Theoretical Computer Science 866 (2021) 160–170

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Minimizing energy on homogeneous processors with shared memory ☆

Vincent Chau [a], Chi Kit Ken Fong [b], Shengxin Liu [c,*], Elaine Yinling Wang [d,e], Yong Zhang [d]

[a] School of Computer Science and Engineering, Southeast University, Nanjing, China
[b] Chu Hai College of Higher Education, Hong Kong, China
[c] School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China
[d] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
[e] School of Mathematical Sciences, Dalian University of Technology, Dalian, China

ARTICLE INFO

Article history:
Received 23 September 2020
Received in revised form 12 February 2021
Accepted 24 March 2021
Available online 26 March 2021

Keywords:
Energy
Scheduling
Shared memory
Approximation algorithm

ABSTRACT

Energy efficiency is a crucial desideratum in the design of computer systems, from small-sized mobile devices with limited battery to large scale data centers. In such computing systems, processors and memory are considered as two major power consumers among all the system components. One recent trend to reduce power consumption is using shared memory in multi-core systems, such architecture has become ubiquitous nowadays. However, implementing the energy-efficient methods to the multi-core processor and the shared memory separately is not trivial. In this work, we consider the energy-efficient task scheduling problem, which coordinates the power consumption of both the multi-core processor and the shared memory, especially focus on the general situation in which the number of tasks is more than the number of cores. We devise an approximation algorithm with guaranteed performance in the multiple cores system. We tackle the problem by first presenting an optimal algorithm when the assignment of tasks to cores is given. Then we propose an approximation assignment for the general task scheduling.

1. Introduction

Over the past decades, energy consumption is one of the major concerns in computer science. This attracts the attention of researchers in designing algorithms for minimizing energy consumption without performance degradation, which is referred to as energy efficiency problems.

One of the technologies used for energy efficiency is speed-scaling, where the processors are capable of varying its speed dynamically (or *Dynamic Voltage Scaling* (DVS)). The faster the processors run, the more energy they consume. The idea is to complete all the given tasks before their deadlines with the slowest possible speed to minimize energy usage. The energy minimization problem of scheduling n tasks with release times and deadlines on a single-core processor that can vary its speed dynamically where preemption is allowed has been first studied in the seminal paper by Yao et al. [2].

☆ A preliminary version of this paper appeared in Proceedings of FAW'2020, LNCS 12340, p 83–95 [1].

* Corresponding author.
E-mail address: sxliu@hit.edu.cn (S. Liu).

https://doi.org/10.1016/j.tcs.2021.03.030
0304-3975/

A lot of the previous research work in the literature focused on optimizing the power consumption of a single-core processor. However, in recent years, with the rapid development of data science and the demand for large-scale data analytics, parallel computing or distributed computing become a critical infrastructure in the market, that is a cluster with multiple processor cores, which can share the heavy workload to speed up the task executions. In such architecture, the main memory is usually shared among multiple processor cores. To minimize the power consumption of the cluster and distribute the resources efficiently, a scheduler is required to assign the tasks to each core simultaneously to utilize the computation resources fully. We refer to this as the system-wide energy consumption problem.

Fu et al. [3] were the first to study the problem of minimizing the scheduling problem with the consideration of both the multi-core processor power and the shared main memory power. The challenge of this problem lies in balancing these two conflicting power consumption: executing tasks at a lower speed leads to a lower processor power consumption but a higher memory power consumption. They focus on the case where the number of cores is unbounded, and tasks are available at the beginning.

In this paper, we consider the energy-efficient task scheduling problem for multiple homogeneous cores and the main memory shared by these cores. Our objective is to find a feasible schedule with minimum energy consumption. We concentrate on the scenario when the number of tasks is more than the number of cores. Fu et al. [3] already pointed out that the setting is NP-hard. Hence, we aim to design an approximation algorithm to obtain a near-optimal performance with a theoretical guarantee. We have made the following technical contributions in this paper:

- For completeness, we present the optimal algorithm when there is a single-core.
- We extend the intuition of the single-core case to the multi-core case, where the problem is tackled in two steps:
  - We first present an optimal polynomial-time algorithm when the assignment of tasks to cores is given;
  - We propose an algorithm to assign tasks to cores, and show that it is a constant approximation algorithm.

Related works. As a standard method to reduce the power consumption of the processor, DVS works by properly scaling the voltage of the processor and has been widely utilized for a single-core processor in recent decades. Yao et al. [2] were the first to give a polynomial-time algorithm to compute an optimal schedule. The time complexity has been further improved in [4] and [5]. More works along this line can be found in the surveys [6,7]. Albers et al. [8] proved the NP-hardness of the multi-core DVS problem when tasks could not migrate[1] and gave several approximation algorithms for various special cases. More recently, there also exist some works focusing on scheduling tasks at an appropriate speed to create an idle period in which the processor can be switched into the sleep state [9–13].

Besides speed-scaling, another way of reducing energy consumption is dynamic resource sleep (DRS) management, which powers off the machines when the machines do not have many tasks to process. For instance, the storage cluster in the data centers can be turned off to save energy during low utilization period. The min-gap strategy is one of the approaches in DRS. When the machines are idle, they are transited to the suspended state without any energy consumption. However, a small amount of energy will be consumed in the process of waking up the machines from the suspended state. Hence, the objective of the min-gap strategy is to find a schedule such that the number of idle periods can be minimized. However, if the wake-up cost is non negligible, i.e., it is more profitable to keep the machine in the active state instead of turning it off then turning it on again, then the problem aims to find a schedule with the minimum number of active periods. This problem has been first studied by Baptiste [14], and has been improved in [15]. More works along this line with various settings can be found in [16,17].

