Electronic Preprint for Journal of Information Processing Vol.27

Regular Paper

Energy-Aware Task Allocation for Heterogeneous Multiprocessor Systems by Using Integer Linear Programming

Yang Qin[1,a)] Gang Zeng[2] Ryo Kurachi[1] Yutaka Matsubara[1] Hiroaki Takada[1]

Received: May 24, 2018, Accepted: November 7, 2018

Abstract: Energy-aware task allocation of embedded systems is one of the most important issues in recent decades. A classical solution to solve the issue is Integer Linear Programming (ILP). However, given the considerable time consumption, it is effective only to the extent that the scale of the problem is small. How to use ILP to solve large allocation problems on heterogeneous multiprocessor systems to minimize energy consumption is still a challenge. This paper proposes two ILP formulations to deal with it. One complete ILP(1) is used to derive a feasible allocation, and the other simplified ILP(2) is for calculating the desired minimum energy. Then the desired minimum energy can be used as a reference to evaluate the intermediate solution of ILP(1) and decide its timeout. Besides, to find out the best-suited platform for a given workload, a flexible design which presents flexibilities and choices in core assignment, is considered for further energy saving. For example, the optimal core number design and core type design are generated as two independent ILP formulations, denoted as ILP(3) and ILP(4). The experimental results on randomly generated task sets demonstrate that, compared with the fixed platform, automatically synthesizing a flexible core assignment saves more energy.

Keywords: energy-aware task allocation, heterogeneous multiprocessor systems, ILP formulations, flexible platform design

1. Introduction

With the increasing demands for computing and communication, power management and energy reduction are becoming more and more important in embedded systems. Multiprocessor system-on-chip (MPSoC) which uses multiple processors, has been widely used for embedded applications. It can be classified into two categories: homogeneous system and heterogeneous system. Modern real-time systems generally consist of heterogeneous multiprocessors, as the designers can take advantage of the processing properties for particular applications. For example, high-performance processors for heavy workloads, and power-efficiency processors for light workloads on ARM big.LITTLE architecture [1]. Compared with homogeneous systems, heterogeneous processing units are more flexible to trade off performance and energy consumption. Furthermore, single-instruction set architecture (ISA) heterogeneous multiprocessor systems have attracted much attention as a solution for energy minimization while improving performance. In this architecture, all the processors share the same instruction set but differ with power and performance behaviors [2]. Therefore, we consider a real-time embedded system that consists of heterogeneous multiprocessors with single-ISA architecture in this paper.

[1] Graduate School of Informatics, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] Graduate School of Engineering, Nagoya University, Nagoya, Aichi 464–8603, Japan
[a)] claire@ertl.jp

A critical issue in real-time embedded systems, especially battery-driven systems, is energy-efficient scheduling. The most widely used technique for reducing energy consumption during processing is Dynamic Voltage Frequency Scaling (DVFS), which is based on the convex relation between voltage/frequency and power consumption [3], [4]. The DVFS approach reduces the energy by decreasing the operational frequency and voltage at its maximum extent with the time constraints guaranteed. Even though some processors are capable of an almost continuous voltage/frequency scaling, most of them are supporting DVFS with discrete voltage/frequency scaling in practical applications. In this work, we assume that each processor core is equipped with discrete DVFS technology on a heterogeneous multiprocessor.

Without loss of generality, the scheduling method can be classified as partitioned scheduling and global scheduling [5]. Given the high overhead of task migration and cache coherence in global scheduling, we consider partitioned scheduling in this paper. The appropriate energy-aware scheduling should not only minimize the overall energy consumption, but also meet functional and timing requirements. This work aims at exploring the most energy-efficient allocation of a set of independent periodic tasks on a real-time heterogeneous system with all the deadlines guaranteed. After the assignment, tasks can be scheduled to execute by dynamic scheduling policy Earliest Deadline First (EDF), which has been proven to be the optimal scheduling policy for independent real-time tasks with dynamic priorities [6].

In Ref. [7], Baruah et al. proposed that the preemptive schedul-

ing of independent, periodic real-time tasks on a heterogeneous multiprocessor is an NP-hard problem in the strong sense. The objective of minimizing energy consumption further complicates the problem with no doubt [8]. A classical solution to solve the scheduling problem is integer linear programming. However, several problems still remain unresolved. Firstly, existing ILP formulations for solving task allocation problems are all based on zero idle power assumption. In practice, idle power consumption is non-negligible for increasing static energy. Meisner et al. referred that one of the major causes contributing to energy inefficiency is idle power waste [9]. Failure to take idle power consumption into consideration leads to a large difference between predicted and actual measured power consumption [10]. This raises the need for modeling a new linear programming formulation that takes the idle power consumption into consideration. Secondly, previous ILP optimizations are usually regarded as ineffective in solving large-scale allocation problems for the challenge of time consumption. Even though the intermediate solution can be calculated, for example, the solution derived within a limited time, the optimality of the solution is unknown, and it cannot be used with confidence instead of the optimal one. Thirdly, the energy-efficient scheduling problem is mostly discussed on a preset platform. In fact, the problem has been researched a lot for decades. How to incorporate the platform design into the task allocation, e.g., the core number and core type, to save energy is still a new issue in this field.

To the best of our knowledge, this is the first work that considers all the issues in energy-aware scheduling. In this paper, we study the problem under two different target platforms: a fixed platform, and a flexible platform. The contributions of the work are summarized as follows:

(1) We formulate the energy-aware task allocation problem as ILP formulation with consideration of idle power consumption.

(2) For a fixed platform, we propose two ILP formulations to deal with large-scale allocation problems. One is used to derive an intermediate solution, the other is used to evaluate the solution. After that, a complete algorithm (Fast Terminate) is proposed to combine the two ILP formulations to achieve a reliable solution within a specified error.

(3) For a flexible platform, we propose ILP formulations to solve the optimal core assignment, where the best decisions of core number and core type are investigated as two independent cases. Simulation results show that compared with the fixed platform, more energy savings can be achieved through a flexible design.

The rest of the paper is organized as follows: Section 2 reviews some related work. Section 3 discusses the system models used in this work. Section 4 generates mathematical formulations on a fixed platform. Section 5 investigates the mathematical formulations on a flexible platform. Section 6 verifies the proposed ILP formulations by experiments. Section 7 makes a conclusion of our work.

2. Related Work

Energy-aware scheduling of real-time embedded systems has been researched for decades. In this section, we give a brief introduction of related work mainly focusing on heterogeneous processing systems. The work is classified into two categories. Firstly, we summarize the research of real-time scheduling on a fixed platform. Then, the recently proposed approaches based on a flexible design are surveyed.

2.1 Real-time Scheduling on a Fixed Platform

For a given task set and platform, the problem of energy-aware scheduling can be addressed as assigning tasks to right cores and setting appropriate operating frequencies. Yu et al. first modeled the problem as an integer linear programming formulation [11]. In their proposed ILP formulation, both the allocation strategies and frequency settings could be attained for each task. By relaxing the model, an LP-heuristic was proposed to solve large allocation problems. Baruah et al. also discussed the real-time partitioning problem among heterogeneous multiprocessors [12]. In their work, the ILP formulation was generated to achieve the minimum makespan (the duration of schedule) but not energy. It was proven in their experiment that the model could be relaxed by appropriate means to achieve a reliable solution in polynomial time. Recently, Zhang et al. also followed the model to consider energy-aware scheduling for heterogeneous platforms [13]. However, according to our best knowledge, the solution only considered the task allocation without frequency setting, which might cause energy waste for DVFS-capable platforms.

There also existed a number of approximate and heuristic algorithms to solve the equivalent problem. Chen et al. introduced a polynomial-time approximation algorithm to partition tasks on the platform consisting of two processing elements [14]. Colin et al. proposed a heuristic by approximating the desired load distribution for heterogeneous systems [15]. Recently, Pagani et al. in Ref. [16] first defined an energy factor to illustrate the energy relation of asymmetric cores and proposed a fixed-frequency-configuration method to get the local optimal allocation. Besides, some intelligence algorithms, such as Ant Colony Optimization (ACO), Genetic Algorithm (GA), Particle Swarm Optimization (PSO) were also introduced to solve the energy-aware task allocation on heterogeneous systems [17], [18], [19]. Given the unacceptable time consumption, existing ILP formulations are difficult to apply to large allocation problems. In this work, we propose two ILP formulations to solve large allocation problems within a reasonable computation time. The basic idea of the method was introduced in Ref. [20]. However, paper [20] did not consider the allocation problem on a flexible platform.

