Energy-Aware Scheduling for Aperiodic Tasks on Multi-core Processors

Dawei Li and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, USA
{dawei.li, jiewu}@temple.edu

***Abstract*—As the performance of modern multi-core processors increases, the energy consumption in these systems also increases significantly. Dynamic Voltage and Frequency Scaling (DVFS) is considered an efficient scheme for achieving the goal of saving energy. In this paper, we consider scheduling a set of independent aperiodic tasks, whose release times, deadlines and execution requirements are arbitrarily given, on DVFS-enabled multi-core processors. Our goal is to meet the execution requirements of all the tasks, and to minimize the overall energy consumption on the processor. Instead of seeking optimal solutions with high complexity, we aim to design lightweight algorithms suitable for real-time systems, with good performances. By applying a subinterval-based method, we come up with a simple algorithm to allocate tasks' available execution times during a heavily overlapped subinterval based on their desired execution requirement during that subinterval. Based on the allocated available execution times, we further consider the final frequency setting and task scheduling, which guarantee that all tasks meet their execution requirements, and tries to minimize the overall energy consumption. Extensive simulations for various platform and task characteristics and evaluations using a practical processor's power configuration indicate that our proposed algorithm has a good performance in terms of saving processor energy, though it has low complexity. Besides, the proposed algorithm is easy to be implemented in practical systems.**

***Index Terms*—Dynamic Voltage and Frequency Scaling (DVFS), multi-core processors, aperiodic tasks, energy-aware scheduling, subinterval approach.**

I. Introduction

High energy consumption in modern computing systems has become an important issue, because it not only results in high electricity bills, but it also increases the requirements for the cooling system and other system components. Currently, the most significant portion of energy is still consumed by processors or processing cores. To facilitate energy-efficient design, the Dynamic Voltage and Frequency Scaling (DVFS) function is widely adopted [11], [15] in modern processors.

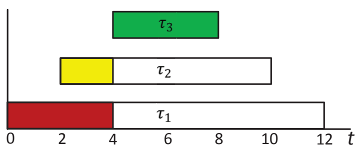
The basic idea of the DVFS strategy is to reduce a processor's (or a core's) processing frequency, as long as tasks' predefined constraints are not violated. Since the power consumption of the processor is a polynomial of the processing frequency, generally with a degree no less than 2 [20], while the overall execution time of a task is just inversely proportional to the processing frequency, DVFS provides the possibility of minimizing energy consumption given a certain performance/timing requirement.

During the past two decades, tremendous works have been done regarding energy-aware scheduling on DVFS-enabled platforms. Both circuit-level design and system scheduling have been studied in [10] and [23], respectively. It is impossible, and not necessary, to provide all of the existing research here; we refer the readers to a comprehensive survey in [13], where typical works on traditional tasks models, namely, frame-based tasks, periodic tasks, sporadic tasks, and tasks with precedence constraints have been included. Later, Li, et al. provide another survey [19], mainly focusing on energy-aware scheduling on multiprocessors; several new trends in this field are also included. For relatively simpler task models, namely, framed-based tasks and periodic tasks, intensive works have been done for energy-aware scheduling on both uniprocessors [5], [6], [14], [21] and multiprocessors [3], [7], [12]. More recent works involve new processor architectures [16], and new task characteristics [17]. Energy-aware scheduling for sporadic tasks on multiprocessors has also been well addressed [22]. Comparatively, energy-aware scheduling for general aperiodic tasks lacks extensive research endeavors.

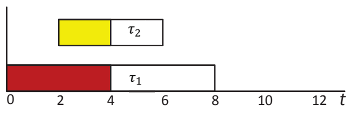
A. Related Work

[23] proposes an optimal offline algorithm for scheduling aperiodic tasks on uniprocessors. Given an aperiodic task set, $T = \{\tau_1, \tau_2, \cdots, \tau_n\}$, each task is characterized by its release time, deadline, and execution requirement, denoted by $\tau_i = (R_i, D_i, C_i)$, and all the processors have same power consumption function $p(f) = f^\alpha$. An off-line scheduling algorithm (referred to as YDS algorithm) is proposed to minimize the energy consumption of task executions. First, a set of subintervals are constructed according to all distinct release times and deadlines. The YDS algorithm is a greedy algorithm that finds the subinterval $[t_1, t_2]$ with the greatest intensity, $C(t_1, t_2)/(t_2 - t_1)$, where $C(t_1, t_2)$ denotes the execution requirement that has a release time no earlier than t_1, and has a deadline no later than t_2. The processor will run at speed $C(t_1, t_2)/(t_2 - t_1)$ during interval $[t_1, t_2]$. Then, the instance is modified as if the time interval $[t_1, t_2]$ does not exist. That is, tasks with deadlines greater than t_1 are reduced to $\max\{t_1, D_i - (t_2 - t_1)\}$ and tasks with release times greater than t_1 are reduced to $\max\{t_1, R_i - (t_2 - t_1)\}$, and the process is repeated. It is proved that the proposed scheduling method is optimal in terms of minimizing overall processor energy consumption.

For the same task model and power consumption model, considering scheduling on multiprocessor platforms, [8] proves that the problem of finding an energy minimal scheduling for execution of a set of tasks on multiprocessor with task migration allowed has polynomial complexity; it requires that the power consumption with respect to the execution frequency function, $p(f)$ is a convex one, and $p(0) = 0$. Then, a polynomial time algorithm which requires repeatedly solving linear programming problems is proposed. For the same problem with $p(f) = f^\alpha$, [2] develops a fully combinatorial algorithm with complexity $O(n^2 f(n))$ that relies on repeated maximum

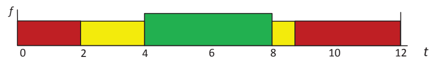


(a) The original example

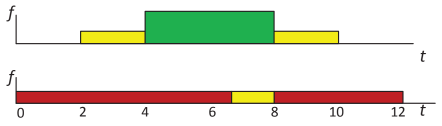


(b) τ_3 scheduled, problem updated

Fig. 1. The problem and the updated problem.



(a) Scheduling on a uniprocessor



(b) Scheduling on a dual-core processor

Fig. 2. Aperiodic task scheduling on uniprocessor and a dual-core processor.

flow computations, where n is the number of tasks, and $f(n)$ is the complexity of finding a maximum flow in a graph with $O(n)$ vertices; independently, [4] proposes a polynomial time combinatorial algorithm which is based on a reduction to the maximum flow problem and with complexity $O(nf(n)\log U)$, where U is the range of all possible values of processors' speed divided by the desired accuracy. All the works in [8], [2] and [4] require that $p(0) = 0$; in other words, static powers of processors are assumed negligible, which is no longer a suitable assumption for modern processors [12], [13], [18]. Different from these existing works, we consider the processor's static power explicitly, i.e., we assume a more practical power model: $p(f) = f^\alpha + p_0$.

B. Introductory Example

We would like to give a simple example to demonstrate how the YDS algorithm schedules tasks on a uniprocessor first. We consider three tasks, which are given in Fig. 1(a), whose release times are $R_1 = 0, R_2 = 2, R_3 = 4$, and deadlines are $D_1 = 12, D_2 = 10, D_3 = 8$. Their execution requirements are $C_1 = 4, C_2 = 2, C_3 = 4$. According to the definition of interval intensity, it is easy to find that the interval with the greatest intensity is $[4, 8]$ and its intensity is $C_3/(8-4) = 1$. Thus, during this interval, the uniprocessor should execute at frequency $f = 1$. After this step, we update the problem instance as described before and get an instance shown in Fig. 1(b). Comparing intervals $[2, 6]$ and $[0, 8]$, interval $[2, 6]$ is with intensity $C_2/(6-2) = 0.5$, while $[0, 8]$ is with intensity $(C_1 + C_2)/(8-0) = 0.75$. Thus, $[0, 8]$ is the interval with the greatest intensity. During this interval, the uniprocessor should execute at frequency $f = 0.75$. The two tasks, τ_1 and τ_2 are scheduled by the Earliest Deadline First (EDF) scheme. Combined with the first step, i.e., the scheduling for τ_3, we can achieve the practical overall scheduling shown in Fig. 2(a). In this paper, we address energy-aware scheduling for general aperiodic tasks on homogeneous multi-core processors, with the explicit consideration of processor's static power. We design lightweight algorithms for the problem and conduct extensive simulation to verify the applicability of the proposed algorithms.

