J Comb Optim (2018) 35:860–894
https://doi.org/10.1007/s10878-017-0229-7

Race to idle or not: balancing the memory sleep time with DVS for energy minimization

Chenchen Fu[1] · **Vincent Chau**[1,2] · **Minming Li**[1] · **Chun Jason Xue**[1]

Published online: 22 December 2017
© Springer Science+Business Media, LLC, part of Springer Nature 2017

Abstract Reducing energy consumption is a critical problem in most of the computing systems today. Among all the computing system components, processor and memory are two significant energy consumers. Dynamic voltage scaling is typically applied to reduce processor energy while sleep mode is usually injected to trim memory's leakage energy. However, in the memory architecture with multiple cores sharing memory, in order to optimize the system-wide energy, these two classic techniques are difficult to be directly combined due to the complicated interactions. In this work, we explore the coordination of the multiple cores and the memory, and present systematic analysis for minimizing the system-wide energy based on different system models and task models. For tasks with common release time, optimal schemes are presented for the systems both with and without considering the static power of the cores. For agreeable deadline tasks, different dynamic programming-based optimal solutions are proposed for negligible and non-negligible static power of cores. For the general task model, this paper proposes a heuristic online algorithm. Furthermore, the scheme is extended to handle the problem when the transition overhead between the active and sleep modes is considered. The optimality of the proposed schemes for common release time and

✉ Vincent Chau
vincentchau@siat.ac.cn

Chenchen Fu
ameliafu1990@gmail.com

Minming Li
minming.li@cityu.edu.hk

Chun Jason Xue
jasonxue@cityu.edu.hk

1 Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China

2 Shenzhen Institutes of Advanced Technology, Shenzhen, China

agreeable deadline tasks are proved. The validity of the proposed heuristic scheme is evaluated through experiments. Experimental results confirm the superiority of the heuristic scheme in terms of the energy saving improvement compared to the most related existing work.

Keywords Schedule algorithm · Multi-core processor · Dynamic voltage scaling (DVS) · Energy efficiency · Main memory

1 Introduction

Energy efficiency is a critical issue in most of the computing environments nowadays. Among all components, processor and main memory typically dominate the energy consumption of computing devices. Different systems or different architectures may report different power consumer portions. But for most of them, the processor and memory are the top three power consumers, while the processor can consume as much as 50% energy consumption of the overall system (Ge et al. 2010; Wallace et al. 2013). Nowadays, the main memory is usually shared by multiple cores in servers, personal computers, and even embedded systems. A conventional and effective method to reduce the energy consumption of the multi-core processor is Dynamic Voltage Scaling (DVS). There are a series of works focusing on improving the energy savings of the multi-core processors by applying DVS (Albers et al. 2015; Angel et al. 2012; Bampis et al. 2014; Hanumaiah and Vrudhula 2014). However, the energy saving problem of the shared memory in the multi-core architecture still remains. In this work, by considering the interactions of the multi-core processor and the memory, we propose techniques to optimize the overall system-wide energy consumption.

Among the overall memory energy consumption, leakage power, including the refresh power and standby power, occupies a significant portion, as the memory chips are becoming denser with smaller technology scales. For example, in DRAM, which is widely used as main memory, leakage power can be as much as 10 times of the dynamic read/write power on the memory chip using a process technology with the size smaller than 50nm (Wilton and Jouppi 1996). Effectively reducing the leakage power can significantly improve the memory energy efficiency.

To reduce the leakage power, the memory can be transformed from the active state to the low power state, such as sleep state, power-down state, self-refresh state (low refresh rate), etc. when it is not accessed (Chen et al. 2011; Liu et al. 2012; Ware et al. 2010; Zhong and Xu 2008). In this work, we abuse the sleep mode to represent any low power state. This is legal because we do not really apply low power state when developing schemes in this paper, but try to maximize the time for low power state. Based on this technique, a series of research works have been done on different objectives. Most of these research works target at only reducing the memory energy consumption, or reducing the energy of single-core processor system with memory (Chen et al. 2011; Zhong and Xu 2007, 2008; Zhuo and Chakrabarti 2005). The system-wide energy saving problem in multi-core processor with shared memory remains blank. The main challenges of the system-wide energy minimization problem of considering both the memory leakage power and the multi-core processor power lie

in two aspects. On one hand, for the sake of cores, executing tasks in lower speed leads to less power consumption. For the memory, however, the processor speed slowdown may result in an increase of the memory accessing time, and thus may lead to the increase of the memory static power, which might be very significant. Hence, balance between the energy consumption of the cores and the memory needs to be achieved for the overall energy minimization. On the other hand, each core may have specific memory access pattern and the shared memory cannot sleep as long as any memory access exists. Consequently, it is the common idle time of all the cores that determines the sleep time of the shared memory, which is a different and new problem compared to the existing multi-core DVS scheduling schemes.

To handle the above challenges, this work proposes optimal solutions to minimize the system-wide energy consumption when the interactions of the multi-core processor and the memory are taken into account. To the best of our knowledge, this work is the first attempt to obtain the optimal solution in minimizing the system-wide energy consumption considering the multiple cores applying DVS and the shared memory.

In this paper, we conduct a systematic study of the system-wide energy minimization problem based on various system and task models. The goal is to schedule tasks among multiple independent DVS cores in proper speeds, while maximizing the time that the shared memory can sleep, so as to minimize the overall energy consumption. Both theoretical and practical techniques are proposed in this work. Experimental results show that the proposed online algorithm can effectively reduce the overall energy consumption compared to a state-of-the-art multi-core DVS scheduling scheme. The main contributions of this paper are:

- We proved that it is NP-hard to minimize the system-wide energy consumption when the number of cores is bounded by the number of tasks;
- When the number of cores is unbounded, for tasks with common release time, optimal schemes are proposed for two cases where cores have negligible and non-negligible static power, respectively;
- With unbounded number of cores, for agreeable deadline tasks (where later release time implies a later deadline) two DP-based (Dynamic Programming) optimal solutions are developed for negligible and non-negligible static power of cores, respectively;
- An online heuristic algorithm is proposed considering the general task model. The evaluation shows the effectiveness of the proposed algorithm;
- The energy overhead caused by mode transitions between active and sleep modes of the memory and the cores are further considered. With the transition overhead, the corresponding schemes over different models are presented.

The rest of this paper is organized as follows. The related work is presented in Sect. 2. Section 3 presents the definitions of the system model and the target problem. In Sect. 4, optimal schemes are proposed for common release time tasks. Optimal solutions for agreeable deadline tasks are presented in Sect. 5. Section 6 proposes an online heuristic algorithm for general tasks. In Sect. 7, the mode transition overhead is analyzed. The experimental results are shown in Sect. 8. Finally we conclude the paper in Sect. 9.

2 Related work

In this section, we introduce three groups of the most related works. First, the schemes focusing on minimizing the single core-based system-wide energy consumption are discussed. Second, we introduce DVS scheduling on multi-core processors. Third, the research works on the speed scaling with sleep state problem are presented.

Dynamic voltage scaling is a widely used energy management technique. Since the power consumption of a processor increases with the voltage of the processor increasing, energy can be saved by scaling the voltage of the processor. Researchers have explored the system-wide energy cost in the architecture consisting of a single DVS core and a single memory. Jejurikar and Gupta (2004) propose a heuristic algorithm for minimizing the energy consumption of the discrete speed level single-core processor system. The proposed method scales the speed of tasks to obtain the minimum system-wide energy. Zhuo and Chakrabarti (2005) study the continuous speed level single-core processor system, and propose a static speed setting policy, together with an online slack distribution scheme. Zhong and Xu (2008) propose to reduce the energy consumption of both the single processor and the memory. The authors developed an offline optimal schedule for periodic jobs, which can be computed in linear time. Furthermore, they propose a best possible online algorithm for sporadic jobs. Zhong and Xu (2007) propose the lower bounds and approximation algorithms, as well as some hardness results in minimizing system-wide energy.

DVS has been widely applied for multi-core processors in the recent decades. The first algorithmic work is given by Chen et al. (2004). For jobs with common release time and deadline, they propose approximation schemes for both with and without the maximum speed constraint. For processors where each core has independent voltage supply, Yang et al. (2005) propose a polynomial time optimal schedule for a given task assignment with common release time and deadline. Albers et al. (2007) prove the NP-hardness of the multi-core DVS problem when tasks cannot migrate and propose several approximation algorithms for various special cases. For jobs with a common release date or a common deadline, the approximation factor is $2(2-1/m)^{\alpha}$, where α is given from the power equation $P(s)=s^{\alpha}$. For arbitrary release dates and deadlines, a polynomial time algorithm is proposed with the approximation ratio of $\alpha^{\alpha}2^{4\alpha}$. Bingham and Greenstreet (2008) study the problem when tasks can migrate between cores and show that the optimal schedule can be obtained by Linear Programming, when the release time and deadline of tasks are arbitrary. To develop a faster optimal scheme towards the same problem, more recently, Albers et al. (2011, 2015) show that in offline setting, optimal schedules can be computed in polynomial time in a combinatorial way. In online setting, they extended two algorithms Optimal Available and Average Rate proposed by Yao et al. (1995) to the multiple processor case. They proved that Optimal Available is α^{α}-competitive, as in the single processor case and Average Rate is $(3\alpha)^{\alpha}/2+2^{\alpha}$. Angel et al. (2012) work on the same problem independently from the work (Albers et al. 2011; Bingham and Greenstreet 2008). The authors propose an optimal algorithm, which can be faster than that of (Albers et al. 2011) when the precision is not highly required.

More recently, several scheduling algorithms focusing on DVS while considering turning the processor to sleep state were proposed. This problem is called *speed scaling*

with sleep state, which was first formally defined and discussed in Irani et al. (2007). The main idea to handle the problem is to schedule tasks at an appropriate speed so as to create an idle period in which the processor can be switched into the sleep state. In this way, both static and dynamic energy consumptions of the processor can be reduced. The energy minimization over speed scaling with sleep state problem on a single-core processor is proved to be NP-hard for tree-structured tasks by Albers and Antoniadis (2012). The authors also propose the best possible lower bound $4/3-$ for the approximation factor in this problem. For the agreeable deadline tasks, Bampis et al. (2012) propose a dynamic programming-based algorithm with time complexity $\mathcal{O}(n^3)$. Most recently, Antoniadis and Huang (2015) settle the complexity of this problem to be weakly NP-hard by proposing a fully polynomial time algorithm. The approximation ratio is reduced from $4/3-$ to $1+\epsilon$. For multi-core processors, Chen et al. (2006) propose polynomial approximation algorithms for periodic tasks. In this work, by applying DVS on multi-core processors, maximizing the memory sleep time is a new problem. It is more complicated than the speed scaling with sleep state problem on multi-processors, because the idle interval of all cores should be adjusted to be close to each other so that the memory sleep time can be maximized. In this work, putting cores into sleep state is also taken into account.

3 Problem definition and analysis

In this section, system and task models that are used in this work are presented. Based on the models, the target problem is formally defined. Furthermore, the complexity analysis of the target problem when the number of cores is bounded is presented and proved.

System model This paper targets at the real-time systems and explores energy-efficient scheduling schemes for multiple homogeneous cores with shared main memory. We assume that each core has individual voltage supply, which can be dynamically scaled. For the systems with different voltage clusters, which allow a group of cores sharing one voltage supply island (Herbert and Marculescu 2007; Marculescu and Choudhary 2006), we leave them as future work.

In this paper, we assume that the speed of cores changes in a continuous manner. The assumption is conventional in the field of DVS task scheduling, the same as in the work (Angel et al. 2012; Yang et al. 2005; Yao et al. 1995; Zhong and Xu 2008). A series of works have been proposed to effectively transform continuous voltage to the most proper discrete voltage (Ishihara and Yasuura 1998). With these techniques and with the number of voltage levels increasing in recent years, there will be no big gap between the continuous voltage and discrete voltage. For the overhead of the voltage adjustment, we firstly assume that it can be ignored in the theoretical analysis. The ignoring of voltage transition overhead is not over-estimated. This is because in the theoretical analysis the proposed schemes do not allow preemptions and can guarantee that the speed of each task stays the same during execution. Hence the voltage adjustment is actually not quite frequent. Furthermore, we remove the assumption in the evaluation part. The experimental results show that the proposed

scheme can effectively reduce the overall system energy when as well considering the frequency transition overhead.

The dynamic power consumption $P_d()$ of the core is a function of the core speed s (Rabaey et al. 2002).

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

where $s = \kappa \frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd} and κ represent respectively the effective switch capacitance, the threshold voltage, the supply voltage and a hardware-design-specified parameter. Let s_{up} represent the upper bound speed level of the processor.

It can be noted that $P_d(s)$ is a convex and increasing function of the core speed s. As analyzed in Chen et al. (2006), Irani et al. (2007) and Yao et al. (1995), the dynamic power consumption $P_d()$ can be represented proportional to s^λ. In this work, we focus on homogeneous cores, which have the same power function:

$$P(s) = \alpha + P_d(s), \tag{1}$$

where $P_d(s) \propto s^\lambda = \beta s^\lambda$, $\lambda > 1$ Yao et al. (1995), and α denotes the static power of the core (Chen et al. 2006). If $\alpha = 0$, it means that the dynamic power dominates the core power consumption and the static power of cores is negligible (Mishra et al. 2003; Yang et al. 2005; Zhong and Xu 2008). Thus the cores do not consume energy when idle. On the other hand, if $\alpha \neq 0$, it implies that the cores consume energy even when no task is executing. Hence turning the cores into sleep state when idle is necessary. In the following analysis, we use these two cases, $\alpha \neq 0$ and $\alpha = 0$, to denote whether the cores need to be turned to sleep or not when idle, respectively.

