Race to Idle or Not: Balancing the Memory Sleep Time with DVS for Energy Minimization

Chenchen Fu, Minming Li, Chun Jason Xue
Department of Computer Science, City University of Hong Kong, Hong Kong

***Abstract*—Reducing energy consumption is a critical problem in most of the computing systems today. In recent years, dynamic voltage scaling (DVS) has been often applied in the multi-core processor systems. The leakage power of the main memory shared by the multiple DVS cores is becoming a larger problem with technology scaling. This paper focuses on minimizing the system-wide energy consumption by applying DVS on each core and turning the memory to sleep when all the cores have common idle time. This work presents systematic analysis for the target problem 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 the general task model, a heuristic online algorithm is proposed. Furthermore, the scheme is extended to handle the problem when the transition overhead between the active and sleep modes is not negligible. The experimental results show that the heuristic algorithm can reduce the energy consumption of the overall system by 8.73% in average (up to 28.44%) compared to a state-of-the-art multi-core DVS scheduling scheme.**

I. 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. It is reported that main memory contributes to about 30-40% of total energy consumption on modern systems [5], while the processor consumes as much as 50% energy consumption of the overall system [2]. 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 processor is Dynamic Voltage Scaling (DVS). There are a series of work focusing on improving the energy saving of the multi-core processors by applying DVS [2, 4, 8, 10]. 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 cores and the memory, we propose techniques to optimize the overall system-wide energy consumption.

Among the overall memory energy consumption, leakage power occupies a significant portion, as the memory chips are becoming denser with smaller technology scales. For example, in Dynamic Random Access Memory (DRAM), which is widely used for main memory, leakage power is as much as 10 times of the dynamic read/write power for the memory chip using a process technology with the size smaller than 50nm [9]. 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 a power-saving state (such as the sleep state) when it is not accessed [7, 12]. The main challenges of the system-wide energy minimization problem of considering both the leakage power of the memory and the multi-core processor power, lie in two aspects. On one hand, from the benefit of cores, executing tasks in lower speed leads to less power consumption, while for the memory, the processor speed slowdown may result in an increase of the 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 problem from the existing multi-core DVS scheduling schemes. To handle these two challenges, this work proposes optimal solutions to minimize the overall system-wide energy consumption when the interactions of the multi-core processors 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 both of the multiple DVS cores and the 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 to maximize the time when all cores are idle, 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 reduce the overall energy consumption by 8.73% in average compared to a state-of-the-art multi-core DVS scheduling scheme. The main contributions of this paper are:

- NP-hardness of the problem is proved 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 , two optimal schemes are proposed for two cases where cores have negligible and non-negligible static power, respectively;
- An online heuristic algorithm is proposed considering the general task model;
- The mode transition overhead of the memory and the cores are further considered.

The rest of this paper is organized as follows. The related work is presented in Section II. Section III presents the definitions of the system model and the target problem. In Section IV, the optimal schemes and an online heuristic algorithm are proposed. Section V analyzes the problem considering mode transition overhead. The experimental results are shown in Section VI. Finally we conclude the paper in Section VII.

978-3-9815370-4-8/DATE15/©2015 EDAA

II. Related Work

In this section, we introduce two groups of the most related work. First we introduce DVS scheduling on multi-core processors. Second, the research work on the speed scaling with sleep state problem is 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. For multi-core processors where each core has independent voltage supply, Yang et al. [10] propose an optimal and polynomial schedule for a given task assignment with common release time and deadline. Albers et al. [2] prove the NP-hardness of the problem when tasks cannot migrate and propose several approximation algorithms. Greenstreet et al. [3] study the problem when tasks can migrate between cores and show the optimal schedule can be obtained by Linear Programming.

More recently, several scheduling algorithms focusing on DVS while considering turning the processor to sleep state were proposed [1, 4, 6]. This problem is called *speed scaling with sleep state*, which was first formally defined and discussed in [6]. 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 sleep state. In this way, both static and dynamic energy consumptions of the processor can be reduced. The speed scaling with sleep state problem on a single-core processor is proved to be NP-hard even for tree-structured tasks by Albers et al. [1]. The authors also propose the best possible lower bound for the approximation factor in this problem. For multi-core processors, Chen et al. [4] 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 common idle time of all cores is the objective that needs to be optimized. Furthermore, in our work, putting the cores into sleep state is also taken into account.