C. Contributions and Paper Organization

Our main contributions can be outlined as follows:

- We address energy-aware scheduling for general aperiodic tasks on multi-core processors. Different from existing works, we explicitly consider processors' static powers. We formulate the problem in a clear and formal way, which helps us find the key aspect of a solution/scheduling. Based on the formulated problem, we also show that the energy-aware scheduling problem with the consideration of static powers is still polynomial time solvable, however, with high complexity.
- Instead of seeking polynomial time solutions with high complexity, we propose a lightweight algorithm suitable for real-time systems to solve the problem efficiently with good performances. Namely, we propose to allocate execution time for a task during an heavily overlapped subinterval, where the number of overlapping tasks is greater than the number of processors, according to the task's Desired Execution Requirement (DER) during this subinterval.
- We demonstrate the practical usage of the proposed algorithms by numerical simulations; also, we evaluate our scheduling algorithms using a practical processor's power consumption characteristics. Our scheduling mechanisms are easy to implement in a practical system.

The rest of the paper is organized as follows. In Section II, we describe a simple example to motivate our work. We give a clear and formal definition of the problem in Section III. In Section IV, we obtain some important characteristics of an optimal solution; then, the original problem is reformulated into a convex optimization problem that can be solved in polynomial time. After that, two algorithms are described in Section V, where an illustration example is also provided to demonstrate our overall approach. Numerical simulations and evaluations using a practical processor's power consumption characteristics are presented in Section VI. We conclude our paper in Section VII.

II. Motivational Example

We demonstrate our motivation using the same task example in Fig. 1(a). We assume that the power consumption of each task is $p(f) = f^\alpha + p_0, \alpha = 3, p_0 = 0.01$, if the task is executed at frequency f, and the energy consumption of task τ_i is $E_i = p(f)(C_i/f) = (f^3 + p_0)C_i/f$. Instead of considering the problem on a uniprocessor, we consider the problem on a homogeneous multi-core processor, where any core can process at most one task at any instant time. In this example with three tasks, we consider scheduling the tasks on a processor with two cores such that the execution requirements are met and the overall energy consumption is minimized.

We can see that before time $R_2 = 2$, only one task, i.e., τ_1, is ready to execute; before time $R_3 = 4$, only two tasks, i.e., τ_1 and τ_2, are ready to execute. Besides, by now, the number of ready tasks does not exceed the number of available cores, 2. Thus, during interval $[0, 4]$, core M_1 can be exclusively allocated to τ_1; during interval $[2,4]$, core M_2 can be exclusively allocated to τ_2. Similarly, M_1 can be exclusively allocated to τ_1 during interval $[8,12]$; M_2 can be exclusively allocated to τ_2 during interval $[8,10]$. However, during interval $[4,8]$, all of the three tasks are ready for execution, while we only have two cores. Denote the time that each task occupies a core during interval $[4,8]$, as x_1, x_2, x_3, respectively. Also, denote the total time that τ_1 occupies a core during intervals $[0,4]$ and $[8,12]$ as y_1, and the total time that τ_2 occupies a core during the intervals $[2,4]$ and $[8,10]$ as y_2. We have the following constraints:

$$\begin{aligned} 0 \le x_1, x_2, x_3 &\le 4; \\ x_1 + x_2 + x_3 &\le 8; \\ 0 \le y_1 &\le 8; \\ 0 \le y_2 &\le 4. \end{aligned}$$

We need to minimize the energy consumption:

$$\begin{aligned} E &= ((4/(x_1+y_1))^3+0.01)(x_1+y_1)+ \\ &\quad ((2/(x_2+y_2))^3+0.01)(x_2+y_2)+((4/x_3)^3+0.01)x_3 \\ &= 64/(y_1+x_1)^2+8/(y_2+x_2)^2+64/x_3^2+ \\ &\quad 0.01(x_1+x_2+x_3+y_1+y_2). \end{aligned}$$

By solving the KKT conditions [9] for this optimization problem with inequality constraints, we can obtain that the optimal values for x_1, x_2, x_3, y_1, y_2 are $8/3, 4/3, 4, 8, 4$ respectively. The minimal energy consumption is $64/(8+8/3)^2 + 8/(4+4/3)^2 + 64/4^2 = 155/32$. Note that the execution core and order for the three tasks during interval $[4,8]$ can be arbitrary. In practice, we can choose the best method to avoid unnecessary preemptions and migrations. The final scheduling for the three tasks is shown in Fig. 2(b). For this simple example, we can use KKT conditions to solve the problem. However, generally, for complex cases, these kinds of problems are difficult to solve, especially when static powers are introduced. One state-of-the-art approach to solve these problems is the Interior Point method, which requires a large number of numeric evaluations and iterations. The time complexity of this method is too high to be used in real-time systems. In this paper, instead of seeking optimal solutions with high complexity as in [2], [4], and [8], we propose lightweight algorithms for the problem.

III. System Model and Problem Definition

A. Task Model

We consider scheduling a set of independent aperiodic tasks $T = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Each task τ_i is represented by a three tuple, $\tau_i = (R_i, D_i, C_i)$, where R_i is the release time of τ_i, $D_i > R_i$ is the deadline of τ_i, and C_i is the execution requirement of τ_i. Tasks do not have precedence constraints. We assume that all of the tasks are preemptive, and migrations are allowed.

B. Platform Model

We consider a multi-core processor with m DVFS-enabled independent processing cores. By independent, we mean that the cores can execute at different frequencies at any time, and can adjust their execution frequencies independently. We assume ideal processing cores whose frequency ranges are continuous on $(0, +\infty)$. Cores can operate in two modes when it is on: *active* mode and *sleep* mode. Active mode refers to the state when it is executing some task and the power consumption is the sum of both dynamic power and static power, $p(f) = f^\alpha + p_0$, $\alpha \ge 2$. When a core has no task to execute, it enters the sleep mode immediately to save energy, and the power consumption becomes zero.

C. Problem Definition

Given a set of preemptive aperiodic tasks $T = \{\tau_1, \tau_2, \cdots, \tau_n\}$, our goal is to schedule all of the tasks on a DVFS-enabled processor with m homogeneous cores, $M_1, M_2, \cdots, M_m$, such that the overall energy consumption is minimized. Since tasks are preemptive, and migrations are allowed, each task τ_i's execution might consist of K_i segments. The jth ($1 \le j \le K_i$) segment of τ_i's execution starts at time $r_{i,j}$ and ends at time $d_{i,j}$. For notational brevity, and to avoid segments overlapping at end points, we denote the jth execution segment of τ_i as $[r_{i,j}, d_{i,j})$. We index these segments sequentially, such that $R_i \le r_{i,1} < d_{i,1} \le r_{i,2} < d_{i,2} \le \cdots \le r_{i,j} < d_{i,j} \le \cdots \le r_{i,K_i} < d_{i,K_i} \le D_i$. Assume that each segment completes an execution requirement of $c_{i,j}$. Each task's execution segments lie within its release time and deadline. We denote τ_i's overall execution intervals as the union of all of its execution segments, namely, $U_i = [r_{i,1}, d_{i,1}) \cup \cdots \cup [r_{i,j}, d_{i,j}) \cup \cdots \cup [r_{i,K_i}, d_{i,K_i})$. Let $f_{i,j}$ be the execution frequency for τ_i's jth execution segment. Then, the execution requirement completed during the jth segment is $c_{i,j} = (d_{i,j} - r_{i,j}) f_{i,j}$. Let $\bar{R} = \min_{i=1}^{n}\{R_i\}$, which is the earliest release time of all the tasks, and $\bar{D} = \max_{i=1}^{n}\{D_i\}$, which is the latest deadline of all the tasks. We introduce a 0-1 function, $exe(i,t)$, to indicate whether task τ_i is executing at time t:

$$exe(i,t) = \begin{cases} 1 & \text{if } t \in U_i. \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Energy consumption of each task:

$$E_i = \sum_{j=1}^{K_i} c_{i,j} \left(f_{i,j}^{\alpha-1} + \frac{p_0}{f_{i,j}} \right) \tag{2}$$

The execution requirement satisfies: $\sum_{j=1}^{K_i} c_{i,j} = C_i$.