2.2 Real-time Scheduling on a Flexible Platform

With a specific workload, the configuration of the platform may have an effect on the optimal task allocation. However, the research of synthesizing the core assignment to save energy is considerably less in real-time scheduling. In 2009, Chen et al. summarized the research based on non-DVFS capable heterogeneous platforms [21]. The problem was cataloged into two cases: MEHEPU and R-MEHEPU by considering if there was limitations on the number of processing cores. Then approximate algorithms were proposed to obtain approximate solutions for each case. Besides, with limited area constraints, Chen et al. first pro-

Table 1 Notations and definitions used in this work.

Notation	Definition
Π	Target platform: $\Pi = \{PU_1, PU_2, ..., PU_M\}$
Num_K	Number of processor cores in PU_K
NUM	Number of processor cores in platform Π
$Type\_num$	Maximum allowed core types to build a platform
$WCET_{i,k}$	Worst-Case-Execution-Time of task τ_i on core k
T_i	Period time of task τ_i
L	Hyper-period time, the minimum repeating interval of all tasks
$u_{i,k}$	Utilization of task τ_i at the maximum frequency on core k
U_k	Sum of utilization at the maximum frequency on core k
$P_{idle,k}$	Idle power of core k
$E(U_k, f_k)$	Energy consumption with utilization U_k at frequency f_k
$u'_{i,j}$	Relative utilization of task τ_i at frequency j on core k, $u'_{i,j} = u_{i,k} f_{k,MAX}/f_j$
$U_{j,k}$	Sum of relative utilization at frequency level j on core k
F_k	Number of frequency levels provided by core k
m	Number of frequency levels provided by system, $m = \sum_{k=1}^{NUM} F_k$
n	Number of tasks for a given task set
$FLG(k)$	A set of frequency levels provided by core k
F'_K	Number of frequency levels provided by one single core in PU_K
$\hat{m}$	Number of frequency levels provided by M different cores, $\hat{m} = \sum_{K=1}^{M} F'_K$
$x_{i,j}$	Decision variable, which represents allocation result of task τ_i at frequency level j

posed a heuristic to find the proper processor allocation together with the task mapping, such that the target workload's execution time was optimized [22]. In contrast to previous work, we assume flexibilities in both core number and core type in this work. Moreover, the study of energy-aware scheduling considering platform design was related to executive components, e.g., memory behaviors [23]. However, the problem would become quite complex when considering the core assignment and memory design at the same time. In this work, when we refer to platform design, we mean the best decisions of core assignment.

3. System Models

In this section, we present a multi-core platform model that consists of heterogeneous processing units, a task model that denotes real-time applications, as well as an energy model for calculating energy consumption. Then the problem of energy-efficient task allocation is discussed. The used notations and their definitions are given in **Table 1**.

A Fixed Platform Model: We consider a multi-core heterogeneous system Π that consists of M types of processing units, $\Pi = \{PU_1, PU_2, ..., PU_M\}$, as Ref. [21]. Each type of PU differs by their power and performance characteristics. Suppose that type PU_K consists of Num_K identical cores, the total core number of system Π can be calculated by $NUM = \sum_{K=1}^{M} Num_K$.

A Flexible Platform Model: If the core assignment, i.e., core number and core type, can be adjusted for designers to build their own platform, we denote the platform as a flexible platform. For flexible core number design, assume that the number of cores in each PU is unknown, while the sum is restricted to NUM. Additionally, for flexible core type design, suppose that only $Type\_num$ from M different types of cores are selectable to build a platform. Also, the sum of processor cores is limited to NUM. Note that all the restrictions are generated for practical consideration.

Task Model: We model a real-time task set Γ that consists of n periodic real-time tasks: $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task is modeled from a real-time application and there is no data dependency among tasks, i.e., we consider independent tasks. Task τ_i is characterized by a 3-tuple $\{WCET_{i,k}, T_i, D_i\}$, where $WCET_{i,k}$ denotes the Worst-Case-Execution-Time of task τ_i executing on processor core k at the maximum frequency, T_i denotes the period time, and D_i denotes the relative deadline, which is assumed to be equal to T_i, i.e., $D_i = T_i$. The utilization of task τ_i executing at the maximum frequency of core k is defined as: $u_{i,k} = WCET_{i,k}/T_i$, and the sum of utilization is: $U_k = \sum_{\tau_i \in \Gamma_k} u_{i,k}$, where Γ_k is a subset of tasks assigned to core k.

Power and Energy Model: Generally, each processor core is supposed to have three power modes, i.e., run, idle and inactive. Among the three modes, assume that the tasks can only be executed in the run mode. In other words, if a core has no task to execute before the next task releases, OS puts the core in the idle mode, which consumes less energy than run mode [24]. In this work, the idle power consumption is considered as a constant for a given core k, denoted as $P_{idle,k}$. Both run mode and idle mode are regarded as the active mode because static power is consumed. Contrarily, if no task is assigned to a core, the core is in the inactive mode and consumes zero power. The power consumption of processor core k in the run mode is given by (1):

$$P_k(f) = \alpha_k f^{\beta_k} + s_k \tag{1}$$

where the first term is frequency-dependent power consumption and the second term is frequency-independent power consumption. All α, β and s are technology-based parameters, which have the same meanings with k, α, β in Ref. [15] respectively.

Note that the tasks may have different periods, thus the energy consumption should be calculated based on the minimum repeating interval, the hyper-period time as follows: $L = LCM(\{T_i : \tau_i \in \Gamma\})$, where LCM is lowest common multiple. The energy consumption of core k during one interval L can be expressed as follows [15], [16]:

$$E(U_k, f_k) = L\left(\frac{U_k f_{k,MAX}}{f_k} P_k(f_k) + \left(1 - \frac{U_k f_{k,MAX}}{f_k}\right) P_{idle,k}\right) \tag{2}$$

where f_k is operating frequency and $f_{k,MAX}$ is the maximum allowed frequency. The operating frequency of core k should be set as: $f_k \geq U_k * f_{k,MAX}$, so that the total utilization after frequency scaling will not be larger than 1 and the tasks can be scheduled by EDF without missing the deadline.

Per-Core DVFS Capability: DVFS scaling is one of the most widely used technologies for energy saving. A general DVFS technique is VFI (voltage/frequency island), which supports different voltage supplies and frequencies for different clusters while the same voltage and frequency setting in one cluster [25]. In other words, the cores in one cluster have to share identical voltage/frequency configurations, denoted as Per-Cluster DVFS. Another DVFS technology is Per-Core DVFS, where each processor

core operates at an individual voltage/frequency level independently [26]. For example, according to the statement of ARM's DynamIQ technology, Per-Core DVFS technology is capable in the DynamIQ big.LITTLE architecture [27]. Compared with Per-Cluster DVFS, Per-Core DVFS allows more flexibilities in controlling power and performance. In this work, we assume that each processor core k is equipped with Per-Core DVFS capability. Moreover, given an almost linear relation between voltage and frequency, DVFS usually scales voltage and frequency simultaneously. Therefore, we use the scaling of discrete frequencies to refer to the changes of voltage and frequency, as Refs. [15], [16], [28].

Static DVFS Scaling: In terms of the scaling algorithm, DVFS can be classified as static and dynamic schemes. The static DVFS is applied to determine the voltage/frequency settings offline, while the dynamic technology controls the configurations online by detecting workload changes. Since the dynamic DVFS may impair the real-time performance, we consider the static scheme in this work. Generally, DVFS scaling brings processor core unavailable time from 10 μs to 650 μs and the overhead is too large to be ignored [29]. Besides, even for the static DVFS, adjusting frequency for each task may cause unanticipated overhead. As a result, we use single-frequency scheme with respect to DVFS, which chooses a single frequency for every execution core throughout the hyper-period. More precisely, the lowest execution frequency will be set to achieve the minimum energy consumption with all the time constraints guaranteed. Even though the single-frequency scheme is not the optimal strategy for energy minimization, it significantly reduces the overhead of DVFS scheduling.

Problem Definition: In this paper, we aim at finding the most energy-efficient task allocation on heterogeneous multiprocessor systems. If the target platform is known in advance, we abbreviate it as Energy-Aware Scheduling on Heterogeneous Multi-core (EASHM). The objective of the work is to partition an input task set Γ onto right cores and set appropriate frequencies with deadlines guaranteed, such that the energy consumption during one hyper-period L is minimized. Otherwise, if the platform is unknown, i.e., core type and core number can be adjusted, we denote it as Flexible Design and Energy-Aware Scheduling on Heterogeneous Multi-Core (FD-EASHM), and the main work includes platform design and task allocation. Briefly, for a given task set, we first compute the best-suited platform, then the problem is equivalent to EASHM, and the task allocation can be decided in the same way.

4. ILP Formulations on a Fixed Platform

In this section, we explore the energy-aware task allocation on a fixed platform. Firstly, the EASHM problem is formulated as ILP(1). Considering the huge time consumption, a relaxed ILP(2) is proposed to calculate the desired minimum energy. After that, a combination of two ILP formulations is generalized to find a reliable allocation within a specified error.