III. Problem Statement and Preliminary Analysis

In this section, we present the system and task models, problem formulation, together with the complexity analysis of the most general case of the target problem.

System model: This paper explores energy-efficient scheduling schemes for the multiple homogeneous DVS cores with shared main memory. Assume that the number of cores is C, and each core has individual dynamic voltage supply. The power function of each core remains the same. The core power can be represented as a convex function of the core speed s: $P(s) = \alpha + \beta s^{\lambda}$, where α denotes the static power of the core [4] and $\lambda > 1$ [11]. In this work, we ignore the overhead of the speed adjustment, and assume that the core speeds change in a continuous manner and no upper bound is given (the assumption is the same as that in [10–12]).

The static power for the shared main memory is 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 transition of the memory can be done instantly, but requires extra energy overhead [4]. Conventionally, the transition energy overhead is represented as *break-even time*, which is the idle length where the memory working in active mode consumes the same energy as the transition overhead. 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. Preemption is allowed but tasks cannot migrate between cores once assigned [8, 10]. Given a set of n tasks, $T_1, T_2, ..., T_n$, each task T_i is associated with release time r_i, deadline d_i, and non-negative workload w_i. Without loss of generality, we assume workload w_i is unique for each task. 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. $s_{fi} = \frac{w_i}{d_i - r_i}$. Note that when $\alpha = 0$, the core scheduling task in the filled speed consumes the least energy while satisfying the deadline constraint.

Problem definition: The goal of the explored problem in this paper is to schedule tasks by applying DVS on each core while turning the core to sleep (when $\alpha \neq 0$) when no task is executing, 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 equal to the sleep time of memory, denoted as Δ. 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, a schedule is feasible meaning that no task misses its deadline; a schedule is optimal denoting that it leads to the least system-wide energy consumption among all feasible schedules.

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

This Theorem can be proved by transforming from the NP-hard problem PARTITION.

The energy minimization problem of scheduling tasks on multiple DVS processors is proved to be NP-hard in [2]. But it does not directly imply the above proof because the problem in [2] only targets the DVS scheduling, without considering the common idle time. Furthermore, in [2], optimal schedule is obtained for common arrival time and common deadline tasks, while SDEM has stronger NP-hardness even with common time constraints. Even though Theorem 1 proves the NP-hardness of the problem, we find that SDEM is solvable when assuming the number of cores is sufficient for loading each task on a single core, i.e. $C \geqslant n$. Considering the complexity of NP-hard problem, this paper focuses on the sufficient number of cores. The following analysis explores the optimal results for tasks with common release time and proposes a heuristic online algorithm for general tasks.

IV. Problem Analysis

This section explores solutions for SDEM problem. In the following, we first consider a set of tasks with common release time and propose optimal schemes when only the memory

can be turned to sleep, and then extend the optimal solution to the system model when the cores can also be turned to sleep when idle. At last, for the general task model, a heuristic online algorithm is developed. In this section, we assume the transition overhead of the memory and the cores, ξ_m and ξ, are both 0. The further analysis including the transition overhead is presented in the next section.

A. $\alpha = 0$: *Only the memory can be turned to sleep*

In this subsection, we assume the dynamic power dominates the core power consumption [8, 10, 12]. Given n tasks with common release time , without loss of generality, we assume all tasks arrive at time 0 and each task has an individual deadline. Index them 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. 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. Assume the optimal solution is obtained when the memory sleeps for Δ length in the right hand side of I, and $\delta_i \leqslant \Delta < \delta_{i-1}, \forall i \in [1, n]$ (let $\delta_n = 0, \delta_0 = \infty$). 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 corresponding task model is given in Fig. 1(a). It can be noted that, the memory sleep time Δ is the only determining factor to the optimal solution. The following optimal scheme aims to obtain the best memory sleep time leading to the optimal solution. The system energy consumption can be represented as

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

Under the assumption of $\delta_i \leqslant \Delta < \delta_{i-1}$, the optimal memory sleep time Δ that minimizes E_i can be obtained by derivation.

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

Without loss of generality, we denote each case under the assumption of $\delta_i \leqslant \Delta < \delta_{i-1}$ as Case i with the local minimal energy consumption E_{min_i}. The overall energy minimization analysis is presented in the following lemma and theorem.