Researchers have also explored the system-wide energy-saving solution in the architecture consisting of a single DVS core and a single memory. Jejurikar and Gupta [18] proposed a heuristic algorithm for minimizing the energy consumption of the discrete speed level single-core processor system while Zhuo and Chakrabarti [19] considered the case of continuous speed. Furthermore, Zhong and Xu [20] proposed algorithms for periodic and sporadic tasks, aiming at reducing the energy consumption of both the single processor and the shared memory. In [21], Zhong and Xu also gave the lower bounds and approximation algorithms, as well as some hardness results in minimizing system-wide energy problems.

The work most related to ours is [3]. They studied the problem of task scheduling for multiple cores with shared memory. However, they only studied the case when the number of cores is larger than the number of tasks. They presented optimal solutions for different task models, such as all tasks are available at the beginning, or for the case where tasks have agreeable deadline, i.e., later release time implies later deadline. Finally, they tested the proposed algorithms with simulations under the condition when the system is reasonably loaded. However, if the system is overloaded (in particular when there are more tasks than the number of available cores), their algorithm cannot be used, and therefore, the primary motivation of this paper is to address this problem.

With the proliferation of computing devices, e.g., mobile devices and servers, energy efficiency is becoming one of the critical issues in the design of computer systems.

Especially, it is reported that the processor consumes as much as 50% energy consumption of the overall system [22]. At the same time, the main memory contributes to about 30-40% of total energy consumption in modern server systems [23].

[1] A task must be scheduled on a single core.

This difficult situation keeps attracting the attention of numerous researchers from both computer systems and theoretical computer science, aiming at proposing energy-efficient solutions for different computer architectures with processors and main memory. The recent trend of this line of research focuses on the computing environments equipped with the multi-core processor and the main memory shared by these cores, as such architecture has become ubiquitous in servers, personal computers, and even embedded systems.

For example, there are a series of works improving the energy savings of the multi-core processor by applying the standard method of Dynamic Voltage Scaling (DVS) [24–27].

On the other hand, to reduce the energy consumption of the main memory, previous studies target at maximizing the time period of sleep mode (which can be seen as a low power state, e.g., the power-down state and low refresh rate, when the memory is not accessed) [19–21].

Nevertheless, we should note that most previous research works either consider optimizing a single power consumption (i.e., multi-core processor power or memory power) or focus on the problem with the system provisioning single-core processor and memory.

They first showed that the problem is NP-hard if the number of cores is fewer than the number of tasks (and the number of cores is at least 2). Thus, to avoid facing the NP-hardness for the feasibility of the problem, they then turned their attention to the case when the number of cores is unbounded where several optimal solutions are proposed regarding different task models.

Paper organization. The rest of the paper is organized as follows. Section 2 describes the system model and presents the problem definition studied in this paper. In Sections 3 and 4, we consider the single-core and multi-core cases, respectively. Finally, we conclude the paper in Section 5.

2. Preliminaries

In this section, we present the system and task models studied throughout this paper. Then we formally define our problem based on these models.

2.1. System and task model

We consider the energy-efficient task scheduling problem for multiple homogeneous cores and the main memory shared by these cores. We assume that each core has an individual voltage supply and the speed of each core changes in a continuous fashion.

The *dynamic power consumption* $P_d(s)$ of the core is a function of the core speed s:

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

where $s = \kappa \frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, κ are used to represent the effective switch capacitance, the threshold voltage, the supply voltage and a hardware-design-specified parameter, respectively. Note that $P_d(s)$ is a convex and increasing function of the core speed s. As analyzed in [9,12], the *dynamic power consumption* $P_d(s)$ can be represented proportional to s^α. The power consumption of each core [9] is measured as $P_d(s) = \gamma + \beta s^\alpha$, where $\alpha > 1$, β are hardware-specified constants [2], and γ denotes the static power of the core and βs^α denotes the dynamic power. When $\gamma > 0$, it means that the cores consume energy even when they are not executing a task; Thus, these cores can be turned into the sleep state for energy saving. When a core is running at speed s for $t > 0$ units time, it can perform a total workload of $s \cdot t$ and consumes $t \cdot P(s)$ energy.

For the shared main memory power consumption, we consider the static power of the memory. That is, as long as one of the cores executes a task, the shared memory needs to be in the active state, which consumes γ_m energy per unit of time. We refer to the *static energy* when the energy consumption is due to the active state of the core or of the shared memory, while the *dynamic energy* refers to the energy consumption due to the execution speed of the cores.

Note that our algorithm can be generalized to the case in which using a new core incurs a (wake-up) cost. We only need to fix the number of used cores $p' \leq p$ and incorporate the cost into the objective function. Formally, let L be the wake-up cost of a core and let $p' \leq p$ be the number of used cores. Then, we compute the schedule with the minimum energy consumption using p' cores, which we add Lp'. Finally, we select the schedule with the minimum cost among all the possible number of cores by increasing the running time by a factor p.

We have a set of n tasks, $\{1, 2, \ldots, n\}$, where each task i is associated with a release time r_i, a deadline d_i, and a non-negative workload w_i. Without loss of generality, we suppose that tasks are sorted in non-decreasing order of their deadlines, i.e., $d_1 \leq d_2 \ldots \leq d_n$. A schedule is *feasible* if and only if all tasks are completed by their deadlines. In this work, we consider the case where tasks are available from the beginning, i.e., $r_i = 0, \forall i$. We also let $W_{i,j} = \sum_{k=i}^{j} w_k$. We use *makespan* to represent the maximum completion time, i.e., the last moment the cores are executing a task.