4.1 ILP(1): Allocation Problem of EASHM

In the first section, we begin our ILP formulation under an idealized assumption that the core consumes zero power in the idle mode. The allocation problem can be generated as mapping tasks onto processing cores, along with setting execution frequencies with reference to previous work [11]. Suppose that the number of discrete frequencies provided by processor core k is F_k, the frequency levels in the system consisting of NUM cores can be calculated as: $m = \sum_{k=1}^{NUM} F_k$. For example, the m'-th frequency level of core k' is labeled as $j = \sum_{k=1}^{k'-1} F_k + m'$. Note that j is not only an index of frequency setting but also a decision of core allocation. As long as j is decided for a task, both the allocated core and operational frequency can be derived.

An index of $e_{i,j}$ is made in advance to represent the energy consumption for executing task τ_i at frequency level j in the run mode during a hyper-period L. For example, let $FLG(k)$ denote a set of frequency levels provided by core k, the relative utilization of task τ_i executing at frequency j is calculated as: $u'_{i,j} = u_{i,k} f_{k,MAX}/f_j$, where j is one of the frequency configurations from core k, denoted as $j \in FLG(k)$. Then we compute $e_{i,j}$ as: $e_{i,j} = Lu'_{i,j}P_k(f_j)$. For each task τ_i in $\{\tau_1, \tau_2, ..., \tau_n\}$, a binary variable $x_{i,j}$ is set as 1 if task τ_i is assigned to frequency level j; otherwise, $x_{i,j}$ is set as 0. Therefore, the objective function without considering the idle power consumption can be formulated as follows:

$$Min \;\; E = \sum_{j=1}^{m} \sum_{i=1}^{n} e_{i,j} x_{i,j} \tag{3}$$

where n is the number of tasks, and m is the number of frequency levels. However, Eq.(3) is based on zero idle power assumption. As explained in Section 1, the idle power consumption is non-negligible with the increasing static energy. According to the energy model presented in Eq. (2), the objective function that takes the idle power into consideration can be formulated as:

$$Min \;\; E = \sum_{j=1}^{m} \sum_{i=1}^{n} e_{i,j} x_{i,j} + L \sum_{k=1}^{NUM} \left(1 - \sum_{j \in FLG(k)} U_{j,k}\right) P_{idle,k} \tag{4}$$

where the first component is the energy consumed in the run mode, and the second is that consumed in the idle mode. The section of $\sum_{j \in FLG(k)} U_{j,k}$ denotes the sum of utilization on core k. As assumed in Section 3, if core k is not chosen to assign tasks, it is set in the inactive mode and consumes zero power. However, the energy consumption of core k calculated by function (4) is: $E_k = LP_{idle,k} \neq 0$, which violates the assumption of our inactive power mode. To deal with the excessive calculated energy consumption, an intuitive method is to define a function $y(k)$ to reflect the core's running states. When core k has no tasks assigned to, it is set in the inactive mode and $y(k)$ is defined as 0; otherwise, core k is set in the active mode and $y(k)$ is 1, as Eq.(5).

$$y(k) = \begin{cases} 0, & \text{if } \sum_{j \in FLG(k)} U_{j,k} = 0 \\ 1, & \text{if } \sum_{j \in FLG(k)} U_{j,k} \neq 0 \end{cases} \tag{5}$$

By observing formulas (4) and (5), it is not difficult to see that the section of $\sum_{j \in FLG(k)} U_{j,k}$ exists in both equations, and it is related to the allocation decisions. In other words, it is impossible to multiply the objective function Eq. (4) by Eq. (5) directly, as

the linear model will be transformed to non-linear thus the solution will be further complicated. In this work, we propose to modify the objective function as Eq. (6) and reformulate function (5) in terms of constraints (9), (10). Together with constraints (7), (8) and (11), the task allocation problem can be formulated as ILP(1) as follows:

$$Min\ \ E = \sum_{j=1}^{m}\sum_{i=1}^{n} e_{i,j}x_{i,j} + L\sum_{k=1}^{NUM}\left(y(k) - \sum_{j\in FLG(k)} U_{j,k}\right)P_{idle,k} \tag{6}$$

Subject to:

$$\sum_{j\in FLG(k)} U_{j,k} \le 1\ \ (k = 1, 2, ..., NUM) \tag{7}$$

where $U_{j,k} = \sum_{i=1}^{n} u'_{i,j}x_{i,j}$.

$$\sum_{j=1}^{m} x_{i,j} = 1\ \ (i = 1, 2, ..., n) \tag{8}$$

where $x_{i,j} \in \{0, 1\}$.

$$y(k) - \sum_{j\in FLG(k)} U_{j,k} < 1\ \ (k = 1, 2, ..., NUM) \tag{9}$$

$$y(k) - \sum_{j\in FLG(k)} U_{j,k} \ge 0\ \ (k = 1, 2, ..., NUM) \tag{10}$$

where $y(k) \in \{0, 1\}$.

$$x_{i,j} + U_b \le 1\ (i = 1, 2, ..., n; k = 1, 2, ..., NUM) \tag{11}$$

where $j, b \in FLG(k)$ and $b \ne j$.

Note that in ILP(1), m, n, L, NUM, $P_{idle,k}$ are preset parameters for a given platform and task set, while $x_{i,j}$ is the decision variable that reflects the final allocation result. In formula (7), $U_{j,k}$ denotes the sum of utilization at frequency level j. This constraint guarantees that the utilization of implicit deadline tasks executed on a core is less than 100% to be scheduled by EDF. Constraint (8) ensures that for each task i from $\{\tau_1, \tau_2, ..., \tau_n\}$, it is executed on one frequency level j without migration or decomposition, where a frequency level corresponds to a decision of core allocation and frequency setting. Constraints (9), (10) are linear expressions of function (5) for calculating right idle power consumption under the assumption of three power modes. In addition, constraint (11) is for the purpose of single-frequency scheme. It ensures that the frequency selection from a processor core is less than or at least equal to 1. For example, if the decision variable $x_{i,j}$ is set as 1, which represents that task i is allocated to frequency level j, constraint (11) guarantees the utilization assigned to any other frequency level U_b is 0, where $b \ne j$ and $j, b \in FLG(k)$. Since b and j are from the same core k, $U_b = 0$ ensures that no task is assigned to other frequency levels except for j.

Taken together, as long as ILP(1) has a feasible solution, all tasks in $\{\tau_1, \tau_2, ..., \tau_n\}$ are assigned to right cores with appropriate frequency settings accordingly. A proper task allocation of EASHM problem can be calculated with guaranteed deadlines for all real-time tasks. Additionally, if the problem of EASHM has a feasible schedule, it is obvious that all the constraints are satisfied and ILP(1) finds a feasible solution. Therefore, the EASHM problem is equivalent to the linear programming formulation ILP(1).

4.2 ILP(2): Desired Minimum Energy Consumption

By observing the linear programming formulation ILP(1), it not only calculates the minimum energy consumption, but also derives the optimal allocation strategy. However, ILP(1) may suffer from an exponential increase of execution time with the growing size of allocation problems, e.g., longer than 24 hours. For this reason, we propose a relaxed model to calculate the desired minimum energy consumption within a reasonable computation time. The desired minimum energy consumption then can be used as a reference to evaluate the optimality of the intermediate solution of ILP(1).

Assume that each task i has a decision of core type instead of specific core. For example, we choose to assign all the tasks to particular processing units from the target platform $\{PU_1, PU_2, ..., PU_M\}$. Let F'_K represent the frequency levels configured by one single core in PU_K, then the frequency levels of M different single cores can be computed as $\hat{m} = \sum_{K=1}^{M} F'_K$. Therefore, the scale of the decision variables can be reduced greatly, from m to $\hat{m}$ to be more specific. For the reminder, m denotes the sum of frequency levels provided by NUM cores.

For example, a platform has two types of processing units PU_1 and PU_2, and each PU consists of four identical cores. Each core in PU_1 has 9 independent frequency configurations, while each core in PU_2 has 13. In that case, the selections of frequency levels in ILP(1) include $(9*4+13*4) = 88$. Note that, one selection of frequency level corresponds to an affirmative allocation for one task. However, if we only consider assigning the tasks to two processing units, the searching space of frequency level is reduced to $(9+13) = 22$, which is a quarter of the original. Therefore, the desired minimum energy consumption is expected to be calculated in a much shorter time. The integer linear programming problem of the relaxed allocation problem is defined as ILP(2):

$$Min\ \ E = \sum_{j=1}^{\hat{m}}\sum_{i=1}^{n} e_{i,j}x_{i,j} + L\sum_{K=1}^{M}\sum_{j\in FLG(K)}\left(N_{j,K} - U_{j,K}\right)P_{idle,K} \tag{12}$$

Subject to:

$$U_{j,K} \le N_{j,K}\ \ (j = 1, 2, ..., \hat{m}) \tag{13}$$

where $U_{j,K} = \sum_{i=1}^{n} u'_{i,j}x_{i,j}$, and $N_{j,K}$ is integer.

$$\sum_{j\in FLG(K)} N_{j,K} \le Num_K\ \ (K = 1, 2, ..., M) \tag{14}$$

$$\sum_{j=1}^{\hat{m}} x_{i,j} = 1\ \ (i = 1, 2, ..., n) \tag{15}$$

where $x_{i,j} \in \{0, 1\}$.