The static power for the shared main memory is denoted by α_m, due to the leakage current. The memory can be turned to sleep when it is not accessed by any core to save the leakage energy. We assume the sleep and active mode transitions of the memory can be done instantly, but require extra energy overhead (Chen et al. 2006; Fan et al. 2001). Conventionally, the transition energy overhead is represented as *break-even time*, which is the length of the time interval where the memory working in the idle but active mode consumes the same energy as the transition overhead (including both of the active-to-sleep and sleep-to-active mode transitions). Let ξ_m represent the break-even time of the memory. Likewise, denote the transition overhead of the core as ξ, if $\alpha \neq 0$.

Task model Tasks, discussed in this work, are independent during executions. We assume that a task accesses the memory during its whole execution period (Zhong and Xu 2007, 2008). We focus on the non-preemption and non-migration case, i.e. tasks are not allowed to be preempted or migrate among cores once assigned (Yang et al. 2005; Mishra et al. 2003). Given a set of n tasks, $T_1, T_2, \ldots, T_n$, each task T_i is associated with release time r_i, deadline d_i, and non-negative workload w_i. All tasks must be completed before their deadlines. The time period $[r_i, d_i]$ of T_i, is called the *feasible region*, denoted as I_i. To clearly state the features of the proposed technique, we define a notation *filled speed* s_{fi} for each task T_i. s_{fi} represents the speed when T_i is executed to occupy the entire feasible region $[r_i, d_i]$, i.e. $s_{fi} = \frac{w_i}{|I_i|} = \frac{w_i}{d_i - r_i}$. Note that when $\alpha = 0$,

the core scheduling task in the filled speed consumes the least energy while satisfying the deadline constraint. W.l.o.g, we assume that each task can be completed before its deadline when scheduled at the upper bound speed, i.e. $s_{up} \geq s_{fi}, \forall i$, otherwise the problem will be meaningless.

In this work, we assume that each core is allocated a part of memory area, which is disjoint from each other. In this way, access congestions from different cores can be avoided. On the other hand, even though for the system of a traditional shared memory, which cannot be divided, many techniques have been proposed and applied to reduce the memory access delay under congestions (Jang and Pan 2011; Kim and Kim 2007). Furthermore, techniques like bank-level parallelism, the NUMA (non-uniform memory access) architecture, etc, guarantee the reasonable memory access delay. Therefore, even though tasks have the potential to be scheduled concentratively to access the memory in the target problem, we assume that the access delay can be ignored, as this work mainly focuses on task scheduling issues. The analysis on the memory access congestions and behaviors are left as future work.

Problem definition The goal of the explored problem in this work is to schedule tasks by applying DVS on each core while turning the core to sleep (when $\alpha \neq 0$) when necessary, and turning the memory to sleep when all cores are idle to minimize the system-wide energy consumption. In this work, we define *common idle time* as the time period when all cores are idle. It is equivalent to the sleep time of memory, denoted as Δ. In contrast, the *busy interval of core* is the maximal interval when the core is working, and the *busy interval of memory* means the maximal interval when at least one core is working. In the following, if not otherwise specified, *busy interval* refers to *busy interval of memory*.

Based on the above defined models, we define the target problem as Sleep and DVS-aware system-wide Energy Minimization (SDEM) problem. For this problem, we say a schedule is feasible if no task misses its deadline and a schedule is optimal if it leads to the least system-wide energy consumption among all feasible schedules.

In this work, SDEM is analyzed for both bounded and unbounded number of cores (Peter et al. 1997). Bounded means that the number of cores, denoted as C, is less than the number of tasks, i.e. $C < n$, while unbounded means that the number of cores is sufficient to schedule each task on an individual core, i.e. $C \geqslant n$. The following theorem proves that SDEM problem is NP-hard for the bounded case, even with the simplest task model and system model: all tasks have the same release time and deadline, and the static power of cores α, as well as the memory mode transition overhead ξ_m is negligible.

Our NP-hardness reduction is inspired by the reduction from Theorems 2 and 3 in (Albers et al. 2007), which uses the PARTITION and MULTI- PARTITION problem. Albers et al. (2007) proved that the energy minimization problem of scheduling tasks on multiple DVS cores is NP-hard for arbitrary release time and deadline tasks. However, there are some differences between their problem and ours. First, for their problem, there exists an optimal polynomial-time algorithm for the common release time and common deadline tasks. However, by considering the common idle time issue, the above Theorem 1 proves that SDEM remains NP-hard even with common time constraints. Second, their problem remains NP-hard when task is of unit-size but in our

problem, we did not consider the unit-size case and the hardness for this subcase is unknown.

Theorem 1 *SDEM is NP-hard when the number of cores* $2 \leqslant C < n$ *and* $C = n - n_x$, *where* $n_x = \omega(1)$ *is a super-constant, even for tasks with common release time and deadline,* $\alpha = 0$ *and* $\xi_m = 0$.

Proof We transform the NP-hard problem PARTITION to this problem. An instance of the PARTITION problem consists of a finite set A and a size $s(a) \in Z^+$ for each $a \in A$. The problem is to determine whether there is a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$.

Given an arbitrary instance of PARTITION, let $A = \{a_1, a_2, \ldots, a_n\}$, $s(a_1)$, $s(a_2), \ldots, s(a_n) \in Z^+$. Create an instance of our problem by constructing a set τ of n tasks, $\tau = \{T_1, T_2, \ldots, T_n\}$. We assume all the tasks have the same release time 0 and deadline D. The number of cores is 2 ($n >> 2$). Let the subset of tasks assigned and scheduled on core 1 be represented by τ', where $\tau' \subseteq \tau$. And the tasks which are scheduled on core 2 are in the subset $\tau - \tau'$. The system-wide energy consumption can be represented as

$$E_{sys} = \beta \left(\frac{\sum_{T_i \in \tau'} w_i}{|I_b|} \right)^{\lambda} |I_b| + \beta \left(\frac{\sum_{T_j \in \tau - \tau'} w_j}{|I_b|} \right)^{\lambda} |I_b| + \alpha_m |I_b|,$$

where I_b is the busy interval for each core (busy interval of memory). It is not hard to notice that the length of busy interval for each core is equal in the optimal solution. The busy interval length $|I_b|$ that minimizes E_{sys} can be obtained by derivation.

$$|I_b| = \left(\frac{(\lambda - 1)\beta \left[(\sum_{T_i \in \tau'} w_i)^{\lambda} + (\sum_{T_j \in \tau - \tau'} w_j)^{\lambda} \right]}{\alpha_m} \right)^{\frac{1}{\lambda}} \tag{2}$$

The corresponding minimum energy consumption is

$$E_{min\_sys} = \alpha_m^{\frac{\lambda - 1}{\lambda}} \beta^{\frac{1}{\lambda}} \lambda (\lambda - 1)^{\frac{1 - \lambda}{\lambda}} \left(\left(\sum_{T_i \in \tau'} w_i \right)^{\lambda} + \left(\sum_{T_j \in \tau - \tau'} w_j \right)^{\lambda} \right)^{\frac{1}{\lambda}} \tag{3}$$

For our instance, we assume that there exists a subset τ' that satisfies

$$\sum_{T_i \in \tau'} w_i = \sum_{T_j \in \tau - \tau'} w_j,$$

which leads to the minimum E_{min_sys}. The optimal solution is obtained if and only if we find the subset τ' that satisfies the above condition, or in other words, find a workload-balanced task assignment to two cores. If we find the solution to our problem, then it can be used to solve an arbitrary instance of the PARTITION problem by constructing

Table 1 Subproblems of SDEM based on various system and task models

Task model	System model	Solutions	Sections
Common release time	$\alpha = 0, \xi_m = 0$	Optimal solution ($\mathcal{O}(nlogn)$)	4.1
	$\alpha \neq 0, \xi_m = 0, \xi = 0$	Optimal solution ($\mathcal{O}(n^2)$)	4.2
Agreeable deadline	$\alpha = 0, \xi_m = 0$	DP-based optimal scheme ($\mathcal{O}(n^4 + n^2)$)	5.1
	$\alpha \neq 0, \xi_m = 0, \xi = 0$	DP-based optimal scheme ($\mathcal{O}(n^5 + n^2)$)	5.2
General model	$\alpha = 0(\alpha \neq 0), \xi_m = 0, \xi = 0$	Online heuristic algorithm	6
All task models	$\alpha = 0(\alpha \neq 0), \xi_m \neq 0, \xi \neq 0$	Extended from corresponding schemes	7

n tasks, and letting $A = \tau$, and $s(a_i) = w_i$. Therefore, our problem is NP-hard when $C = 2$.

When considering $C > 2$, the problem can be similarly reduced from 3- PARTITION and MULTI- PARTITION (Albers et al. 2007). It remains hard to find a workload-balanced partition to multiple cores for our problem, and the problem can only be optimal when the partition is workload-balanced. It should be noted that when the number of cores C is approaching to n, the problem is NOT NP-hard. For example, if $C = n - 1$, exactly two tasks have to share a core. It is trivial as we only need to test $O(n^2)$ possibilities to obtain the optimal solution. To eliminate the trivial cases, we need to introduce $n_x = \omega(1)$. When $C = n - n_x$, the time complexity is $O(n^{n_x}) = O(n^{\omega(1)}) = \omega(n^c)$, (where c is a constant) which is super-polynomial time, i.e. exponential time. In this work, we say that our problem is NP-hard when $2 \leq C < n$ and $C = n - n_x$, where $n_x = \omega(1)$. □

Even though Theorem 1 proves the NP-hardness of the problem, we find that SDEM is solvable in polynomial time when the number of cores is unbounded. Considering the complexity of NP-hard problems, this paper focuses on the case of sufficient number of cores ($C \geq n$), named as the unbounded case. There are a series of related works focusing on the unbounded case analysis in real-time system (Edwin Cheng et al. 2001; Feng et al. 2013; Khandekar et al. 2010; Peter et al. 1997). The unbounded case is meaningful for the real-time system. Consider the scenario that tasks, the number of which is more than the number of cores, burst at some time instant. For the real-time system, it is difficult to guarantee the schedulability no matter with which scheduling algorithm. Tasks cannot be scheduled at a speed as fast as possible due to the maximum frequency limitation. Hence the number of tasks in execution at any time instant is limited for the real-time system. We can assume that the overall number of tasks is much more than the number of cores during a long period of time. But within a short period of time, the unbounded case is common. The following analysis explores the optimal results for tasks with common release time and agreeable deadlines, and proposes a heuristic online scheme for general tasks. The subproblems of SDEM and the corresponding solutions analyzed in each section are summarized in Table 1.

4 Common release time tasks

This section explores solutions for SDEM problem with common release time tasks. Optimal solutions are proposed for both the $\alpha = 0$ and $\alpha \neq 0$ cases. In this section, we assume the transition overhead of the memory and the cores are both negligible. Further analysis considering the transition overhead is presented in Sect. 7.

4.1 $\alpha = 0$

In this subsection, we consider the case that the dynamic power dominates the core power consumption, and the cores do not need to be turned to sleep (i.e. $\alpha = 0$). For this case, a faster scheme is proposed, with time complexity of $\mathcal{O}(nlogn)$.

Given a set of tasks with common release time, index the tasks in the increasing order of their deadlines. Let $I_i = [0, d_i]$ represent the feasible region of each task T_i, and $I = I_n = [0, d_n]$ be the maximal interval. Different from the definition in Sect. 4.2, we use

$$\delta_i = d_n - d_i, \forall i \in [1, n-1]$$

to represent the time period right after each task's feasible region. The corresponding input task model and notations are given in the left part of Fig. 1.

Define each case under the assumption of $\delta_i \leqslant \Delta < \delta_{i-1}$ as Case i. Without violating the time constraints, tasks from T_1 to T_{i-1} should be scheduled in their own filled speed, while tasks from T_i to T_n are scheduled to finish at time $|I| - \Delta$. The task schedule for case i is presented in the right part of Fig. 1.

For case i, the system energy consumption can be represented as

$$E_i = \alpha_m(|I| - \Delta) + \beta \sum_{j=1}^{i-1} w_j^{\lambda} |I_j|^{1-\lambda} + \beta \sum_{k=i}^{n} w_k^{\lambda} (|I| - \Delta)^{1-\lambda}$$

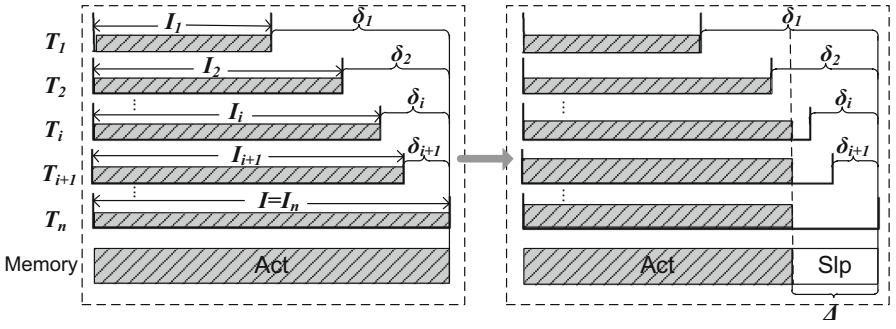

Fig. 1 Common release time task models with $\alpha = 0$

The extreme value that minimizes E_i can be obtained by derivation.

$$\Delta_{mi} = |I| - \left(\frac{\beta(\lambda - 1) \sum_{j=i}^{n} w_j^{\lambda}}{\alpha_m} \right)^{\frac{1}{\lambda}} \tag{4}$$

By examining whether Δ_{mi} falls inside or outside the feasible domain $[\delta_i, \delta_{i-1})$, we can obtain the local optimal solution for Case i, denoted as $E_{\min\_i}$. The formal equation for developing the local optimal solution is just the same as that in Lemma 2 (the notations should be replaced with the ones in this subsection).

