Energy Minimization for Periodic Real-Time Tasks on Heterogeneous Processing Units*

Jian-Jia Chen, Andreas Schranzhofer, Lothar Thiele
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
Email: {jchen, schranzhofer, thiele}@tik.ee.ethz.ch

Abstract

Adopting multiple processing units to enhance the computing capability or reduce the power consumption has been widely accepted for designing modern computing systems. Such configurations impose challenges on energy efficiency in hardware and software implementations. This work targets power-aware and energy-efficient task partitioning and processing unit allocation for periodic real-time tasks on a platform with a library of applicable processing unit types. Each processing unit type has its own power consumption characteristics for maintaining its activeness and executing jobs. This paper proposes polynomial-time algorithms for energy-aware task partitioning and processing unit allocation. The proposed algorithms first decide how to assign tasks onto processing unit types to minimize the energy consumption, and then allocate processing units to fit the demands. The proposed algorithms for systems without limitation on the allocated processing units are shown with an $(m+1)$-approximation factor, where m is the number of the available processing unit types. For systems with limitation on the number of the allocated processing units, the proposed algorithm is shown with bounded resource augmentation on the limited number of allocated units. Experimental results show that the proposed algorithms are effective for the minimization of the overall energy consumption.

Keywords: Power-aware design, Task partitioning, Processing unit allocation, Real-time systems, Heterogeneous processing units.

1 Introduction

In the past decade, energy-efficient and low-power design has become important issues in a wide range of computer systems. The pursuit of energy efficiency could be not only useful for mobile devices for the improvement on operating duration but also helpful for server systems for the reduction of power bills. Dynamic power consumption due to switching activities and static power consumption due to the leakage current are two major sources of power consumption of a processing unit [17].

As multiprocessor system-on-chip (MPSoC) platforms, which are composed of multiple heterogeneous processors (processing units), have been widely adopted, a designer can take advantage of the particular processing units' properties to increase the flexibility of the system. The hardware platform may not need to allow for all processing units to execute in parallel. This introduces interesting new options in the embedded systems design process. For example, a field-programmable gate array (FPGA) might be adopted to provide flexibility to execute tasks/jobs in hardware for acceleration. Moreover, due to the dramatic increase on power density, multiprocessor platforms or platforms with co-processing units have become more and more popular in architecture designs. For example, chip makers, such as Intel and AMD, are releasing multi-core chips to improve the system performance instead of increasing the operating frequencies. Such configurations have triggered research in hardware/software co-design to improve the system performance with energy-efficiency considerations.

Power-aware and energy-efficient scheduling for multiprocessor systems has been widely explored in recent years in both academics and industry, especially for real-time systems, e.g., [1, 3, 4, 21, 22, 28], whereas [5] provides a comprehensive survey. However, only few results have been developed for energy-efficient considerations for systems with heterogeneous processing units (processors). By considering the energy consumption for dynamic voltage scaling (DVS) systems with negligible leakage power consumption, heuristic algorithms and approximation algorithms are proposed, e.g., [6, 7, 13–15, 18, 22, 27].

Unfortunately, in nano-meter manufacturing, leakage current contributes significant static power consumption to the system, while the static power consumption is comparable to the dynamic power dissipation [17]. By applying a

*This work is sponsored in part by Taiwan National Science Council NSC-096-2917-I-564-121 and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator (Grant 216008).

dormant (sleep) mode and DVS to reduce the energy consumption in homogeneous multiprocessor systems, Xu et al. [25] and Chen et al. [4] propose polynomial-time algorithms to derive task mappings to try to execute at a critical execution frequency. For homogeneous multiprocessor systems with discrete voltage levels, de Langen and Juurlink [9] provide heuristic algorithms for energy-aware scheduling. For heterogeneous processing units, Andrei et al. [2] develop genetic-based algorithms to map applications with mode execution probabilities to heterogeneous processing units with non-negligible static power consumption. When the number of processing units is small (a constant), Yang et al. [26] develop fully polynomial-time approximation schemes to derive approximation solutions with worst-case guarantees. However, how to design analytical algorithms for achieving energy-efficiency for an arbitrary number of heterogeneous processor units with non-negligible leakage power consumption is still an open question, and to our best knowledge, the to-be-proposed algorithms in this paper are the first ones, for such cases, that have analytical bounds in the worst-case performance.

We explore energy-efficient task partitioning and processing unit allocation for periodic real-time tasks on a given library of processing unit types. By considering both static (leakage) power consumption and dynamic power consumption, the objective of this research is to minimize the overall energy consumption of the system. To simplify the presentation, we only present the results for processing unit types that have only one mode for execution. The extension can be easily achieved for processing unit types that have multiple execution modes, e.g., DVS systems with discrete supply voltages/speeds. Our contributions are as follows:

- We formulate the problem as an integer linear programming problem, and, based on a relaxation of the integral constraints, we provide polynomial-time algorithms to assign tasks onto processing unit types and allocate processing units by applying existing algorithms for the bin-packing problem.
- The algorithms for systems without limitation on the allocated processing units are analyzed to have an $(m+1)$-approximation factor, where m is the number of the available processing unit types. When there is a limitation on the number of the allocated processing units, the proposed algorithm is shown with bounded resource augmentation.
- Experimental results show the effectiveness of the proposed algorithms.

The rest of this paper is organized as follows: Section 2 provides system models. Section 3 shows the hardness analysis of the studied problem. Based on the relaxed linear programming presented in Section 4, we present approximation algorithms in Section 5 when there is no limitation on the number of the allocated processing units. Extensions to systems with limitation on the number of allocated processing units are provided in Section 6. Experimental results are presented in Section 7. Section 8 concludes this paper.

2 System Models

This section presents the problem definition, models of processing unit (abbreviated as PU) types in power consumption and execution, and the task model.

Models of Processing Units The power consumption function P on a PU type has two parts P_d and P_s, where P_d is the dynamic power consumption dissipated for task/job execution and P_s is the static power consumption to maintain the activeness of the PU. For notational brevity, for a PU type M_j, the static (resp. dynamic) power consumption is denoted by $P_{s,j}$ (resp. $P_{d,j}$). The results in this work can be easily extended to systems with multiple dynamic power consumption modes in a PU type, i.e., by applying dynamic voltage scaling (DVS) with discrete levels of supply voltages. Due to space limitation, we will only present the results for systems without DVS capability.

The set of m available PU types is denoted by $\mathbf{M}$. We consider PU types that have significant overhead for turning off [29]. Therefore, when a PU of type M_j executes a job with power characteristics h, the power consumption is assumed to be $P_{s,j} + hP_{d,j}$. On the other hand, when the PU is idle without executing any jobs/tasks, the power consumption is $P_{s,j}$. In other words, if we allocate a PU for executing tasks, when the PU is idle, we cannot turn the PU off, and, hence, have to consume the static power consumption to maintain the activeness of the PU.

Task Model This work explores the scheduling of periodic real-time tasks that are independent in execution. A periodic task τ_i is an infinite number of task instances (jobs) released with periodicity [20]. A task τ_i is characterized by its period p_i. The relative deadline of task τ_i is assumed equal to its period. The (worst-case) execution time of task τ_i on PU type M_j is $c_{i,j}$. Let $\mathbf{T}$ be the input task set of n periodic real-time tasks. To execute a task instance of task τ_i on PU type M_j, the energy consumption is $(h_{i,j}P_{d,j} + P_{s,j})c_{i,j}$, in which $h_{i,j}$ is the *power characteristics* of task τ_i on PU type M_j. Note that, if $c_{i,j} > p_i$, then $c_{i,j}$ is set to ∞, since it is not possible to complete the task on PU type M_j in time. For notational brevity, the *utilization* of task τ_i on PU type M_j is denoted by $u_{i,j} = \frac{c_{i,j}}{p_i}$.

The earliest-deadline-first (EDF) policy is an optimal uniprocessor scheduling policy for independent real-time tasks. A set of tasks is schedulable by EDF if and only if the total utilization of the set of tasks is no more than

100%, where the utilization of a task is defined as its execution time divided by its period. For the rest of this work, we focus on systems that apply EDF scheduling on a PU.

The *hyper-period* of $\mathbf{T}$, denoted by L, is the minimum positive number L so that the released jobs are repeated every L time units. For example, L is the least common multiple (LCM) of the periods of tasks in $\mathbf{T}$ when the periods of tasks are integers. This work focuses on the minimization of the overall energy consumption in the hyper-period L. Another equivalent measurement is the *average power consumption*. Suppose that $\mathbf{T}_j$ is the set of tasks assigned on a PU of type M_j. The average power consumption of the PU is $(\sum_{\tau_i \in \mathbf{T}} u_{i,j} h_{i,j}) P_{d,j} + P_{s,j}$. The average power consumption times the hyper-period (if exists) is the energy consumption of the PU. For the rest of this paper, we assume the existence of L, while the results also hold for minimizing the average power consumption when L does not exist.