In this relaxed formulation, the decision variables include $x_{i,j}$ and $N_{j,K}$, where $N_{j,K}$ denotes the number of minimum execution cores at frequency level j. For the reminder, j is a frequency level configured by one single core in PU_K. For example, when the relative utilization $U_{j,K}$ is calculated as 2.5, constraint (13) limits that at least three cores in PU_K should be chosen to provide the computing capacity. Since the energy consumption of Eq. (12) is positively correlated to $N_{j,K}$, ILP(2) computes the minimum decision of $N_{j,K}$, where $N_{j,K} = 3$. Besides, constraint (14)

ensures that the execution cores in PU_K is restricted to Num_K. Unlike ILP(1), when we restrict the computing capacity as 100% for each core in ILP(1), we limit the maximum allowed capacity as $Num_K * 100\%$ for each PU_K. However, the constraint of $\sum_{j \in FLG(K)} U_{j,K} \leq Num_K * 100\%$ cannot be used directly, as the assumption of the single-frequency scheme is not satisfied. Therefore, constraints (13), (14) are proposed instead. Given that the number of execution cores is equal to selectable frequency levels in each PU, the restriction of core number also ensures the assumption of single-frequency scheme.

Let *opt_Energy* denote the energy of the optimal allocation from ILP(1), and *desired_Energy* denote the minimum energy calculated by ILP(2). As presented in Section 4.1, ILP(1) is equivalent to the EASHM problem, thus *opt_Energy* must be the true optimum. On the contrary, the allocation problem in ILP(2) is relaxed, since the core decision for each task is not considered. It can be verified that any basic solution of ILP(1) constitutes a basic solution to ILP(2), while for some solutions of ILP(2), it may not form a solution to ILP(1). Therefore, the minimum energy consumption calculated by ILP(2) must be smaller or at least equal to the minimum energy of ILP(1):

$$desired\_Energy \leq opt\_Energy \tag{16}$$

Clearly, when the solution of ILP(1) is computed without termination, it must be optimal. But when the scale of the problem becomes large, it is difficult to use ILP(1) to solve the optimal solution because of time consumption. Then the *desired_Energy* of ILP(2) can be calculated to evaluate ILP(1)'s intermediate solution solved within a limited time.

4.3 Algorithm: Fast Terminate

Note that the objectives of ILP(1) and ILP(2) are different. While ILP(1) is to obtain a feasible task allocation, ILP(2) is to compute *desired_Energy*. In the previous section, we presented the calculation method of *desired_Energy*. In this section, we describe how the two formulations ILP(1) and ILP(2) are combined to find a reliable allocation in a limited computation time by specifying an error. The pseudocode is in Algorithm 1.

We employ LINGO tool as the ILP solver, which is able to restore and report the best solution found so far [30], i.e., the minimum energy of the current best solution of ILP(1). In this section, we denote the energy as *Energy*1. The flow of Algorithm 1 can be concluded as four steps:

(1). For a given large task set, use ILP(2) to calculate *desired_Energy*, as line 1.

(2). Execute ILP(1) and update the minimum energy *Energy*1 at all times, as line 2.

(3). If *Energy*1 is close to *desired_Energy* within a specified error, computed as: $(Energy1 - desired\_Energy)/desired\_Energy \leq Err$, terminate the execution of ILP(1) and output the best allocation solution found so far, as line 6.

(4). Otherwise, continue the execution of ILP(1) until the error is acceptable or the given time is out, as lines 3–5.

Algorithm 1 Fast Terminate

Input: task set Γ, platform $\Pi = \{PU_1, PU_2, ..., PU_M\}$, specified error **Err**, upper bound of execution time **UppTime**;
Output: task allocation queue Θ and frequency settings of each core Q;
1: Execute ILP(2) to compute **desired_Energy**
2: Execute ILP(1) and output **Energy1**
3: **while** the difference of **Energy1** and **desired_Energy** is larger than **Err**, and the execution time is within **UppTime do**
4: continue the execution of ILP(1) and update **Energy1**
5: **end while**
6: interrupt ILP(1) and output the current best solution
7: **return** Θ and Q

5. ILP Formulations on a Flexible Platform

In Section 4, we discussed the energy-efficient scheduling on a fixed platform. However, it is not difficult to see that designing a specific platform for executing different workloads results in significant energy savings in task allocation. In other words, great energy savings can be achieved through the task allocation, incorporated with the platform design. In this section, flexible core number design and flexible core type design are discussed as two independent cases, denoted as ILP(3) and ILP(4).

Since the energy table $e_{i,j}$ in Eq.(6) cannot be calculated without knowing the core assignment, it is impossible to use ILP(1) to compute the task allocation. To solve this problem, we propose ILP(3) and ILP(4) to calculate the optimal platform first. The goal of the formulation is to find the most appropriate core assignment for a given workload, so that the tasks can be assigned without deadline missing and the energy can be minimized. Note that, after the platform is determined, ILP(1) should be applied again to compute a feasible task allocation.

5.1 ILP(3): Flexible Core Number Design

With the announcement of Arm's newest DynamIQ big.LITTLE architecture, a mixing and matching of big and little CPU cores, with up to eight cores total in a cluster, is allowed for designers to build their own platform [27]. In summary, the optimal core number that minimizes the energy consumption can be theoretically calculated for a given workload.

In the case study of DynamIQ big.LITTLE architecture, Cortex-A75 and Cortex-55 are combined into a single and fully-integrated cluster, where Cortex-A75 is regarded as the high-performance core, and Cortex-A55 is chosen to be the energy-efficient core. The integrated cluster supports up to eight CPUs, or combinations thereof, as **Fig. 1**. Rather than achieving a typical octa-core design using two clusters, DynamIQ can now achieve this with one. For example, $Num_1 = 1$, $Num_2 = 7$ means one Cortex-A75 core and seven Cortex-A55 cores mixed in a cluster. Compared with conventional big.LITTLE architecture, flexible CPU design of DynamIQ technology saves more energy.

In this section, we assume that the core types are preset in advance, while the number of each type is unknown and the sum is restricted to NUM. In that case, constraint (14) in ILP(2) is no longer applicable, as Num_K is unknown. To figure out the most energy-efficient core number design, a naive method is to verify all possible combinations of Num_K and find out the best solution. However, the exhaustive solution would be time-consuming when

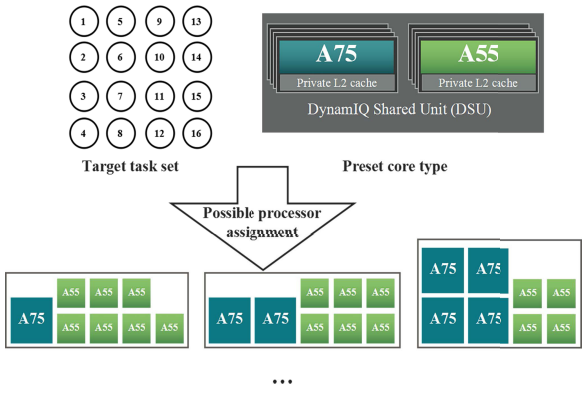


Fig. 1 Flexible core number design announced with DynamIQ technology.

core type M or core number NUM becomes large. Even though the execution time of each test is very short, the overall execution time is non-negligible, increasing exponentially with M and NUM. Therefore, we replace constraint (14) of ILP(2) with (17), in which the total number of cores does not go higher than NUM. The new model is denoted as ILP(3).

$$\sum_{K=1}^{M} \sum_{j \in FLG(K)} N_{j,K} \leq NUM \tag{17}$$

As introduced in Section 4.2, $N_{j,K}$ means the minimum number of execution cores at frequency level j, thus $\sum_{j \in FLG(K)} N_{j,K}$ represents the number of cores in PU_K. The restriction of Eq. (17) is based on the statement of ARM for practical consideration. If the restriction is deleted, the optimal solution would always choose energy-efficient cores, which makes the flexible design meaningless. It is noteworthy that the objective of ILP(3) is to compute $\sum_{j \in FLG(K)} N_{j,K}$. Once $\sum_{j \in FLG(K)} N_{j,K}$ is decided, the optimal combination of processing cores can be derived and the platform can be determined.