Lemma 1. *The local minimal energy consumption E_{min_i} for Case i is obtained as*

$$E_{min\_i} = \begin{cases} E_i(\Delta_{mi}) & \text{if } \delta_i \leqslant \Delta_{mi} < \delta_{i-1} \\ E_i(\delta_i) & \text{if } \Delta_{mi} < \delta_i \\ E_i(\delta_{i-1}) & \text{if } \Delta_{mi} \geqslant \delta_{i-1} \end{cases} \tag{2}$$

when $2 \leqslant i \leqslant n-1$. Specifically, when $\Delta \geqslant \delta_1$,

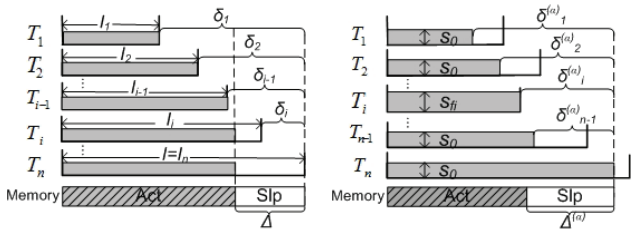
$$E_{min\_1} = \begin{cases} E_1(\Delta_{m1}) & \text{if } \Delta_{m1} \geqslant \delta_1 \\ E_1(\delta_1) & \text{if } \Delta_{m1} < \delta_1 \end{cases} \tag{3}$$

and when $\Delta < \delta_{n-1}$,

$$E_{min\_n} = \begin{cases} E_n(\Delta_{mn}) & \text{if } \Delta_{mn} < \delta_{n-1} \\ E_n(\delta_{n-1}) & \text{if } \Delta_{mn} \geqslant \delta_{n-1} \end{cases} \tag{4}$$

Proof is omitted due to the space limitation. In Theorem 2, we present a scheme which goes over these n cases to find the global minimal energy $E_m = \min\{E_{min\_i}\}, \forall i \in [1, n]$ for a given task set. To clearly describe the theorem, we call Δ_{mi} *valid* when E_{min_i} chooses Δ_{mi} as its solution, for example, Δ_{m1} is valid when $\Delta_{m1} \geqslant \delta_1$, 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*.

Theorem 2. *The optimal solution can be obtained by going through all n cases as defined in Lemma 1 from Case n to Case 1. For each Case i, the global optimal result is obtained when Δ_{mi} is valid or just-fit as the corresponding result given in (2). When Δ_{mi} is invalid, the scheme goes to the next Case $i-1$, which needs to be checked in the same way.*



(a) $\alpha = 0$. (b) $\alpha \neq 0$.

Fig. 1: Common release time task models

Proof: First of all, it can be noted from (1), that for the same task set,

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

In the following analysis, we prove this theorem by induction. Set a counter k to represent the case that is discussed, and initialize it to n. For the base case when $k = n$, if $\Delta_{mn} < \delta_{n-1}$, according to (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 (2) (3) (4), it can be noted that under this case, for $\forall i \in [2, k-1]$ (now $k = n$)

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

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 minimal 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 $k = k - 1$ 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 (6).). So the global optimal solution can be obtained accordingly. Likewise the following process. If at last the scheme goes to Case 1, then it stops and obtains the global optimal result according to (3).

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 minimal energy consumption. Assume there are two solutions, Δ_{min} and Δ'_{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 Δ_{min} (Δ'_{min}) equals to either Δ_{mi} ($\Delta_{mi'}$) or δ_i ($\delta_{i'}$) (Equation (2)). If $\Delta_{min} = \Delta_{mi}$, $\Delta'_{min} = \Delta_{mi'}$, according to (5), we have $\Delta_{min} > \Delta'_{min}$ (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. ■

B. $\alpha \neq 0$: Both the memory and cores can be turned to sleep

In this section, we discuss a more complicated problem, by considering the static power of the core $\alpha \neq 0$, which 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, a notation critical speed is presented to guide the following scheme.