Problem Definition This work explores the Minimization of Energy consumption of periodic real-time tasks on platforms with HEterogeneous PUs (abbreviated as the MEHEPU problem). The objective is to partition the input task set $\mathbf{T}$ of n tasks into several disjoint subsets such that the energy consumption in the hyper-period is minimized, in which all the tasks in a partition of tasks are executed on an allocated processing unit in $\mathbf{M}$ of m types without violating their timing constraints. Moreover, when the allocated number of processing unit type M_j is restricted to be no more than F_j, the problem is denoted by the R-MEHEPU problem.

Suppose that the number of allocated units of PU type M_j is K_j. For each task τ_i in $\mathbf{T}$, a binary variable $z_{i,j,k}$ is set as 1 if τ_i is assigned to execute on the k-th allocated unit of PU type M_j; otherwise, $z_{i,j,k} = 0$. The set $\mathbf{T}_{j,k}$ of tasks assigned onto the k-th allocated unit of type M_j is schedulable by EDF if $\sum_{\tau_i \in \mathbf{T}_{j,k}} u_{i,j} \leq 1$. Clearly, $K_j \leq F_j$ for the R-MEHEPU problem, while $K_j \leq n$ for the MEHEPU problem. Therefore, without loss of generality, we can set F_j as n for the MEHEPU problem, and, then, the studied problems can be formulated as an integer linear programming problem as follows:

$$
\begin{aligned}
\text{min} \quad & L\left(\sum_{M_j \in \mathbf{M}} K_j \cdot P_{s,j}\right) \\
& + L\left(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}} \sum_{k=1}^{K_j} u_{i,j} h_{i,j} \cdot z_{i,j,k} P_{d,j}\right) \\
\text{s.t.} \quad & \sum_{M_j \in \mathbf{M}} \sum_{k=1}^{K_j} z_{i,j,k} = 1 \quad , \forall \tau_i \in \mathbf{T}, \\
& \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot z_{i,j,k} \leq 1 \quad , \forall M_j \in \mathbf{M}, k = 1 \ldots K_j, \\
& z_{i,j,k} \in \{0,1\}, \forall \tau_i \in \mathbf{T}, \ \forall M_j \in \mathbf{M}, k = 1 \ldots K_j, \\
& K_j \in \{0,1,2,\ldots,F_j\}, \ \forall M_j \in \mathbf{M}
\end{aligned}
\tag{1}
$$

where the first constraint requires that each task τ_i must execute on one allocated unit only, and the second constraint means that the total utilization of the tasks executing on one allocated PU must be no more than 100%.

3 Hardness Analysis

It is not difficult to see that the MEHEPU problem is $\mathcal{NP}$-hard in a strong sense even when there is only one PU type, since the bin packing problem is its special case. Furthermore, if we are restricted to allocate at most F_j PUs of type M_j, deriving a feasible task partition and PU allocation is $\mathcal{NP}$-complete. In other words, when $F_j < n$ for all PU types M_js, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time algorithm for deriving a feasible solution. Therefore, instead of finding optimal solutions exhaustively, we look for algorithms that can efficiently derive approximated solutions under some bounded factor to the optimal solutions. An algorithm is said to have an *approximation factor* β if the objective function of its derived feasible solution is at most β times of the optimal objective solution for any input instance. The following theorem shows the hardness for deriving approximation algorithms.

Theorem 1 *Unless $\mathcal{P} = \mathcal{NP}$, there does not exist any polynomial-time approximation algorithm with a constant approximation factor for the* MEHEPU *problem.*

Proof. The MEHEPU problem is a special case of the power-aware scenario-mapping problem in [23] with only one scenario. The hardness comes directly from the same L-reduction from the set cover problem in [23]. □

Moreover, the following theorem shows that there does not exist any polynomial-time algorithm to determine whether there exists a feasible task partition or not.

Theorem 2 *Determining whether there exists a feasible task partitioning for the* R-MEHEPU *problem is $\mathcal{NP}$-complete in a strong sense.*

Proof. Consider the special case that $m = 1$ and $F_1 = k$ for some $k \geq 2$. The problem is equivalent to the decision version of the makespan problem, which is known $\mathcal{NP}$-complete in a strong sense [10]. □

As a result, there does not exist any polynomial-time approximation algorithm for the R-MEHEPU problem. Therefore, constraint violation is necessary for designing polynomial-time algorithms. We adopt the resource-augmentation approach [4, 19] so that the resulting schedule is feasible by assuming some bounded augmentation on some F_js.

4 Relaxations of the MEHEPU Problem

Even though we can formulate the problem as an integer linear programming problem, deriving an optimal solution of Equation (1) is still a $\mathcal{NP}$-hard problem. We have to relax the constraints in Equation (1). This section presents how to relax the constraints and the objective function in Equation (1). The naive relaxation will be shown to lead to

an unbounded lower bound to the optimal solution of Equation (1), and, hence, relaxation must be done carefully.

The first relaxation will be on the objective function to reduce the number of variables required in the programming. For each task τ_i in $\mathbf{T}$, a binary variable $y_{i,j}$ is set as 1 if τ_i is assigned to execute on a unit of PU type M_j; otherwise, $y_{i,j} = 0$. As a result, the number $\lceil \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j} \rceil$ is the lower bound of the required units of PU type M_j. Equation (1) could be relaxed into the following integer linear programming problem:

$$\begin{array}{ll} \min & L(\sum_{M_j \in \mathbf{M}} \lceil \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j} \rceil \, P_{s,j}) \\ & + L(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}} u_{i,j} h_{i,j} \cdot y_{i,j} P_{d,j}) \\ \text{s.t.} & \sum_{M_j \in \mathbf{M}} y_{i,j} = 1 \ \ , \forall \tau_i \in \mathbf{T}, \text{ and} \\ & y_{i,j} \in \{0,1\}, \forall \tau_i \in \mathbf{T}, \forall M_j \in \mathbf{M}. \end{array} \tag{2}$$

For any feasible solution of Equation (2), each task is assigned to exactly one PU type. Let task set $\mathbf{T}_j$ be the set of the tasks in $\mathbf{T}$ assigned on type M_j for a solution of Equation (2), i.e., $\mathbf{T}_j = \{\tau_i \in \mathbf{T} \mid y_{i,j} = 1\}$. To allocate the units of type M_j to execute tasks in $\mathbf{T}_j$, we apply the algorithms for the traditional bin packing problem such as the first-fit, last-fit, worst-fit, and best-fit strategies by taking 100% as the capacity of a bin (a PU) and the utilization of a task on the PU type as the size of a corresponding item.

Unfortunately, deriving an optimal solution for Equation (2) is still $\mathcal{NP}$-hard. We could relax the integral constraint of Equation (2) as well as the ceiling function of the objective function of Equation (2) as follows:

$$\min L(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j}(P_{s,j} + h_{i,j} P_{d,j})) \tag{3a}$$

$$\text{s.t.} \sum_{M_j \in \mathbf{M}} y_{i,j} = 1 \ \ , \forall \tau_i \in \mathbf{T}, \text{ and} \tag{3b}$$

$$y_{i,j} \geq 0, \forall \tau_i \in \mathbf{T}, \forall M_j \in \mathbf{M}. \tag{3c}$$

However, the following theorem shows that such a relaxation might lead to an unbounded relaxed solution.

Theorem 3 *There exists a set of input instances for the* MEHEPU *problem such that the ratio of the optimal solution of Equation (2) to the optimal solution of Equation (3) is* ∞.

Proof. The proof is in the appendix by providing a set of input instances that satisfy the statement. □

Theorem 3 reveals that the relaxation in Equation (3) is too much so that the optimal solution of Equation (3) might be unbounded to the optimal solution of the MEHEPU problem. Deriving a solution based on the optimal solution of Equation (3) cannot have any worst-case guarantee by comparing to the optimal relaxed solution of Equation (3) since the relaxation is not bounded. As a result, other non-trivial relaxations are needed to have worst-case guarantees. The reason why the gap between the optimal solutions in Equation (2) and Equation (3) is unbounded is because of the under-estimation of the static energy consumption. To overcome this issue, once we decide to assign a task to a PU type, we have to take its static energy consumption as the minimum cost. However, it takes exponential time to decide which PU unit type should be allocated or not. This paper adopts a greedy approach for the relaxation of Equation (2).