In the following theorem, for a given task set, a binary search-based efficient scheme is proposed to find the global minimum energy

$$E_m = \min\{E_{\min\_i}\}, \quad \forall i \in [1, n].$$

To clearly describe the theorem, we call Δ_{mi} *valid* when $E_{\min\_i}$ chooses Δ_{mi} as its solution, and call Δ_{mi} *just-fit* when $E_{\min\_i}$ chooses δ_i. When δ_{i-1} is chosen as the solution to $E_{\min\_i}$, Δ_{mi} is called *invalid*. This can be referred to the corresponding results in Eq. (9).

Theorem 2 *The optimal solution can be obtained by going through all n cases from Case n to Case* 1. *For each Case i, the global optimal result is obtained when* Δ_{mi} *is valid or just-fit. When* Δ_{mi} *is invalid, the scheme goes to the next Case* $i - 1$, *and checks in the same way. Note that when some task's speed exceeds* s_{up} *in Case k, stop and report* Δ_{mk} *as the optimal solution.*

Proof First of all, it can be noted from Eq. (4) that for the same task set,

$$\Delta_{mi} > \Delta_{m(i-1)} \forall i \in [2, n] \tag{5}$$

In the following analysis, we prove this theorem by induction. Set a counter k to represent the case that is currently discussed, and initialize it to n. For the base case when $k = n$, if $\Delta_{mn} < \delta_{n-1}$, according to Eq. (5), it means that from Δ_{m1} to $\Delta_{m(n-1)}$, they are all smaller than δ_{n-1}, which implies that Δ_{mi} in each case is just-fit. According to the corresponding conditions showed in Eqs. (9), (10) and (11), it can be noted that under this case, $\forall i \in [2, k-1]$

$$E_{\min\_i} = E_i(\delta_i) < E_i(\delta_{i-1}) = E_{i-1}(\delta_{i-1}) = E_{\min\_(i-1)} \tag{6}$$

and

$$E_n(\Delta_{mn}) < E_n(\delta_{n-1}) = E_{\min\_(n-1)}$$

Therefore, the local optimal values in all the other cases cannot be less than $E_n(\Delta_{mn})$. Then the global minimum energy is obtained as $E_m = E_n(\Delta_{mn})$. Otherwise, when $\Delta_{mn} \geqslant \delta_{n-1}$, the other cases cannot be guaranteed to be smaller or larger than the

local optimal of Case n without further analysis. So we let $k = n - 1$ and go to the next case.

In the induction steps, we assume Case $k = i + 1$ satisfies Theorem 2 and the scheme goes to Case $k = i$, which implies that $\Delta_{m(i+1)}$ is invalid. For Case $k = i$, if Δ_{mi} is valid or just-fit, then all the Δ_{mk}, where $k \in [1, i-1]$, are just-fit and can never lead to less energy value than $E_i(\Delta_{mi})$ (it can be analyzed in the similar way by applying Eq. (6)). So the global optimal solution can be obtained accordingly. In this way, Case i is proved to satisfy Theorem 2 as well. For the worst case, the scheme goes to Case 1 at last, then it stops and obtains the global optimal result according to Eq. (10).

Next, to prove that it is feasible to stop checking the next case when the current solution is valid or just-fit, we prove that there is only one Δ that leads to the global minimum energy consumption. Assume that there are two solutions, $\Delta_{\min}$ and $\Delta'_{\min}$ that both lead to the optimal results. Without loss of generality, we assume $\delta_i \leqslant \Delta_{\min} < \delta_{i-1}$, $\delta_{i'} \leqslant \Delta'_{\min} < \delta_{i'-1}$, $\delta_{i-1} \leqslant \delta_{i'}$ and $E_i(\Delta_{\min}) = E_{i'}(\Delta'_{\min})$. It can be noted that $\Delta_{\min}$ (respectively $\Delta'_{\min}$) equals either Δ_{mi} (respectively $\Delta_{mi'}$) or δ_i (respectively $\delta_{i'}$) (Eq. (9)). If $\Delta_{\min} = \Delta_{mi}$, $\Delta'_{\min} = \Delta_{mi'}$, then we have $\Delta_{\min} > \Delta'_{\min}$ according to Eq. (5) (note that $i - 1 \geqslant i'$ from the assumption $\delta_{i-1} \leqslant \delta_{i'}$), which violates the assumption $\Delta_{\min} < \Delta'_{\min}$. If $\Delta_{\min} = \delta_i$, $\Delta'_{\min} = \delta_{i'}$, $E_i(\delta_i)$ cannot be equal to $E_{i'}(\delta_{i'})$ unless $\delta_i = \delta_{i'}$, which violates the assumption $\delta_{i-1} < \delta_{i'}$. We also can deduce that if $\Delta_{\min} = \delta_i$ and $\Delta'_{\min} = \Delta_{mi'}$, $E_i(\delta_i)$ cannot be equal to $E_{i'}(\Delta_{mi'})$, and vice versa. Hence the optimal solution is unique and the optimality is proved. The proof corresponds to the speed s_{up} is straightforward and is omitted here. □

The above optimal scheme can be further accelerated by applying binary search, as stated in Lemma 1. In this way, the time complexity can be improved to $\mathcal{O}(nlogn)$.

Lemma 1 *For each Case i, if Δ_{mi} is just-fit, then the binary search should go on the side of Case i to Case n; if Δ_{mi} is invalid, then the binary search repeats on the side of Case i to Case* 1. *Otherwise the valid Δ_{mi} is the optimal solution. Note that if the binary search ends without finding a valid Δ_{mi}, then the just-fit solution is the global optimal solution.*

Proof To prove the effectiveness of the binary search, the key point is to prove that the optimal solution is obtained when finding a valid solution in some case. In other words, if Δ_{mi} is valid in Case i, then other cases (to be more specifically, other cases from Case $i + 1$ to n), cannot have valid or just-fit solutions. For the sake of contradiction, assuming that for both Case i and Case j, Δ_{mi} and Δ_{mj} are valid. W.l.o.g, let $i > j$, then $\delta_i < \delta_j$, and $\Delta_{mi} < \Delta_{mj}$. However, according to Eq. (5), we have $\Delta_{mi} > \Delta_{mj}$, which leads to a contradiction. The contradiction still holds when Case i obtains the just-fit solution, and Case j obtains the valid result, with $i > j$. In this way, the unique valid solution is proved to be the global optimal solution. The binary search direction is guided by the cases of just-fit and invalid. The analysis is directly based on the feature of the original scheme, and thus is omitted here. □

4.2 $\alpha \neq 0$

In this subsection, we discuss a general problem, by considering that the static power of the core is non-negligible, i.e. $\alpha \neq 0$. It implies that each core can be independently turned into sleep state according to the completion time of the task loading on it. The memory can be turned into sleep state during the common idle time for all cores. In the following analysis, *critical speed* is firstly presented to guide the proposed scheme, and then the optimal solution is presented.

Critical speed Consider a system only consisting of a single core. By executing an arbitrary task T_i, the energy consumption of the core can be represented as $E_{core} = \beta s^{\lambda} \frac{w_i}{s} + \alpha \frac{w_i}{s}$. The minimum energy is obtained when the execution speed is $\sqrt[\lambda]{\frac{\alpha}{\beta(\lambda-1)}}$, denoted as s_m, which is independent from T_i (Irani et al. 2007). As each task cannot violate its time constraint or the upper bound speed level, we define the critical speed

$$s_0 = \min\{\max\{s_m, s_{fi}\}, s_{up}\}.$$

It can be noted that this guarantees that $s_{fi} \leq s_0 \leq s_{up}$, and thus tasks can always be feasibly scheduled when executing at critical speed s_0.

Given n tasks with common release time, without loss of generality, we assume that all tasks arrive at time 0 and each task has an individual deadline. Let these tasks be executed at the critical speed s_0. Index them in the increasing order of their completion time, denoted as c_i, where $c_i = \frac{w_i}{s_0}$. We set

$$\delta_i^{(\alpha)} = |I|^{(\alpha)} - c_i,$$

where the interval length $|I|^{(\alpha)} = |c_n|$. Denote the sleep length of memory as $\Delta^{(\alpha)}$. The input task model is given in the left part of Fig. 2.

Assume that the optimal solution is obtained when the memory sleeps for a length of $\Delta^{(\alpha)}$ in the right hand side of $I^{(\alpha)}$. It can be noted that the memory sleep time $\Delta^{(\alpha)}$ is the only determining factor to obtain the optimal solution. Intuitively, in the optimal

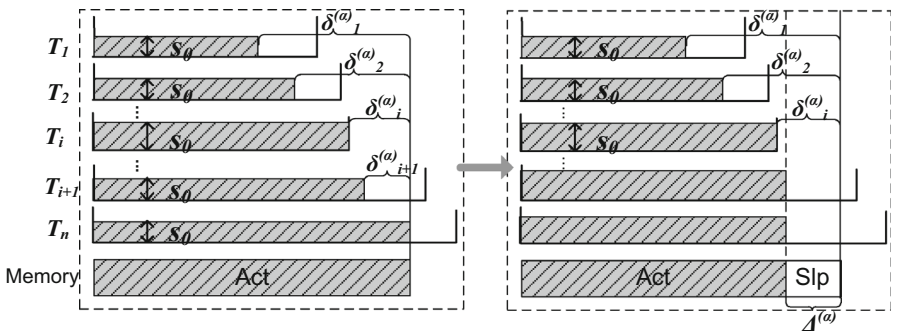

Fig. 2 Common release time task models with $\alpha \neq 0$

solution, tasks that satisfy $|I|^{(\alpha)} - c_i > \Delta^{(\alpha)}$ maintain the critical speed on their cores, while the other tasks with $|I|^{(\alpha)} - c_i \leqslant \Delta^{(\alpha)}$ need to increase their execution speed to align the sleep time of their cores to that of the memory to minimize the energy. The corresponding task schedule is presented in the right part of Fig. 2.

In the proposed scheme, we construct $n^{(\alpha)}$ cases based on the length of the $\Delta^{(\alpha)}$. For each case $i^{(\alpha)}$, let

$$\delta_i^{(\alpha)} \leqslant \Delta^{(\alpha)} < \delta_{i-1}^{(\alpha)}, \forall i \in [1, n^{(\alpha)}] \ \ (let\ \delta_n^{(\alpha)} = 0, \delta_0^{(\alpha)} = \infty).$$

For simplicity, we call $[\delta_i^{(\alpha)}, \delta_{i-1}^{(\alpha)})$ the feasible domain of $\Delta^{(\alpha)}$.

The following optimal scheme aims to obtain the best memory sleep time $\Delta^{(\alpha)}$ leading to the optimal solution. The system energy for case $i^{(\alpha)}$, excluding the cores loading tasks whose $|I|^{(\alpha)} - c_i > \Delta^{(\alpha)}$ is represented in Eq. (7). It can be used to obtain the optimal solution to the overall system (including all cores), because the tasks with $|I|^{(\alpha)} - c_i > \Delta^{(\alpha)}$ do not affect the optimal solution $\Delta^{(\alpha)}$.

$$E_i^{(\alpha)} = [(n - i + 1)\alpha + \alpha_m](|I|^{(\alpha)} - \Delta^{(\alpha)}) + \sum_{j=i}^{n} \beta w_i^{\lambda} (|I|^{(\alpha)} - \Delta^{(\alpha)})^{1-\lambda} \tag{7}$$

The extreme value $\Delta_{mi}^{(\alpha)}$ that leads to the minimum system energy can be obtained as

$$\Delta_{mi}^{(\alpha)} = |I|^{(\alpha)} - \left(\frac{\beta(\lambda - 1) \sum_{j=i}^{n} w_j^{\lambda}}{(n - i + 1)\alpha + \alpha_m} \right)^{\frac{1}{\lambda}} \tag{8}$$

Note that the extreme value $\Delta_{mi}^{(\alpha)}$ should fall inside the feasible domain $[\delta_i^{(\alpha)}, \delta_{i-1}^{(\alpha)})$ assumed for $\Delta^{(\alpha)}$. Otherwise the local optimal solution is achieved at one of the boundary values $\delta_i^{(\alpha)}$ and $\delta_{i-1}^{(\alpha)}$. In this way, the local optimal solution for Case $i^{(\alpha)}$ can be obtained, denoted as $\Delta_{opti}^{(\alpha)}$. In this local optimal solution, all tasks from T_n to T_i are executed to finish at $|I|^{(\alpha)} - \Delta_{opti}^{(\alpha)}$, and other tasks maintain the critical speed. Note that the speed of each task should not exceed the upper bound level s_{up}. This constraint is guaranteed in Theorem 3.