We summarize the notations used in this paper in Table 1.

Table 1
Notation summary.

n	number of tasks
p	number of cores
s	speed/frequency of a core
$s^\star$	critical speed/frequency of a core
α, β	parameters of a core used in the power function
γ	static energy rate of a core
γ_m	static energy rate of the shared memory
$P(\cdot)$	power function of a core
$W_{i,j}$	total workload of tasks $\{i, \ldots, j\}$
c_i	completion time of core i
Δ	length of sleep time
$B_{i,k}$	total workload of the k-th block of tasks on core i

2.2. Problem definition

In this paper, we consider the following problem. Given a set of n tasks that need to be scheduled on a system, as described above, where the system is associated with p cores and a shared memory. Our objective is to find a schedule with the minimum energy consumption such that the tasks are scheduled on a unique core, and such that they are completed before their deadline. Thus, we aim to minimize the following function:

$$E = \sum_{i=1}^{p} \left(\gamma c_i + \beta \int_0^{d_n} s_i(t)^\alpha \, dt \right) + \gamma_m \max_{i=1}^{p} \{c_i\} \tag{1}$$

where $s_i(t)$ is the running speed of the core i at time t, c_i is the completion time of the core i.

3. Warm-up: single core case

This section studies the single-core case in which we propose an algorithm that computes the optimal schedule. We first consider the situation where $\gamma_m = 0$, and we will discuss the case of $\gamma_m > 0$ subsequently. The proposed algorithm is a consequence of [10]. They studied the problem of task scheduling with a single speed-scalable core, with static energy and when tasks have agreeable deadline, i.e., $r_j \leq r_i \Leftrightarrow d_j \leq d_i$. The running complexity time of the algorithm is $O(n^3)$. We show in that when all tasks are released at time 0, and the running time becomes $O(n^2)$. The algorithm in [10] is based on the algorithm in [2] that computes the minimum energy consumption schedule by only considering the *dynamic energy*; In each iteration, we find the *maximum* intensity interval of tasks, which is the minimum speed such that the set of selected tasks must be scheduled in order to be completed before their respective deadlines. Such a speed can be calculated by dividing the sum of the workload of the tasks over the length of the interval. Finally, we remove the considered interval from the instance/schedule, and we repeat this procedure until all tasks are scheduled. In our case, the *static energy* has to be taken into account. We observe that, since the energy function $P(s)/s$ is convex, there exists a unique speed s^* such that a task with any other speed $s' \neq s^*$ will consume more energy. Formally, we have the following definition:

Definition 1 *([10,9]).* For the single core case, the critical speed is defined as: $s^\star = \arg\min_s \frac{P(s)}{s} = \sqrt[\alpha]{\frac{\gamma}{\beta(\alpha-1)}}$.

Since tasks are available at time 0, the considered interval at each step will be in the following form $[0, d_j]$ for some task j. After removing this interval, the next interval will be in the following form $[d_j, d_i]$ for some tasks $j < i$. Since at each step, we find the *maximum* intensity interval of tasks, it means that the future intervals will have lower intensity (task execution speed). We then have the following proposition.

Proposition 1. *The schedule returned by Algorithm 1 has a non-increasing running speed.*

The idea is to consider the tasks in groups where the speed of a group is calculated as the minimum speed for these tasks to ensure that they can be completed before their deadlines. According to Definition 1, tasks should be scheduled at speed $s^\star$ if it is possible. In particular, the strategy of speed selection is as follows:

- When the speed of a group of tasks is less than $s^\star$, we speed up the schedule in order to minimize the energy consumption.
- When the speed is larger than $s^\star$, it means that it is not possible to decrease the speed (to $s^\star$) because it will violate the deadline constraints.

Algorithm 1 Optimal algorithm for single core and common release time tasks.

1: Set $t \leftarrow 0$ and $i \leftarrow 1$.
2: Sort the set of tasks in non-decreasing order of their deadline
3: Compute the critical speed $s^\star \leftarrow \arg\min_s P(s)/s$.
4: **while** $i \leq n$ **do**
5: Compute $W_{i,j} \leftarrow \sum_{k=i}^{j} w_k,\ \forall i \leq j \leq n$.
6: Compute $j^\star \leftarrow \arg\max_{i \leq j \leq n} \frac{W_{i,j}}{d_j - t}$ and $s \leftarrow \frac{W_{i,j^\star}}{d_{j^\star} - t}$.
7: **if** $s \leq s^\star$ **then**
8: Schedule tasks $\{i, \ldots, j^\star\}$ at speed $s^\star$ until $d_{j^\star}$.
9: **else**
10: Schedule tasks $\{i, \ldots, j^\star\}$ at speed s until $d_{j^\star}$.
11: **end if**
12: Set $t \leftarrow d_{j^\star}$ and $i \leftarrow j^\star + 1$.
13: **end while**

By combining the algorithms from [10,2] and the observation of Definition 1, we obtain Algorithm 1. After initializing in lines 1-2, we compute the critical speed of s^* in line 3. Then we schedule the tasks in the while-loops (lines 4-13). For each iteration, we compute a group of consecutive tasks starting from the current task T_i with minimum speed s such that no deadlines are violated in lines 5-6. Then we schedule this group of tasks at speed s^* if $s \leq s^*$ (line 8); s, otherwise (line 10). The iteration keeps looping until we have scheduled all the tasks. The analysis of Algorithm 1 is shown as follows.