The proposed relaxation first sorts PU types according to their static power consumptions. Therefore, for notational brevity, we re-index the processing unit types so that $P_{s,1} \leq P_{s,2} \leq \cdots \leq P_{s,m}$. Then, for a specified parameter $\hat{m}$, we specify the solution by allocating (1) at least one processing unit of PU type $\hat{m}$ and (2) no processing unit of PU type M_j with $j > \hat{m}$. Therefore, the relaxation of Equation (2) based on the above restriction for a given parameter $\hat{m}$ is as follows:

$$\min \ L\left(\begin{array}{c} P_{s,\hat{m}} x_{\hat{m}} + \sum_{j=1}^{\hat{m}-1} \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j} P_{s,j} \\ + \sum_{j=1}^{\hat{m}} \sum_{\tau_i \in \mathbf{T}} u_{i,j} h_{i,j} \cdot y_{i,j} P_{d,j} \end{array} \right) \tag{4a}$$

$$\text{s.t.} \ \sum_{j=1}^{\hat{m}} y_{i,j} = 1 \ \ , \forall \tau_i \in \mathbf{T}, \tag{4b}$$

$$y_{i,j} \geq 0, \forall \tau_i \in \mathbf{T}, \ \forall j = 1, 2, \ldots, \hat{m}, \tag{4c}$$

$$\sum_{\tau_i \in \mathbf{T}} y_{i,\hat{m}} u_{i,\hat{m}} \leq x_{\hat{m}}, \text{ and} \tag{4d}$$

$$x_{\hat{m}} \geq 1, \tag{4e}$$

where $x_{\hat{m}}$ is a variable no less than 1 to indicate whether PU type $M_{\hat{m}}$ is allocated with more than 1 units.

By setting $\hat{m}$ from 1 to m, let m^* be the index $\hat{m}$ with the minimum objective function in Equation (4a). The following lemma shows that we can build a lower bound based on the above relaxation.

Lemma 1 *Suppose m^* is the index $\hat{m}$ with the minimum value in Equation (4a). The objective function in Equation (4a) by setting parameter $\hat{m}$ as m^* is no more than the optimal solution of Equation (2).*

Proof. Suppose that PU type M_{j^*} is the one with the largest static power consumption among those PU types M_j with $\sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j} > 0$ in the optimal solution of Equation (2). Therefore, the optimal solution of Equation (4) by setting $\hat{m}$ as j^* is no more than the optimal solution of Equation (2). □

5 Algorithms for the MEHEPU Problem

This section provides approximation algorithms for the MEHEPU problem, in which we are allowed to allocate any

amount of processing unit types. We will first show how to derive an optimal solution for Equation (4) under a specified parameter $\hat{m}$, and then, based on the optimal solution of the relaxed problem, we present how to derive solutions with worst-case guarantees in terms of energy consumption minimization.

5.1 Deriving optimal solutions for the relaxation

Deriving an optimal solution for Equation (4) in polynomial time can be achieved by applying linear-programming toolkits, such as GLPK [11] or CPLEX [16]. This subsection provides a combinatorial approach to derive the optimal solution for Equation (4), which is much more efficient. Note that, in this subsection, we only deal with the case that $\hat{m}$ is given as a fixed parameter.

To derive an optimal solution of Equation (4) for a specified $\hat{m}$, we will first use the *extreme point theory* [8] to show the number of fractional variables, and then provide a solution based on the property. According to the feasible domains of linear programming [8], the feasible solutions of a linear programming problem with ρ variables form a *convex set* in ρ dimensions. Furthermore, there exists an optimal solution for a linear programming problem at an *extreme point* of the convex set [8, 24], where an extreme point of the convex set is a member in the convex set which can not be expressed by the convex combination of any two distinct members in the convex set. For detail, we refer interested readers to [8, 24]. Specifically, an extreme point of the convex set of the feasible solutions for a linear programming problem with ρ variables makes at least ρ inequalities in the linear programming problem tight. A solution is said to be an *extreme-point* solution if it is at an extreme point. By applying the extreme point theory, we can reach the following lemma.

Lemma 2 *For an optimal solution at an extreme point of Equation (4) with a specified $\hat{m}$, (1) if $x_{\hat{m}} > 1$, all the variables $y_{i,j}$s are integral in the solution, and (2) if $x_{\hat{m}} = 1$, at most two variables $y_{i,j}$s are fractional.*

Proof. Clearly, there are $\hat{m}n+1$ variables in Equation (4a), and there must be at least $\hat{m}n + 1$ tight (sub-)equations among the number of (sub-)equations in Equations (4b), (4c), and (4d). Here are two cases:

- $x_{\hat{m}} > 1$**:** For an optimal solution in such a case, it is not difficult to see that $\sum_{\tau_i \in \mathbf{T}} y_{i,\hat{m}} u_{i,\hat{m}}$ is equal to $x_{\hat{m}}$; otherwise, it contradicts the optimality. Therefore, there are $n + 1$ (sub-)equations that are tight from Equation (4b) and Equation (4d). Among the (sub-)equations in Equation (4c), there must be at least $\hat{m}n - n$ tight (sub-)equations. Therefore, there are at most n (sub-)equations in Equation (4c) that are not tight. As a result, there must be at most n variables of $y_{i,j}$s that are greater than 0. By Equation (4b) and the extreme point theory, when $x_{\hat{m}} > 1$, there exists an optimal solution that has no fractional variables in $y_{i,j}$s.
- $x_{\hat{m}} = 1$**:** Similar to the other case, if $\sum_{\tau_i \in \mathbf{T}} y_{i,\hat{m}} u_{i,\hat{m}}$ is less than 1, the extreme point theory shows that there exists an optimal solution that has no fractional variables in $y_{i,j}$s as well as $x_{\hat{m}}$. However, if $\sum_{\tau_i \in \mathbf{T}} y_{i,j} u_{i,j}$ is equal to 1, since there are at least $\hat{m}n - n - 1$ (sub-)equations in Equation (4c) that are tight, an extreme-point solution has at most $n + 1$ nonzero variables of $y_{i,j}$s. As a result, there is at most one task τ_i with $0 < y_{i,j} < 1$ and at least $n - 1$ tasks with integral variables in an extreme-point solution.

□

We now show how to assign tasks onto PU types by applying the statement in Lemma 2. The combinatorial algorithm for deriving an optimal solution for Equation (4) is illustrated in Algorithm 1.

For the case that $x_{\hat{m}} > 1$, since all the variables $y_{i,j}$ in an extreme-point solution are either 0 or 1, we should assign task τ_i to the PU type with the minimum cost in the objective function. That is, we just have to assign task τ_i to the PU type M_j ($j \leq \hat{m}$) with the minimum $u_{i,j}(P_{s,j} + h_{i,j}P_{d,j})$, while ties are broken by choosing the largest index j. For notational brevity, let $g_{\hat{m}}(\tau_i)$ be the index j of PU type M_j with the minimum energy consumption above, i.e.,

$$g_{\hat{m}}(\tau_i) = \max\left(\arg_{j=1,2,\ldots,\hat{m}} \min\{u_{i,j}(P_{s,j} + h_{i,j}P_{d,j})\}\right). \quad (5)$$

As a result, if assigning a task to the PU type with the minimum energy consumption (by considering the static power and the dynamic power together) for execution and the total utilization of tasks assigned on PU type $M_{\hat{m}}$ is greater than 100%, the derived solution is optimal for Equation (4).

For the other case that the total utilization of tasks assigned on PU type $M_{\hat{m}}$ is no more than 100% after assigning tasks to the PU type with the minimum energy consumption, we need to perform task reassignment. Since, in this case, the static energy consumption has been counted already and can be treated as a constant in the objective function in Equation (4a), we can reassign some tasks τ_is that have less *dynamic energy consumption* on PU type $M_{\hat{m}}$ than the overall energy consumption (including the static and the dynamic energy) on PU type $M_{g_{\hat{m}}(\tau_i)}$. Therefore, tasks τ_is with $g_{\hat{m}}(\tau_i) \neq \hat{m}$ and $u_{i,g_{\hat{m}}(\tau_i)}(P_{s,g_{\hat{m}}(\tau_i)} + h_{i,g_{\hat{m}}(\tau_i)}P_{d,g_{\hat{m}}(\tau_i)}) > u_{i,\hat{m}}h_{i,\hat{m}}P_{d,\hat{m}}$ are candidates for reassignment. Moving a task τ_i with υ portion ($0 \leq \upsilon \leq 1$) from PU type $M_{g_{\hat{m}}(\tau_i)}$ to PU type $M_{\hat{m}}$ reduces the energy consumption by $\upsilon L(u_{i,g_{\hat{m}}(\tau_i)}(P_{s,g_{\hat{m}}(\tau_i)} + h_{i,g_{\hat{m}}(\tau_i)}P_{d,g_{\hat{m}}(\tau_i)}) - u_{i,\hat{m}}h_{i,\hat{m}}P_{d,\hat{m}})$ with utilization increase on PU type

Algorithm 1

Input: $\mathbf{T}$, $\mathbf{M}$, and $\hat{m}$;
Output: the optimal solution for Equation (4) along with a variable assignment;