Without loss of generality, we denote each case under the assumption of $\delta_i^{(\alpha)} \leqslant \Delta^{(\alpha)} < \delta_{i-1}^{(\alpha)}$ as Case $i^{(\alpha)}$ with the local minimum energy consumption $E_{\min\_i}^{(\alpha)}$. The formal equation for developing the local optimal solution for each Case $i^{(\alpha)}$ is given in Lemma 2. Based on Lemma 2, Theorem 3 presents and proves the global energy minimization analysis.

Lemma 2 *The local minimum energy consumption $E_{\min\_i}^{(\alpha)}$ for Case $i^{(\alpha)}$ is obtained as*

$$E_{\min\_i}^{(\alpha)} = \begin{cases} E_i^{(\alpha)}(\Delta_{mi}^{(\alpha)}) & if\ \delta_i^{(\alpha)} \leqslant \Delta_{mi}^{(\alpha)} < \delta_{i-1}^{(\alpha)} \\ E_i^{(\alpha)}(\delta_i^{(\alpha)}) & if\ \Delta_{mi}^{(\alpha)} < \delta_i^{(\alpha)} \\ E_i^{(\alpha)}(\delta_{i-1}^{(\alpha)}) & if\ \Delta_{mi}^{(\alpha)} \geqslant \delta_{i-1}^{(\alpha)} \end{cases} \tag{9}$$

when $2 \leqslant i^{(\alpha)} \leqslant n-1$. *Specifically, when* $\Delta^{(\alpha)} \geqslant \delta_1^{(\alpha)}$,

$$E^{(\alpha)}_{\min\_1} = \begin{cases} E_1^{(\alpha)}(\Delta_{m1}^{(\alpha)}) & if\, \Delta_{m1}^{(\alpha)} \geqslant \delta_1^{(\alpha)} \\ E_1^{(\alpha)}(\delta_1^{(\alpha)}) & if\, \Delta_{m1}^{(\alpha)} < \delta_1^{(\alpha)} \end{cases} \tag{10}$$

and when $\Delta^{(\alpha)} < \delta_{n-1}^{(\alpha)}$,

$$E^{(\alpha)}_{\min\_n} = \begin{cases} E_n^{(\alpha)}(\Delta_{mn}^{(\alpha)}) & if\, \Delta_{mn}^{(\alpha)} < \delta_{n-1}^{(\alpha)} \\ E_n^{(\alpha)}(\delta_{n-1}^{(\alpha)}) & if\, \Delta_{mn}^{(\alpha)} \geqslant \delta_{n-1}^{(\alpha)} \end{cases} \tag{11}$$

The proof is straightforward and is omitted here. The following theorem presents that for a given task set, the global minimum energy can be obtained by going over these n cases.

Theorem 3 *The optimal solution can be obtained by going through all n cases as defined in Lemma 2 from Case* $n^{(\alpha)}$ *to Case* $1^{(\alpha)}$. *The local optimal result is recorded when* $\delta_i^{(\alpha)} \leqslant \Delta_{mi}^{(\alpha)} < \delta_{i-1}^{(\alpha)}$ *or* $\Delta_{mi}^{(\alpha)} < \delta_i^{(\alpha)}$. *When* $\Delta_{mi}^{(\alpha)} \geqslant \delta_{i-1}^{(\alpha)}$, *the scheme skips the recording and goes to the next case. The global optimal solution is obtained referring to the minimum value of all the* $n^{(\alpha)}$ *local optimal results. Note that when some task's speed exceeds* s_{up} *in Case* $k^{(\alpha)}$, *skip and go to the next case.*

The proof is straightforward and is omitted here. The time complexity of the above scheme is $\mathcal{O}(n^2)$. For the special case when all tasks have the common release time and common deadline, the global optimal solution can be directly obtained by applying Eqs. (7) and (8), while setting $i = 1$, as all tasks share the same feasible region. Furthermore, all the proposed schemes in Sect. 4 can be applied for heterogeneous cores with different power functions, i.e. different α, β. Under this case, different cores will have different critical speed s_0; and when developing the optimal system energy $E_i^{(\alpha)}$ in Eq. (7) for Case $i^{(\alpha)}$, the dynamic power of different cores should be added up separately.

5 Agreeable deadline tasks

In this section, the optimal solutions for SDEM problem with agreeable deadline tasks are explored. Tasks that have agreeable deadlines satisfy the following condition. For two arbitrary tasks T_i, T_j, if $r_i \geqslant r_j$, then $d_i \geqslant d_j$. In the following two subsections, SDEM problem for agreeable deadline tasks is analyzed for $\alpha = 0$ and $\alpha \neq 0$ cases, respectively. The transition overhead is discussed in Sect. 7.

5.1 $\alpha = 0$

In this subsection, without turning the cores into the sleep state, an optimal solution is proposed to minimize the system-wide energy consumption. The main idea of the solution can be divided into two parts. In the first part, the local optimal solution of

a single task subset is obtained. Let τ' represent an arbitrary subset of the whole task set, in which all tasks in τ' are scheduled in a single busy interval of memory. A local optimal solution can be obtained by finding a proper busy interval of memory that minimizes the energy consumption for these tasks. In the second part, for the whole task set, a Dynamic Programming based method can be constructed to determine the subset division, and the optimal solution can be developed accordingly. In the following, we describe the proposed scheme in detail.

5.1.1 Local optimal solution of one task subset ($\alpha = 0$)

Given a set τ of n tasks with agreeable deadlines, let τ' represent an arbitrary subset $\tau' \subseteq \tau$ consisting of n' tasks. Tasks in τ' are scheduled within a single, continuous busy interval. Note that the agreeable deadline property still holds in the subset. Index tasks in τ' in the increasing order of deadlines. The interval that is covered by these n' tasks can be represented as $[r_1, d_{n'}]$. W.l.o.g, we set $r_1 = 0$. Let s' and e' represent the start and end points of the busy interval. Define the idle time before and after the busy interval $[s', e']$ as Δ_1 and Δ_2, where $\Delta_1 = s'$ and $\Delta_2 = d_{n'} - e'$.

Based on the above definitions, the processing interval of a task T_k can be classified into one of the following 4 cases: (1) $[s', d_k]$; (2) $[r_k, d_k]$; (3) $[s', e']$ and (4) $[r_k, e']$. Figure 3 presents an example to show these 4 processing cases. Limited by the property of the agreeable deadline tasks, a task satisfying case (2) and another task satisfying case (3) cannot be scheduled in a single busy interval. Otherwise the two tasks satisfying case (2) and (3) respectively will be nested rather than satisfying the agreeable deadline property. According to the above observations, the detailed analysis is given as follows.

Assume that $s' \in (r_i, r_{i+1}]$, and $e' \in (d_{n'-j}, d_{n'-j+1}]$. We can obtain an (i, j) pair, where i means the ith task from left to right (from T_1 to T_i), and j denotes the j th task from right to left (from $T_{n'}$ to $T_{n'-j+1}$). Note that if two tasks have the same release time or the same deadline, s' or e' will become a specific time instant instead of falling inside a range, which makes the following procedure much simpler. In this section, we focus in the general case. The system energy consumption of the task subset can be represented given an (i, j) pair, as shown in Eqs. (12), (13), (14) in Lemma 3.

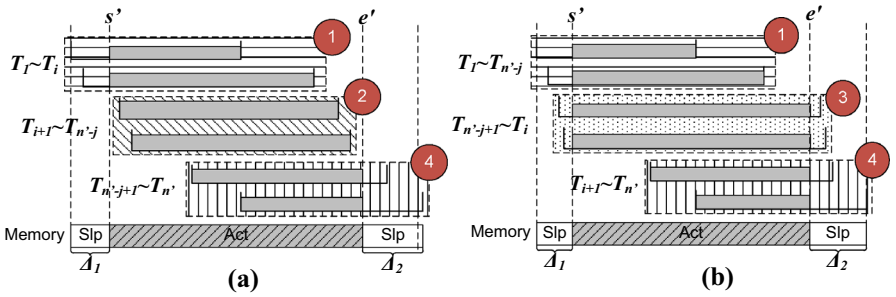

Fig. 3 Agreeable deadline tasks given an (i, j) pair. Case 1: $[s', d_k]$; Case 2: $[r_k, d_k]$; Case 3: $[s', e']$ and Case 4: $[r_k, e']$. **a** If $i < n' - j$. **b** If $i > n' - j$

In the following lemma, we first show how to develop the solution to minimize the energy consumption for each (i, j) pair. Then the optimal solution for the subset τ' can be obtained by finding the one leading to the minimum energy among all (i, j) pairs, accordingly.

Lemma 3 *When $\Delta_1 \neq 0$ and $\Delta_2 \neq 0$, given a pair of (i, j) for a task subset τ', if $i < n' - j$, then each task in τ' satisfies one of the three processing cases (1), (2) and (4), as shown in Fig. 3a. The system energy cost of tasks in τ' can be represented as:*

$$\begin{aligned} E_{i,j} = {} & \alpha_m(d_{n'} - \Delta_1 - \Delta_2) + \beta \textstyle\sum_{k=1}^{i} w_k^{\lambda}(d_k - \Delta_1)^{1-\lambda} \\ & + \beta \textstyle\sum_{k=i+1}^{n'-j} w_k^{\lambda}(d_k - r_k)^{1-\lambda} \\ & + \beta \textstyle\sum_{k=n'-j+1}^{n'} w_k^{\lambda}(d_{n'} - \Delta_2 - r_k)^{1-\lambda} \end{aligned} \tag{12}$$

If $i > n' - j$, then each task in τ' satisfies one of the three processing cases (1), (3) and (4), as shown in Fig. 3b, with the system energy consumption:

$$\begin{aligned} E_{i,j} = {} & \alpha_m(d_{n'} - \Delta_1 - \Delta_2) + \beta \textstyle\sum_{k=1}^{n'-j} w_k^{\lambda}(d_k - \Delta_1)^{1-\lambda} \\ & + \beta \textstyle\sum_{k=n'-j+1}^{i} w_k^{\lambda}(e' - s')^{1-\lambda} \\ & + \beta \textstyle\sum_{k=i+1}^{n'} w_k^{\lambda}(d_{n'} - \Delta_2 - r_k)^{1-\lambda} \end{aligned} \tag{13}$$

For the special case when $i = n' - j$, then each task in τ' satisfies one of the two processing cases (1) and (4).

$$\begin{aligned} E_{i,j} = {} & \alpha_m(d_{n'} - \Delta_1 - \Delta_2) + \beta \textstyle\sum_{k=1}^{i} w_k^{\lambda}(d_k - \Delta_1)^{1-\lambda} \\ & + \beta \textstyle\sum_{k=i+1}^{n'} w_k^{\lambda}(d_{n'} - \Delta_2 - r_k)^{1-\lambda} \end{aligned} \tag{14}$$

Note that when $\Delta_1 = 0$ or $\Delta_2 = 0$, the equations and the corresponding analysis are similar. The solution leading to the minimum energy consumption for each (i, j) pair can be obtained by finding the optimal solution $(\Delta_1^{(0)}, \Delta_2^{(0)})$ pair to the above equations. In the meantime, the solution is guaranteed to satisfy the constraints $r_i < \Delta_1^{(0)} \leqslant r_{i+1}$ and $d_{n'} - d_{n'-j+1} \leqslant \Delta_2^{(0)} < d_{n'} - d_{n'-j}$. Otherwise, the boundary points $r_i, r_{i+1}, d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j}$ should be set as the the minimum solutions instead.

For each subset τ', the optimal solution is the one leading to the minimum energy consumption among all the (i, j) pairs, with the constraints $s' = \Delta_1 \leqslant d_1$ and $e' = d_{n'} - \Delta_2 \geqslant r_n$.

The proof to Lemma 3 is straightforward and hence is omitted here. The only issue we need to verify is that the minimum solution does exist for each (i, j) pair, in the range of $\Delta_1 \in (r_i, r_{i+1}]$ and $\Delta_2 \in [d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j})$. We take Eq. (12) as an example. Equations (13) and (14) can be analyzed similarly.

To develop extreme values, let $\frac{\partial E_{i,j}}{\partial \Delta_1} = 0$ and $\frac{\partial E_{i,j}}{\partial \Delta_2} = 0$. We have

$$\sum_{k=1}^{i} \left(\frac{w_k}{d_k - \Delta_1} \right)^{\lambda} = \sum_{k=n'-j+1}^{n'} \left(\frac{w_k}{d_{n'} - r_k - \Delta_2} \right)^{\lambda} = \frac{\alpha_m}{\beta(\lambda - 1)}$$

The solution $(\Delta_1^{(0)}, \Delta_2^{(0)})$ pair to the above equation are the extreme values. For the second derivative, we find that $\frac{\partial^2 E_{i,j}}{\partial \Delta_1^2} > 0$, $\frac{\partial^2 E_{i,j}}{\partial \Delta_2^2} > 0$, and $\frac{\partial^2 E_{i,j}}{\partial \Delta_1 \partial \Delta_2} = 0$. Hence the solutions are extreme minimum values. When the minimum values are not in the range of $\Delta_1^{(0)} \in (r_i, r_{i+1}]$ and $\Delta_2^{(0)} \in [d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j})$, the boundary points $r_i, r_{i+1}, d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j}$, should be used instead to obtain the local minimum solutions.