One important constraint we should keep in mind is that, at any time, the number of executing tasks must be less than or equal to the number of cores. Obviously, the time span we need to consider is from $\bar{R}$ to $\bar{D}$.

$$\sum_{i=1}^{n} exe(i,t) \le m, \forall t \in [\bar{R}, \bar{D}]. \tag{3}$$

The optimization problem can be formulated as follows:

$$\begin{aligned} \min \quad & E_{total} = \textstyle\sum_{i=1}^{n} E_i & (4) \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{K_i} c_{i,j} = C_i, \forall i = 1, 2, \cdots, n; & (5) \\ & \textstyle\sum_{i=1}^{n} exe(i,t) \le m, \forall t \in [\bar{R}, \bar{D}]; & (6) \\ & R_i \le r_{i,1} < d_{i,1} \le r_{i,2} < d_{i,2} \le \cdots \le r_{i,j} & (7) \\ & < d_{i,j} \le \cdots \le r_{i,K_i} < d_{i,K_i}, \forall i = 1, 2, \cdots, n. & (8) \end{aligned}$$

It is no easy task to solve this optimization problem directly. In the following, we will attack the problem step by step, starting with considering the ideal optimal situation of the problem. Related notations are provided in Table I; some of the meanings will be made clear later.

IV. Preliminaries

In this section, we uncover some important characteristics of an optimal solution. Also, the original problem is reformulated into a convex optimization problem for guiding efficient lightweight algorithms.

TABLE I. NOTATIONS USED IN THIS PAPER

Notation	Description
n, m	the number of tasks and processing cores.
T	task set $\{\tau_1, \tau_2, \cdots, \tau_n\}$; τ_i is the ith task.
$\bar{R}$	the earliest release time of all the tasks.
$\bar{D}$	the latest deadline of all the tasks.
R_i, D_i, C_i	release time, deadline, execution requirement of τ_i.
$r_{i,j}$	the left end of task τ_i's jth execution interval.
$d_{i,j}$	the right end of task τ_i's jth execution interval.
$f_{i,j}$	execution frequency of τ_i during its jth segment.
$exe(i,t)$	binary function indicating if task τ_i is executing at time t.
M_j	the jth processing core ($j = 1, 2, \cdots, m$).
$p(f)$	core's power when it is running at f.
$f^O_{\tau_i}$	optimal frequency setting for task τ_i in the ideal situation where the number of cores is unlimited.
$[t_j, t_{j+1}]$	the jth constructed subinterval.
n_j	the number of overlapping tasks during the jth subinterval.
$\tau_{j,i}$	the ith overlapping tasks during the jth subinterval.
$t(\tau_{j,i})$	available execution time of $\tau_{j,i}$ during the jth subinterval.
$c(\tau_{j,i})$	desired execution requirement of $\tau_{j,i}$ during the jth subinterval.
$x_{i,j}$	the execution time of τ_i during the jth subinterval.
E^O	optimal energy consumption of the ideal case, where the number of cores is unlimited.
$\bar{E}^O$	practically achievable optimal energy consumption of the problem.
$A^{F_1}_i$, $A^{F_2}_i$	task τ_i's total available execution time calculated by the evenly and the DER-based allocating methods, respectively.
E^{I_1},E^{F_1}	overall energy consumption of the intermediate and final schedulings based on the evenly allocating method.
E^{I_2},E^{F_2}	overall energy consumption of the intermediate and final schedulings based on the DER-based allocating method.

A. Characteristics of an Optimal Solution

Observation 1: in an optimal solution, no matter how many segments a task's execution consists of, the execution frequencies for this task during all its intervals should be equal, i.e., $f_{i,1} = f_{i,2} = \cdots = f_{i,K_i}, \forall i = 1, 2, \cdots, n$.

Illustration: consider a scheduling, in which task τ_i's execution frequency during its jth interval is $f_{i,j}$. The execution requirement completed during each interval is $c_{i,j} = f_{i,j}(d_{i,j} - r_{i,j})$. The following condition holds:

$$\sum_{j=1}^{K_i} f_{i,j}(d_{i,j} - r_{i,j}) = C_i. \tag{9}$$

Energy consumption of task τ_i can be calculated as:

$$E_i = \sum_{j=1}^{K_i} (f^{\alpha}_{i,j} + p_0)(d_{i,j} - r_{i,j}). \tag{10}$$

To minimize E_i, we can apply the Lagrange Multiplier Method, which tells that:

$$\alpha f^{\alpha-1}_{i,j}(d_{i,j} - r_{i,j}) - \lambda(d_{i,j} - r_{i,j}) = 0, \forall j = 1, \cdots, K_i, \tag{11}$$

where λ is the Lagrange Multiplier. Thus, $f_{i,1} = f_{i,2} = \cdots = f_{i,K_i} = \sqrt[\alpha-1]{\lambda/\alpha}$. Applying the first constraint, we have

$$f_{i,j} = C_i / \sum_{j=1}^{K_i} (d_{i,j} - r_{i,j}), \forall j = 1, 2, \cdots, K_i. \tag{12}$$

Thus, the execution frequencies of τ_i during all its intervals should be equal, and the common execution frequency is equal to the overall execution requirement of τ_i divided by the overall execution time of τ_i [20].

We sort all R_i and D_i values in ascending order, and relabel the distinct values as $t_1, t_2, \cdots, t_N$, where $N \le 2n$ is the total number of distinct R_i and D_i values; $t_1 = \bar{R}$ is the earliest release time, and $t_N = \bar{D}$ is the latest deadline. Through this method, we also construct a set of $N-1$ subintervals: $\{[t_1, t_2], [t_2, t_3], \cdots, [t_{N-1}, t_N]\}$. After this, the key of the problem is to determine the execution time of each task τ_i's during each subinterval $[t_j, t_{j+1}]$.

B. Problem Reformulation

We are now ready to reformulate the problem into a convex optimization problem. Denote the execution time of the ith task during the jth subinterval, $[t_j, t_{j+1}]$ by $x_{i,j}$, $i = 1, \cdots, n, j = 1, \cdots, N-1$. An obvious fact is that, task τ_i's execution segment(s) lies within subintervals that are covered by interval $[R_i, D_i]$. We have

$$\begin{cases} x_{i,j} = 0 & \text{if } [t_j, t_{j+1}] \notin [R_i, D_i]. \\ 0 \le x_{i,j} \le t_{j+1} - t_j & \text{if } [t_j, t_{j+1}] \in [R_i, D_i]. \end{cases} \tag{13}$$

Also, the total execution time of all tasks during subinterval $[t_j, t_{j+1}]$ should be less than the total execution time that is available:

$$\sum_{i=1}^{n} x_{i,j} \le m(t_{j+1} - t_j), \forall j = 1, 2, \cdots, N-1. \tag{14}$$

The original problem can be reformulated as follows:

$$\min \quad E_{total} = \textstyle\sum_{i=1}^{n}((\sum_{j=1}^{N-1} x_{i,j})((\frac{C_i}{\sum_{j=1}^{N-1} x_{i,j}})^{\alpha} + p_0)) \tag{15}$$

$$\text{s.t.} \quad (13) \text{ and } (14). \tag{16}$$

Theorem 1: The energy minimal scheduling of aperiodic tasks on multi-core processors with static power consumptions and migrations allowed is polynomial time solvable.

Proof: First, the reformulated problem is a convex programming program that can be solved in polynomial time by the Interior Point method [9]. We demonstrate this by showing both the constraints and objective function are convex. Obviously, the constraints in Equations (13) and (14) are linear, thus, are convex. In Equation (15), $\sum_{j=1}^{N-1} x_{i,j}$ is the total execution time of task τ_i. $\alpha \ge 2$ guarantees that $C_i^3/(\sum_{j=1}^{N-1} x_{i,j})^{\alpha-1}$ is convex. Apparently, $p_0 \sum_{j=1}^{N-1} x_{i,j}$ is convex. Thus, the objective function is also convex. Second, given the optimal solution of the convex programming problem, i.e., the optimal values for $x_{i,j}$'s, a valid scheduling can be derived if all the tasks are preemptive and migrations are allowed, as will be shown later in Algorithm 1. ■

We denote the practically achievable optimal energy consumption by $\bar{E}^O$. However, achieving the optimal solution using the Interior Point method requires a large number of numerical evaluations and iterations. Besides, the reformulated problem has $O(n^2)$ number of variables; this fact also incurs significant time complexity for achieving the optimal solution. Instead of seeking optimal solutions with high complexity, as in [8], [2], and [4], we consider developing lightweight algorithms which are suitable for real-time systems.