1: find $g_{\hat{m}}(\tau_i)$;
2: **if** there exists a task τ_i with $c_{i,g_{\hat{m}}(\tau_i)} = \infty$ **then**
3: return ∞;
4: $y^*_{i,j} \leftarrow 0$ for every task $\tau_i \in \mathbf{T}$ with $j = 1, 2, \ldots, \hat{m} \neq g_{\hat{m}}(\tau_i)$, and $y^*_{i,g_{\hat{m}}(\tau_i)} \leftarrow 1$ for every task $\tau_i \in \mathbf{T}$;
5: **if** $\sum_{\tau_i \in \mathbf{T}} u_{i,\hat{m}} y^*_{i,\hat{m}} > 1$ **then**
6: return the solution of Equation (4) by assigning $y_{i,j}$ as $y^*_{i,j}$ and $x_{\hat{m}}$ as $\sum_{\tau_i \in \mathbf{T}} u_{i,\hat{m}} y^*_{i,\hat{m}}$;
7: let $U \leftarrow \sum_{\tau_i \in \mathbf{T}} u_{i,\hat{m}} y^*_{i,\hat{m}}$;
8: **while** $U < 1$ **do**
9: find the task $\tau_{i'}$ with $y^*_{i',\hat{m}} = 0$ and the *largest* positive $\frac{u_{i',\gamma}(P_{s,\gamma}+h_{i',\gamma}P_{d,\gamma})-u_{i',\hat{m}}h_{i',\hat{m}}P_{d,\hat{m}}}{u_{i',\hat{m}}}$, where $\gamma = g_{\hat{m}}(\tau_{i'})$ for simplicity;
10: **if** task $\tau_{i'}$ does not exist **then**
11: $U \leftarrow 1$; break;
12: **if** $U + u_{i',\hat{m}} < 1$ **then**
13: $y^*_{i',\hat{m}} \leftarrow 1$, $U \leftarrow U + u_{i',\hat{m}}$, and $y^*_{i',g_{\hat{m}}(\tau_{i'})} \leftarrow 0$;
14: **else**
15: $\upsilon \leftarrow \frac{(1-U)}{u_{i',\hat{m}}}$, $y^*_{i',\hat{m}} \leftarrow \upsilon$, $U \leftarrow 1$, and $y^*_{i',g_{\hat{m}}(\tau_{i'})} \leftarrow 1-\upsilon$;
16: return the solution of Equation (4) by assigning $y_{i,j}$ as $y^*_{i,j}$ and $x_{\hat{m}}$ as U;

$M_{\hat{m}}$ by $\upsilon u_{i,\hat{m}}$. As a result, to maximize the energy saving, we always choose the task τ_i with the maximum $\frac{L(u_{i,g_{\hat{m}}(\tau_i)}(P_{s,g_{\hat{m}}(\tau_i)}+h_{i,g_{\hat{m}}(\tau_i)}P_{d,g_{\hat{m}}(\tau_i)})-u_{i,\hat{m}}h_{i,\hat{m}}P_{d,\hat{m}})}{u_{i,\hat{m}}}$ to reassign until the utilization on PU type $M_{\hat{m}}$ is 100% or such a candidate task τ_i does not exist anymore.

For a specified $\hat{m}$, we say that a task τ_i is *with fractional variables* if there exists some $0 < y_{i,j} < 1$. Based on the above observation, it is not difficult to see that Algorithm 1 derives an optimal solution for Equation (4) in polynomial time.

Theorem 4 *Algorithm 1 derives an optimal solution for Equation (4) for a specified $\hat{m}$ in $O(nm + n \log n)$.*

Proof. For the case that the optimal solution is without fractional variables, i.e., the utilization on PU type $M_{\hat{m}}$ is not equal to 100%, the optimality is clear, because, among the solutions without fractional variables, the derived solution is the best. For the other case, the optimality can be proved by applying the proof procedure of the greedy approach for the fractional knapsack problem as shown in [12]. One can imagine that when U in Algorithm 1 is less than 100%, we select tasks, by allowing fractional selection, with the maximum rate of the energy reduction to the increase of total utilization on PU type $M_{\hat{m}}$, which is the same as the fractional knapsack problem. The time complexity is $O(nm + n \log n)$, in which the first term is for the calculation of $g_{\hat{m}}()$ of and the second term comes from Step 9 by applying a heap data structure. □

5.2 Deriving approximation solutions

Based on the solution derived from Algorithm 1, we now show how to assign tasks and allocate PU types. Our first proposed algorithm, denoted as Algorithm S-GREEDY, first derives a minimum feasible solution among the m equations of all combinations in Equation (4). For a specified $\hat{m}$, let $\vec{y}_{\hat{m}}$ be the vector of the variables derived by applying Algorithm 1, where variable $y_{i,j}$ in vector $\vec{y}_{\hat{m}}$ is set as 0 if $j > \hat{m}$. If no feasible solution is derived from these m linear programming problems, there does not exist any feasible solution for such an input instance. Otherwise, let the variable assignment with the minimum value in the objective function of Equation (4) be $\vec{y}_{m^*}$, where $y^{m^*}_{i,j}$ is the corresponding variable assignment.

Then, Algorithm S-GREEDY simply assigns task τ_i to the PU type M_j with $y^{m^*}_{i,j} = 1$. For task τ_{i^*} with fractional variables (if it exists), we greedily assign this task to the PU type M_j ($j \leq m^*$) with the minimum dynamic energy consumption. Let $\mathbf{T}_j$ be the set of tasks assigned on PU type M_j. Then, for each PU type M_j with non-empty task set $\mathbf{T}_j$, heuristic algorithms, such as first-fit, last-fit, best-fit, or worst-fit algorithms, for the bin packing problem are applied to minimize the number of allocated units of PU type M_j. That is, tasks in $\mathbf{T}_j$ are considered one by one, and are assigned to the allocated unit that is fit firstly, is fit lastly, is fit with the maximal utilization, and is fit with the minimal utilization for the first-fit, the last-fit, the best-fit, and the worst-fit bin packing algorithms, respectively. Once there is no allocated unit of M_j that can fit the considered task in $\mathbf{T}_j$, all the fitting algorithms allocate a new processing unit of PU type M_j and assign the task to the newly allocated unit. The time complexity of Algorithm S-GREEDY is $O(m(nm + n \log n) + n^2)$, in which $O(n^2)$ comes from the adopting of the bin packing algorithms. Moreover, it is clear that all the tasks can meet their timing constraints by applying EDF scheduling on each allocated PU.

Analysis of Algorithm S-GREEDY After presenting Algorithm S-GREEDY, we now show that the approximation factor of Algorithm S-GREEDY is $m + 1$. As fitting algorithms, such as first-fit, last-fit, best-fit, or worst-fit, of the bin-packing problem are adopted as a subroutine, the following lemma shows the upper bound of the number of the allocated processing units of a given PU type.

Lemma 3 *Given a non-empty task set $\mathbf{T}_j$ to be allocated on PU type M_j, the number of the allocated processing units of type M_j by applying the first-fit, the last-fit, the best-fit, or the worst-fit algorithm is at most $\max\{1, 2\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}\}$.*

Algorithm 2 S-GREEDY

Input: $\mathbf{T}$ and $\mathbf{M}$, m, where $P_{s,1} \leq P_{s,2} \leq \cdots \leq P_{s,m}$;
Output: a feasible solution for the MEHEPU problem;

1: find the best solution $\vec{y}_{\hat{m}}$ by setting $\hat{m}$ from 1 to m by applying Algorithm 1;
2: let $\vec{y}_{m^*}$ be the vector among the m vectors $\vec{y}_{\hat{m}}$ with the minimum objective function in Equation (4);
3: assign task τ_i to PU type M_j if $y_{i,j}^{m^*}$ is 1;
4: let τ_i^* be the task with fractional variables $0 < y_{i^*,j}^{m^*} < 1$ for some M_j;
5: assign task τ_{i^*} to PU type M_j with the minimum $u_{i^*,j}h_{i,j}P_{d,j}$ where $j \leq m^*$;
6: let $\mathbf{T}_j$ be the set of tasks assigned on PU type M_j;
7: **for each** PU type M_j with non-empty task set $\mathbf{T}_j$ **do**
8: allocate PUs of type M_j by applying bin-packing algorithms (e.g., first-fit, last-fit, best-fit, or worst-fit algorithms) for $\mathbf{T}_j$ so that the utilization of an allocated PU is no more than 100%;
9: return the task assignment as well as the PU allocation;

Proof. If $\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}$ is no more than 1, all the fitting algorithms would allocate only one unit and assign all of the tasks in $\mathbf{T}_j$ on that unit. Therefore, only one unit of M_j is allocated in this case.