Critical speed: Considering a system only consisting of a single core, for 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 minimal energy value is obtained when the executing speed is $\sqrt[\lambda]{\frac{\alpha}{\beta(\lambda-1)}}$, denoted as s_m, which is independent from T_i [6]. As each task cannot violate its time constraint, we define the critical speed $s_0 = \max\{s_m, s_{fi}\}$.

Given a set of tasks with common release time and executing in the critical speed, index the tasks in the increasing order of their completion time, denoted as c_i, where $c_i = \frac{w_i}{s_0}$. Note that different from Section IV.A, we set $\delta_i^{(\alpha)} = |I| - c_i$, where the interval length $|I| = |c_n|$. Denote the sleep length of memory as $\Delta^{(\alpha)}$. The task model is given in Fig. 1(b). Intuitively, in the optimal solution, tasks that satisfy $|I| - c_i > \Delta^{(\alpha)}$ maintain the critical speed on their cores, while the other tasks whose $|I| - c_i \leqslant \Delta^{(\alpha)}$ need to increase their executing speed to align the sleep time of their cores to that of the memory to minimize the energy. The system energy excluding the cores loading tasks whose $|I| - c_i > \Delta^{(\alpha)}$ is represented in (7). It can be used to obtain the optimal solution to the overall system (including all cores), because the tasks with $|I| - c_i > \Delta^{(\alpha)}$ do not affect the optimal solution of the memory length $\Delta^{(\alpha)}$.

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

When $\delta_i^{(\alpha)} \leqslant \Delta^{(\alpha)} < \delta_{i-1}^{(\alpha)}$, the $\Delta_{mi}^{(\alpha)}$ that leads to the minimal system energy is

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

, which means that all tasks from T_n to T_i execute to finish at $|I| - \Delta_{mi}^{(\alpha)}$, and other tasks maintain the critical speed. Denote each case under the assumption of $\delta_i^{(\alpha)} \leqslant \Delta^{(\alpha)} < \delta_{i-1}^{(\alpha)}$ with the minimal energy value $E_{min\_i}^{(\alpha)}$ as Case $i^{(\alpha)}$.

Theorem 3. *The optimal solution can be obtained by going through all n cases 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 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 different from Theorem 2, $\Delta_{mi}^{(\alpha)} > \Delta_{m(i-1)}^{(\alpha)}$ is not satisfied for all cases. The relationship between two successive cases are not fixed as in Theorem 2. Hence all the local optimal results must be recorded and compared to obtain the global optimal solution.

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 (7) and (8), while setting $i = 1$, as all tasks share the same feasible region.

C. Online algorithm for general tasks

In this section, focusing on the general task model, an online heuristic algorithm is proposed. The main idea of the algorithm is presented as follows. When a new task T_i arrives, active the algorithm, and set all unfinished tasks' release time the same as that of T_i. By applying the analysis in Section IV.A (Section IV.B), the local optimal solution can be obtained for the current tasks. Keep the memory in 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. Note that all the proposed schemes in Section IV can be applied for heterogeneous cores with different power functions.

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 IV.A (IV.B), and record each task's corresponding execution time p_j;
4: Mark the latest executing 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 executing 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);

A detailed algorithm description by an example is given as follows. Without loss of generality, assume the first task T_1 arrives at time 0. We calculate the optimal memory sleep time Δ_1^{on} as shown in Section IV.A (IV.B) but with the single task. Based on the optimal solution, the executing time of T_1 is developed as $p_1 = d_1 - \Delta_1^{on}$. Considering that more tasks might come later, postponing the execution of T_1 is more likely to have a chance of obtaining longer execution overlap with other tasks. Hence we keep the memory (and cores) in sleep state until T_1 meets its latest executing 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 Section IV.A (IV.B) can be used to obtain the optimal solution Δ_2^{on}, and similar processes are followed to calculate the latest executing time of two tasks. Once one task meets its latest executing 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 executing time.

V. Transition Overhead Analysis

In this section, we analyze the solutions for problem SDEM when $\xi_m \neq 0$ and $\xi \neq 0$. Both optimal solutions for tasks with common release time and the online heuristic algorithm developed in the former section are extended.