Before further discussion, we give the following definitions. The *overlapping tasks during a subinterval*, $[t_j, t_{j+1}]$, is the set of tasks whose release times are less than or equal to t_j, and whose deadlines are greater than or equal to t_{j+1}. Denote the number of overlapping tasks during subinterval $[t_j, t_{j+1}]$ by n_j. A *heavily overlapped subinterval* is a subinterval from $\{[t_1, t_2], [t_2, t_3], \cdots, [t_{N-1}, t_N]\}$, during which, the number of overlapping tasks is greater than the number of cores. A *lightly overlapped subinterval* is an interval from $\{[t_1, t_2], [t_2, t_3], \cdots, [t_{N-1}, t_N]\}$, during which, the number of

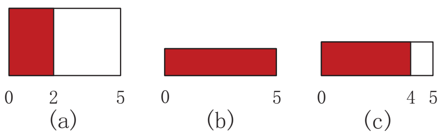


Fig. 3. Available execution time versus actual execution time. (a) The execution requirement and its available execution time. (b) The scheduling use all available execution time. (c) The scheduling use a part of the available execution time.

overlapping tasks is less than or equal to the number of cores.

Observation 2: during a subinterval $[t_j, t_{j+1}]$, if it is a lightly overlapped subinterval, the overlapping tasks during this subinterval are valid to occupy a processing core for the whole subinterval.

Illustration: this is obvious, since the number of overlapping tasks during such a subinterval is less than or equal to the number of cores. Any task other than the overlapping tasks during this subinterval either are not ready for execution or have been completed (deadline has passed).

According to our problem reformulation and observations, it can be noticed that the key of the problem lies in how to allocate the execution time to the overlapping tasks during each subinterval, i.e., how to determine $x_{i,j}$'s. Due to the existence of static power, a certain amount of execution time allocated to a task might not be actually used by this task. For example, in Fig. 3, assume that the task is valid to occupy a core from 0 to 5. The power consumption of a core is $p(f) = f^2 + 0.25$. Using all available execution time from 0 to 5 (Fig. 3(b)) results in an energy consumption of 2.05 (at frequency 0.4); using only available execution time from 0 to 4 (Fig. 3(c)) results in an energy consumption of 2.00 (at frequency 0.5). Thus, using only a part of the available execution time may be better. We first consider how to allocate "available" execution times to tasks. According to *Observation 2*, for a lightly overlapped subinterval, we can allocate $(t_{j+1} - t_j)$ to each of its overlapping tasks' available execution time. In the next section, we address how to allocate available execution times to overlapping tasks during a heavily overlapped subinterval.

V. Subinterval-based Scheduling

As for how to allocate available execution time during a heavily overlapped subinterval, two approaches can be applied. One allocates available execution times evenly among all overlapping tasks. The other method, which is intuitively more reasonable, is to allocate the available execution times based on tasks' DERs during this subinterval. We consider an ideal case, where the number of cores is unlimited, to define the DER.

A. An Ideal Case

Consider an ideal case, where the number of processing cores is unlimited. In this situation, we do not need to consider the collisions among tasks, and can simply execute one task on one core. In this ideal case, the only constraint is that the execution time of task τ_i should not exceed $D_i - R_i$. Assume that task τ_i's execution frequency is f_i. Then, energy consumption of τ_i is

$$E_i^{ideal} = C_i \left(f_i^{\alpha-1} + p_0/f_i \right). \tag{17}$$

The optimal energy consumption can be achieved by solving the following optimization problem:

$$\min \quad \sum_{i=1}^{n} E_i^{ideal} \tag{18}$$

$$\text{s.t.} \quad f_i \geq C_i/(D_i - R_i). \tag{19}$$

We denote the optimal frequency setting for τ_i by $f_{\tau_i}^O$. Applying the KKT conditions for each optimization problem, we can easily get the analytical expression for $f_{\tau_i}^O$:

$$f_{\tau_i}^O = \max\{ \sqrt[\alpha]{p_0/(\alpha-1)}, C_i/(D_i - R_i)\}. \tag{20}$$

Denote the execution interval of τ_i, in this ideal scheduling, $\mathcal{S}^O$, by $U_{\tau_i}^O = [R_i, R_i + C_i/f_{\tau_i}^O]$, and the optimal energy consumption of τ_i by $E_{\tau_i}^O$, which can be calculated as follows:

$$E_{\tau_i}^O = C_i \left((f_{\tau_i}^O)^{\alpha-1} + p_0/f_{\tau_i}^O \right). \tag{21}$$

Also, the optimal energy consumption of this ideal case is: $E^O = \sum_{i=1}^{n} E_{\tau_i}^O$.

B. Scheduling by the Evenly Allocating Method

1) An Intermediate Scheduling: We consider allocating available execution times evenly among all overlapping tasks during a heavily overlapped subinterval $[t_j, t_{j+1}]$. First, we construct an intermediate scheduling, $\mathcal{S}^{I_1}$, in which the execution requirement completed in interval $[t_j, t_{j+1}]$ is equal to the ideal optimal case, $\mathcal{S}^O$, where the number of cores is unlimited. Since the number of overlapping tasks during interval $[t_j, t_{j+1}]$ is $n_j, n_j > m$, we allocate each task τ_i an available execution time of $m(t_{j+1} - t_j)/n_j$. If in $\mathcal{S}^O$, the execution time of τ_i is less than or equal to this amount, we leave the execution frequency unchanged. If in $\mathcal{S}^O$, the execution time of τ_i is greater than this amount, say, $t^O (m(t_{j+1} - t_j)/n_j < t^O \leq t_{j+1} - t_j)$, in order for the task to complete the same amount of execution requirement, we need to increase the execution frequency to $f_{\tau_i}^O t^O/(m(t_{j+1} - t_j)/n_j)$, which is at most $f_{\tau_i}^O n_j/m$.

By increasing the execution frequency, the dynamic energy consumption during this subinterval will be increased by at most $(n_j/m)^{\alpha-1}$, while the static energy consumption will reduce. Thus, the total energy consumption of task τ_i will not be greater than $(n_j/m)^{\alpha-1} E_i^O$. Note that we do not specify which task we are considering, which means that, for each overlapping task in interval $[t_j, t_{j+1}]$, its energy consumption is not greater than $(n_j/m)^{\alpha-1}$ times its optimal energy consumption in the ideal case. Denote $n_j^{max} = \max\{m, \max_{j=1}^{N-1} n_j\}$. It is easy to notice that this intermediate scheduling, $\mathcal{S}^{I_1}$, has an energy consumption, E^{I_1}, that is no greater than $(n_j^{max}/m)^{\alpha-1} E^O$.

Again, the allocated available execution time may not be fully used by each task, due to the existence of static power. In the following, we will use "task scheduling" and "task's available execution time scheduling" interchangeably. In our approach, we require that each task's executions are only mapped to its available execution intervals. By now, we have allocated available execution times to tasks; to avoid task collisions during a heavily overlapped subinterval, we still need to schedule the overlapping tasks in a safe way. Algorithm 1 provides a safe way to schedule these tasks' available execution times. By the evenly allocating method, each task's allocated execution time during this interval is $t(\tau_{j,1}) = t(\tau_{j,2}) \cdots = t(\tau_{j,n_j}) = m(t_{j+1} - t_j)/n_j$. P_k, initialized as $P_k = t_j, k = 1, 2, \cdots, m$, represents the earliest available time of core M_k during subinterval $[t_j, t_{j+1}]$.