We now consider the other case that $\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}$ is more than 1. Suppose that the number of the allocated units of PU type M_j by a fitting algorithm is σ, where σ must be at least 2 because of the total utilization on the PU type. By applying these fitting algorithms, there must be at most one allocated unit of PU type M_j with utilization less than $\frac{1}{2}$; otherwise, the fitting strategy is contradicted. If there is no allocated PU of type M_j with utilization less than $\frac{1}{2}$, we know that $\sigma \leq 2\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}$. Otherwise, let u^* be the utilization of the allocated unit of PU type M_j with utilization less than $\frac{1}{2}$. All of the other $\sigma - 1$ allocated units of type M_j must have utilization no less than $1 - u^*$ due to the greedy fitting strategy. As a result, we know that $\sum_{\tau_i \in \mathbf{T}_j} u_{i,j} \geq u^* + (\sigma - 1)(1 - u^*) \geq \sigma/2$ since $u^* < 1/2$ and $\sigma \geq 2$.

Therefore, we reach the conclusion that the number of the allocated processing units of type M_j by applying the first-fit, the last-fit, the best-fit, or the worst-fit algorithm is at most $\max\{1, 2\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}\}$. □

Based on the lemma, we now show the approximation factor of Algorithm S-GREEDY.

Theorem 5 *Algorithm* S-GREEDY *is a polynomial-time* $(m+1)$*-approximation algorithm for the* MEHEPU *problem.*

Proof. We prove this theorem by showing that the resulting solution of Algorithm S-GREEDY would have energy consumption no more than $(m^* + 1)$ times of the optimal solution, where m^* is the variable $\hat{m}$ with the best solution $\vec{y}_{\hat{m}}$ by applying Algorithm 1. By Lemma 3, we know that the resulting allocation of Algorithm S-GREEDY uses at most $\max\{1, 2\sum_{\tau_i \in \mathbf{T}_j} u_{i,j}\}$ units of PU type M_j. Here are two cases, determined by whether the task τ_{i^*} with fractional variables exists or not.

If task τ_{i^*} does not exist and $\sum_{\tau_i \in \mathbf{T}_{m^*}} u_{i,m^*} > 1$, let E^* be the resulting objective function of Equation (4), where $E^* = L \times (\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}(h_{i,j}P_{d,j} + P_{s,j}))$. Let set $\mathbf{M}^\dagger$ be the set of PU types with utilization larger than 1, i.e., $\left\{M_j \in \mathbf{M} \mid \sum_{\tau_i \in \mathbf{T}_j} u_{i,j} > 1\right\}$. Since $\sum_{\tau_i \in \mathbf{T}_{m^*}} u_{i,m^*} > 1$, we know that $\mathbf{M}^\dagger$ is a non-empty set. Similarly, let set $\mathbf{M}^\sharp$ be the set of PU types with assigned tasks and with utilization no more than 1, i.e., $\left\{M_j \in \mathbf{M} \mid 0 < \sum_{\tau_i \in \mathbf{T}_j} u_{i,j} \leq 1\right\}$. The resulting energy consumption of Algorithm S-GREEDY for tasks on the PU types in $\mathbf{M}^\dagger$ is at most $L \times \left(\sum_{M_j \in \mathbf{M}^\dagger} \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}(2P_{s,j} + h_{i,j}P_{d,j})\right)$. The resulting energy consumption of Algorithm S-GREEDY to schedule tasks on the PU types in $\mathbf{M}^\sharp$ is

$$
\begin{aligned}
& L \times \Big(\sum_{M_j \in \mathbf{M}^\sharp} (P_{s,j} + \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}h_{i,j}P_{d,j}) \Big) \\
\leq & L \times (P_{s,m^*}|\mathbf{M}^\sharp| + \sum_{M_j \in \mathbf{M}^\sharp} \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}h_{i,j}P_{d,j}),
\end{aligned}
$$

where $|\mathbf{M}^\sharp|$ is the cardinality of set $\mathbf{M}^\sharp$ and the inequality comes from the definition of $P_{s,1} \leq P_{s,2} \leq \cdots \leq P_{s,m^*}$. Therefore, as $|\mathbf{M}^\sharp| \leq m^* - |\mathbf{M}^\dagger| \leq m^* - 1$, we know that the energy consumption E of the resulting solution is no more than $(m^* + 1)E^*$ as follows:

$$
\begin{aligned}
E \leq & \; L(P_{s,m^*}(m^* - |\mathbf{M}^\dagger|)) \\
& + 2L\Big(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}(h_{i,j}P_{d,j} + P_{s,j}) \Big) \\
\leq & \; (m^* - 1)E^* + 2E^* \leq (m^* + 1)E^*,
\end{aligned}
$$

where the second inequality comes from $L \times P_{s,m^*} \leq E^*$ and $L(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}_j} u_{i,j}(h_{i,j}P_{d,j} + P_{s,j})) \leq E^*$. If task τ_{i^*} does not exist and $\sum_{\tau_i \in \mathbf{T}_{m^*}} u_{i,m^*} \leq 1$, the analysis is similar, in which the energy consumption is at most $(m^* + 1) \times E^*$.

For the other case that τ_{i^*} exists, let E^* be the resulting objective function of Equation (4), where $E^* = L \times (P_{s,m^*} + \sum_{j=1}^{m^*-1} \sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j}^{m^*} P_{s,j} + \sum_{j=1}^{m^*} \sum_{\tau_i \in \mathbf{T}} u_{i,j}h_{i,j} \cdot y_{i,j}^{m^*} P_{d,j})$. Suppose that task τ_{i^*} is assigned to PU type M_{j^*}, in which $j^* \leq m^*$. Clearly, by definition, we have $u_{i^*,j^*}h_{i^*,j^*}P_{d,j^*} \leq u_{i^*,j}h_{i^*,j}P_{d,j}$ for any $j \leq m^*$. Therefore, by definition, we know that $(u_{i^*,j^*}h_{i^*,j^*}P_{d,j^*} + P_{s,j^*}) \leq (u_{i^*,j^*}h_{i^*,j^*}P_{d,j^*} + P_{s,m^*}) \leq E^*$. Define $\mathbf{M}^\dagger$ and $\mathbf{M}^\sharp$ similarly by excluding task τ_{i^*}. That is, we force PU type M_{m^*} to exist in $\mathbf{M}^\sharp$ instead of $\mathbf{M}^\dagger$. Then, we know that $L(P_{s,m^*} + \sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}_j \setminus \{\tau_{i^*}\}} u_{i,j}h_{i,j}P_{d,j} +$

Algorithm 3 E-GREEDY

Input: $\mathbf{T}$ and $\mathbf{M}$, m, where $P_{s,1} \leq P_{s,2} \leq \cdots \leq P_{s,m}$;
Output: a feasible solution for the MEHEPU problem;

1: **for** $\hat{m} \leftarrow 1; \hat{m} \leq m; \hat{m} \leftarrow \hat{m} + 1$ **do**
2: find the solution $\vec{y}_{\hat{m}}$ by applying Algorithm 1;
3: **if** the optimal solution of Equation (4) is ∞ **then**
4: continue;
5: schedule $S_{\hat{m}}$ assigns τ_i to PU type M_j if $y^*_{i,j}$ is 1;
6: let τ^*_i be the task with fractional variables $0 < y^*_{i^*,j} < 1$ for some M_j;
7: schedule $S_{\hat{m}}$ assigns task τ_{i^*} to PU type M_j with the minimum $u_{i^*,j}h_{i,j}P_{d,j}$ where $j \leq \hat{m}$;
8: let $\mathbf{T}_j$ be the set of tasks assigned on PU type M_j;
9: **for each** PU type M_j assigned with non-empty task set $\mathbf{T}_j$ in $S_{\hat{m}}$ **do**
10: allocate PUs of type M_j by applying bin-packing algorithms (e.g., first-fit, last-fit, best-fit, or worst-fit algorithms) for $\mathbf{T}_j$ so that the utilization of an allocated PU in $S_{\hat{m}}$ is no more than 1;
11: return the task assignment and the PU allocation in S_j with the minimum energy consumption;

$\sum_{M_j \in \mathbf{M}^\dagger} \sum_{\tau_i \in \mathbf{T}_j \setminus \{\tau_{i^*}\}} u_{i,j}P_{s,j}) \leq E^*$. Therefore, the energy consumption E of the resulting solution is no more than $(m^* + 1)E^*$:

$$\begin{aligned} E &\leq L(P_{s,m^*}(m^* - |\mathbf{M}^\dagger|) + u_{i^*,j^*}h_{i^*,j^*}P_{d,j^*} + P_{s,j^*}) \\ &\quad + L\Big(\sum_{M_j \in \mathbf{M}} \sum_{\tau_i \in \mathbf{T}_j \setminus \{\tau_{i^*}\}} u_{i,j}h_{i,j}P_{d,j}\Big) \\ &\quad + 2L\Big(\sum_{M_j \in \mathbf{M}^\dagger} \sum_{\tau_i \in \mathbf{T}_j \setminus \{\tau_{i^*}\}} u_{i,j}P_{s,j}\Big) \\ &\leq (m^* + 1)E^*. \end{aligned}$$

As Algorithm S-GREEDY is with polynomial-time complexity and $m^* \leq m$, we reach the conclusion of this theorem. □

Algorithm E-GREEDY, illustrated in Algorithm 3, can further improve the performance of Algorithm S-GREEDY by choosing the best solution among the m possible task assignment and PU allocations. If the linear programming of Equation (4) has feasible solutions for a given $\hat{m}$, Algorithm E-GREEDY determines a task assignment and PU allocation based on the linear-programming result, and returns the best one as the final solution. The time complexity of Algorithm E-GREEDY is at most $O(m)$ times of that of Algorithm S-GREEDY. Clearly, a solution of Algorithm E-GREEDY is no worse than that of Algorithm S-GREEDY for any input instance. Algorithm E-GREEDY is also a polynomial-time $(m + 1)$-approximation algorithm for the MEHEPU problem.