5.2 ILP(4): Flexible Core Type Design

In the previous section, we introduced a model to compute the optimal core number for a specific workload. However, it is unrealistic to use ILP(3) if the core types are not known. Since different processors may have various micro-architectures, flexible core type design has a higher advantage to customize the platform to achieve more energy savings.

Take ARM big.LITTLE architecture for example. High-performance processor Cortex-A15 and ultra-high-efficiency processor Cortex-A7 can be selected to construct a target platform, while Cortex-57 and Cortex-A53 may also work together playing big and small CPUs. Theoretically, any two types from the Cortex-A series supported by ARMv7 architecture, e.g., A5, A7, A8, A9, A15, A17, and any two from A32, A35, A53, A57, A72, A73 that are supported by ARMv8, can be selected as a pair, as **Fig. 2**. Additionally, with respect to Big.Medium.Little architecture [31], any three of them can be chosen to build a Big.Medium.Little platform.

In this section, we propose a general model to solve the platform design in the case that neither the core number nor the core type is known. For example, we assume six types of Cortex-A cores are given ($M = 6$), while only two of them are allowed to

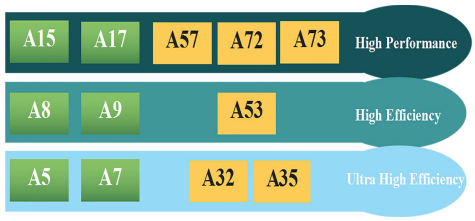


Fig. 2 Flexible core type design for Cortex-A series.

construct a big.LITTLE architecture ($Type\_num = 2$). Also, the total core number is restricted to 8 ($NUM \leq 8$). To calculate the optimal core assignment, one additional constraint should be added to restrict the selections of core type. We define a binary variable $Type_K$ to represent the decision of type K, and the function is expressed as follows:

$$Type_K = \begin{cases} 0, & \text{if } \sum_{j \in FLG(K)} N_{j,K} = 0 \\ 1, & \text{if } \sum_{j \in FLG(K)} N_{j,K} \neq 0 \end{cases} \tag{18}$$

For the reminder, the section of $\sum_{j \in FLG(K)} N_{j,K}$ denotes the execution cores in PU_K. If $\sum_{j \in FLG(K)} N_{j,K}$ is calculated as 0, PU_K is eliminated and $Type_K$ is set as 0; otherwise, PU_K is chosen and $Type_K$ is set as 1. Thus, the new model by replacing (14) of ILP(2) with following constraints is denoted as ILP(4).

$$C * Type_K \geq \sum_{j \in FLG(K)} N_{j,K} \quad (K = 1, 2, ..., M) \tag{19}$$

where $Type_K \in \{0, 1\}$.

$$Type_K \leq \sum_{j \in FLG(K)} N_{j,K} \quad (K = 1, 2, ..., M) \tag{20}$$

$$\sum_{K=1}^{M} Type_K \leq Type\_num \tag{21}$$

$$\sum_{K=1}^{M} \sum_{j \in FLG(K)} N_{j,K} \leq NUM \tag{22}$$

In ILP(4), constraints (19) and (20) are linear expressions of Eq.(18). C is a user-denoted constant for proper calculation of $Type_K$, which should be set larger than NUM. For example, when the section of $\sum_{j \in FLG(K)} N_{j,K}$ is calculated as 2, $Type_K$ can be derived as 1 according to expression (19). In addition, constraints (21) and (22) restrict that at most $Type\_num$ types of cores can be selected, and the core number is limited to NUM. In contrast to ILP(3), only when $Type_K$ and $\sum_{j \in FLG(K)} N_{j,K}$ are calculated at the same time that the platform can be determined.

Theoretically speaking, it happens that the best-suited platform calculated by ILP(3) and ILP(4) cannot derive a feasible allocation for a given task set, as the problem is relaxed following ILP(2). In that case, some popular backtracking algorithms could be considered to solve it. Given its infrequence in practice (not observed in our experiments), and limited space of the paper, this part is left for future work.

6. Experiment Results

To verify the effectiveness of proposed ILP method, simulation

Table 3 Performance comparisons of different allocation methods: average execution time with varying number of tasks.

Number of tasks	5	10	15	20	25	30	35	40	45	50	55	60	65
Execution time of ILP(1)	1 s	10 s	4 min23 s	>24 h	>24 h	>24 h	>24 h	>24 h	>24 h	>24 h	>24 h	>24 h	>24 h
Execution time of ILP(2)	1 s	1 s	1 s	1 s	2 s	3 s	3 s	3 s	4 s	4 s	9 s	16 s	22 s
Execution time of Greedy	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s
Execution time of HIT-LTF	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s	<0.1 s

Table 2 Power parameters of different core types.

	α(mW/Mhz^3)	β	s(mW)	P_{idle}(mJ)
Cortex-A7	$1.35 * 10^{-5}$	2.27	18.01	17.49
Cortex-A9	$1.71 * 10^{-6}$	2.53	67.5	28.82
Cortex-A15	$3.42 * 10^{-7}$	2.88	135.07	57.64

*1 P_{idle} is obtained from calculation.

experiments based on the latest DynamIQ big.LITTLE architecture are conducted. The DynamIQ big.LITTLE is a heterogeneous architecture that allows for mixing and matching of 'LITTLE' cores and 'big' cores. All the cores are combined into an integrated cluster and each core is capable of independent DVFS scaling [27].

We compare the evaluation results against a classic greedy algorithm [11], [32], which is commonly used in heterogeneous task allocation problems, and the latest heuristic algorithm HIT-LTF [16]. The basic idea of greedy algorithm is to calculate the best core and frequency decision for each task, thus the system energy is minimized. Clearly, in the task allocation problem, there always exists an optimal assignment for each task. Thus the extension work satisfies the original objective of classic greedy. The newest heuristic HIT-LTF is initially proposed for clustered heterogeneous multiprocessors, where the cores from one cluster share the uniform voltage and frequency setting [16]. However, the clustered DVFS scaling may cause energy waste compared with Per-Core DVFS. For a fair comparison, we realize HIT-LTF algorithm on the platform that supports Per-Core DVFS technology.

6.1 Setup

In this work, we consider using processors Cortex-A7 and Cortex-A15 to simulate the experiments. The power parameters of Cortex-A7 and Cortex-A15 are obtained from Ref. [15] by fitting on a real board EXYNOS 5422, as **Table 2** (the intention of Cortex-A9 will be explained later). Besides, the performance variance of two processors caused by internal microarchitecture and program characteristics are also taken into consideration, which is changing from $x1.9$ to $x3.0$ [33]. Since the most energy-efficient frequency could be higher than the minimum provided frequency, we assume that 'LITTLE' Cortex-A7 has a range of frequencies between 600 MHz and 1,400 MHz, while 'big' Cortex-A15 scales frequencies from 800 MHz to 2,000 MHz, all with a minimum step of 100 MHz.

For real-time tasks, we model them from real applications as the following settings: the minimum utilization u_{min} of a task is set as 0.05, and the maximum utilization u_{max} is 0.3. The workload U is calculated as the total utilization of a given task set. In the remainder of the following experiments, when we refer to utilization u_{min}, u_{max} and U, we mean the utilization calculated at the maximum frequency on little core. Additionally, the period time T for each task is generated in the interval [10, 1,000] [34].

6.2 Experiment Results on a Fixed Platform

In this section, we compare different allocation algorithms on a fixed platform. The proposed formulations are implemented by LINGO Solver [30] on processor Intel(R) Core(TM) *i*7-3770 CPU (@3.4 GHz) with 8.00 GB installed memory. We randomly generate 10 task sets for each experiment and the average performance and energy consumption are calculated from 10 groups of experiments. All the experiment results are listed in Table 3, Fig. 3 and Fig. 4.

Performance Comparisons of Different Methods: In the first experiment, we compare the average execution time of ILP(1), ILP(2) and two representative heuristics. We fix the maximum task number n as 65 for each task set because when n is increased to 65, the compared heuristics may cause allocation failures of some tasks. The average execution time of different n varying from 5 to 65 is listed in **Table 3**. In this work, as long as the solution cannot be computed within an acceptable time, e.g., 24 hours, we consider the problem as a large-scale allocation problem. As seen in Table 3, when the task number is increased to large enough, i.e., $n = 20$, the average execution time of ILP(1) exceeds 24 hours and we regard it as a large task allocation problem. It is also easy to observe that ILP(1) always performs the worst and the optimal solution is difficult to be derived in 24 hours when n is increased to 20. On the contrary, ILP(2) solves the *desired_Energy* in a very short time, which proves the efficiency of ILP(2). Additionally, the Greedy and HIT-LTF obtain the allocation results in the shortest execution time which also reflects the high performance of heuristic method.