Algorithm 1 Available Execution Time Scheduling During a Heavily Overlapped Subinterval

```
Input: a heavily overlapped subinterval, $[t_j, t_{j+1}]$; the set of overlapping tasks during this interval, $T_j = \{\tau_{j,1}, \tau_{j,2}, \cdots, \tau_{j,n_j}\}$, $n_j > m$; Each task's allocated execution time during this interval $t(\tau_{j,i}), \forall i = 1, 2, \cdots, n_j$;
Output: A scheduling of tasks without collision;
1: $P_k = t_j, \forall k = 1, 2, \cdots, m$; $k = 1$;
2: for $i := 1$ to $n_j$ do
3:     if $P_k + t(\tau_{j,i}) > t_{j+1}$ then
4:         Schedule the first part of $\tau_{j,i}$ on core $M_{k+1}$ from time $t_j$ to time $t_j + P_k + t(\tau_{j,i}) - t_{j+1}$; $P_{k+1} = t_j + P_k + t(\tau_{j,i}) - t_{j+1}$;
5:         Schedule the second part of $\tau_{j,i}$ on core $M_k$ from time $P_k$ to time $t_{j+1}$; $P_k = t_{j+1}$;
6:         $k = k + 1$;
7:     else
8:         Schedule $\tau_{j,i}$ on core $M_k$ from time $P_k$ to time $P_k + t(\tau_{j,i})$; $P_k = P_k + t(\tau_{j,i})$;
```

2) Final Scheduling of the Evenly Allocating Method: A refined scheduling can be constructed based on $\mathcal{S}^{I_1}$. Since we have allocated available execution times for each task during every lightly overlapped subinterval and every heavily overlapped subinterval, we can calculate the total available execution time for each task τ_i, denoted by $A_i^{F_1}$. The optimal frequency setting for τ_i can be determined by solving the following optimization problem:

$$\min \quad C_i \left(f_i^{\alpha-1} + p_0/f_i\right) \tag{22}$$

$$\text{s.t.} \quad f_i \geq C_i/A_i^{F_1}. \tag{23}$$

which has the solution, $f_i = \max\{\sqrt[\alpha]{p_0/(\alpha-1)}, C_i/A_i^{F_1}\}$.

We denote this final scheduling as $\mathcal{S}^{F_1}$. Since $\mathcal{S}^{F_1}$ is further optimized based on $\mathcal{S}^{I_1}$, the energy consumption of these three schedulings has the following relation: $E^{F_1} \leq E^{I_1} \leq (n_j^{max}/m)^{\alpha-1} E^O$.

C. Scheduling by the DER-based Allocating Method

The evenly allocating method ignores the execution requirements of overlapping tasks; thus, it may result in tasks not efficiently utilizing the available execution times. In the following, we propose another method, which allocates available execution times to tasks, according to their desired execution requirements, in a heavily overlapped subinterval.

1) An Intermediate Scheduling: In the ideal case, each task whose $[R_i, D_i]$ contains $[t_j, t_{j+1}]$ is valid to occupy the entire subinterval. We define the DER of $\tau_{j,i}$ during this heavily overlapped subinterval as:

$$c(\tau_{j,i}) = |U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}]| f^O_{\tau_{j,i}}. \tag{24}$$

where $f^O_{\tau_{j,i}}$ is the optimal frequency setting of task $\tau_{j,i}$ in the ideal case, $\mathcal{S}^O$. $|U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}]|$ represents $\tau_{j,i}$'s execution time of the scheduling of the ideal case, during subinterval $[t_j, t_{j+1}]$. This value may be not equal to $[t_j, t_{j+1}]$, due to the existence of the static power. Moreover, if $U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}] = \emptyset$, $c(\tau_{j,i}) = 0$. Different tasks' execution times and optimal execution frequencies may be different, resulting in the fact that the desired execution requirements in this interval are different. For example, if a task's execution requirement is quite small, while its valid execution time, $D_i - R_i$, is very large, then, the optimal execution frequency of this task will be low (assuming a low static power). For such a task in a heavily overlapped subinterval, allocating more available execution time to this task may not reduce the overall energy consumption. Intuitively, we can allocate more available execution time to tasks whose desired execution requirement is high.

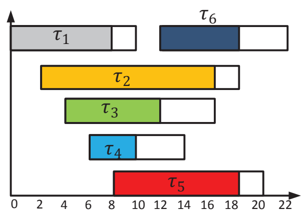
Algorithm 2 DER-based Available Execution Time Allocation

```
Input: a heavily overlapped subinterval, $[t_j, t_{j+1}]$; the set of overlapping tasks during this interval, $T_j = \{\tau_{j,1}, \tau_{j,2}, \cdots, \tau_{j,n_j}\}$; the desired execution requirement of each task during this interval $c(\tau_{j,i}), \forall i = 1, 2, \cdots, n_j$; the number of cores, $m$;
Output: task $\tau_{j,i}$'s execution time during this interval, $t(\tau_{j,i}), \forall i = 1, 2, \cdots, n_j$;
1: $C = \sum_{i=1}^{n_j} c(\tau_{j,i})$;
2: Sort tasks in $T_j$ in descending order of their $c(\tau_{j,i})$ values. Denote the sorted order set as $\{\tau_{j,i_1}, \tau_{j,i_2}, \cdots, \tau_{j,i_{n_j}}\}$; //$i_1, i_2, \cdots, i_{n_j}$ is a permutation of $1, 2, \cdots, n_j$; $c(\tau_{j,i_1}) \geq c(\tau_{j,i_2}) \geq \cdots \geq c(\tau_{j,i_{n_j}})$.
3: for $k := 1$ to $n_j$ do
4:     if $\frac{c(\tau_{j,i_k})}{C} \geq \frac{1}{m}$ then
5:         $t(\tau_{j,i_k}) = t_{j+1} - t_j$;
6:         $C = C - c(\tau_{j,i_k})$;
7:         $m = m - 1$;
8:     else
9:         $t(\tau_{j,i_k}) = \frac{c(\tau_{j,i_k})}{C} m(t_{j+1} - t_j)$;
```

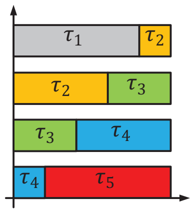
We apply Algorithm 2 to allocate available execution times during each heavily overlapped subinterval. In Algorithm 2, C represents the total execution requirement of all the overlapping tasks during this subinterval. When allocating available execution times, this algorithm considers the task with the greatest DER first. For example, if $\tau_{j,1}$ is the task with the greatest DER, the algorithm attempts to allocate $c(\tau_{j,1})/C$ of the total execution time of all cores, $m(t_{j+1}-t_j)$. If $c(\tau_{j,i})/C > 1/m$, the desired execution time of $\tau_{j,1}$ is $c(\tau_{j,1})m(t_{j+1}-t_j)/C > t_{j+1}-t_j$, which is not valid. Thus, $\tau_{j,1}$ is allocated $t_{j+1}-t_j$. If $c(\tau_{j,1})/C \leq 1/m$, $\tau_{j,1}$ can be allocated its desired execution time. After applying Algorithm 2, we can also apply Algorithm 1 to derive a safe task scheduling during this subinterval.

We also consider an intermediate scheduling first, in which the execution requirement of each task during each subinterval is equal to that of $\mathcal{S}^O$. Denote $t(\tau_{j,i})$ as the available execution time allocated to $\tau_{j,i}$ by Algorithm 2. if $|U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}]| \leq t(\tau_{j,i})$, a task's execution frequency does not need to be changed. However, if $|U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}]| > t(\tau_{j,i})$, we increase its execution frequency to $|U^O_{\tau_{j,i}} \cap [t_j, t_{j+1}]| f^O_{\tau_{j,i}}/t(\tau_{j,i})$. We denote this intermediate scheduling by $\mathcal{S}^{I_2}$.

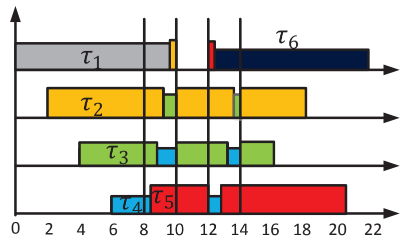
2) Final Scheduling of the DER-based Allocating Method: Similarly, we can design a final scheduling $\mathcal{S}^{F_2}$ based on $\mathcal{S}^{I_2}$. Note that, after applying Algorithm 2 for every heavily overlapped subinterval, the total available execution time for each task can also be easily calculated. Denote $A_i^{F_2}$ as the total available execution time for each task τ_i using the DER-based allocating method. To further optimize the frequency setting and energy consumption, while still meeting the execution requirement of each task, we can solve another optimization



(a)

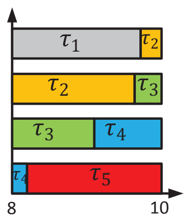


(b)

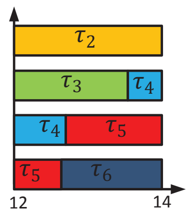


(c) Overall scheduling

Fig. 4. Illustration example. (a) The original task set. (b) Scheduling in the first subinterval. (c) Overall scheduling.