We have shown that the approximation factors of Algorithm S-GREEDY and Algorithm E-GREEDY are both $(m + 1)$. The proof in Appendix shows that the analysis is almost tight as there exists a set of input instances with a gap close to m between the derived solutions and the optimal ones. Moreover, it is not difficult to see that applying bin-packing algorithms by using the linear programming solution of Equation (3) for task assignment will lead to unbounded solutions for some input instances.

6 Algorithms for the R-MEHEPU Problem

This section presents how to cope with the R-MEHEPU problem, in which the number of the allocated units of a PU type M_j is restricted by a parameter $F_j \neq \infty$. Recall that deriving a feasible task partition and PU allocation for the R-MEHEPU problem is $\mathcal{NP}$-complete in a strong sense in Theorem 2. Unless $\mathcal{P} = \mathcal{NP}$, it is not possible to have polynomial-time approximation algorithms for the challenging problem. Therefore, in this section, we will apply the widely adopted resource-augmentation approach in the literature, e.g., [4, 19]. Specifically, the resulting solution will augment the allocation constraint F_j with a bounded factor.

Again, the naive relaxation does not work well for the R-MEHEPU problem. We extend the linear programming relaxation in Equation (4) by introducing one additional inequality to restrict that the utilization of a PU type is no more than F_j. Therefore, for a specified parameter $\hat{m}$, the relaxed linear programming of the R-MEHEPU problem is as follows:

$$\min \quad L\left(\begin{array}{c} P_{s,\hat{m}}x_{\hat{m}} + \sum_{j=1}^{\hat{m}-1}\sum_{\tau_i \in \mathbf{T}} u_{i,j} \cdot y_{i,j}P_{s,j} \\ + \sum_{j=1}^{\hat{m}}\sum_{\tau_i \in \mathbf{T}} u_{i,j}h_{i,j} \cdot y_{i,j}P_{d,j} \end{array}\right) \tag{6a}$$

$$\text{s.t.} \quad \sum_{j=1}^{\hat{m}} y_{i,j} = 1 \quad , \forall \tau_i \in \mathbf{T}, \tag{6b}$$

$$y_{i,j} \geq 0, \forall \tau_i \in \mathbf{T}, \forall j = 1, 2, \ldots, \hat{m}, \tag{6c}$$

$$\sum_{\tau_i \in \mathbf{T}} y_{i,\hat{m}}u_{i,\hat{m}} \leq x_{\hat{m}} \tag{6d}$$

$$x_{\hat{m}} \geq 1, \text{ and} \tag{6e}$$

$$\sum_{\tau_i \in \mathbf{T}} y_{i,j}u_{i,j} \leq F_j, \forall j = 1, 2, \ldots, \hat{m}. \tag{6f}$$

Then, Algorithm S-GREEDY and Algorithm E-GREEDY can be extended to assign tasks and allocate processing units based on the optimal solution of the linear programming in Equation (6) for a specified $\hat{m}$. Unfortunately, for such cases, we have no combinatorial algorithm so far, and, hence, the linear-programming toolkits, such as CPLEX [16] or GLPK [11], are applied to solve Equation (6) optimally in polynomial time. By applying the extreme point theory, the following lemma shows that the number of tasks with fractional variables for an optimal solution in an extreme point of Equation (6) is bounded by $\hat{m}$.

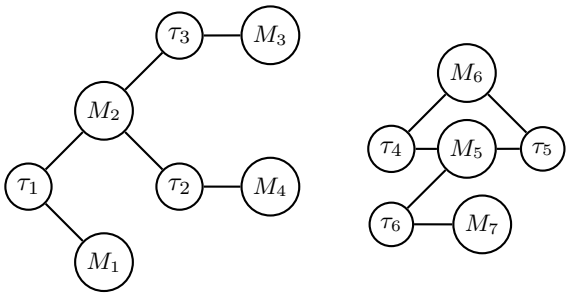


Figure 1. An example for graph G to assign tasks.

Lemma 4 *There are at most $\hat{m}$ tasks with fractional variables $y_{i,j}$ for an extreme point solution of Equation (6).*

Proof. Clearly, there are $\hat{m}n + 1$ variables in Equation (6). Therefore, at least $\hat{m}n + 1$ constraints in Equation (6) must be tight for an extreme point solution. If $F_{\hat{m}} = 1$, then $x_{\hat{m}}$ is a constant. If $F_{\hat{m}} > 1$, then at most one inequality will be tight in Equation (6d) and Equation (6e) since either $x_{\hat{m}}$ is 1 or $\sum_{\tau_i \in \mathbf{T}} y_{i,\hat{m}} u_{i,\hat{m}} = x_{\hat{m}}$. There are n tight equations in Equation (6b) and at most $\hat{m} - 1$ tightness in Equation (6f) when $j \neq \hat{m}$. Therefore, at least $\hat{m}n - n - \hat{m}$ inequalities of Equation (6c) are tight. As a result, at most $n + \hat{m}$ variables are non-zero, and at most $\hat{m}$ tasks are with fractional variables $y_{i,j}$. □

Algorithm S-GREEDY can be revised as follows: For a task τ_i without fractional variables, we again simply assign this task to the PU type M_j with $y_{i,j}^{m^*} = 1$. At most m^* tasks are with fractional variables. We can simply revise Step 4 and Step 5 in Algorithm 2 by assigning each of tasks τ_{i^*}s with fractional variables to the corresponding PU type M_j ($j \leq m^*$) with the minimum dynamic energy consumption. It is not difficult to see that the approximation factor is $2m$. However, there might be a PU type M_j, in which the number of the allocated units of the type is $2F_j + m$, which might not be acceptable when F_j is small and m is large. As there is no ambiguity between the above revised algorithm and Algorithm S-GREEDY, the above revised algorithm is denoted by Algorithm S-GREEDY as well.

Another alternative is to try not to augment the allocated units so much by assigning tasks with fractional variables carefully. We first construct an undirected graph $G = (V, \mathcal{E})$ as follows: (1) for each task in $\mathbf{T}$ and each PU type M_j with $j \leq m^*$, we create a vertex in V. If the variable $y_{i,j}^{m^*}$ is with $0 < y_{i,j}^{m^*} < 1$, we create an edge in $\mathcal{E}$ to connect the vertex of task τ_i and the vertex of PU type M_j. By the extreme point theory and a similar proof in [24, §15], each connected component in G has at most ρ edges if there are ρ vertexes in the connected component. Therefore, each connected component in G is either a tree or a tree plus one edge, in which G is a *pseudo-forest*. Figure 1 illustrates an example for the pseudo-forest structure. If there is a vertex in G whose degree is 1, e.g., M_1, M_3, M_4, or M_7 in Figure 1, we can simply assign the task connected to this vertex onto the PU type represented by the vertex with degree 1, and remove both vertices of the task and the PU type from G. At the end, the left graph is a set of cycles, then, we can follow the cycle in clockwise or counterclockwise to find a perfect matching to assign the remaining tasks onto the PU types. Therefore, each task is assigned onto a PU type, and each PU type is assigned with at most one task with fractional variables. By applying the first-fit, the last-fit, the worst-fit, or the best-fit algorithm, the number of the allocated units of PU type M_j is at most $2F_j + 1$. The above algorithm for the R-MEHEPU problem is denoted by Algorithm S-GREEDY-GV. Similar to the extension in Algorithm E-GREEDY, we can also have an extended Algorithm E-GREEDY-GV by selecting the best achieved solution as the answer.

When all the tasks on a PU type have the same power consumption characteristics, i.e., $h_{i,j} = h_{k,j}$ for every τ_i, τ_k in $\mathbf{T}$, Algorithm S-GREEDY-GV is a $2m$-approximation algorithm with at most $F_j + 1$ augmentation. However, whether the approximation factor of Algorithm S-GREEDY-GV for different power characteristics on the same PU type is bounded by $2m$ is unknown. As the above approaches have bounded resource augmentation, one could artificially set the allocation constraint of PU type m_j as $\frac{F_j - 1}{2}$ for applying Algorithm S-GREEDY-GV or $\frac{F_j - m}{2}$ for applying Algorithm S-GREEDY. Then, the resulting solution is feasible for the original constraints.