Theorem 1. *Algorithm 1 computes the optimal schedule in $O(n^2)$ time.*

Proof. This algorithm is a direct consequence of the algorithm in [10]. The idea is to compute the critical intervals. When the density of tasks in an interval is larger than $s^\star$, i.e., the ratio between the total workload of the tasks over the length of the interval, then the tasks must be scheduled with speed equal to the density. However, when the density is smaller than $s^\star$, then they must be scheduled with a speed of $s^\star$ because the energy consumption will be lower. For the running time, it is obvious that we have at most n while-loops (lines 4-13) since at least one task is scheduled in a while-loop (line 8 or 10). For each iteration, the algorithm requires $O(n)$ time to compute $W_{i,j},\ \forall i \leq j \leq n$ in line 5, to compute $j^\star$ in line 6, and to schedule tasks in either line 8 or 10. Thus, the total running time of Algorithm 1 is $O(n^2)$. □

We remark that Algorithm 1 can be applied to the case of $\gamma_m > 0$ by changing $s^\star \leftarrow \arg\min_s (P(s) + \gamma_m)/s$ in line 2. The optimality and the running time of the algorithm remain unchanged.

4. Multi-core case

In this section, we propose a polynomial-time approximation algorithm with a bounded performance guarantee. We first present an optimal algorithm when the task assignment is given in Section 4.1, then we propose an assignment algorithm that generates an approximate solution with a bounded ratio in Section 4.2.

4.1. Computing optimal schedule for a given task assignment

According to the assignment of the tasks, we compute the minimum energy consumption of each core with Algorithm 1. We aim to calculate the optimal length of the sleeping time of the shared memory, which is the length between the end of the schedule and the maximum deadline. We then divide the schedule into several intervals, which are delimited by different relevant completion times of each core. The rationale of this approach is that the energy consumption function is convex in the function of the memory sleeping time between two critical consecutive time points, which may not be the case overall. Thus, for our case, it is also crucial to find such a set of critical time points for separating the feasible domain of the memory sleeping time. We first consider the single-core case and observe the behavior of energy consumption when we increase the memory sleeping time.

Given a schedule on core i, by Proposition 1, let $s_1 > s_2 > \ldots > s_{k_i}$ be the set of different speeds. We also let $\mathcal{Z}_k$ be the set of tasks that are scheduled at speed s_k, i.e., $j \in \mathcal{Z}_k$ if and only if task j is scheduled at speed s_k. We refer to the set of tasks running at the same speed as a *block of tasks*. Because of the convexity of the power function, each task is scheduled at a constant speed. Thus there are at most n different speeds in an optimal solution, and in particular, there are at most n blocks of tasks.

A first observation is that when we increase the length of the sleeping time, only the execution speed of the last block of tasks increases. This comes from the fact that the last block has the slowest speed among all blocks of tasks, and increasing the execution speed of another block of tasks will incur a higher energy consumption. Therefore, we only need to find out how far we can increase the sleeping time until the last block's speed reaches the second last block's speed. When the execution speed of the last block is equal to the speed of the second last block, they are merged into a new block. See Fig. 1 for an illustration of the different completion times. In the first schedule, we increase the speed of the last block of tasks until the execution speed is equal to the execution speed of the second last block of tasks (with hatched lines). We can

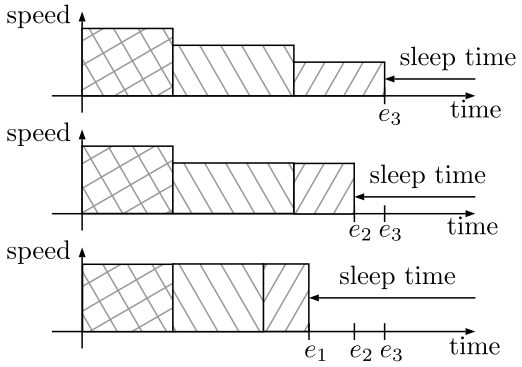

Fig. 1. Illustration of different ending time when we modify the speed of the last block of tasks. From the first schedule, we increase the execution speed of the last block until it has the same execution speed as the previous block, and we get the second schedule as well as the completion time e_2.

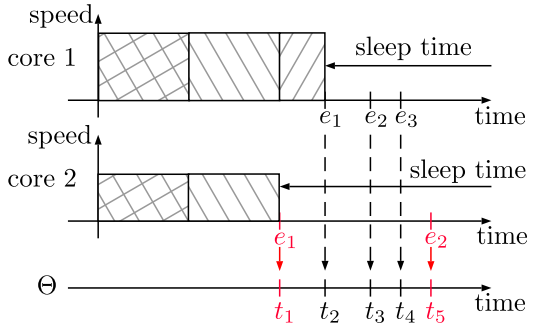

Fig. 2. The set Θ is the union of different completion times of 2 cores.

obtain the completion time e_2 at this moment. Similarly, we can obtain the completion time e_1 by changing the execution speed of the second last and the last blocks of tasks.

Thus, we can compute the different completion time in the following way: let e_{k-u} be the completion time of the schedule if we only use the speeds in $\{s_1, s_2, \ldots, s_{k-u}\}$ and we have:

$$e_{k-u} = \frac{\sum_{j=k-u}^{k} \sum_{i \in \mathcal{Z}_j} w_i}{s_{k-u}} + \sum_{j=1}^{k-u-1} \frac{\sum_{i \in \mathcal{Z}_j} w_i}{s_j} \quad \text{where } 1 \leq u \leq k. \tag{2}$$

We apply the same principle to the multiple cores case. See Fig. 2 for an example on two cores.