(a)



(b)

Fig. 5. Subinterval scheduling by the DER-based allocating method. (a) Scheduling in the first subinterval. (b) Scheduling in the second subinterval

problem similar to (22) and (23), with only $A_i^{F_1}$ replaced by $A_i^{F_2}$. The optimal frequency setting for this problem can also be easily calculated: $f_i = \max\{\sqrt[\alpha]{p_0/(\alpha-1)}, C_i/A_i^{F_2}\}$. Since $\mathcal{S}^{F_2}$ is further optimized based on $\mathcal{S}^{I_2}$, their energy consumption has the following relation: $E^{F_2} \leq E^{I_2}$.

D. Example

Look at an example of six tasks: $\tau_1 = (0,8,10)$, $\tau_2 = (2,14,18)$, $\tau_3 = (4,8,16)$, $\tau_4 = (6,4,14)$, $\tau_5 = (8,10,20)$, $\tau_6 = (12,6,22)$ as shown in Fig. 4(a). As has been defined, $\tau_i = (R_i, C_i, D_i), i = 1, 2, \cdots, 6$, R_i, C_i, and D_i represent the release time, execution requirement and deadline of task τ_i, respectively. We consider scheduling these tasks on a quad-core (4-core) processor, with each core's power consumption being $p(f) = f^3$.

With the release time and deadline at hand, we construct the subintervals. In this example, there are a total of 12 distinct values of R_i and D_i. Thus, we can construct 11 subintervals: $\{[t_j, t_{j+1}], j = 1, 2, \cdots, 11\}$, where $t_j = 2(j-1), \forall j = 1, \cdots, 12$. It is easy to notice that only during intervals $[8,10]$ and $[12,14]$, the number of overlapping tasks is greater than the number of cores. Thus, only intervals $[8,10]$ and $[12,14]$ are heavily overlapped subintervals. If we allocate the available execution time evenly among each interval's overlapping tasks, each overlapping tasks will be allocated $(4/5) \times 2 = 8/5$. Applying Algorithm 1, we can derive a safe scheduling during inter $[8,10]$. The scheduling is detailed in Fig. 4(b). The scheduling during interval $[12,14]$ is similar, and thus, is omitted. Final frequency settings for τ_1, τ_2, τ_3, τ_4 and τ_5 are $8/(8+8/5)$, $14/(12+16/5)$, $8/(8+16/5)$, $4/(4+16/5)$ and $10/(8+16/5)$, respectively. The final frequency setting for τ_6 is $6/(8+8/5)$. The overall scheduling based on this method is derived as Fig. 4(c). The overall energy consumption of $\mathcal{S}^{F_1}$ is 33.0642.

The optimal execution frequency for each task in $\mathcal{S}^O$ can be calculated as $f_{\tau_i}^O = C_i/(D_i - R_i)$. In this example, $f_{\tau_1}^O = 4/5$, $f_{\tau_2}^O = 7/8$, $f_{\tau_3}^O = 2/3$, $f_{\tau_4}^O = 1/2$, $f_{\tau_5}^O = 5/6$, $f_{\tau_6}^O = 3/5$. Thus, the desired execution requirements of tasks $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5 during interval $[8,10]$ are $8/5, 7/4, 4/3, 1$ and $5/3$, respectively. Applying Algorithm 2, we can determine the allocation time of tasks $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5, as $1.7415, 1.9048, 1.4512, 1.0884, 1.8141$. The scheduling during interval $[8,10]$ can be derived as Fig. 5(a). Similarly, the desired execution requirement of tasks $\tau_2, \tau_3, \tau_4, \tau_5$ and τ_6 during interval $[12,14]$ are $7/4, 4/3, 1, 5/3$ and $6/5$. Applying Algorithm 2 again, we can determine the allocation time of tasks $\tau_2, \tau_3, \tau_4, \tau_5$ and τ_6, as $2, 1.5385, 1.1538, 1.9231, 1.3846$. Also, we can derive the scheduling during interval $[12,14]$ as shown in Fig. 5(b). With the allocated execution times during interval $[8,10]$ and $[12,14]$, we can calculate the overall available execution time of each task. Thus, we can further optimize the execution frequency of each task. We omit the final scheduling of the second method, since it is a straightforward process. The overall energy consumption of $\mathcal{S}^{F_2}$ is 31.8362. We can see that allocating available execution times based on desired execution requirements will save more energy than evenly allocating available execution times.

VI. Experiments and Simulations

We design various numerical simulations to evaluate our proposed scheduling methods in this section. On the task side, the release times, deadlines, and execution requirements of all of the tasks can vary. What also matters is the total number of tasks. On the platform side, the power consumption characteristics, namely, the values of α and p_0, also have a significant influence. Another important parameter is the number of cores of the multi-core processor. Although there are many parameters that might influence the energy consumption of a scheduling, we notice that, it is not the absolute values that matter. Combined or comparative parameter values dictate the influence. Also, we need to consider the situations that are close to practical processor and task characteristics. With these considerations, we design our simulation settings as follows. We randomly generate tasks' release times on interval $[0, 200]$; the values are uniformly distributed. We generate tasks' execution requirements on interval $[10, 30]$; values are also uniformly distributed. Intuitively, a combined parameter of R_i, C_i, D_i: $C_i/(D_i - R_i)$, the intensity of a task, may have a significant influence. Thus, we first generate a random intensity value for τ_i, denoted by $intensity_i$, for which we choose its value less than or equal to 1, and then set the deadline of task τ_i as: $D_i = C_i/intensity_i + R_i$.

Recall that the reformulated convex optimization problem can be solved in polynomial time; denote the energy consumption of the optimal solution by $\bar{E}^O$. Thus, we normalize the energy consumption of each scheduling, divided by the optimal energy consumption $\bar{E}^O$. We denote the Normalized Energy

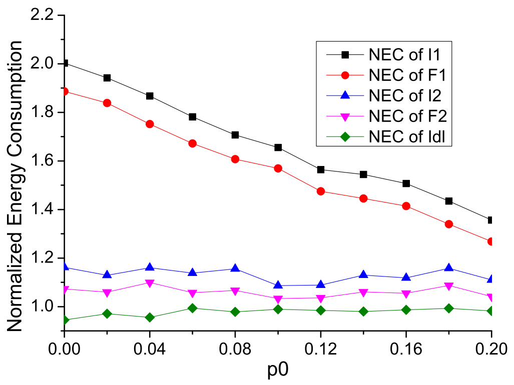


Fig. 6. Normalized energy consumption for various static power values.

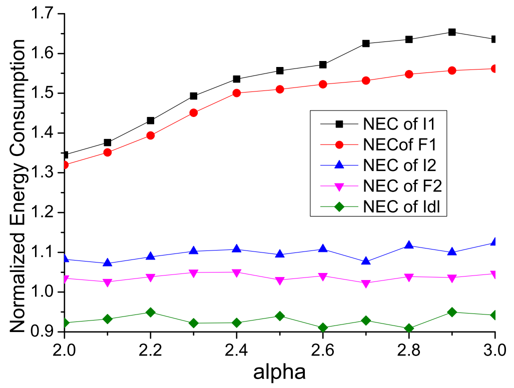


Fig. 7. Normalized energy consumption for various α values.

Consumption (NEC) of $\mathcal{S}^{I_1}$, $\mathcal{S}^{F_1}$, $\mathcal{S}^{I_2}$ and $\mathcal{S}^{F_2}$ by "NEC of I1", "NEC of F1", "NEC of I2" and "NEC of F2", respectively. We also normalize the optimal energy consumption of the ideal case, and denote $E^O/\bar{E}^O$ as "NEC of Idl". Notice that, due to the existence of static power, E^O may be greater than $\bar{E}^O$; however, for most cases, $E^O \leq \bar{E}^O$, as will be shown by various experiment results.

A. *Influence of Platform's Characteristics*

To investigate the performance of our scheduling algorithm on different platform characteristics, we choose three important parameters: the values of α, p_0, and the number of cores.