7 Evaluation Results

This section provides evaluations for the proposed algorithms, including Algorithm S-GREEDY, Algorithm E-GREEDY, Algorithm S-GREEDY-GV, and Algorithm E-GREEDY-GV with the first-fit, the last-fit, the best-fit, and the worst-fit strategies. As different fitting algorithms applied for allocating PUs might lead to different results, we denote an algorithm by leading with how task assignment onto PU types is done and ending with its algorithm used for PU allocation, e.g., E-GREEDY-FIRST-FIT, E-GREEDY-LAST-FIT, E-GREEDY-BEST-FIT, and E-GREEDY-WORST-FIT.

Setups The performance evaluations are done by using synthetic real-time task sets. The period of task τ_i is a random variable in the range of $[1, 100]$ms. The execution time $c_{i,j}$ of jobs of task τ_i on PU type M_j is a random variable uniformly distributed in the range of $[0, \kappa] \times p_i$, and $h_{i,j}$ is a random variable in the range of $[0.5, 1.5]$. For each PU type M_j, the dynamic power consumption $P_{d,j}$ is a random variable uniformly distributed in the range of $[10, 1000]$ mWatt, and the static power consumption $P_{s,j}$ is a random variable uniformly distributed in the range of $[0, pr] \times P_{d,j} + 500$ mWatt, where the parameter pr is called *power ratio*. For a configuration, if the number m of PU types is specified, the

number of tasks in $\mathbf{T}$ is an integral random variable in the range of $[5, \chi \times m + 5]$.

Various experiments have been done, but, due to space limitation, we only present representative results for three configurations. For the first configuration, we evaluate the performance of different fitting algorithms for different power ratios. For the second configuration, by fixing the power ratios as 2, we evaluate the performance by varying the number of PU types, the parameter χ to change the numbers of tasks, and the parameter κ to change the execution time distributions. For the third configuration, we evaluate the restriction of the number of the allocated PUs by setting the *restriction factor* ϕ, in which F_j is an integral random variable in the range of $[1, \phi]$.

The *normalized energy* is adopted as the performance index in the experiments. The normalized energy consumption of an algorithm for an input instance is the ratio of the energy consumption of the solution derived from the algorithm to that of the lower bound derived by solving the m linear programming equations in Equation (4) or Equation (6). Clearly, an algorithm with less normalized energy consumption has better performance. For the R-MEHEPU problem, we evaluate how much resource augmentation the algorithms have. For an input instance, the *resource augmentation number* of an algorithm is defined as the maximum excess allocated processing units among the PU types, i.e., $\max_{m_j \in \mathbf{M}}\{\max\{0, f_j - F_j\}\}$, where f_j is the actual allocated units of PU type M_j. Similarly, the *resource augmentation rate* of an algorithm is defined as $\max_{m_j \in \mathbf{M}}\{\max\{0, \frac{f_j - F_j}{F_j}\}\}$. Each point of the configurations is an average of 512 independent runs.

Results

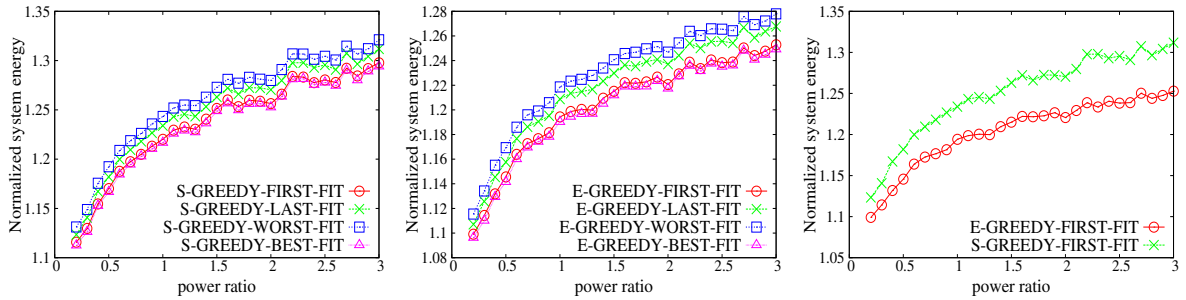
Figure 2 presents the results of Algorithm S-GREEDY and Algorithm E-GREEDY, for the MEHEPU problem, of different fitting algorithms for the first configuration by varying the power ratios from 0.1 to 3 under the settings of χ as 15, κ as 1, and m as an integral random variable in the range of $[2, 12]$. For clarity, Figure 2(a) (resp. Figure 2(b)) shows the results for different fitting algorithms of Algorithm S-GREEDY (resp. E-GREEDY). In general, the higher the power ratio, the more the normalized energy consumption of all the evaluated algorithms is. This comes from the under-estimation of Equation (6). As the lower bound becomes more under-estimated, the normalized energy consumption increases. Even though the normalized energy consumption increases with higher power ratios, the growing tendency is quite slow. Among the fitting algorithms, the first-fit algorithm, in most cases, is the best among the other evaluated algorithms, while the improvement is more when Algorithm E-GREEDY is applied with a higher power ratio. The performance of the best-fit algorithm is quite similar to that of the first-fit algorithm. Figure 2(c) summarizes the normalized energy consumption of Algorithm S-GREEDY and Algorithm E-GREEDY with the first-fit algorithm. The improvement of Algorithm E-GREEDY becomes more when the power ratio increases. Moreover, with the same fitting algorithm, Algorithm E-GREEDY outperforms Algorithm S-GREEDY in all the cases.

Figure 3 presents the experimental results of Algorithm S-GREEDY and Algorithm E-GREEDY of different variations for the second configuration. As applying the first-fit algorithm outperforms the other fitting algorithms in most cases, for clarity, we only present the results of Algorithm S-GREEDY-FIRST-FIT and Algorithm E-GREEDY-FIRST-FIT in Figure 3. Figure 3(a) illustrates the results by varying m for $\kappa = 1$, $\chi = 15$, and $pr = 2$. Figure 3(b) is the result by varying χ for $\kappa = 1$, $pr = 2$, and a random variable m in the range of $[2, 12]$. Figure 3(c) shows the results by varying κ for $\chi = 15$, $pr = 2$, and a random variable m in the range of $[2, 12]$. Algorithm S-GREEDY and Algorithm E-GREEDY become worse for (1) a larger number of PU types in Figure 3(a) due to the more under-estimation of the energy consumption by using the lower bound when m is large, (2) less number of tasks in Figure 3(b) due to the more allocated units with low utilization, or (3) less utilization in Figure 3(c) due to the more low-utilized units. Moreover, the performance gap between Algorithm S-GREEDY and Algorithm E-GREEDY becomes wider for worse performance of Algorithm S-GREEDY. By the results in Figures 2 and 3, applying Algorithm E-GREEDY could improve the normalized energy consumption very much, and is not too far from the optimal solutions.

Figure 4 shows the results for the R-MEHEPU problem by setting the restriction factor ϕ from 4 to 16 for $\chi = 15$, $\kappa = 1$, $pr = 2$, and a random variable m in the range of $[2, 12]$. As shown in Figure 4(a), with looser resource constraints, the performance of Algorithm E-GREEDY and Algorithm E-GREEDY-GV becomes similar, since the solution of Equation (6) becomes the same when the restriction factor ϕ is large enough. Moreover, as shown in Figure 4(b) and Figure 4(c), the resource augmentation numbers and the resource augmentation rates become lower for the higher restriction factor ϕ. In general, Algorithm E-GREEDY-GV outperforms Algorithm E-GREEDY in resource augmentation, and vice versa in energy consumption.

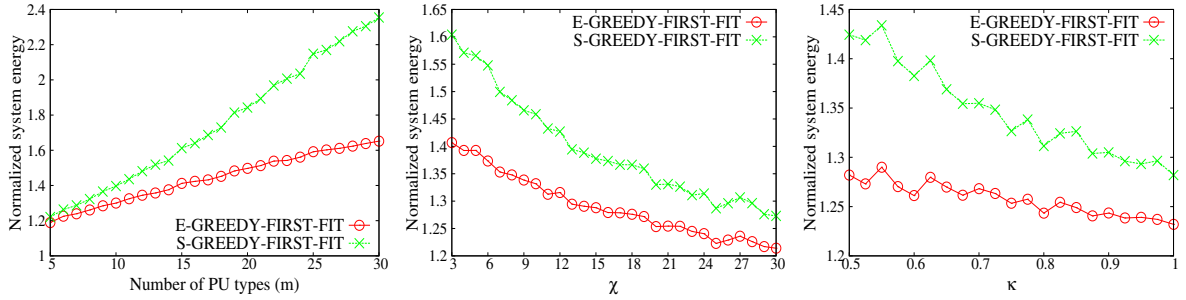
8 Conclusion

This work explores energy-efficient task partitioning and processing unit allocation for periodic real-time tasks on a given library of processing unit types. By considering both static (leakage) power consumption and dynamic power consumption, we propose polynomial-time energy-efficient scheduling algorithms based on the relaxation of the integer linear programming. The algorithms first find a good mapping of tasks onto processing unit types, and then apply bin-packing algorithms to allocate processing units. By analyzing the performance of the proposed algo-



(a) Algorithm S-GREEDY (b) Algorithm E-GREEDY (c) First-fit for Algorithm E-GREEDY and Algorithm S-GREEDY

Figure 2. The results by applying different fitting algorithms.