Definition 2. Let Θ be the set of relevant completion times on all cores.

$$\Theta = \bigcup_{i=1}^{p} \{e_{n_i-u} \mid 1 \leq u \leq n_i\}$$

At each step, we increase the speed of the last block of tasks. If there are n_i tasks on core i, then such a modification occurs at most $n_i - 1$ times which creates $n_i - 1$ new completion time. Thus adding the original completion time, we have n_i for core i, which implies that the set Θ contains at most n completion time. Let $t_1 < t_2 < \ldots < t_{|\Theta|}$ be the time in Θ. We analyze each zone of Θ which is delimited by two consecutive time. In any interval $[t_\ell, t_{\ell+1}]$, the time t_ℓ corresponds to either an actual completion time of a core, or the time when one block's speed reaches another block's speed (see Fig. 1). Then, the cost of the schedule is a convex function depends on the makespan.

Proposition 2. *The cost of the schedule is convex depending on the makespan for each interval* $[t_\ell, t_{\ell+1}]$ *where* $1 \leq \ell \leq |\Theta|$.

Proof. To simplify the notation, let $B_{i,k}$ be the total workload of the k-th block of tasks on core i, k_i be the number of blocks on core i and c_i be the completion time of core i. Without loss of generality, assume that the cores are sorted in non-decreasing order of completion time, i.e., $c_1 \leq c_2 \leq \ldots \leq c_p$. Since the core p has the largest completion time among all

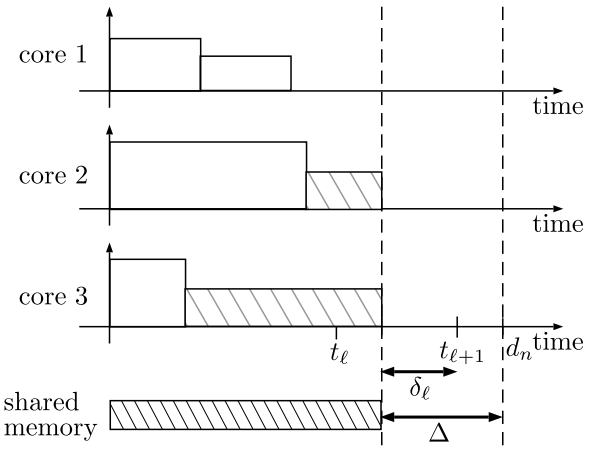

Fig. 3. Illustration of how the sleeping time δ_ℓ affects the schedule. Only the blocks of tasks with hatched lines are affected by the sleeping time.

cores, we know that the shared memory will be active until c_p. Moreover, let $s_{i,b}$ be the running speed of the b-th block on core i. The energy consumption of a schedule S can be defined as follows: $E(S) = \sum_{i=1}^{p} \sum_{b=1}^{k_i} \frac{B_{i,b}}{s_{i,b}} P(s_{i,b}) + \gamma_m c_p$.

Let δ_ℓ denote the length of sleeping time of the new schedule, i.e., the period between the end of the schedule and the end of the zone ℓ. For a given interval $[t_\ell, t_{\ell+1}]$, we can separate the cost into two parts: the first part is a constant according to the makespan, while the other part depends on the makespan. We denote p' as the number of cores whose makespan is less than $c_p = t_{\ell+1}$. The energy consumption of the first p' cores is not affected when the maximum makespan is in $[t_\ell, t_{\ell+1}]$. In fact, only the last block on cores $p'+1, \ldots, p$ will be affected by the sleeping time δ_ℓ (see Fig. 3). Therefore, the speed of the last block for each core will be increased when the length of sleeping time δ_ℓ increases. More formally, the new speed will be $s_{i,k_i}(\delta_\ell) = \frac{B_{i,k_i}}{\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell}$. We are now able to define the energy cost function depending on $\delta_\ell \in [0, t_{\ell+1} - t_\ell]$ as follows:

$$E_\ell(S, \delta_\ell) = \sum_{i=1}^{p'} \sum_{b=1}^{k_i} \frac{B_{i,b}}{s_{i,b}} P(s_{i,b}) + \sum_{i=p'+1}^{p} \sum_{b=1}^{k_i-1} \frac{B_{i,b}}{s_{i,b}} P(s_{i,b}) + \sum_{i=p'+1}^{p} \left(\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell \right) P(s_{i,k_i}(\delta_\ell)) + \gamma_m (c_p - \delta_\ell).$$

Since the first part does not depend on δ_ℓ, it is a constant. We only need to show that the second part of the cost function is convex on δ_ℓ. In particular, we can show that the second derivative is positive.

$$\begin{aligned}
f(\delta_\ell) &= \sum_{i=p'+1}^{p} \left(\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell \right) P(s_{i,k_i}(\delta_\ell)) + \gamma_m (c_p - \delta_\ell) \\
&= \sum_{i=p'+1}^{p} \left(\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell \right) \left(\left(\frac{B_{i,k_i}}{\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell} \right)^\alpha + \gamma \right) + \gamma_m (c_p - \delta_\ell) \\
&= \sum_{i=p'+1}^{p} \left(\frac{B_{i,k_i} - s_{i,k_i}\delta_\ell}{s_{i,k_i}} \right) \left(\frac{B_{i,k_i} s_{i,k_i}}{B_{i,k_i} - s_{i,k_i}\delta_\ell} \right)^\alpha + \sum_{i=p'+1}^{p} \gamma \left(\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell \right) + \gamma_m (c_p - \delta_\ell) \\
&= \sum_{i=p'+1}^{p} \frac{B_{i,k_i}^\alpha s_{i,k_i}^{\alpha-1}}{(B_{i,k_i} - s_{i,k_i}\delta_\ell)^{\alpha-1}} + \sum_{i=p'+1}^{p} \gamma \left(\frac{B_{i,k_i}}{s_{i,k_i}} - \delta_\ell \right) + \gamma_m (c_p - \delta_\ell) \\
f'(\delta_\ell) &= \sum_{i=p'+1}^{p} \frac{(B_{i,k_i}^\alpha s_{i,k_i}^{\alpha-1})(1-\alpha)(-s_{i,k_i})}{(B_{i,k_i} - s_{i,k_i}\delta_\ell)^\alpha} - \gamma_m - \sum_{i=p'+1}^{p} \gamma
\end{aligned}$$

Algorithm 2 Optimal schedule for a given task assignment.