In this way, by going over all the local optimal solutions for each (i, j) pair, we can obtain the minimum energy consumption, denoted as $E_{\min}^{\tau'}$ for any task subset τ'.

5.1.2 Dynamic programming-based subset division ($\alpha = 0$)

Now that the optimal solution for any arbitrary task subset can be developed, the global optimal solution for the whole task set depends on how to determine the subsets, in which tasks are executing in a single busy interval. In the following analysis, we call each busy interval, where a subset of tasks is scheduled, as the scheduling *block*. Firstly, a lemma is presented to identify the execution order of tasks in blocks. Then Dynamic Programming is applied based on the execution order to develop the global optimal solution.

Lemma 4 *Given n tasks sorted by their deadlines, there is an optimal solution, when $p < q$, $\forall p, q$, task T_p is not scheduled in the block after the block where task T_q is scheduled.*

Proof Assume that there is an arbitrary optimal solution, denoted as OPT, where tasks with earlier deadlines are scheduled in the block after the block where tasks with later deadlines are scheduled. In the following, we prove that this optimal solution can be transformed into a solution satisfying Lemma 4 while maintaining the optimality.

In the optimal solution, we assume task pair (T_p, T_q) is the first pair that violates the lemma conditions, based on the increasing order of deadlines. In other words, T_p is scheduled in the later block, while T_q is scheduled in the earlier block, with $p < q$. Let p_p and p_q represent the processing time of two tasks. If $p_p \geqslant p_q$, then task T_q can be moved forward to the later block, without affecting the schedulability, and leading to the same or less energy than the previous scheduling. Similarly, if $p_p < p_q$, then task T_p can be moved backward to be executed in the former block. All the task pairs that violate the lemma conditions can be transformed in this way. Hence we can draw the conclusion that there is an optimal solution, where any arbitrary task with earlier deadline is not scheduled in the block after the block where a task with later deadline is scheduled. □

According to Lemma 4, a Dynamic Programming based algorithm is designed on the order of increasing deadlines of tasks. Let $OPT(T_p)$ represent the global optimal solution for tasks from T_1 to T_p. Then

$$OPT(T_q) = \min_{\forall p \leqslant q} \left\{ OPT(T_p) + E_{\min}^{\{T_{p+1}, \ldots, T_q\}} \right\},$$

where $E_{\min}^{\{T_{p+1},\ldots,T_q\}}$ represents the minimum energy consumption for tasks from T_{p+1} to T_q, obtained based on the analysis in Sect. 5.1.1. By applying the proposed DP, the global optimal solution for tasks from T_1 to T_q can be divided into two parts. The first part includes tasks from T_1 to T_p that are scheduled in the optimal way. The second part consists of tasks from T_{p+1} to T_q, that are executed in the same block. The time complexity of the proposed Dynamic Programming based scheme is $\mathcal{O}(n^4)$.

5.2 $\alpha \neq 0$

In this subsection, the SDEM problem of the agreeable deadline tasks is analyzed when taking the static power of cores into consideration. The proposed scheme is constructed based on two parts, similar as that in $\alpha = 0$ case. The first part is to find the optimal solution to a subset of tasks scheduled in one block, and the second part is to construct the Dynamic Programming to divide the subsets and develop the global optimal result. In the following analysis, a new definition *memory-associated critical speed* is firstly presented to help guide the development of the scheme.

Memory-associated critical speed Consider a system consisting of a single core and the main memory. When executing an arbitrary task T_i, the energy consumption of the system can be represented as $E_{cm} = \beta s^{\lambda} \frac{w_i}{s} + (\alpha + \alpha_m) \frac{w_i}{s}$. The minimum energy is obtained when the execution speed is $\sqrt[\lambda]{\frac{\alpha+\alpha_m}{\beta(\lambda-1)}}$, denoted as s_{cm}. As a task cannot violate its deadline constraint or the upper bound speed level, we define the memory-associated critical speed

$$s_1 = \min\{\max\{s_{cm}, s_{fi}\}, s_{up}\}.$$

While s_1 leads to the minimum energy consumption of the system consisting of a single core and the memory, the critical speed s_0 is the optimal solution to the energy consumption of a single core. It is easy to note that $s_1 \geqslant s_0$. Based on the above definition, the detailed analysis is given as follows.

5.2.1 Local optimal solution of one task subset ($\alpha \neq 0$)

Given a set τ of n tasks with agreeable deadlines, let $\tau' \subseteq \tau$ represent an arbitrary subset consisting of n' tasks. Tasks in τ' are scheduled in a single busy interval. The notations s', e', Δ_1, Δ_2 and (i, j) pair used in the following analysis are the same as previously defined in the $\alpha = 0$ case.

According to the analysis for the common release time tasks when $\alpha \neq 0$, we know that in the final optimal solution, not all tasks are scheduled "aligned with" the busy interval of memory. Here, "aligned with" means that the execution period of a task has the same (i) start point, (ii) end point, or (iii) both start and end points, with the busy interval $[s', e']$ (Refer to the black rectangles in (i) Case 1, (ii) Case 4, and (iii) Case 3 in Fig. 4, respectively). Those tasks that are not "aligned with" the busy interval, are scheduled in the speed s_0. In the following, if not otherwise specified, the meaning of

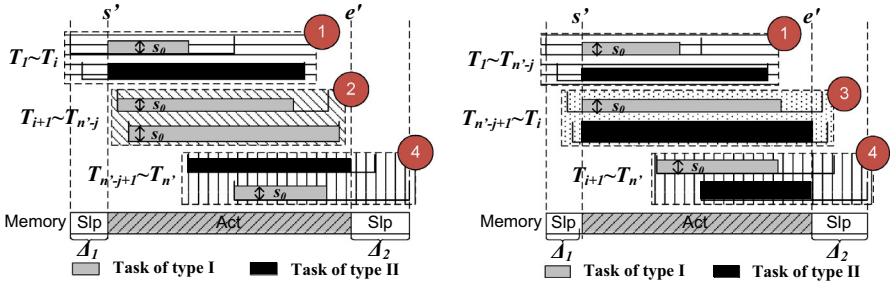

Fig. 4 When $\alpha \neq 0$, tasks with speed less than s_0 are scheduled at s_0 after Step 2. When tasks are scheduled at s_0, classify them into Type-I. When tasks are scheduled aligned with the busy interval $[s', e']$, denote them as Type-II. **a** If $i < n' - j$. **b** If $i > n' - j$

Table 2 Classification of tasks, and the corresponding properties in the optimal solution

	Type-I	Type-II
Speed	s_0	$[s_0, s_1]$
Execution period	Shorter than busy interval	Aligned with the busy interval
Property	Achieve the minimum energy for the cores loading Type-I tasks	Achieve the minimum energy for the memory and the cores loading Type-II tasks

aligned with the busy interval, without double quotations, is what we have explained above.

Based on the above knowledge, in the following analysis we classify tasks into two types: Type-I and Type-II, in terms of their execution cases in the optimal solution. Here, in Sect. 5.2.1, the optimal solution means the local optimal solution for each (i, j) pair. Tasks of Type-I execute at their critical speed s_0. Tasks of Type-II are scheduled aligned with the busy interval of memory. The execution period of the tasks of Type-I is covered by that of Type-II tasks, which determine the busy interval. Here, "execution period A (e.g. $[A_s, A_e]$) is covered by execution period B (e.g. $[B_s, B_e]$)" means that $A_s \geq B_s$, and $A_e \leq B_e$. Hence the main challenge of this problem is to identify the two types of tasks to develop the minimum energy consumption given an (i, j) pair. Table 2 summarizes the differences of these two types.

From Table 2, it can be noted that the speed of tasks of Type-II are within $[s_0, s_1]$. The reason is stated below. If any Type-II task T_i has speed larger than s_1 in the optimal solution, for the memory and the core who loads T_i, the energy consumption is reduced if we slow down T_i to approach the speed s_1. Now that T_i has been slowed down and has a longer execution period, for the other tasks of Type-II with speed in the range of $[s_0, s_1]$, we can further prolong them. This is legal because for these tasks, prolonging their processing time to approach s_0 can minimize the energy consumption of the cores loading them. In this way, considering both task T_i and the other tasks of Type-II, we can gain benefit by prolonging them, until all tasks with speed lager than s_1 are slowed down to execute within $[s_0, s_1]$.

Algorithm 1: For each (i, j) pair

while some task's speed is less than s_0 **do**
  Step 1: assuming all tasks are aligned with the busy interval, find the solution minimizes Eq. (15);
  Step 2: let tasks, whose speed are less than s_0 to execute at s_0;
  Step 3: evict the tasks with speed s_0;
  if the solution obtained in Step 1 violates the constraints **then**
    Set the boundary as the new solution, and quit;
  end if
end while
Temporary output: task subset $\tau_{ex\_s_0}$, where tasks have speed larger than s_0.
while some task's speed is larger than s_1 **do**
  Step 4: find a new solution minimizes Equation (15) just considering tasks with speed larger than s_1;
  Step 5: prolong other tasks' processing time (whose speed within $[s_0, s_1]$) to be aligned with the new busy interval. If some tasks' speeds fall below s_0 because of the prolonging. Set their speed as s_0, and evict them.
  if the solution obtained in Step 4 violates the constraints **then**
    Set the boundary as the new solution, and quit;
  end if
end while
Output: tasks that are classified into two types, with the local optimal solution for the given (i, j) pair;

In the following analysis, given an arbitrary (i, j) pair, we identify the two types of tasks in 5 iterative steps. The algorithmic flow and the details of the five steps are described in Algorithm 1 and the following paragraphs.

Step 1 Assume that all tasks are scheduled aligned with the busy interval. Similar to the $\alpha = 0$ case, each (i, j) pair can divide tasks into 4 processing cases: (1) $[s', d_k]$; (2) $[r_k, d_k]$; (3) $[s', e']$ and (4) $[r_k, e']$, with Case 2 and Case 3 not appearing simultaneously in one block.

When $\Delta_1 \neq 0$ and $\Delta_2 \neq 0$, if $i < n' - j$, the energy consumption for Case 1, 2 and 4 can be represented as

$$\begin{aligned} E_{i,j}^{(\alpha)} = {} & \alpha_m (d_{n'} - \Delta_1 - \Delta_2) \\ & + \beta \textstyle\sum_{k=1}^{i} w_k^{\lambda} (d_k - \Delta_1)^{1-\lambda} + \sum_{k=1}^{i} \alpha (d_k - \Delta_1) \\ & + \beta \textstyle\sum_{k=i+1}^{n-j'} w_k^{\lambda} (d_k - r_k)^{1-\lambda} + \sum_{k=i+1}^{n-j'} \alpha (d_k - r_k) \\ & + \beta \textstyle\sum_{k=n'-j+1}^{n'} w_k^{\lambda} (d_{n'} - \Delta_2 - r_k)^{1-\lambda} \\ & + \textstyle\sum_{k=n'-j+1}^{n'} \alpha (d_{n'} - \Delta_2 - r_k) \end{aligned} \tag{15}$$

When $i > n' - j$ ($i = n' - j$), the energy consumption for Case 1, 3 and 4 (Case 1 and 4) can be represented similarly, and is omitted here. A similar analysis can be developed when either $\Delta_1 = 0$ or $\Delta_2 = 0$.

According to the above equations, the temporary solution $(\Delta_1^{(\alpha)}, \Delta_2^{(\alpha)})$ leading to the minimum energy and the execution speed of all tasks can be obtained. Note that the solution should satisfy the constraints $r_i < \Delta_1^{(\alpha)} \leqslant r_{i+1}$ and $d_{n'} - d_{n'-j+1} \leqslant \Delta_2^{(\alpha)} < d_{n'} - d_{n'-j}$. Otherwise, the boundary points $r_i, r_{i+1}, d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j}$ should be set as the the minimum solutions instead. After Step 1, the tasks' scheduling states are just the same as shown in Fig. 3.

Step 2 For tasks with speed smaller than s_0, we accelerate them to s_0 to achieve the minimum energy cost, with no influence to other cores or the memory. For example, tasks in Case 2 can be scheduled at s_0 to minimize the energy consumption, which do not affect the scheduling of other tasks. After Step 2, tasks are scheduled as shown in Fig. 4.

Step 3 According to the above steps, now there are some tasks aligned with the busy interval, and the other tasks are scheduled at speed s_0. Tasks executing at s_0 are ignored in the following analysis, because they neither determine the busy interval nor affect the optimal solution.

After excluding several tasks with speed s_0, for the remaining tasks, a new busy interval that minimizes Eq. (15) can be obtained. In this new busy interval, the speed of some tasks might fall below s_0. Hence we should go to Step 1 to repeat the three steps until none of the tasks has speed lower than s_0. If the new solution falls outside the range of $\Delta_1^{(\alpha)} \in (r_i, r_{i+1}]$ and $\Delta_2^{(\alpha)} \in [d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j})$, set the boundary values as the local optimal solution for the given (i, j) pair and quit the algorithm.

After the above three iterative steps, now all the remaining tasks have speed larger than s_0. Subsequent steps only focus on these remaining tasks. Denote the task subset of the remaining tasks as $\tau_{ex\_s_0}$.

Step 4 In the remaining task subset $\tau_{ex\_s_0}$, some tasks may have speed within $[s_0, s_1]$, and some tasks may have speed larger than s_1. For tasks whose speeds are larger than s_1, we apply Eq. (15) to these tasks to develop an optimal solution, i.e. a new busy interval, which is longer than the previous one. The reason will be explained in Lemma 5.