Considering the influences of α and p_0, we choose the number of cores fixed as $m=4$, which is a common configuration of modern multi-core processors. We generate $n=20$ tasks, with their intensities randomly choosing values from $[0.1, 0.2, \cdots, 1.0]$. To evaluate the influence of static power consumption, we fix $\alpha=3$, and vary the static power consumption as $\{0, 0.02, 0.04, \cdots, 0.20\}$. We run our algorithm on each setting 100 times and calculate the five average NEC values. The result is shown in Fig. 6. To evaluate the influence of the dynamic parameter α, we fix the static power consumption as $p_0 = 0$, and vary the values of α as $2.0, 2.1, \cdots, 3.0$. The result is shown in Fig. 7. We further run simulations for each pair of (α, p_0) values, where $\alpha \in \{2.0, 2.1, \cdots, 3.0\}$, and $p_0 = \{0, 0.02, 0.04, \cdots, 0.20\}$. The results are shown in Table II.

From Fig. 6 and Fig. 7, we can see that, the intermediate scheduling, $\mathcal{S}^{I_1}$ and its corresponding final scheduling, $\mathcal{S}^{F_1}$ have much greater energy consumption, especially when p_0 is low and α is high; the intermediate scheduling $\mathcal{S}^{I_2}$ and its corresponding final scheduling, $\mathcal{S}^{F_2}$, have more stable and

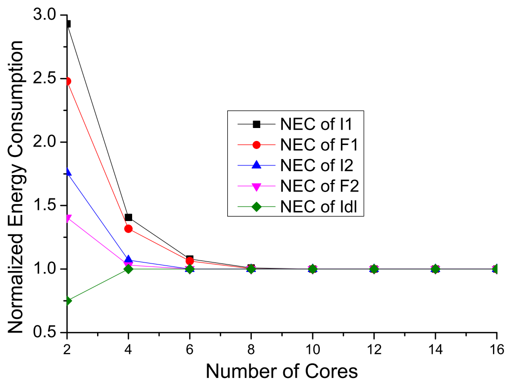


Fig. 8. Normalized energy consumption for various numbers of cores.

lower energy consumption. This demonstrates the advantage of the DER-based allocating method over the evenly allocating method. Compared to $\mathcal{S}^{I_2}$, $\mathcal{S}^{F_2}$ has a further reduced near-optimal energy consumption.

Also, from Table II, we can see that the normalized energy consumption of $\mathcal{S}^{F_2}$ remains at at a low level when the static power changes from 0 to 0.20. The reason for this lies in the smart aspect of the subinterval-based scheduling. For low static power, namely, $p_0 = 0$, the ideal optimal scheduling is to stretch all of the tasks during their interval $[R_i, D_i]$; our subinterval-based scheduling detects subintervals where the cores are heavily loaded and share some amount of the load with other lightly loaded subintervals. For high static power, even the optimal solution will not choose to stretch task executions as much as possible; our proposed approach adopts a frequency refining after allocating available execution times; thus, it will not lead to great increase of static energy consumption either. Consequently, in both cases, the normalized energy consumption remains at a satisfactorily low level. Besides, from Table II, we can see that the DER-based allocating method obviously outperforms the evenly allocating method. When the processor's static power is zero, the corresponding NEC of the DER-based method is about 1.1. When the processor's static power increases, NEC of the DER-based method decreases from around 1.1 to around 1.03, which demonstrates that the DER-based allocating method achieves near-optimal energy consumption.

To evaluate the influence of the total number of cores, we fix $\alpha = 3$, $p_0 = 0.2$, and vary the number of cores as $2, 4, 6, 8, 10, 12$. The results are shown in Fig. 8. Though, when the number of cores is 2, $\mathcal{S}^{F_2}$ has a worse-than-general performance, the NEC of $\mathcal{S}^{F_2}$ sharply reduces when the number of cores increases.

B. *Influence of Tasks' Characteristics*

To investigate the influence of tasks' characteristics, we fix the following values: number of cores, $m = 4$; dynamic power parameter, $\alpha = 3$; and static power, $p_0 = 0.2$. To investigate the influence of task intensity, we set the number of tasks, $n = 20$, and vary the intensity generation range from $\{[0.1, 1], [0.2, 1], \cdots, [1.0, 1.0]\}$. The result is shown in Figure 9. To investigate the influence of the number of tasks, we set the task intensity generation range as $[0.1, 1.0]$, and vary the number of tasks as $5, 15, 20, 25, 30, 35, 40$. The result is shown in Fig. 10.

In Fig. 9, the energy consumptions of $\mathcal{S}^{F_2}$ is quite stable when task intensity changes significantly, though other schedulings have significant fluctuations. From Fig. 10, we

TABLE II. NORMALIZED ENERGY CONSUMPTION OF THE FINAL SCHEDULING APPROACHES FOR VARIOUS (α, p_0) VALUES PAIRS

α	NECs	static power, p_0										
		0	0.02	0.04	0.06	0.08	0.1	0.12	0.14	0.16	0.18	0.2
2.0	NEC of F1	1.3607	1.3083	1.2468	1.1893	1.1798	1.1883	1.1881	1.1655	1.1340	1.1175	1.1268
	NEC of F2	1.0961	1.0751	1.0783	1.0527	1.0264	1.0391	1.0036	1.0227	1.0229	1.0093	1.0181
2.1	NEC of F1	1.3737	1.3488	1.3283	1.2090	1.2302	1.2288	1.2168	1.9566	1.1680	1.1422	1.1583
	NEC of F2	1.1014	1.0429	1.0256	1.0150	1.0101	1.0429	1.0036	1.0040	1.0016	1.0013	1.0256
2.2	NEC of F1	1.4387	1.4066	1.3244	1.3049	1.2600	1.2366	1.2391	1.2169	1.1847	1.1521	1.1644
	NEC of F2	1.0972	1.0709	1.0537	1.0450	1.0491	1.0409	1.0353	1.0438	1.0226	1.0324	1.0237
2.3	NEC of F1	1.4752	1.4579	1.3071	1.2309	1.2037	1.2579	1.2073	1.1805	1.1543	1.1651	1.1771
	NEC of F2	1.1020	1.1028	1.0977	1.0774	1.0530	1.0528	1.0557	1.0350	1.0436	1.0331	1.0377
2.4	NEC of F1	1.5270	1.5039	1.3690	1.2589	1.3124	1.3039	1.0852	1.0657	1.0572	1.0489	1.1990
	NEC of F2	1.1098	1.0995	1.0886	1.0623	1.0749	1.0595	1.0475	1.0469	1.0245	1.0340	1.0286
2.5	NEC of F1	1.5857	1.4862	1.3101	1.3647	1.2558	1.2862	1.2915	1.2708	1.2370	1.2495	1.2101
	NEC of F2	1.1110	1.0935	1.0879	1.0783	1.0520	1.0435	1.0479	1.0272	1.0356	1.0266	1.0279
2.6	NEC of F1	1.6968	1.6507	1.4671	1.5144	1.4193	1.3807	1.3580	1.4009	1.2906	1.3407	1.2671
	NEC of F2	1.1191	1.1002	1.0935	1.0771	1.0530	1.0602	1.0696	1.0785	1.0561	1.0357	1.0335
2.7	NEC of F1	1.7137	1.6625	1.4798	1.5038	1.3888	1.3625	1.3488	1.2856	1.3928	1.3014	1.2798
	NEC of F2	1.1167	1.0916	1.0875	1.0796	1.0656	1.0616	1.0639	1.0576	1.0492	1.0474	1.0375
2.8	NEC of F1	1.8496	1.7301	1.6569	1.6026	1.4532	1.4401	1.4350	1.4069	1.3520	1.3752	1.3269
	NEC of F2	1.1309	1.1183	1.0913	1.0766	1.0550	1.0583	1.0617	1.0406	1.0395	1.0470	1.0413
2.9	NEC of F1	1.9250	1.8763	1.8438	1.6508	1.5889	1.4763	1.4571	1.4794	1.4099	1.3952	1.3438
	NEC of F2	1.1245	1.1060	1.0956	1.0810	1.0723	1.0660	1.0605	1.0548	1.0435	1.0402	1.0356
3.0	NEC of F1	2.0214	1.5298	1.3848	1.2829	1.2076	1.5298	1.5164	1.4759	1.4360	1.4019	1.3848
	NEC of F2	1.1386	1.1208	1.0932	1.0731	1.0750	1.0688	1.0701	1.0531	1.0567	1.0477	1.0432

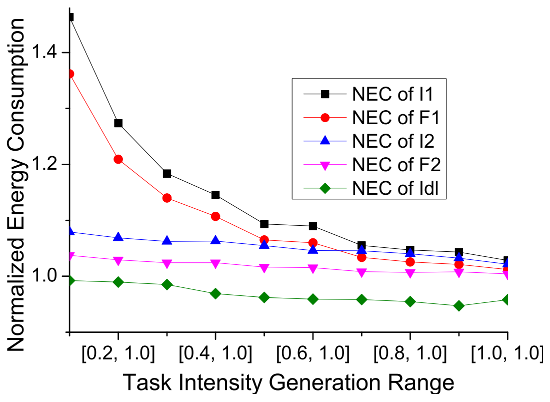


Fig. 9. Normalized energy consumption for various task intensity generation ranges.