The Effectiveness of Intermediate Solution: As described above, the optimal solution may not be computed by ILP(1) within 24 hours for large-scale allocation problems. Moreover, the optimality of the intermediate solution, i.e., the solution calculated in a limited time, cannot be evaluated. Therefore, we propose to use ILP(2) to calculate the *desired_Energy* first. Then the *desired_Energy* can be taken as a reference to evaluate the intermediate solution of ILP(1). Also, the classic greedy and heuristic HIT-LTF are tested for the same allocation problems. The comparisons among ILP(1) (executed in 1 hour), classic greedy and HIT-LTF heuristic are drawn in **Fig. 3**. It is easy to see, ILP(1) executed in 1 hour achieves very close results to *desired_Energy*, as seen in Fig. 3, the blue bars are roughly near 1. On the contrary, the two heuristic methods contribute to significant deviations. Experimental results show that ILP(1) executed in 1 hour achieves 1.005 over the *desired_Energy* in average, while classic greedy achieves 1.32 over the *desired_Energy*,

Table 4 Flexible core number design compared with fixed 4L+4B design.

Number of tasks	5	10	15	20	25	30	35	40	45
Workload U	0.93	2.04	2.99	3.91	4.8	5.64	6.57	7.55	8.64
ILP(3) Design	2L	5L	7L	8L	7L+1B	7L+1B	6L+2B	5L+3B	4L+4B
Saved Energy	0	7.7%	15%	15.1%	10.3%	8.2%	4.2%	1.4%	0%
Number of tasks	50	55	60	65	70	75	80	85	90
Workload U	9.72	10.46	11.21	12.17	13.06	13.98	14.83	15.67	16.59
ILP(3) Design	3L+5B	2L+6B	1L+7B	8B	8B	8B	8B	8B	8B
Saved Energy	1.6%	3.7%	6.2%	10.2%	13.9%	17.1%	19.8%	22.1%	24.9%

and HIT-LTF achieves 1.19 over the *desired_Energy*. In summary, the intermediate solution of ILP(1) performs better than two other heuristics. Besides, given that the energy of intermediate solution must be larger than the real optimum *opt_Energy*, we have: $opt\_Energy/desired\_Energy < 1.005$. In other words, the *desired_Energy* can be very close to *opt_Energy*, which implies that using *desired_Energy* instead of the *opt_Energy* is reasonable and efficient.

The Effectiveness of Fast Terminate: Even though the intermediate solution of ILP(1) achieves better allocation results than two heuristics, it is difficult to evaluate its optimality and decide the timeout. In this experiment, we evaluate the effectiveness of algorithm Fast Terminate. The average execution time to achieve specific percentage error, e.g., 5% off *desired_Energy*, 3% off *desired_Energy* and 1% off *desired_Energy* is drawn in **Fig. 4**. We find that when the maximum allowable error is given, the execution time of ILP(1) to obtain a feasible solution can be reduced. For existing LP solvers, the current best solution is getting closer to the optimal solution with the calculation of solver. As a result, we conclude that the larger error we set, the faster the acceptable solution can be calculated. As seen in Fig. 4, the blue bars (5% off *desired_Energy*) are always below the red bars (3% off *desired_Energy*), meanwhile the red bars are lower than the green bars (1% off *desired_Energy*). Furthermore, the execution time is also related to the scale of the task allocation. When the scale becomes larger, for example, the task number is increased from 10 to 65, the execution time to reach the specified error becomes longer. Some exceptions happen because the parameters of a task set may affect the execution time of ILP(1). For some cases, it also exists that a feasible solution with a specific error is not able to be obtained within a limited time. Therefore, an upper bound of execution time should be set, e.g., 1 hour. When the timeout is exceeded, the solver is terminated. The number of interruptions caused by the timeout in 10 experiments is listed above the bars. It is noteworthy that the calculation of average execution time does not take these terminations into account.

6.3 Experiment Results on a Flexible Platform

In the previous section, we evaluated the ILP formulations under a fixed platform assumption. In this section, we assume the core assignment is unknown for a target platform. Simulation experiments based on flexible core number design and core type design are evaluated as two study cases individually. All the experiment results are listed in Table 4, Table 5, Fig. 5, Fig. 6, Fig. 7 and Fig. 8. As different workloads may contribute to different optimal platform designs, a random task set is generated to make illustrations.

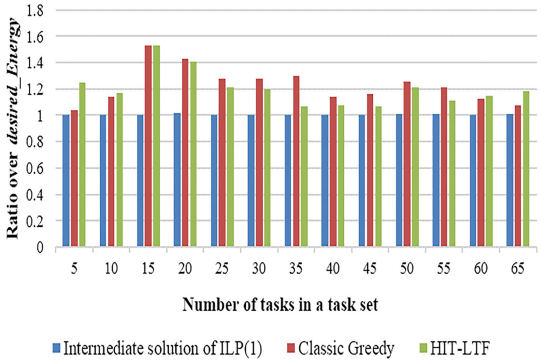


Fig. 3 The effectiveness of intermediate solution: ratio over *desired_Energy* of different allocation methods.

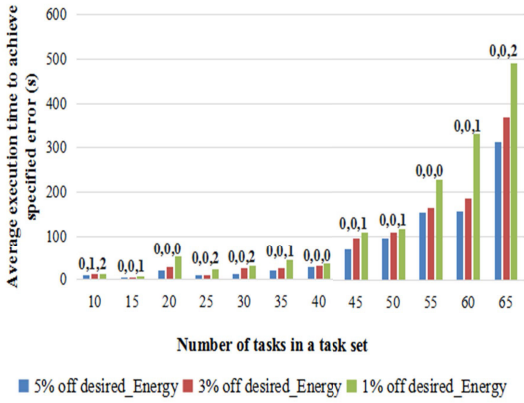


Fig. 4 The effectiveness of Fast Terminate: average execution time to achieve specific percentage error.

The Effectiveness of Flexible Core Number Design: In this experiment, we evaluate the effectiveness of flexible core number design under one restriction, i.e., up to eight cores for the combination of Cortex-A7 and Cortex-A15. With a given task set, we follow two steps to determine the platform design and allocation strategy. Firstly, we execute ILP(3) to find the most energy-efficient core number design, which is the optimal combination of Cortex-A7 and Cortex-A15 in this experiment. After that, we apply ILP(1) to compute a feasible task allocation. The same as above, we set the timeout of ILP(1) as 1 hour. If the optimal solution is unable to be calculated in 1 hour, an intermediate solution is used instead. The optimal design of ILP(3) is shown in **Table 4**, where "L" denotes Cortex-A7, and "B" denotes Cortex-A15. The energy comparison with fixed "4L+4B" design for the same task set is seen in **Fig. 5**. Since the energy consumption is calculated over a constant duration, which is the hyper-period L in this work, we calculate the average power within one hyper-period to reflect the energy difference.

Table 5 Flexible core type design compared with fixed 4L+4B design.

Number of tasks	5	10	15	20	25	30	35	40	45
Workload U	0.93	2.04	2.99	3.91	4.8	5.64	6.57	7.55	8.64
ILP(4) Design	2L	5L	7L	7L+1M	7L+1B	7L+1B	6L+2B	5L+3B	4L+4B
Saved Energy	0	7.7%	15%	16.8%	10.4%	8.2%	4.2%	1.4%	0%
Number of tasks	50	55	60	65	70	75	80	85	90
Workload U	9.72	10.46	11.21	12.17	13.06	13.98	14.83	15.67	16.59
ILP(4) Design	3M+5B	4M+4B	2M+6B	1M+7B	1M+7B	1M+7B	1M+7B	1M+7B	1M+7B
Saved Energy	2%	7.5%	8%	12%	15%	18%	20.3%	22.3%	25.2%

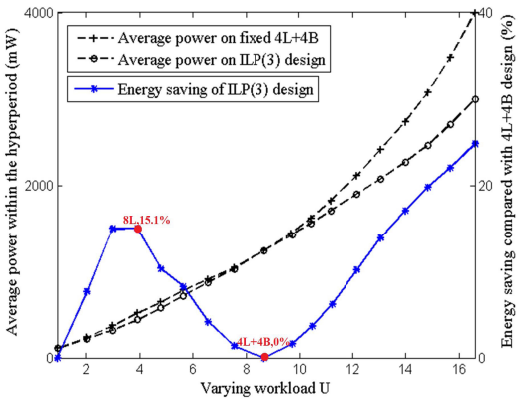


Fig. 5 The effectiveness of core number design: the tendency of energy saving with varying workload.