Require: Assignment of tasks to cores.
1: **for** each core i **do**
2: Apply Algorithm 1 and obtain schedule S.
3: **end for**
4: Compute different completion time on each core as defined in Eq. (2) which are then collected in Θ.
5: Compute the optimal solution $E_\ell(S)$ for each interval $[t_\ell, t_{\ell+1}]$ s.t. $t_\ell, t_{\ell+1} \in \Theta$.
6: **return** $\min_{t_\ell \in \Theta} E_\ell(S)$

Algorithm 3 Lower bound of an instance.

Require: Set of tasks.
1: **for** each task j **do**
2: Divide w_j into p equal parts.
3: Assign each part to a different core with deadline d_j.
4: **end for**
5: Apply Algorithm 2 on the new assignment of tasks.

$$= \sum_{i=p'+1}^{p} \frac{(B_{i,k_i}^{\alpha} s_{i,k_i}^{\alpha})(\alpha-1)}{(B_{i,k_i} - s_{i,k_i}\delta_\ell)^{\alpha}} - \gamma_m - \sum_{i=p'+1}^{p} \gamma$$

$$f''(\delta_\ell) = \sum_{i=p'+1}^{p} \frac{(B_{i,k_i}^{\alpha} s_{i,k_i}^{\alpha})(\alpha-1)(-s_{i,k_i})(-\alpha)}{(B_{i,k_i} - s_{i,k_i}\delta_\ell)^{\alpha+1}}$$

$$= \sum_{i=p'+1}^{p} \frac{(B_{i,k_i}^{\alpha} s_{i,k_i}^{\alpha+1})(\alpha-1)\alpha}{(B_{i,k_i} - s_{i,k_i}\delta_\ell)^{\alpha+1}} \geq 0$$

From the construction of the completion time in Θ, we know that the starting time of a block is not in $[t_\ell, t_{\ell+1}]$. We show this by contradiction. Suppose that there exists a core such that the last block of tasks starts at a time t in $[t_\ell, t_{\ell+1}]$. When we increase the execution speed of the last block of tasks, the completion time e cannot be before t. However, there exists a time $t_\ell < t < e < t_{\ell+1}$ such that the execution speed of the last block of tasks is equal to the execution speed of the second last block of tasks. We have a contradiction because e should have been computed and added to Θ. Thus, we have $B_{i,k_i} - s_{i,k_i}\delta_\ell \geq 0$ when $\delta_\ell \in [0, t_{\ell+1} - t_\ell]$ which leads to a feasible solution. Thus, the function $E_\ell(S, \delta_\ell)$ is convex. □

Let $E_\ell(S)$ be the optimal solution when the makespan of the schedule is in $[t_\ell, t_{\ell+1}]$. Then, we have $E_\ell(S) = \min_{\delta_\ell \in [0, t_{\ell+1}-t_\ell]} E_\ell(S, \delta_\ell)$, $\forall t_\ell, t_{\ell+1} \in \Theta$. Combining everything, our optimal algorithm for this part is described in Algorithm 2. In particular, we apply Algorithm 1 for the single-core case to each core in lines 1-3. Then we compute the completion time for each core in line 4, followed by the computation of the optimal solution in each interval, as shown in Proposition 2. Finally, we return the optimal schedule. The analysis of Algorithm 2 is listed as follows.

Theorem 2. *Given an assignment of tasks to cores, Algorithm 2 computes the optimal solution in $O(n^2 p)$ time.*

Proof. By Theorem 1, the schedule S can be obtained in $O(n^2)$ time for each core, the running time is $O(n^2 p)$ for lines 1-3. Once the schedule S is obtained, the set Θ can be computed in $O(n)$ time. Finally, the value $E_\ell(S)$ can be computed by solving a convex minimization problem using Newton's method in $O(n^2)$ time. Thus, the overall running time is $O(n^2 p)$. □

4.2. Task assignment

As shown in Section 4.1, we can compute the schedule with the minimum energy consumption if the assignment of tasks is given. We now present a task assignment scheme that produces a bounded approximate solution.

Lower bound. We show a simple lower bound of our problem, which means that given a set of tasks, the cost of the optimal solution must be at least this cost. The algorithm that gives the lower bound is shown in Algorithm 3.

The intuition of our lower bound is to allow scheduling the same task on different cores at the same time. Since we relax a constraint of the problem, the energy consumption of such a schedule must be at least the optimal energy consumption of our problem. In particular, by relaxing this constraint, the workload of the cores will be balanced, and the execution speeds of the cores are identical at any time, so the energy consumption will be lower because of the convexity of the power function.

Approximation algorithm. Our approximation algorithm, as shown in Algorithm 4, is based on a simple task assignment method. Initially, we sort tasks in non-decreasing order of their deadlines in line 1. Then, we assign the task to the core

with the lowest load (i.e., the core with the least total assigned workload) one by one in lines 2-4. Note that this assignment is similar to the one proposed by Albers et al. in [8]. Finally, we apply Algorithm 2 on this assignment and generate the corresponding schedule in line 5. In the rest of this section, we will prove that Algorithm 4 has a constant approximation ratio.