Step 5 For the other tasks in $\tau_{ex\_s_0}$, i.e. tasks whose speeds are within $[s_0, s_1]$, prolong their processing time to be aligned with the new solution, which is developed in Step 4. Some tasks' speed might fall below s_0 because of the prolonging. Set their speed to s_0, and exclude them in the following analysis. Stop the algorithm if the solution violates the constraints $\Delta_1^{(\alpha)} \in (r_i, r_{i+1}]$ and $\Delta_2^{(\alpha)} \in [d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j})$. Repeat Step 4-5 until no task has speed exceeding s_1.

In this way, the proposed scheme ends with the optimal solution for an (i, j) pair, where some tasks execute at s_0, and the other tasks are aligned with the final busy interval.

The proposed scheme reduces the energy consumption with the processing of iterative steps. The first three steps obtain better solutions iteration by iteration. By evicting tasks with speed less than s_0, and setting their speed as s_0, the system energy consumption can be reduced. The latter two steps further reduce the system energy cost, which can be explained in terms of two aspects. On one hand, by developing a new busy interval for the tasks with speed larger than s_1, it is proved in Lemma 5 that the new busy interval is longer than the original busy interval when considering all tasks in $\tau_{ex\_s_0}$. With the speed of these tasks being smaller and approaching s_1, the energy cost for the memory and the cores loading these tasks is reduced. On the other hand, for the tasks whose speed are within $[s_0, s_1]$, it is better to prolong them to approach the speed s_0 to minimize the energy cost for the cores loading them. In this way, the system energy consumption is reduced step by step.

Specially, when the solution violates the constraints $\Delta_1^{(\alpha)} \in (r_i, r_{i+1}]$ and $\Delta_2^{(\alpha)} \in [d_{n'} - d_{n'-j+1}, d_{n'} - d_{n'-j})$, the reason that we can stop the algorithm is presented as follows. On one hand, the busy interval obtained in each iteration keeps being longer with the advance of the algorithm, which is proved in Lemma 5. Therefore, once in the current iteration, the solution has already violated the constraints, the following iterations can only be farther from the constraint boundary. On the other hand, even though we set the boundary values as the local optimal solution for one (i, j) pair, the potential that the tasks need a longer busy interval can be examined in other (i, j) pairs. In the following analysis, if not otherwise specified, the boundary cases are not discussed.

Recall that in the optimal solution, tasks should be classified into two types: Type-I and Type-II. To prove the optimality of the scheme, a lemma is firstly presented to guarantee that the proposed scheme can come to an end, with a division of two types of tasks.

Lemma 5 *In the proposed scheme, once a task is identified to execute at s_0 in some iteration, it will never be scheduled in a speed larger than s_0 in the following iterations. Finally, the scheme outputs tasks that are classified into two types.*

Proof The proof is conducted based on two parts. In the first part we explain Step 1–Step 3, and Step 4–Step 5 are analyzed in the second part.

For Step 1–Step 3, without loss of generality, we assume that in an arbitrary iteration ρ, there are m tasks scheduled aligned with the busy interval. Define the length of the busy interval of these tasks as $|I_b(\rho)| = e' - s'$. Among these m tasks, we assume that task T_k is executed with speed less than s_0. Step 2 then squeezes the processing time of T_k to make it scheduled at s_0. In the next iteration $\rho + 1$, for the remaining $m - 1$ tasks, Eq. (15) is applied to obtain the new busy interval length, denoted as $|I_b(\rho + 1)| = e'' - s''$. In the following, we prove that interval $I_b(\rho + 1)$ covers $I_b(\rho)$, and thus T_k will never need to be executed at a speed larger than s_0 in next iterations.

In iteration $\rho + 1$, the $m - 1$ tasks are scheduled aligned with the busy interval $I_b(\rho + 1)$. The energy consumption of $m - 1$ cores and the memory achieves the minimum value with $I_b(\rho + 1)$ as the solution, as long as assuming all $m - 1$ tasks are scheduled aligned with $I_b(\rho + 1)$. Similarly, the energy consumption of m cores and the memory is minimized under the assumption that m tasks are all scheduled aligned with the interval $I_b(\rho)$.

For the sake of contradiction, we assume $|I_b(\rho)| > |I_b(\rho + 1)|$. In iteration ρ, for task T_k, it is better to accelerate it to approach the speed s_0 for less energy consumption of core k. Besides, as $|I_b(\rho)| > |I_b(\rho + 1)|$, for the other $m - 1$ tasks, it is also better to squeeze their processing time to approach $|I_b(\rho + 1)|$. Because the solution $I_b(\rho + 1)$ leads to the minimum energy consumption for the $m - 1$ cores and the memory. Hence, all m tasks can be further squeezed to obtain less energy consumption, which violates the fact that $I_b(\rho)$ leads to the minimum energy consumption in iteration ρ. The above proof results in a contradiction and thus the assumption fails. In this way it is proved that $|I_b(\rho)| \leqslant |I_b(\rho + 1)|$, and it is straightforward that $I_b(\rho + 1)$ covers $I_b(\rho)$.

Similar analysis can be developed for Step 4 - Step 5. Each time when tasks with speed s_0 are excluded in the current iteration, the length of busy interval can only be prolonged. In this way, as the busy interval keeps being longer iteration by iteration,

once T_k is set in speed s_0 in some iteration, it never needs to be executed at a speed larger than s_0. By evicting tasks with speed s_0 in each iteration, the task set is reduced, and the algorithm can finally come to an end. All the evicted tasks can be eventually executed at speed s_0. □

The following lemma and theorem prove the optimality of the proposed scheme.

Lemma 6 *Given an optimal solution, we only consider the tasks of Type-II. Assume that for a subset of i arbitrary Type-II tasks, the busy interval that minimizes the energy consumption is $I_b^{(i)}$. When adding one more arbitrary task to this subset, the new solution leading to the minimum energy cost is $I_b^{(i+1)}$. There is $|I_b^{(i+1)}| > |I_b^{(i)}|$, for $\forall i \in [1, n]$.*

Proof When an arbitrary task T_1 is aligned with the busy interval, denoted as $I_b^{(1)}$, executing at s_1 leads to the minimum system energy cost. When considering another task T_2, to minimize the energy, both of T_1, T_2 should be scheduled within $[s_0, s_1]$, which determines a longer busy interval $I_b^{(2)}$. When introducing task T_3, we assume that the newly obtained busy interval $|I_b^{(3)}| < |I_b^{(2)}|$. Then for the memory and the cores loading T_1 and T_2, the energy cost can be reduced if we prolong $I_b^{(3)}$, and slow down T_1, T_2 to approach the solution $I_b^{(2)}$. Meanwhile, for the new comer T_3, it is also better to prolong its processing time to approach s_0. Hence $I_b^{(3)}$ should be longer, which violates the fact that it is optimal for $\{T_1, T_2, T_3\}$. The assumption fails, and thus $|I_b^{(3)}| \geqslant |I_b^{(2)}|$. Analysis is similar for more tasks.

In this way, the busy interval becomes longer when more Type-II tasks are involved. Besides, it is straightforward that the longer busy interval covers the shorter one, similar to $I_b(\rho + 1)$ covering $I_b(\rho)$ in Lemma 5. □

Theorem 4 *Given an arbitrary task subset τ', the optimal solution leading to the minimum energy consumption in a block can be obtained by the proposed scheme, as described in the following. For each (i, j) pair, obtain the optimal solution by the iterative five steps, and then choose the one leading to minimum energy consumption among all (i, j) pairs.*

Proof For each (i, j) pair, given an arbitrary optimal solution (denoted by OPT in the following), tasks can be classified into Type-I and Type-II. Denote the proposed scheme as SCM. According to Lemma 5, we denote all the evicted tasks with speed s_0 as tasks of Type-I $_{scm}$ in SCM, and the other tasks, who are aligned with the busy interval are of Type-II $_{scm}$. In the following, it is proved that all the Type-I $_{scm}$ tasks found in SCM are the Type-II tasks in OPT; and all the Type-II $_{scm}$ tasks are the Type-II tasks in OPT.

For the sake of contradiction, we assume some tasks are misclassified in SCM. For simplicity, we call them "wrong Type-I $_{scm}$ tasks" and "wrong Type-II $_{scm}$ tasks" to represent the tasks that should be of Type-II and Type-II, respectively. Let *Case A* represent that there are wrong Type-I $_{scm}$ tasks, and *Case B* denote there are wrong Type-II $_{scm}$ tasks in SCM, respectively. Figure 5 shows the tasks classification and the main idea of the proof. Note that in SCM, all tasks of Type-I $_{scm}$, which are correctly

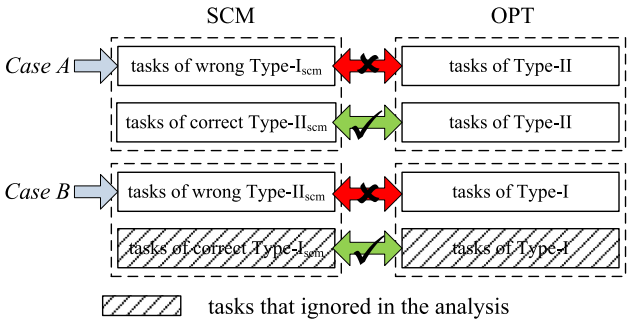

Fig. 5 Task classifications in SCM and OPT, which guides the proof of Theorem 4

classified can be ignored in the following analysis, as they do not affect the solutions. In the following, we prove the optimality based on the two cases.

If there is only Case A: tasks of wrong Type-I $_{scm}$ are executed in s_0 in SCM, while scheduled within $[s_0, s_1]$ in OPT. The number of Type-II tasks in OPT is more than the correct Type-II $_{scm}$ tasks in SCM. Denote the busy interval in SCM and OPT as b_{scm} and b_{opt}, respectively. Based on Lemma 6, we have

$$|b_{scm}| \leqslant |b_{opt}|$$

On the other hand, while the wrong Type-I $_{scm}$ tasks (speed within $[s_0, s_1]$) are aligned with busy interval b_{opt} in OPT, b_{scm} needs to cover a longer processing period of the wrong Type-I $_{scm}$ tasks (at speed s_0) in SCM. Therefore,

$$|b_{scm}| > |b_{opt}|$$

In other words, it is impossible for b_{scm}, which is shorter, to cover a longer processing period of the wrong Type-I $_{scm}$ tasks in SCM. This leads to a contradiction.
If there is only Case B: the proof is similar and is omitted here.

When the misclassification is bidirectional, i.e. both Case A and Case B exist: the proof can be conducted as follows. While the wrong Type-II $_{scm}$ tasks (speed within $[s_0, s_1]$) are aligned with busy interval b_{scm} in SCM, b_{opt} needs to cover a longer processing period of the wrong Type-II $_{scm}$ tasks (at speed s_0) in OPT. In this way,

$$|b_{scm}| < |b_{opt}|$$

However, while the wrong Type-I $_{scm}$ tasks (speed within $[s_0, s_1]$) are aligned with busy interval b_{opt} in OPT, b_{scm} needs to cover a longer processing period of the wrong Type-I $_{scm}$ tasks (at speed s_0) in SCM. Therefore,

$$|b_{scm}| > |b_{opt}|$$

Online algorithm (executes when a new task T_i arrives)

1: Record the time $t_i = r_i$;
2: Delete all the completed tasks before t_i, update the workload of all the existing tasks and reset their release time as t_i;
3: Obtain the optimal solution for all tasks and memory using the analysis in Section 4.1 (4.2), and record each task's corresponding execution time p_j;
4: Mark the latest execution point for each task T_j as $d_j - p_j$;
5: Keep the memory (and cores) in sleep state (from t_i), and wake up the memory when the first task meets its latest execution point;
6: All tasks begin to execute as long as the memory is waked up (wake up the core as long as the loaded task begins to execute).

This leads to a contradiction and the assumption fails. Finally, the optimal solution for the subset τ' is the one with the minimum energy consumption among all (i, j) pairs. □

5.2.2 Dynamic programming-based subset division ($\alpha \neq 0$)

The Dynamic Programming part is the same as $\alpha = 0$ case. To obtain the local optimal solution in each block requires $\mathcal{O}(n^3)$. The time complexity of the entire scheme is $\mathcal{O}(n^5)$.

6 Online algorithm for general tasks

The algorithms that have been presented in the previous sections are all optimal solutions for specified task models. In this section, we explore the SDEM problem and propose a solution in the realistic environment. Assume that tasks of the general model are scheduled in the online scenario, and preemption is allowed for tasks. Based on this model, an online heuristic algorithm is proposed.

The main idea of the algorithm is presented as follows. When a new task T_i arrives, run the algorithm, and set all unfinished tasks' release times the same as that of T_i. By applying the scheme presented in Sect. 4.1 (Sect. 4.2), the local optimal solution can be obtained for the current tasks. Keep the memory in the sleep state until a new task arrives or some task has to be executed to guarantee the local optimality. The online algorithm is applied for both cases of with and without considering the static power of cores.