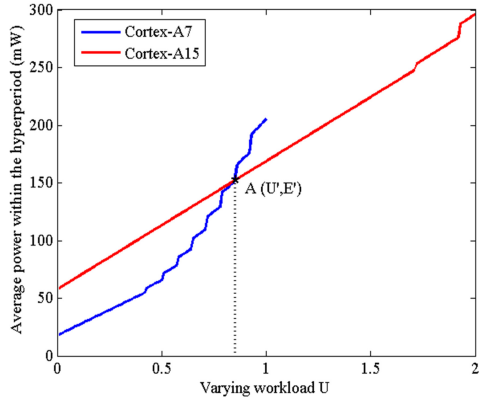


Fig. 6 The energy relation of big and little cores with varying workload.

From the experiments, it is easy to see that the optimal solution of the core number is always changing with different workloads. Besides, compared with fixed "4L+4B" design, more energy can be saved through the flexible design. Since the formulation is extended from ILP(2), the execution time of ILP(3) to achieve the optimal design is very short, maximum to 40 s according to the experimental results. In respect of energy saving, the conclusions can be summarized as follows:

(1) With the increase of workload, more and more big cores are selected for the platform construction. The reason is that for the same task, the big core shows larger computing capacity compared with the little core. As a result, to guarantee the time constraints of the tasks, more big cores should be chosen to perform the heavy workload.

(2) When the optimal design is 8L, the curve of energy saving first reaches the peak. After that, it shows a decreasing tendency until 4L+4B. If we assume the performance difference between Cortex-A7 and Cortex-A15 is not changing with workload, the energy relation can be drawn as **Fig. 6** (calculated by Eq.(2)). As can be seen, when the workload is less than U', little core Cortex-A7 consumes less energy. Therefore, choosing little core has more advantage for energy saving. When the number of little core reaches eight, it achieves the most energy saving at the first time. Afterwards big core Cortex-A15 has to be chosen to provide higher computing capacity, thus the curve of energy saving shows a decreasing tendency.

(3) When the optimal design is 4L+4B, it achieves the minimum energy saving, 0% to be more precise. After that, the saved energy increases again. As seen in Fig. 6, when the workload U increases to U', big core Cortex-A15 turns to be more energy-efficient, and the heavier the workload is, the greater the difference becomes. In conclusion, after the workload increases to the turning point A in Fig. 6, replacing small cores with big cores saves more energy.

The Effectiveness of Flexible Core Type Design: In this experiment, we consider evaluating the effectiveness of flexible core type design. Suppose that three different types of cores are given: Cortex-7, Cortex-9 and Cortex-A15. Since no real platform is composed of the three processors, it is difficult to obtain precise power parameters of Cortex-A9. For the above reason, we assume its parameters based on the energy relation of Cortex-A15 and Cortex-A9, as Table 2. Besides, we assume that Cortex-A9 scales the same frequencies as Cortex-A15 and the performance difference is between $x1.5$ and $x2.6$ [35]. Even though three different processors are prepared, only two of them are permitted to build a big.LITTLE architecture. Besides, the sum of the core number is restricted to eight. The optimal solution of ILP(4) is listed in **Table 5**, where "L" denotes Cortex-A7, "M" denotes Cortex-A9, and "B" denotes Cortex-A15. Also, it is observed that the execution time of ILP(4) is very short, maximum to 50 s according to the experimental results. After the platform design, ILP(1) with the timeout of 1 hour is used to calculate a feasible allocation. The energy saving is drawn in **Fig. 7**. It can be seen that compared with fixed "4L+4B" design and flexible core number design, flexible core type design consumes the least energy. Besides, the tendency of energy saving is the same as the flexible core number design.

The Experiment on Big.Medium.Little Architecture: Besides the big.LITTLE architecture, our method is also applicable to other heterogeneous multiprocessor systems. To demonstrate the validity and generality of platform design, we conduct experiments on a Big.Medium.Little architecture. Assume that five different types of cores are prepared in advance: Cortex-A7, Cortex-A9, Cortex-A15, in-order type A and out-of-order type

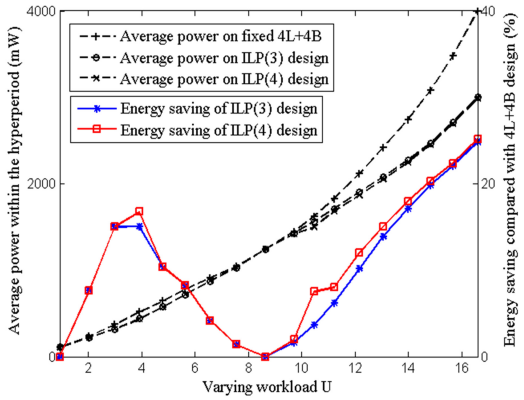


Fig. 7 The effectiveness of core type design: the tendency of energy saving with varying workload.

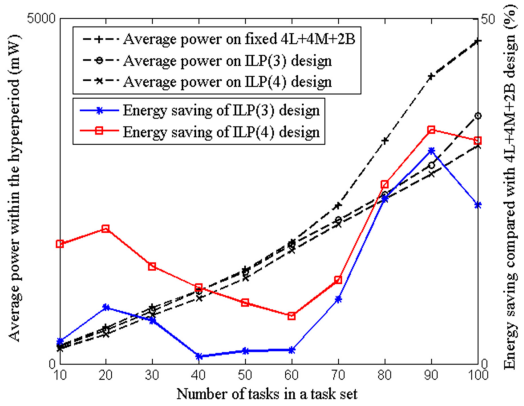


Fig. 8 A supplementary experiment on Big.Medium.Little architecture.

B. Type A is the most energy-efficient core, and type B has the highest performance. The parameters of type A and type B are generated according to the above requirements. A typical Big.Medium.Little architecture consists of three different types, with up to ten cores [31]. For comparison, we choose four Cortex-A7 cores, four Cortex-A9 cores and two Cortex-A15 cores to construct a typical and fixed Big.Medium.Little architecture. In contrast, ILP(3) and ILP(4) are applied to calculate the optimal number and types of cores. After designing the platform, ILP(1) with the termination of 1 hour is applied to find a feasible allocation. The comparisons of the average power consumption with different platform designs are drawn in **Fig. 8**. From the figure, it is observed that the fixed "4L+4M+2B" design consumes the most energy, while ILP(4) consumes the least. Besides, we find that compared with Fig. 7, larger freedom of design in terms of selective core type and permitted maximum core number saves more energy through the flexible platform design. The experimental results show that ILP(3) achieves maximum to 30.9% energy saving than the fixed design, while ILP(4) saves 33.9%. Meanwhile, the execution time of ILP(3) and ILP(4) is reasonable. In the randomly generated experiment, the best-suited platform is able to be computed within six minutes.

7. Conclusion

This work explores the energy-aware allocation for periodic real-time tasks on heterogeneous multiprocessors. After considering the power consumption in both run and idle modes, a general optimization formulation ILP(1) is proposed. The solution derived from ILP(1) not only maps tasks to optimal cores, but also decides operating frequencies. If the optimal solution is unable to be calculated in a reasonable time, a simplified ILP(2) is proposed to calculate the desired minimum energy. By referring to the desired minimum energy, the intermediate solution of ILP(1) can be evaluated, and a reliable solution within a specified error can be derived. In contrast to previous work, our method makes it possible to use integer linear programming to solve large-scale allocation problems for energy optimization within a limited time. Furthermore, a flexible platform design is also considered to incorporate with the task allocation for more energy savings. We generate ILP(3) to solve the flexible core number design, and propose ILP(4) to deal with the flexible core type design. After deciding the best-suited platform for a given workload, ILP(1) is used to find a feasible allocation. Experiment results show that compared with a fixed design, flexible core assignment achieves more energy savings. For future work, extension experiments based on a real board will be considered to evaluate our proposed models.

Acknowledgments The author Yang Qin thanks for the financial support from China Scholarship Council (CSC, 201606090182).