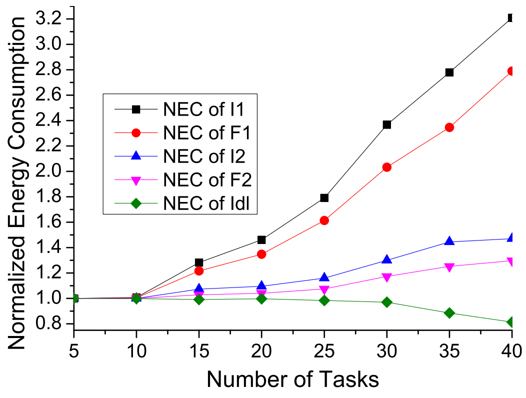


Fig. 10. Normalized energy consumption for various numbers of tasks.

can see that when the number of tasks increases, though the energy consumptions of schedulings, $\mathcal{S}^{I_1}$ and $\mathcal{S}^{F_1}$ increase significantly, the proposed scheduling, $\mathcal{S}^{F_2}$ will not. $\mathcal{S}^{F_2}$ still has a much better performance than $\mathcal{S}^{F_1}$.

C. Applying the Scheduling Method on a Practical Processor's Power Configuration

We also consider a multi-core processor with practical power configuration. We are aware that practical processing cores are only able to execute on a set of discrete frequency

TABLE III. FREQUENCY AND POWER CHARACTERISTICS OF THE INTEL XSCALE PROCESSOR

k	1	2	3	4	5
frequency, f_k (MHz)	150	400	600	800	1000
power, p_k (mW)	80	170	400	900	1600

values, instead of arbitrary continuous values. For a practical multi-core processor, we first apply the curve-fitting technique for the frequency and power characteristics using the form of $p(f) = \gamma f^{\alpha} + p_0$. We use the power characteristics of the Intel XScale processor, which is shown in Table III [1], as the power characteristics of a core on a quad-core processor. Since practical processors have similar power characteristics, we just choose Intel XScale as an representative. Applying the curve-fitting technique, we achieve a fitting function: $p(f) = 3.855 \times 10^{-6} f^{2.867} + 63.58$. Then, we apply our scheduling method and derive the frequency setting for each of the tasks, though these frequency values may not appear in the available frequency set of the core. After this, we round each derived frequency value to the closest higher frequency. Though other techniques that use both the closest lower frequency and the closest higher frequency can be used, we choose the simple rounding up strategy to show the advantage of our final practical scheduling against other scheduling methods.

For each task, we generate the tasks' execution requirement C_i within $[4000, 8000]$. Tasks' release times are uniformly generated between 0 and 200s. A reasonable deadline is chosen as, $D_i = R_i + C_i/(intensity_i \times f_2)$, where $f_2 = 400$(MHz) is the second available execution frequency. Task intensity is still within $[0.1, 1.0]$. The results are shown in Fig. 11, in which the practical scheduling based on $\mathcal{S}^{F_2}$ still has the best performance in terms of saving energy, and is also very close to the optimal energy consumption. Since $\mathcal{S}^{I_1}$ and $\mathcal{S}^{I_2}$ may require significantly increasing the execution frequency during heavily overlapped subintervals, their energy consumption may increase significantly. Besides, when a frequency higher than f_5 is required, tasks' deadlines may be missed. During experiments, we notice the probability of $\mathcal{S}^{I_1}$ and $\mathcal{S}^{I_2}$ missing deadline(s) is significant; the probability of $\mathcal{S}^{F_1}$ missing deadline(s) is non-negligible, while the probability of $\mathcal{S}^{F_2}$ missing deadline(s) is negligible.

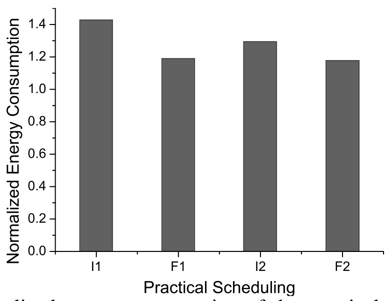


Fig. 11. Normalized energy consumption of the practical scheduling based on different approaches

D. Additional Remarks

So far, we assume using all of the cores available. However, we can choose how many cores to use before actual scheduling. Basically, before actually running the aperiodic task set, we can simulate the energy consumption of a scheduling that uses one core, then two cores, until the maximum number of cores. Among all these scheduling strategies, we choose the one that consumes the minimum amount of energy. In the practical execution, we use the scheduling with the minimum energy consumption and the corresponding number of cores. Also notice that both algorithms 1 and 2 are with low complexity, and obtaining the desired execution times and the final optimal frequency settings only needs several simple calculations. Thus, our overall scheduling algorithm is easy and suitable to be implemented in real-time systems.

VII. Conclusion

The energy-aware scheduling for general aperiodic tasks on multi-core processors is addressed. We formulate the problem on multi-core processors in a formal way, which shows that it is polynomial time solvable (though requiring high complexity), and helps us find that the key aspect of a solution/scheduling lies in how to allocate available execution times during a heavily overlapped subinterval. Instead of seeking optimal solutions with high complexity, we design a lightweight algorithm to solve the problem efficiently with good performance. Specifically, we allocate the available execution time for a task according to its desired execution requirement during a heavily overlapped subinterval, where the number of overlapping tasks is greater than the number of cores. We demonstrate the practical usage of the proposed algorithms by numerical simulations; also, we evaluate our scheduling method using a practical multi-core processor's power consumption characteristics. Results show that the lightweight algorithm can achieve near-optimal energy consumption in general cases. Besides, our proposed scheduling mechanisms are easy to be implemented in a practical system.

References

[1] Intel xscale microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[2] S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration: Extended abstract. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 279–288, 2011.

[3] 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*, pages 213–223, 2005.

[4] E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Proceedings of the 18th International Conference on Parallel Processing*, Euro-Par, 2012.

[5] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *13th Euromicro Conference on Real-Time Systems*, pages 225–232, 2001.

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

[7] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of International Parallel and Distributed Processing Symposium*, page 9, 2003.

[8] B. D. Bingham and M. R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 153–161, Dec. 2008.

[9] S. Boyd and L. Vandenberghe. Convex optimization. In *Cambridge University Press*, 2004.

[10] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, volume 1, pages 288–297, Jan. 1995.

[11] A. Chandrakasan, A. Burstein, and R. W. Brodersen. A low power chipset for portable multimedia applications. In *Solid-State Circuits Conference, 1994. Digest of Technical Papers., IEEE International*, pages 82–83, Feb. 1994.

[12] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 408–417, 2006.

[13] J.-J. Chen and C.-F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 28–38, Aug. 2007.

[14] J.-J. Chen, T.-W. Kuo, and C.-S. Shih. $1+\epsilon$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *Proceedings of the 5th ACM International conference on Embedded Software*, pages 247–250, 2005.

[15] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8–11, Oct. 1994.

[16] F. Kong, W. Yi, and Q. Deng. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Proceedings of Design, Automation Test in Europe Conference and Exhibition*, pages 1–6, March 2011.

[17] W. Y. Lee. Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors. In *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 216–223, 2009.

[18] V. Legout, M. Jan, and L. Pautet. A scheduling algorithm to reduce the static energy consumption of multiprocessor real-time systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 99–108, 2013.

[19] D. Li and J. Wu. Energy-aware scheduling on multiprocessor platforms. In *Springerbriefs on Computer Science*, Oct. 2012.

[20] K. Li. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *Computers, IEEE Transactions on*, 61(12):1668–1681, 2012.

[21] P. Mejia-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embed. Comput. Syst.*, 3(2):284–306, May 2004.

[22] V. Nelis and J. Goossens. Mora: An energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 210–215, August 2009.

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