \leq 5$ when $M = 2$ or $N \leq 9$ when $M = 4$, under the proportional model, Algorithm TE+EGA can significantly beat both Algorithms EGA and ES+EGA. This is because Step 10 in Algorithm 4 can be reached by rejecting one or two tasks with higher ratio in their penalty divided by their computation requirement in the task model. When the number of tasks increases, Algorithm TE+EGA and Algorithm ES+EGA have almost the same performance. This is because Step 10 is seldom reached since rejecting more than two tasks in the task model increases a lot of penalty. As in these figures, Algorithm TE+EGA can effectively derive solutions to the EFFECT problem.

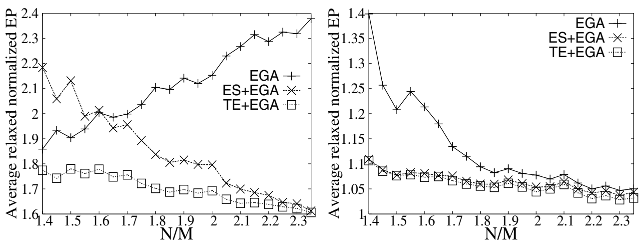
Table 1 shows the running time of the branch-and-bound approach under different pruning methods when M is 4 running on a machine with Intel Pentium4 3GHz CPU and 512M RAM. The LB pruning method uses Algorithm LEP as the lower bound for pruning as shown in Procedure DFSBB in Algorithm 2. The UB pruning method accumulates the EP of the tasks decided so far instead of applying Algorithm LEP in Step 11 in Procedure DFSBB in Algorithm 2. The feasibility pruning method eliminates the steps between Step 9 and Step 14 in Procedure DFSBB in Algorithm 2. As shown in Table 1, applying LB pruning can effectively reduce the running time of the branch-and-bound approach.

We also evaluate the performance of the proposed polynomial-time algorithms for larger input instances. For a given ratio K of N to M, the number of processors is an integral random variable in $[4, 16]$, and the number of tasks in $\mathbf{T}$ is $\lceil KM \rceil$. Figure 2(a) and Figure 2(b) show the average relaxed normalized EP by varying the ratio of M to N when the proportional and the inverse models are applied, respectively. Algorithm TE+EGA is the best among the proposed polynomial-time algorithms. The reason why Algorithm

Pruning methods \ Number of tasks	10	11	12	13	14	15	16	17	18	19
LB pruning	0.19	0.42	1.2	3.9	20.1	80.1	177	988	3621	17232
UB pruning	0.33	0.75	2.80	10.5	59.5	263	797	4507	26140	$>$ 1day
Feasibility pruning	0.8	3.91	20.3	111	521	2352	14261	50134	$>$ 1day	$>$ 1day

unit: sec

Table 1. Running time for different pruning methods in the branch-and-bound approach for $M = 4$.



(a) proportional model (b) inverse model

Figure 2. Average relaxed normalized energy-penalty (EP) for the evaluated algorithms under different models.

EGA outperforms Algorithm ES+EGA when N to M is small (≤ 1.6) for the proportional model in Figure 2(a) is because Algorithm EGA performs task eviction for overloaded processors in Step 5 to Step 8 in Algorithm 3 but Algorithm ES+EGA does not. (It also explains the relation between Algorithms EGA and ES+EGA when $M = 4$ and $N = 6$ in Figure 1(d).) The reason why the average relaxed normalized EP in Figure 2(a) is much greater than that in Figure 2(b) is due to the precision of the derived lower bound by Algorithm LEP.

As shown in Figure 1 and Figure 2, Algorithm TE+EGA and Algorithm ES+EGA have better performance when N to M is higher in most cases, but Algorithm SGA might not. Algorithm TE+EGA is the best among the evaluated algorithms.

5. Conclusion

This research explores systems with the possibility for task rejection in a homogeneous multiprocessor system with continuously available speeds or discretely available speeds. The objective is to minimize the linear combination of the total rejection cost for the tasks that are not completed in time and the energy consumption of the system. We show the $\mathcal{NP}$-hardness of the studied problem, and provide analysis on the non-existence of polynomial-time approximation algorithms, provided that $\mathcal{P} \neq \mathcal{NP}$. We also propose branch-and-bound and efficient algorithms. The proposed algorithms are evaluated by extensive experiments, in which the branch-and-bound approach reduce the running time effectively and Algorithm TE+EGA is shown to provide very effective solution for energy-penalty minimization.

For future research, we will consider systems with heterogeneous multiprocessors.

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. 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.

[4] 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.

[5] 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.

[6] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, pages 408–417, 2006.

[7] 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.

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

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

[14] 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.

[15] 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.

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

[17] INTEL. Strong ARM SA-1100 Microprocessor Developer's Manual, 2003. INTEL.

[18] INTEL-XSCALE, 2003. http://developer.intel.com/design/xscale/.

[19] 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.

[20] 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.

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