Constrained critical speed: When $\xi \neq 0$, $s_m = \sqrt[\lambda]{\frac{\alpha}{\beta(\lambda-1)}}$ is optimal only when $|I_i| - \frac{w_i}{s_m} \geqslant \xi$. Otherwise, it is easy to note that the core consumes the least energy by executing T_i in s_{fi}. In the following, we use s_c to represent the constrained critical speed of a task T_i. Set $s_c = s_m$ when $|I_i| - \frac{w_i}{s_m} \geqslant \xi$, 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 $\frac{w_i}{s_c}$. An original task executing model, together with n cases, can be constructed similar to Section IV.B. For each case, the system energy consumption function can be represented the same as in (7), because the transition overhead is independent from the memory sleep time, which means that it does not affect the optimal solution. Thus the preliminary local optimal memory sleep time that leads to the minimal $E_i^{(\alpha)}$ is the same as in (8). 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 4. *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 I, which presents the relationships among $\Delta_{mi}^{(\alpha)}$ and ξ, ξ_m. Otherwise the scheme does nothing and enters to the next case.*

TABLE I: 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 in 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 in s_c
$\Delta_{mi}^{(\alpha)} < \xi, \xi_m$	$\Delta_{mi}^{(\xi)} = 0$, all cores executing tasks in s_c

Due to the space limitation, we only focus on the proof of the third case, as it is the most complicated. When $\xi_m \leqslant \Delta_{mi}^{(\alpha)} < \xi$, three subcases need to be fully 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 (1). 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 appears 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 minimal energy consumption should be set as the minimum value of $\{E_i^{(\alpha)}(\Delta_{mi}), E_i^{(\alpha)}(\xi), E_i^{(\alpha)}(0)\}$.

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 proposed in this section (Line 4). Before turning the memory into sleep state (Line 6), we should make sure that the time period between the new arriving time t_i and the minimum of all tasks' latest executing points is larger than ξ. Otherwise all tasks start to execute at t_i.

VI. Evaluation

In this section, we evaluate the effectiveness of the proposed online heuristic algorithm compared with another online multi-core DVS scheduling algorithm proposed in [2]. The algorithm in [2], denoted as MBKP, achieves satisfying results among multiple DVS-cores in terms of energy saving, but does not consider the leakage energy consumption of the memory. In the following experiment, we compare the proposed algorithm, denoted as SDEM-ON with the original MBKP, which does not turn memory to sleep and a modified MBKP approach, denoted as MBKPS, by applying a simple sleep transition scheme.

We evaluate the algorithm over various core utilizations [12, 13], different memory static power settings [13] and different memory transition overhead by generating random tasks. Note that randomly generated tasks is a common validation method in previous work [4, 10, 12, 13].

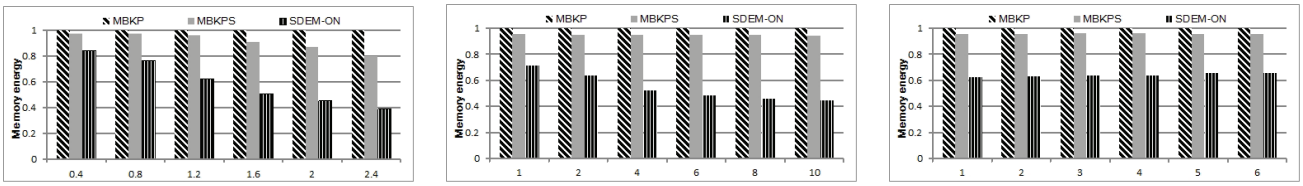
Recall that the proposed online heuristic assumes the number of cores is sufficient for scheduling tasks. This assumption, which is important for pursuing theoretical analysis, actually does not imply over-optimistic results in practice. The actual number of executing tasks at a time is reasonable. In the experiment, 100 tasks is randomly generated as follows. The feasible region length of each task is randomly ranged from $0.4s$ to $3s$. The inter-arrival time between two successive tasks is randomly ranged between $[0, x]$. Set the number of cores to be 6. For a high utilization system, the inter-arrival time between the first task and the seventh task, which is as much as $6x$, should be comparable with the processing length of a task. Considering that the processing time, which is determined by the executing 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. Hence, we set $x = 0.4s$ for a high utilization system, which implies that all 6 cores are most likely to be used at any time, and range x from $0.4s$ to $2.4s$ with a step size of 0.4 to evaluate results based on various utilization systems, where $x = 2.4s$ implies that a single core might be sufficient to schedule all tasks. The weight of a task is set to the range between $[3, 15]$.