Algorithm 4 Approximation algorithm.

Require: Set of tasks J
1: Sort tasks in non-decreasing order of deadline.
2: **for** each task $j \in J$ **do**
3: Assign task j to the core with the lowest load.
4: **end for**
5: Apply Algorithm 2 on the assignment of tasks to compute minimal energy consumption.

First, we need to show that there is an upper bound of the speed of the cores when scheduling the last block of tasks.

Proposition 3. *The execution speed of a core at the end of the schedule (the last block of tasks) such that any task of this block does not end at its deadline, is at most* $s_m^\star = \sqrt[\alpha]{\frac{\gamma+\gamma_m}{\beta(\alpha-1)}}$.

Proof. We prove this claim by contradiction. Suppose we have an optimal schedule, and the last block of tasks of a core has an execution speed, which is strictly higher than $s_m^\star$ such that no task of this block ends at its deadline. By considering this particular core and the shared memory, we can decrease the total energy consumption by decreasing the execution speed to $s_m^\star$ or until a task of this block reaches a deadline, which decreases the energy consumption. Thus we have a contradiction with the fact that the schedule is optimal. □

Theorem 3. *Algorithm 4 has an approximation ratio of* $\max\left\{1+\frac{\gamma_m}{\gamma}, 2^{\alpha+2}\right\}$.

Proof. The proof is divided into two parts: **Part (a)** is on the *static energy* while **Part (b)** is devoted to the cost induced by the *dynamic energy*. Then, the approximation ratio is bounded by the ratios obtained by either the ratio in static energy or the one in dynamic energy.

Let LB (resp. AMS) be the lower bound of the schedule (resp. the schedule returned by the algorithm in [8]). Noted that in AMS, the static energy is not taken into account, so the execution speeds of the tasks are as slow as possible.

We construct a schedule S whose cost is higher than the cost of the schedule returned by the algorithm. The schedule S is constructed as follows. For each task, we select the maximum execution speed between AMS and LB. Since AMS is a feasible schedule, by scheduling tasks faster, the resulting schedule is still feasible. We recall that the energy consumption of the schedule from Algorithm 4 is no more than the energy consumption of S, since Algorithm 4 returns a schedule with minimum energy consumption for the same assignment of tasks.

Part (a). By construction, the execution speed of tasks in S is higher or equal to the execution speed of the same task in LB, then the static energy of the cores in S is at most the static energy of the cores in LB. Similarly, the energy consumption of the shared memory in S is at most γ_m/γ times the static energy of the cores in LB. It is because the makespan is at most the total running time of all cores, so by multiplying γ_m/γ (we schedule some tasks with speed $s_m^\star$ if the initial speed was in $[s^\star, s_m^\star]$), we obtain an upper bound of the energy cost of the shared memory from Proposition 3. More formally, we have the following equations: $S_{core}^{(s)} \le LB_{core}^{(s)}$ and $S_{mem}^{(s)} \le \frac{\gamma_m}{\gamma} LB_{core}^{(s)}$, where $S_{core}^{(s)}$ (resp. $S_{mem}^{(s)}$) is the energy consumption of the static energy of the cores (resp. shared memory) in S, and $LB_{core}^{(s)}$ and $LB_{mem}^{(s)}$ are defined similarly. Thus $LB_{core}^{(s)} + LB_{mem}^{(s)} \le S_{core}^{(s)} + S_{mem}^{(s)} \le LB_{core}^{(s)} + \frac{\gamma_m}{\gamma} LB_{core}^{(s)}$. And the approximation ratio is $\frac{LB_{core}^{(s)} + \frac{\gamma_m}{\gamma} LB_{core}^{(s)}}{LB_{core}^{(s)} + LB_{mem}^{(s)}} \le \frac{LB_{core}^{(s)}(1+\frac{\gamma_m}{\gamma})}{LB_{core}^{(s)}} = 1 + \frac{\gamma_m}{\gamma}$.

Since the optimal static energy is at least $LB_{core}^{(s)} + LB_{mem}^{(s)}$, we have that the approximation ratio of the static energy is $(1+\frac{\gamma_m}{\gamma})$.

Part (b). On the other hand, we study the *dynamic energy* of the schedule S, i.e., the energy consumption induced by the execution speed of the cores. Clearly, we have $AMS \le S^{(d)}$ and $LB^{(d)} \le S^{(d)}$ by construction.

We also know that the dynamic energy consumption is at most the sum of dynamic energy consumptions of AMS and LB. That is, $S^{(d)} \le AMS + LB^{(d)}$. Let $LB_{AMS}^{(d)}$ be the lower bound of the dynamic energy of AMS. Albers et al. [8] shows that $LB_{AMS}^{(d)} \le AMS \le 2^{\alpha+1} LB_{AMS}^{(d)}$. So we have the following equations:
$LB^{(d)} \le S^{(d)} \le 2^{\lambda+1} LB_{AMS}^{(d)} + LB^{(d)}$ and $LB_{AMS}^{(d)} \le S^{(d)} \le 2^{\alpha+1} LB_{AMS}^{(d)} + LB^{(d)}$
By summing these two equations, we have:

$$LB^{(d)} + LB^{(d)}_{AMS} \leq 2S^{(d)} \leq 2\left(2^{\alpha+1}LB^{(d)}_{AMS} + LB^{(d)}\right)$$

$$\frac{LB^{(d)} + LB^{(d)}_{AMS}}{2} \leq S^{(d)} \leq 2^{\alpha+1}LB^{(d)}_{AMS} + LB^{(d)}$$

Since the optimal dynamic energy is at least $\frac{LB^{(d)}+LB^{(d)}_{AMS}}{2}$, the approximation ratio of the dynamic energy is $\frac{2^{\alpha+1}LB^{(d)}_{AMS}+LB^{(d)}}{\frac{LB^{(d)}+LB^{(d)}_{AMS}}{2}} \leq 2^{\alpha+2}$.