A detailed algorithm description by an example is given as follows. W.l.o.g., assume the first task T_1 arrives at time 0. We calculate the optimal memory sleep time Δ_1^{on} as shown in Sect. 4.1 (4.2) but with the single task. Based on the optimal solution, the execution time of T_1 is $p_1 = d_1 - \Delta_1^{on}$. Considering that more tasks might come later, postponing the execution of T_1 is more likely to have a chance of obtaining a longer execution overlap with other tasks. Hence we keep the memory (and cores) in sleep state until T_1 meets its latest execution point $d_1 - p_1$. If the second task arrives before $d_1 - p_1$, the optimal solution should be re-calculated to deal with two tasks. The analysis in Sect. 4.1 (4.2) can be used to obtain the optimal solution Δ_2^{on}, and similar processes are followed to calculate the latest execution time of two tasks. Once one task meets its latest execution point, both tasks begin to execute and the memory

(and cores) will be waked up. In this way, anytime a new task arrives, we re-calculate the optimal solution for the existing tasks and keep the memory (and cores) sleep before the first-met latest execution time.

7 Transition overhead analysis

In this section, solutions for SDEM are analyzed when the energy overhead caused by transitions between active mode and sleep mode is not negligible, i.e. $\xi_m \neq 0$ and $\xi \neq 0$. Both optimal solutions for tasks with common release time, agreeable deadline, and the online heuristic algorithm can be modified to apply in the system with non-negligible mode transition overhead.

Common release time tasks A new notion *constrained critical speed* is firstly proposed to guide the modification of the scheme, and then the detailed analysis is presented.

Constrained critical speed When $\xi \neq 0$, $s_m = \sqrt[\lambda]{\frac{\alpha}{\beta(\lambda-1)}}$ is optimal only when $|I| - \frac{w_i}{s_m} \geqslant \xi$ (recall that $I = I_n = [0, d_n]$ represents the maximal interval of the given task set). Otherwise, the core consumes the least energy by executing T_i at s_{fi}. In the following, let s_c represent the constrained critical speed of a task T_i. Set

$$s_c = \min\{\max\{s_m, s_{fi}\}, s_{up}\},$$

when $|I| - \frac{w_i}{\min\{s_m, s_{up}\}} \geqslant \xi$, and $s_c = s_{fi}$ otherwise.

Given a set of tasks with common release time and each executing at the speed of s_c, index the tasks in the increasing order of their completion time $\frac{w_i}{s_c}$. The task execution model, together with n cases, can be constructed similar to Sect. 4.2. For each case, the system energy consumption function can be represented the same as in Eq. (7), because the transition overhead is independent from the memory sleep time, which means that it does not affect the optimal solution. Thus the local optimal memory sleep time that leads to the minimum $E_i^{(\alpha)}$ is the same as in Eq. (8). Each local optimal solution $\Delta_{mi}^{(\alpha)}$ classifies tasks into two types: Type-I tasks executing at speed s_c, and Type-II tasks executing aligned with memory busy time $|I| - \Delta_{mi}^{(\alpha)}$. For any task T_i of Type-I , if $|I| - c_i \geqslant \xi$, then the core can be directly turned to sleep after this task completion and the execution of T_i does not affect the memory behavior. In the following analysis, we ignore these tasks and the corresponding cores as their behaviors are fixed. Denote $\Delta_{mi}^{(\xi)}$ as the final local optimal memory sleep time for each case. Let $\Delta_{mi}^{(\alpha)} = \delta_i^{(\alpha)}$, when $\Delta_{mi}^{(\alpha)} < \delta_i^{(\alpha)}$.

Theorem 5 *The optimal scheme goes over n cases in the decreasing order. For each case, if $\Delta_{mi}^{(\alpha)} < \delta_{i-1}^{(\alpha)}$, the optimal memory sleep time can be obtained by referring to Table 3, which presents the relationships among $\Delta_{mi}^{(\alpha)}$ and ξ, ξ_m. Otherwise the scheme does nothing and enters the next case.*

We only focus on the proof of the third case, as it is the most complicated and other cases can be analyzed similarly. When $\xi_m \leqslant \Delta_{mi}^{(\alpha)} < \xi$, three subcases need

Table 3 Optimal results of $\Delta_{mi}^{(\xi)}$ based on different cases

Cases	Optimal results of $\Delta_{mi}^{(\xi)}$
$\Delta_{mi}^{(\alpha)} \geqslant \xi, \xi_m$	$\Delta_{mi}^{(\xi)} = \Delta_{mi}^{(\alpha)}$
$\xi \leqslant \Delta_{mi}^{(\alpha)} < \xi_m$	$\Delta_{mi}^{(\xi)} = 0$, all cores executing tasks at s_c
$\xi_m \leqslant \Delta_{mi}^{(\alpha)} < \xi$	$\Delta_{mi}^{(\xi)} =$ one of $\{\Delta_{mi}, \xi, 0\}$ that minimizes $E_i^{(\alpha)}$ (when $\Delta_{mi}^{(\xi)} = 0$, all cores executing tasks at s_c
$\Delta_{mi}^{(\alpha)} < \xi, \xi_m$	$\Delta_{mi}^{(\xi)} = 0$, all cores executing tasks at s_c

to be analyzed. (1) Turning the memory to sleep and keeping all cores active (idle but not sleep) all the time. Then the memory sleep time that minimizes the energy consumption is Δ_{mi}, which is defined in Eq. (4). If $\Delta_{mi} \geqslant \xi_m$, then Δ_{mi} leads to the local optimal solution for this case. Otherwise the memory should be kept active during the whole interval to minimize the energy consumption. (2) Turning both the cores and the memory to sleep state. For this subcase, it brings no benefit to keep the cores sleeping for less than ξ time, which wastes extra energy overhead. Besides, the energy consumption increases with the memory sleep time being larger than ξ, as the optimal memory sleep time is smaller than ξ. Hence, the local optimal solution $\Delta_{mi}^{(\xi)}$ should be set as ξ. (3) Keeping the memory active all the time and executing tasks at the speed of s_c, which means that the memory sleep time is 0. There are no fixed relationships among the above three subcases, hence the minimum energy consumption should be set as the minimum value of $\{E_i^{(\alpha)}(\Delta_{mi}), E_i^{(\alpha)}(\xi), E_i^{(\alpha)}(0)\}$.

Agreeable deadline tasks When the transition overhead is considered for the agreeable deadline tasks, the part of developing local optimal solution of a task subset does not need to be changed for both $\alpha = 0$ and $\alpha \neq 0$ cases. This is because the single busy interval of the task subset leads to one mode transition (including both the sleep-to-active and the active-to-sleep mode transitions), which is fixed. The Dynamic Programming part needs to be revised. Let $OPT(T_p)$ and $E_{\min}^{\{T_{p+1}, T_q\}}$ represent the same meanings as in Sect. 5 respectively. For both $\alpha = 0$ and $\alpha \neq 0$ case, we have the same DP function

$$OPT(T_q) = \min_{\forall p \leqslant q} \{OPT(T_p) + E_{\min}^{\{T_{p+1}, T_q\}} + \alpha_m \xi_m\}.$$

Online heuristic algorithm For the online heuristic solution with transition overhead considered, the main revision is briefly illustrated as follows. In each iteration, the local optimal solution is obtained by applying the scheme regarding the common release time tasks proposed in this section.

8 Evaluation

All the proposed schemes except the online heuristic algorithm, have been proved to be optimal in energy minimization. In this section, we evaluate the effectiveness

of the proposed online heuristic algorithm compared with another online multi-core DVS scheduling algorithm proposed in Albers et al. (2007), denoted as MBKP. MBKP achieves satisfying results among multiple DVS-cores in terms of energy saving, but does not consider the static processor power or the static memory cost. In the experiment, we compare the following algorithms:

(1) the proposed scheme, denoted as SDEM-ON;
(2) the original MBKP, which does not turn memory into sleep state;
(3) a modified MBKP approach, denoted as MBKPS, which is applied with a simple sleep transition scheme. This scheme turns the memory into sleep state whenever the memory has an idle time as they do not target at maximizing the idle time.

These algorithms are compared over different core utilizations (Zhong and Xu 2008; Zhuo and Chakrabarti 2005), memory static power settings (Zhuo and Chakrabarti 2005) and memory transition overheads.

8.1 Simulation setup

8.1.1 Benchmark

We firstly evaluate the scheme with two different benchmarks from DSPstone (Hsieh and Huang 2008): FFT and matrix multiply. A task is an instance of either of these two benchmarks (Pagani et al. 2015a, b). For the instance of FFT, the input is randomly generated 1024-point discrete signals. For the matrix, the input is the randomly constructed $[X \times Y]$, $[Y \times Z]$ matrices. The instance of each benchmark is released sporadically. The time interval between the release time and deadline is set as the processing time when the task instance executes at $16.5MHz$, where the values are collected based on the simulator xsim2101 of Analog Devices, provided in the DSPstone benchmark. The length of period is set to $|d_i - r_i| \times U$, where U is scaled in the range of [2, 3, 4, 5, 6, 7, 8, 9] to simulate different utilizations. It can be noted that larger U implies lower core utilizations.

8.1.2 Task set synthesis

To observe more features of the proposed scheme, besides the above two benchmarks, a large set of random generated tasks are conducted for evaluation. Randomly generating tasks is a common validation method in the area of DVS-simulation (Aydin et al. 2001; Chen et al. 2006; Yang et al. 2005; Zhong and Xu 2008; Zhuo and Chakrabarti 2005). Based on the real life task sets, the workload of a task is set randomly in the range of $[2, 5] \times 10^6$ cycles (Zhong and Xu 2007). The feasible regions of tasks are randomly set in the range of [10 ms, 120 ms] (Jejurikar and Gupta 2004).

Recall that the proposed online heuristic assumes that the number of cores is sufficient for scheduling tasks. This assumption, which is important for theoretical analysis, actually does not imply over-optimistic results in practice. The actual number of tasks in execution at a time is reasonable and limited in the real time system. Because if tasks, the number of which is more than the number of cores, burst at some time

Table 4 Parameter setting over various core utilizations ($1/x$), memory static power α_m and break-even time ξ_m

Point	1	2	3	4	5	6	7	8
x (ms)	100	200	300	400*	500	600	700	800
α_m (W)	1	2	3	4*	5	6	7	8
ξ_m (ms)	15	20	25	30	40*	50	60	70

instant, the real-time system will most likely fail to schedule the task set no matter with which scheduling algorithm. In the experiment, we set the number of cores to be 8, and assume that tasks that are executed at any time instant is less than 8. Let x represent the maximum inter-arrival time between two successive tasks. Assume that the first 8 tasks are assigned to 8 cores separately, and the 9th task will be assigned to the first core, so on and so forth. For a system with 100% utilization, all cores should be occupied all the time. Under this case, the inter-arrival time between the 1st task and the 9th task, which is at most $8x$, should be close to the processing length of a task. Considering that the processing time, which is determined by the execution speed, is the variable we try to optimize and cannot be estimated beforehand, we use $0.8\times$ *the task's feasible region length* instead. As the feasible regions of tasks are randomly generated in the range of [10 ms, 120 ms], we set $x = 120 * 0.8 \approx 100$ms for a high utilization system, which implies that all 8 cores are most likely to be used at any time, and range x from 100ms to 800 ms with a step size of 100 ms to evaluate results based on various utilizations. $x = 800$ ms implies that a single core might be sufficient to schedule all tasks.

8.1.3 System configuration

The system configurations are set based on the actual device specifications. The core we simulate in the following experiment is ARM Cortex-A57 (ARM 2013), where the power parameters are set based on ARM Cortex-A57 technical reference manual (ARM 2013) and the power consumption data collected by AnandTech (ARM 2012). We set the dynamic power parameter $\beta = 2.53 \times 10^{-7} \frac{\text{mW}}{\text{MHz}^3}$, static power $\alpha = 310$ mW, and $\lambda = 3$, referring to Eq. (1). The minimum and the maximum frequency of ARM Cortex-A57 is 700 and 1900 MHz, respectively. Set the number of the homogeneous cores to be 8.

The parameters of memory is modeled based on the 50nm DRAM, and are collected using CACTI (Fu et al. 2014; Wilton and Jouppi 1996). The static power of the memory is different with the memory size scales. We vary the memory static power α_m from 1W to 8W in steps of 1W. The break-even time of the memory is set based on metric and values shown in Fan et al. (2001). Let ξ_m vary in the range of [15, 20, 25, 30, 40, 50, 60, 70] ms. The detailed parameter setting is given in Table 4, where $*$ represents the default value of each parameter when evaluating other parameters. Note that when evaluating the benchmark based tasks, α_m and ξ_m are set as the default values in Table 4 as well.

This work mainly focuses on the static energy consumption of the memory, and does not aim to reduce the dynamic power, which is mainly caused by memory accesses.

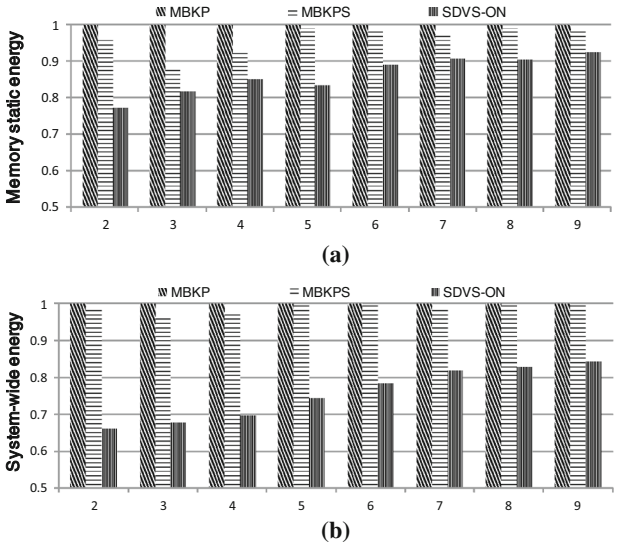

Fig. 6 **a** Memory static energy saving of FFT and matrix multiply benchmark results over different utilizations U. **b** System-wide energy saving of FFT and matrix multiply benchmark results over different utilizations U

Hence the system-wide energy consumption shown in the following simulation results only includes the energy cost of processor and the static energy consumption of memory.