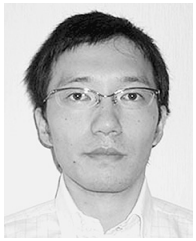
References

[1] Jeff, B.: Advances in big. little technology for power and energy savings, *ARM White paper* (2012).
[2] Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P. and Tullsen, D.M.: Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction, *Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36*, pp.81–92, IEEE (2003).
[3] Liu, J.W.: Real-Time Systems, chapter Priority-Driven Scheduling of Periodics Tasks (2000).
[4] Bambagini, M., Marinoni, M., Aydin, H. and Buttazzo, G.: Energy-aware scheduling for real-time systems: A survey, *ACM Transactions on Embedded Computing Systems* (*TECS*), Vol.15, No.1, p.7 (2016).
[5] Davis, R.I. and Burns, A.: A survey of hard real-time scheduling for multiprocessor systems, *ACM Computing Surveys* (*CSUR*), Vol.43, No.4, p.35 (2011).
[6] Liu, C.L. and Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM* (*JACM*), Vol.20, No.1, pp.46–61 (1973).
[7] Baruah, S.: Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms, *Proc. 25th IEEE International Real-Time Systems Symposium*, pp.37–46, IEEE (2004).
[8] Aydin, H. and Yang, Q.: Energy-aware partitioning for multiprocessor real-time systems, *Proc. 17th International Parallel and Distributed Processing Symposium*, p.9, IEEE (2003).
[9] Meisner, D., Gold, B.T. and Wenisch, T.F.: PowerNap: Eliminating server idle power, *ACM Sigplan Notices*, Vol.44, No.3, pp.205–216, ACM (2009).
[10] Daud, S., Ahmad, R.B., Lynn, O.B., Kareem, Z.I.A., Kamaruddin, L.M., Ehkan, P., Warip, M.N.M. and Othman, R.R.: The effects of cpu load & idle state on embedded processor energy usage, *2014 2nd International Conference on Electronic Design* (*ICED*), pp.30–35, IEEE (2014).
[11] Yu, Y. and Prasanna, V.K.: Power-aware resource allocation for independent tasks in heterogeneous real-time systems, *Proc. 9th International Conference on Parallel and Distributed Systems*, pp.341–348, IEEE (2002).
[12] Baruah, S.K.: Partitioning real-time tasks among heterogeneous multiprocessors, *International Conference on Parallel Processing* (*ICPP 2004*), pp.467–474, IEEE (2004).
[13] Zhang, W., Bai, E., He, H. and Cheng, A.M.: Solving energy-aware real-time tasks scheduling problem with shuffled frog leaping algorithm on heterogeneous platforms, *Sensors*, Vol.15, No.6, pp.13778–13804 (2015).
[14] Chen, J.-J. and Thiele, L.: Energy-efficient task partition for peri-

odic real-time tasks on platforms with dual processing elements, *14th IEEE International Conference on Parallel and Distributed Systems* (*ICPADS'08*), pp.161–168, IEEE (2008).
[15] Colin, A., Kandhalu, A. and Rajkumar, R.R.: Energy-efficient allocation of real-time applications onto single-ISA heterogeneous multicore processors, *Journal of Signal Processing Systems*, Vol.84, No.1, pp.91–110 (2016).
[16] Pagani, S., Pathania, A., Shafique, M., Chen, J.-J. and Henkel, J.: Energy efficiency for clustered heterogeneous multicores, *IEEE Trans. Parallel and Distributed Systems*, Vol.28, No.5, pp.1315–1330 (2017).
[17] Chen, H., Cheng, A.M.K. and Kuo, Y.-W.: Assigning real-time tasks to heterogeneous processors by applying ant colony optimization, *Journal of Parallel and Distributed Computing*, Vol.71, No.1, pp.132–142 (2011).
[18] Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y.C., Talbi, E.-G., Zomaya, A.Y. and Tuyttens, D.: A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, *Journal of Parallel and Distributed Computing*, Vol.71, No.11, pp.1497–1508 (2011).
[19] Zhang, W., Xie, H., Cao, B. and Cheng, A.M.: Energy-aware real-time task scheduling for heterogeneous multiprocessors with particle swarm optimization algorithm, *Mathematical Problems in Engineering*, Vol.2014 (2014).
[20] Qin, Y., Zeng, G., Kurachi, R., Matsubara, Y. and Takada, H.: Energy-aware task allocation for large task sets on heterogeneous multiprocessor systems, *2018 IEEE International Conference on Embedded and Ubiquitous Computing* (*EUC*), pp.158–165, IEEE (2018).
[21] Chen, J.-J., Schranzhofer, A. and Thiele, L.: Energy minimization for periodic real-time tasks on heterogeneous processing units, *IEEE International Symposium on Parallel & Distributed Processing* (*IPDPS 2009*), pp.1–12, IEEE (2009).
[22] Chen, Y.-J., Chang, W.-W., Liu, C.-Y., Wu, C.-E., Chen, B.-Y. and Tsai, M.-Y.: Processors Allocation for MPSoCs With Single ISA Heterogeneous Multi-Core Architecture, *IEEE Access*, Vol.5, pp.4028–4036 (2017).
[23] Wang, Y., Li, K., Chen, H., He, L. and Li, K.: Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints, *IEEE Trans. Emerging Topics in Computing*, Vol.2, No.2, pp.134–148 (2014).
[24] Zeng, G., Matsubara, Y., Tomiyama, H. and Takada, H.: Energy-aware task migration for multiprocessor real-time systems, *Future Generation Computer Systems*, Vol.56, pp.220–228 (2016).
[25] Elewi, A., Shalan, M., Awadalla, M. and Saad, E.M.: Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems, *ACM Trans. Embedded Computing Systems* (*TECS*), Vol.13, No.2s, p.71 (2014).
[26] Lin, C.-C., Syu, Y.-C., Chang, C.-J., Wu, J.-J., Liu, P., Cheng, P.-W. and Hsu, W.-T.: Energy-efficient task scheduling for multi-core platforms with per-core DVFS, *Journal of Parallel and Distributed Computing*, Vol.86, pp.71–81 (2015).
[27] ARM: ARM Developer (accessed 2017).
[28] Xie, G., Zeng, G., Li, R. and Li, K.: Energy-Aware Processor Merging Algorithms for Deadline Constrained Parallel Applications in Heterogeneous Cloud Computing, *IEEE Trans. Sustainable Computing*, Vol.2, No.2, pp.62–75 (2017).
[29] Huang, M.-K., Chang, J.M. and Chen, W.-M.: Grouping-based dynamic power management for multi-threaded programs in chip-multiprocessors, *International Conference on Computational Science and Engineering* (*CSE'09*), Vol.2, pp.56–63, IEEE (2009).
[30] LINGO: LINDO Systems (accessed 2017).
[31] Villebonnet, V., Da Costa, G., Lefevre, L., Pierson, J.-M. and Stolf, P.: "Big, Medium, Little": Reaching energy proportionality with heterogeneous computing scheduler, *Parallel Processing Letters*, Vol.25, No.3, 1541006 (2015).
[32] Braun, T., Ali, S., Siegel, H. and Maciejewski, A.: Using the Min-Min heuristic to map tasks onto heterogeneous high-performance computing systems, *2nd Symposium of the Los Alamos Computer Science Institute* (2001).
[33] Greenhalgh, P.: Big.LITTLE Processing with ARM Cortex™-A15 & Cortex-A7, *Citado na*, p.46 (2011).
[34] Davis, R.I. and Burns, A.: Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, *Real-Time Systems*, Vol.47, No.1, pp.1–40 (2011).
[35] Rajovic, N., Rico, A., Puzovic, N., Adeniyi-Jones, C. and Ramirez, A.: Tibidabo1: Making the case for an ARM-based HPC system, *Future Generation Computer Systems*, Vol.36, pp.322–334 (2014).

Yang Qin is a Ph.D student in Graduate School of Informatics, Nagoya University. She received her master degree in Software Engineering from Southeast University, China in 2015. Her research interests mainly include embedded systems, real-time systems and power-aware scheduling.

Gang Zeng is an associate professor at the Graduate School of Engineering, Nagoya University. He received his Ph.D. degree in Information Science from Chiba University in 2006. From 2006 to 2010, he was a researcher, and then assistant professor at the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing, real-time embedded system design. He is a member of IEEE and IPSJ.



Ryo Kurachi is a Designated Associate Professor of Center for Embedded Computing Systems at Nagoya University. He graduated from Tokyo University of Science with undergraduate majors in applied electronics. After a few years working at AISIN AW CO., LTD. as a software engineer, he received his master's degree in Management of Technology from Tokyo University of Science in 2007, followed by his Ph.D. in information science from the Nagoya University in 2012. His research interests include embedded systems and real-time systems. Within that domain, he has investigated topics such as in-vehicle networks and real-time scheduling theory and embedded systems security.

Yutaka Matsubara is an Associate Professor at the Graduate School of Informatics, Nagoya University. He received his Ph.D. degree in Information Science from Nagoya University in 2011. From 2009 to 2018, he was a Researcher, and then an Assistant Professor at the Center of Embedded Computing Systems (NCES), Nagoya university. His research interests include real-time operating systems, real-time scheduling theory, and system safety and security for embedded systems. He is a member of IEEE, USENIX, IPSJ and IEICE.

Hiroaki Takada is a professor at Institutes of Innovation for Future Society, Nagoya University. He is also a professor and the Executive Director of the Center for Embedded Computing Systems (NCES), the Graduate School of Informatics, Nagoya University. He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, IEICE, JSSST, and JSAE.