(a) varying number of PU types (b) varying number of tasks (c) varying utilization

Figure 3. The results by applying Algorithm S-GREEDY and Algorithm E-GREEDY for different settings.

rithms, if there is no limitation on the number of allocated processing units, we show that Algorithm S-GREEDY and Algorithm E-GREEDY are with a $(m+1)$-approximation ratio, in which m is the number of applicable processing unit types. Algorithm S-GREEDY-GV (resp. Algorithm E-GREEDY-GV) extended from Algorithm S-GREEDY (resp. Algorithm E-GREEDY) is shown to have bounded resource augmentation with good performance. Experimental results show that the proposed algorithms are quite effective in average cases. Due to space limitation and for clarity, we only present the results for non-DVS systems. It is easy to extend the results here to cope with DVS systems with discrete levels of supply voltages by revising the linear programming formulation in Equation (4) or Equation (6). Moreover, for energy-aware synthesis problems under area/cost constraints, e.g., in [18], the approaches in Section 5 and Section 6 can be extended. For future research, we would like to explore analytical algorithms for heterogeneous multiprocessor systems that can be shut down dynamically.

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] A. Andrei, P. Eles, Z. Peng, M. T. Schmitz, and B. M. Al-Hashimi. Energy optimization of multiprocessor systems on chip by voltage selection. *IEEE Trans. VLSI Syst.*, 15(3):262–275, 2007.

[3] 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.

[4] 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.

[5] J.-J. Chen and C.-F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *RTCSA*, pages 28–38, 2007.

[6] J.-J. Chen and T.-W. Kuo. Allocation cost minimization for periodic hard real-time tasks in energy-constrained DVS systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 255–260, 2006.

[7] J.-J. Chen and L. Thiele. Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements. In *International Conference on Parallel and Distributed Systems (ICPADS)*, pages 161–168, 2008.

[8] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer Verlag, 1997.

[9] P. J. de Langen and B. H. H. Juurlink. Leakage-aware multiprocessor scheduling for low power. In *IPDPS*, 2006.

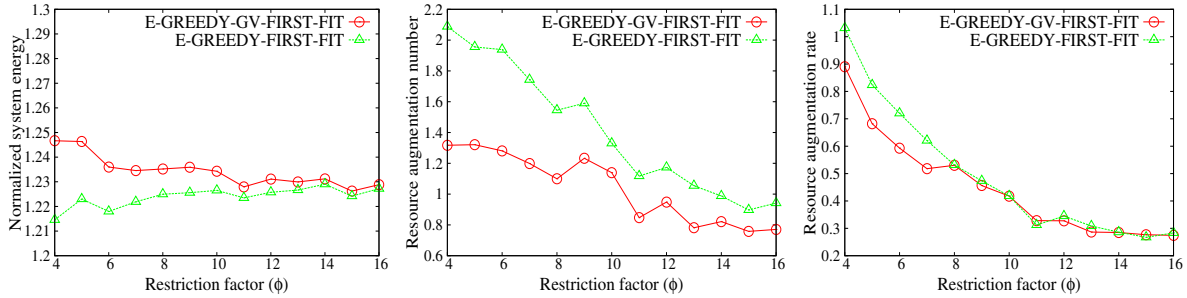
[10] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Co., 1979.

[11] GNU Linear Programming Kit. http://www.gnu.org/software/glpk/glpk.html.

[12] E. Horowitz, S. Sahni, and S. Rajasckaran. *Computer Algorithms: C++*. W. H. Freeman & Co., New York, NY, USA, 1996.

[13] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo. Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint. In *ACM/IEEE Conference of Design, Automation, and Test in Europe (DATE)*, pages 1061–1066, 2006.

[14] T.-Y. Huang, Y.-C. Tsai, and E. T.-H. Chu. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup



(a) energy consumption (b) resource augmentation number (c) resource augmentation rate

Figure 4. The results by applying Algorithm E-GREEDY and Algorithm E-GREEDY-GV.

problem. In *21th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.

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

[16] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[17] 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.

[18] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian. Energy-aware cosynthesis of real-time multimedia applications on mpsocs using heterogeneous scheduling policies. *Trans. on Embedded Computing Sys.*, 7(2):1–19, 2008.

[19] 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.

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

[21] 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 (IPDPS)*, page 21, 2003.

[22] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2002.

[23] A. Schranzhofer, J.-J. Chen, and L. Thiele. Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

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

[25] 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)*, pages 1–10, 2005.

[26] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In *Conference of Design, Automation, and Test in Europe (DATE)*, 2009.

[27] Y. Yu and V. K. Prasnna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In *Proceedings of the Ninth International Conference on Parallel and Distributed Systems(ICPADS'02)*. IEEE, 2002.

[28] 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.

[29] 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.

Appendix

Proof of Theorem 3. We prove this theorem by providing a set of input instances that satisfy the statement. Suppose that all the tasks have the same period p. For task τ_i in $\mathbf{T}$, the execution time at the first PU type M_1 is $c_{i,1} = \frac{p\epsilon}{n}$, while the execution time at any other PU type is p, where ϵ is a positive number less than 1. The static power consumption and the dynamic power consumption of PU type M_1 are both σ. The static power consumption and the dynamic power consumption of the other PU types are equal to $\frac{\sigma}{n}$. The optimal solution for the above set of input instances in Equation (2) is to allocate one unit of PU type M_1 to assign all the tasks, which leads to a solution with $\sigma p(1+\epsilon)$ in the objective function. An optimal solution for the above set of input instances in Equation (3) also assigns all the tasks on PU type M_1, which leads to a solution with $2p\epsilon\sigma$ in the objective function. Clearly, when $\epsilon \to 0$, for the above set of input instances, the ratio $\frac{p\sigma(1+\epsilon)}{2p\epsilon\sigma}$ of the optimal solution of Equation (2) to that of Equation (3) is ∞. □

Proof of tightness of the $(m+1)$-approximation factor. We now show that the analysis of the $(m+1)$-approximation factor is almost tight by providing a set of input instances with a gap close to m between the solutions of Algorithm S-GREEDY (or Algorithm E-GREEDY) and the optimal solutions. Suppose that the power consumptions of the m PU types are as follows: $P_{s,1} = \epsilon$, $P_{s,2} = P_{s,3} = \cdots = P_{s,m-1} = \kappa - \epsilon$, $P_{s,m} = \kappa$, $P_{d,1} = \eta$, $P_{d,2} = \cdots = P_{d,m} = \frac{\kappa\eta}{\epsilon}$, where $0 < \epsilon < \kappa \neq \infty$ and $0 < \eta$. There are m tasks, $\tau_1, \tau_2, \ldots, \tau_m$ with the same period 1. For τ_i in $\{\tau_2, \tau_3, \ldots, \tau_m\}$, let $c_{i,j}$ be 1 for $j \neq i$, $c_{i,i}$ be $\frac{\epsilon}{\kappa} - \delta$, and $h_{i,j}$ be 1 for $1 \leq j \leq m$, where $\delta > 0$. For task τ_1, let $c_{1,m}$ be 1 and $c_{1,j}$ be ∞ for any $j < m$, where $h_{1,m}$ is set as $\frac{\epsilon}{\kappa}$. For the above set of input instances, applying Algorithm S-GREEDY leads to a solution with energy consumption $2\kappa + (m-2)(\kappa - \epsilon) + m\eta - (m-1)\frac{\kappa\eta\delta}{\epsilon}$ by assigning tasks τ_i in task set $\{\tau_2, \tau_3, \ldots, \tau_m\}$ to PU type M_i and task τ_1 to PU type M_m. The energy consumption by assigning all the tasks in task set $\{\tau_2, \tau_3, \ldots, \tau_m\}$ to PU type M_1 and task τ_1 to PU type M_m is $(m-1)(\epsilon+\eta)+\kappa+\eta$. Therefore, when δ, ϵ, and η are small, the solution derived by Algorithm S-GREEDY is with a gap m to an optimal one. □