8.2 Simulation result

In the experiment, the energy consumption results of SDEM-ON and MBKPS are compared to the output of MBKP, to show the energy saving over MBKP. The experimental results of the compared three algorithms based on the benchmark are shown in Fig. 6. All the schemes are scaled based on different utilizations $U = [2, 3, 4, 5, 6, 7, 8, 9]$ (recall that larger U implies lower utilization). Figure 7a, b shows the evaluations over the synthesis tasks, in terms of the system-wide energy saving improvement ratio. To generate convincing results, for each data point in all task sets, we randomly generate 10 different cases, and use the average value as the final evaluation result.

In Fig. 6a, it can be found that the proposed scheme SDEM-ON can turn the memory into sleep state for a longer period than MBKPS, by which, the static power of memory can be reduced. Besides, the memory energy saving improvement of SDEM-ON increases slightly with the utilization being lower. This indicated that the memory can save much more energy when the system is not busy. The average memory saving ratio is of SDEM-ON compared to MBKPS is 10.02%.

The system-wide energy saving is shown in Fig. 6b. The average system energy saving ratio of SDEM-ON compared to MBKPS is 23.45%. In Fig. 6b, SDEM-ON has a different trend with the scaling of utilizations compared to that in Fig. 6a. SDEM-ON performs better when the system has higher utilizations, in terms of the system-wide

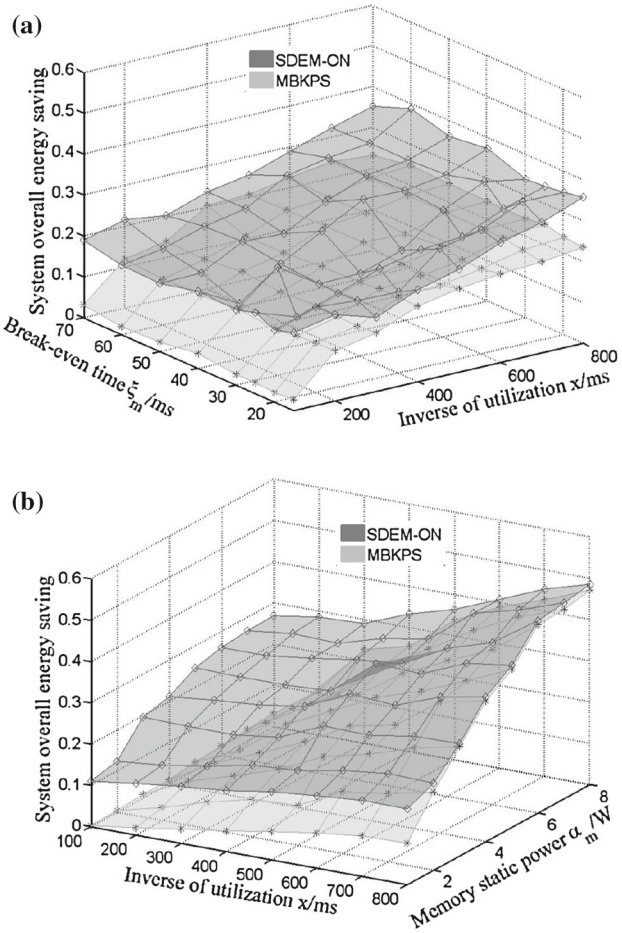

Fig. 7 **a** The system-wide energy saving improvement over different parameters over memory static power setting and core utilizations. **b** The system-wide energy saving improvement over different parameters over memory transition overhead and core utilizations

energy saving. This indicates that for the system with high and normal utilizations, SDEM-ON can obtain benefits for both the processor and memory. But when the system is extremely idle (when U is large), the potential in improving the processor energy saving is not much compared to MBKPS. This is because both of MBKPS and SDEM-ON are most likely to schedule tasks at lower speed when the utilization is low. The different between speeds is little.

Figure 7a shows the energy consumption reduction of the synthesis tasks, evaluated over different system utilizations, and memory static powers. The average energy saving improvement of SDEM-ON compared to MBKPS is 9.74% including the memory transition overhead. The evaluation results over different system utilizations and memory transition overhead are shown in Fig. 7(b). The average energy saving improvement of MBKPS is 10.52%.

MBKPS has close relationship with system utilizations as shown in Figs. 6b and 7a, b. For a system with high utilization ($x \to 100$ms or $U \to 2$), MBKPS can barely idle the memory, as it performs very close to the non-sleep version MBKP. SDEM-ON, on the contrary, can develop the proper balance between the speed and sleep mode for the system, and thus always outperforms MBKPS no matter how busy the system is. For the transition overhead evaluated in Fig. 7b, there is basically no difference with the varying of break-even time, for both of the schemes.

In conclusion, the proposed scheme SDEM-ON performs stably well over different system configurations and execution status. Compared with MBKPS, SDEM-ON is more proper to be applied in the real life.

9 Conclusion

In order to reduce the overall system energy consumption in a multi-core architecture, this paper proposes scheduling schemes to apply DVS on each core and maximize the memory sleep time, which is equal to the common idle time of all cores. When the number of cores is bounded, we prove the problem to be NP-hard even for tasks with common release time and deadline. Assuming that the number of cores is unbounded, optimal schemes are proposed for tasks with common release time, and tasks with agreeable deadlines. Furthermore, an online heuristic algorithm is developed for general tasks. Both theoretical and practical solutions for the target problem based on different system models are presented. Evaluations show that the proposed heuristic algorithm can effectively reduce the overall system energy consumption compared to a state-of-the-art work.

Acknowledgements The work described in this paper was partly supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117913] and [Project No. CityU 11278316].

References

Albers S, Antoniadis A (2012) Race to idle: new algorithms for speed scaling with a sleep state. In: SODA, pp 1266–1285

Albers S, Müller F, Schmelzer S (2007) Speed scaling on parallel processors. In: Proceedings of SPAA, pp 404–425

Albers S, Antoniadis A, Greiner G (2011) On multi-processor speed scaling with migration. In: Proceedings of the twenty-third annual ACM symposium on parallelism in algorithms and architectures. ACM, pp 279–288

Albers S, Antoniadis A, Greiner G (2015) On multi-processor speed scaling with migration. J Comput Syst Sci 81(7):1194–1209

Angel E, Bampis E, Kacem F, Letsios D (2012) Speed scaling on parallel processors with migration. In: European Conference on Parallel Processing. Springer, Berlin, pp 128–140

Antoniadis A, Huang CC, Ott S (2015) A fully polynomial-time approximation scheme for speed scaling with sleep state. In: SODA

ARM (2012) Arm a53/a57/t760 investigated by anandtech

ARM (2013) Arm cortex-a57 mpcore processor technical reference manual

Aydin H, Melhem R, Mossé D, Mejía-Alvarez P (2001) Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In: Proceedings of the 13th Euromicro conference on real-time systems, ECRTS, pp 225–232

Bampis E, Dürr C, Kacem F, Milis I (2012) Speed scaling with power down scheduling for agreeable deadlines. SUSCOM 2:184–189
Bampis E, Kononov A, Letsios D, Lucarelli G, Sviridenko M (2014) Energy efficient scheduling and routing via randomized rounding. arXiv preprint arXiv:1403.4991
Bingham B, Greenstreet M (2008) Energy optimal scheduling on multiprocessors with migration. In: ISPA, pp 153–161. https://doi.org/10.1109/ISPA.2008.128
Chen JJ, Hsu HR, Chuang KH, Yang CL, Pang AC, Kuo TW (2004) Multiprocessor energy-efficient scheduling with task migration considerations. In: ECRTS, pp 101–108. https://doi.org/10.1109/EMRTS.2004.1311011
Chen JJ, Hsu HR, Kuo TW (2006) Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In: RTAS, pp 408–417. https://doi.org/10.1109/RTAS.2006.25
Chen M, Wang X, Li X (2011) Coordinating processor and main memory for efficientserver power control. In: Proceedings of the international conference on Supercomputing. ACM, pp 130–140
Edwin Cheng T, Liu Z, Yu W (2001) Scheduling jobs with release dates and deadlines on a batch processing machine. IIE Trans 33(8):685–690
Fan X, Ellis C, Lebeck A (2001) Memory controller policies for dram power management. In: ISLPED, pp 129–134
Feng Q, Yuan J, Liu H, He C (2013) A note on two-agent scheduling on an unbounded parallel-batching machine with makespan and maximum lateness objectives. Appl Math Model 37(10):7071–7076
Fu C, Zhao M, Xue CJ, Orailoglu A (2014) Sleep-aware variable partitioning for energy-efficient hybrid pram and dram main memory. In: ISLPED, pp 75–80
Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW (2010) Powerpack: energy profiling and analysis of high-performance systems and applications. IEEE Trans Parallel Distrib Syst 21(5):658–671
Hanumaiah V, Vrudhula S (2014) Energy-efficient operation of multicore processors by DVFs, task migration, and active cooling. IEEE Trans Comput 63(2):349–360
Herbert S, Marculescu D (2007) Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In: ISLPED, pp 38–43
Hsieh MC, Huang CT (2008) An embedded infrastructure of debug and trace interface for the DSP platform. In: Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE. IEEE, pp 866–871
Irani S, Shukla S (2007) Gupta R (2007) Algorithms for power savings. ACM Trans Algorithms 3(4):41. https://doi.org/10.1145/1290672.1290678
Ishihara T, Yasuura H (1998) Voltage scheduling problem for dynamically variable voltage processors. In: 1998 international symposium on low power electronics and design, 1998. Proceedings. IEEE, pp 197–202
Jang W, Pan D (2011) Application-aware NoC design for efficient SDRAM access. TCAD 30(10):1521–1533
Jejurikar R, Gupta R (2004) Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In: ISLPED, pp 78–81. https://doi.org/10.1109/LPE.2004.1349313
Khandekar R, Schieber B, Shachnai H, Tamir T (2010) Minimizing busy time in multiple machine real-time scheduling. In: LIPIcs-Leibniz International Proceedings in Informatics, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, vol 8
Kim T, Kim J (2007) Integration of code scheduling, memory allocation, and array binding for memory-access optimization. TCAD 26(1):142–151
Liu S, Pattabiraman K, Moscibroda T, Zorn BG (2012) Flikker: saving dram refresh-power through critical data partitioning. ACM SIGPLAN Not 47(4):213–224
Marculescu D, Choudhary P (2006) Hardware based frequency/voltage control of voltage frequency island systems. In: CODES+ISSS, pp 34–39
Mishra R, Rastogi N, Zhu D, Mosse D, Melhem R (2003) Energy aware scheduling for distributed real-time systems. In: IPDPS, p 21. https://doi.org/10.1109/IPDPS.2003.1213099
Pagani S, Chen JJ, Henkel J (2015a) Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. TCAD 34(9):1415–1428. https://doi.org/10.1109/TCAD.2015.2406862
Pagani S, Chen JJ, Li M (2015b) Energy efficiency on multi-core architectures with multiple voltage islands. TPDS 26(6):1608–1621. https://doi.org/10.1109/TPDS.2014.2323260
Peter B, Andrei G, Han H, Mikhail KY, Chris P, Thomas T, van de Steef V (1997) Scheduling a batching machine. Eindhoven University of Technology. https://doi.org/10.1002/(SICI)1099-1425(199806)1:1<31::AID-JOS4>3.0.CO;2-R
Rabaey JM, Chandrakasan AP, Nikolic B (2002) Digital integrated circuits, vol 2

Wallace S, Vishwanath V, Coghlan S, Lan Z, Papka ME (2013) Measuring power consumption on IBM BLUE Gene/Q. In: 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). IEEE, pp 853–859

Ware M, Rajamani K, Floyd M, Brock B, Rubio JC, Rawson F, Carter JB (2010) Architecting for power management: the IBM® power^{TM} approach. In: HPCA-16 2010 the sixteenth international symposium on high-performance computer architecture. IEEE, pp 1–11

Wilton SJE, Jouppi N (1996) Cacti: an enhanced cache access and cycle time model. IEEE J Solid-State Circuits 31(5):677–688. https://doi.org/10.1109/4.509850

Yang CY, Chen JJ, Kuo TW (2005) An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In: DATE, pp 468–473. https://doi.org/10.1109/DATE.2005.51

Yao F, Demers A, Shenker S (1995) A scheduling model for reduced CPU energy. In: FOCS, pp 374–382. https://doi.org/10.1109/SFCS.1995.492493

Zhong X, Xu CZ (2007) Frequency-aware energy optimization for real-time periodic and aperiodic tasks. ACM SIGPLAN Not 42(7):21–30

Zhong X, Xu CZ (2008) System-wide energy minimization for real-time tasks: lower bound and approximation. TECS 7(3):28:1–28:24. https://doi.org/10.1145/1347375.1347381

Zhuo J, Chakrabarti C (2005) System-level energy-efficient dynamic task scheduling. In: DAC, pp 628–631. https://doi.org/10.1109/DAC.2005.193887