As the total energy consumption consists of both static and dynamic energy costs, combining the inequalities (3) and (4), the approximation ratio of Algorithm 4 is $\max\left\{1+\frac{\gamma_m}{\gamma}, 2^{\alpha+2}\right\}$. □

5. Conclusion

This paper considers the system-wide energy-efficient task scheduling problem in a system architecture equipped with the multi-core processor and the shared main memory. Especially, we focus on the case when the number of tasks is more than the number of cores, which has been proved to be NP-hard if the number of cores is at least 2. For the multiple cores case, based on the observation we gain from the single-core case, we devise a polynomial-time approximation algorithm with guaranteed performance.

As a future direction, considering the case with different release time will be great of interest.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] V. Chau, C.K.K. Fong, S. Liu, E.Y. Wang, Y. Zhang, Minimizing energy on homogeneous processors with shared memory, in: Proceedings of International Workshop on Frontiers in Algorithmics (FAW), 2020, pp. 83–95.
[2] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proceedings of IEEE Annual Symposium on Foundations of Computer Science (FOCS), 1995, pp. 374–382.
[3] C. Fu, V. Chau, M. Li, C.J. Xue, Race to idle or not: balancing the memory sleep time with DVS for energy minimization, J. Comb. Optim. 35 (3) (2018) 860–894.
[4] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, Proc. Natl. Acad. Sci. USA 103 (11) (2006) 3983–3987.
[5] M. Li, F.F. Yao, H. Yuan, An O(n^2) algorithm for computing optimal continuous voltage schedules, in: Proceedings of Annual Conference on Theory and Applications of Models of Computation (TAMC), 2017, pp. 389–400.
[6] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (5) (2010) 86–96.
[7] S. Albers, Algorithms for dynamic speed scaling, in: Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS), 2011, pp. 1–11.
[8] S. Albers, F. Müller, S. Schmelzer, Speed scaling on parallel processors, Algorithmica 68 (2) (2014) 404–425.
[9] S. Irani, S. Shukla, R. Gupta, Algorithms for power savings, ACM Trans. Algorithms 3 (4) (2007).
[10] E. Bampis, C. Dürr, F. Kacem, I. Milis, Speed scaling with power down scheduling for agreeable deadlines, Sustain. Comput., Inform. Syst. 2 (4) (2012) 184–189.
[11] A. Antoniadis, C. Huang, S. Ott, A fully polynomial-time approximation scheme for speed scaling with a sleep state, Algorithmica 81 (9) (2019) 3725–3745.
[12] J.-J. Chen, H.-R. Hsu, T.-W. Kuo, Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems, in: Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2006, pp. 408–417.
[13] W. Wu, M. Li, K. Wang, H. Huang, E. Chen, Speed scaling problems with memory/cache consideration, J. Sched. 21 (6) (2018) 633–646.
[14] P. Baptiste, Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management, in: Proceedings of the 17th Annual ACM-SIAM (SODA), 2006, pp. 364–367.
[15] P. Baptiste, M. Chrobak, C. Dürr, Polynomial-time algorithms for minimum energy scheduling, ACM Trans. Algorithms 8 (3) (2012) 26:1–26:29.
[16] E. Angel, E. Bampis, V. Chau, Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Appl. Math. 175 (2014) 1–10.
[17] E.D. Demaine, M. Ghodsi, M. Hajiaghayi, A.S. Sayedi-Roshkhar, M. Zadimoghaddam, Scheduling to minimize gaps and power consumption, J. Sched. 16 (2) (2013) 151–160.
[18] R. Jejurikar, R. Gupta, Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems, in: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2004, pp. 78–81.
[19] J. Zhuo, C. Chakrabarti, System-level energy-efficient dynamic task scheduling, in: Proceedings of the Annual Design Automation Conference (DAC), 2005, pp. 628–631.
[20] X. Zhong, C.-Z. Xu, System-wide energy minimization for real-time tasks: lower bound and approximation, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 28:1–28:24.
[21] X. Zhong, C.-Z. Xu, Frequency-aware energy optimization for real-time periodic and aperiodic tasks, in: Proceedings of the ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), 2007, pp. 21–30.
[22] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K.W. Cameron, Powerpack: energy profiling and analysis of high-performance systems and applications, IEEE Trans. Parallel Distrib. Syst. 21 (5) (2010) 658–671.

[23] G. Dhiman, R. Ayoub, T. Rosing, PDRAM: a hybrid PRAM and DRAM main memory system, in: Proceedings of the Annual Design Automation Conference (DAC), 2009, pp. 664–669.
[24] E. Angel, E. Bampis, F. Kacem, D. Letsios, Speed scaling on parallel processors with migration, J. Comb. Optim. 37 (4) (2019) 1266–1282.
[25] E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, M. Sviridenko, Energy efficient scheduling and routing via randomized rounding, J. Sched. 21 (1) (2018) 35–51.
[26] S. Albers, A. Antoniadis, G. Greiner, On multi-processor speed scaling with migration, J. Comput. Syst. Sci. 81 (7) (2015) 1194–1209.
[27] V. Chau, X. Chen, K.C.K. Fong, M. Li, K. Wang, Flow shop for dual CPUs in dynamic voltage scaling, Theor. Comput. Sci. 819 (2020) 24–34.