For the core power and memory static power setting, firstly we set them to be comparable, and then fix the core power parameter $\beta = 1$ and scale the memory static power to observe the effect on the energy saving [4, 13]. We fix the core transition overhead $\xi = 1$ and scale the memory transition overhead, considering the leakage power is more significant in memory than in the multi-core processor. The detailed parameter setting is given in Table II, where $*$ represents the default value of each parameter when evaluating other parameters. To generate convincing results, for each data point, we randomly generate ten different cases, and use the average value as the final evaluation result for each data point. Note that all the energy values shown in the experiments are normalized to the corresponding MBKP results.

The experimental results of memory energy consumption over three algorithms are given in Fig. 2. Fig. 2(a) shows the results of memory energy consumption based on various core utilizations. The average energy saving improvement compared to MBKPS is 35.40%. Fig. 2(b) shows the evaluation results of different static power settings. The average energy saving improvement compared to MBKPS is 43.14%. The improve-

TABLE II: Parameter setting over various core utilizations, memory power settings and memory transition overheads.

Point	1	2	3	4	5	6
1/utilization (x)	0.4	0.8	1.2*	1.6	2.0	2.4
Power setting (α_m/β)	1	2*	4	6	8	10
Transition overhead (ξ)	1	2*	3	4	5	6



(a) The inverse of core utilizations. (b) Memory static power setting. (c) Memory transition overhead.

Fig. 2: The memory energy consumption comparison over different parameters.

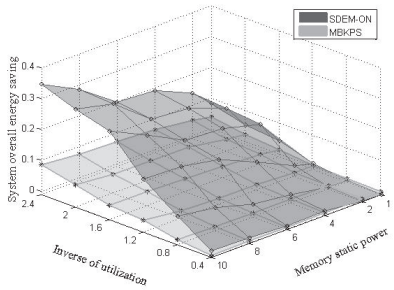


Fig. 3: The system-wide energy saving improvement over different memory static power setting and core utilizations.

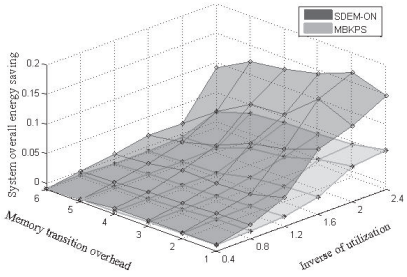


Fig. 4: The system-wide energy saving improvement over different memory transition overhead and core utilizations.

ment over different memory transition overhead is 33.59% in average, shown in Fig. 2(c). These three figures show that SDEM-ON can turn the memory into sleep state for a longer period than MBKPS, by which, the leakage power of memory is reduced. Besides, it can be noted that the energy saving improvement increases with the core utilization decreasing and the memory static power being larger, and decreases a little bit with the transition overhead increasing. This indicates that the proposed algorithm can obtain more significant benefit when the system has larger flexibility.

The proposed schemes in this paper do not aim to reduce the dynamic energy of the memory. Therefore, in the overall energy consumption analysis, we assume the dynamic energy remains the same for three algorithms. We set the dynamic power of memory $\alpha_d = \frac{1}{4}\alpha_m$ [9]. The overall system energy consumption is shown in Fig. 3 and Fig. 4. The varying trend of the energy saving is similar to that of the memory energy saving. Note that SDEM-ON performs worse than MBKP in the bottom-left corner of Fig. 4. This is because when the system has very high utilization and transition overhead, the time that SDEM-ON turns the memory to sleep is more likely to be unfortunately interrupted by a new arriving task, paying a very high transition mode energy costs. However, for most situations, SDEM-ON leads to a much better trade-off between the processor and the memory than MBKPS. SDEM-ON reduces the system-wide energy by 4.10% (up to 12.68%) and 8.73% (up to 28.44%) in average compared to MBKPS varying over different parameters as shown in Fig. 3 and Fig. 4, respectively.

VII. 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 the number of cores is unbounded, optimal schemes are proposed for tasks with common release time. Furthermore, an online heuristic algorithm is developed for general tasks. This paper is the first attempt in minimizing the system-wide energy consumption based on a multiple DVS cores system. Both theoretical and practical solutions for the target problem based on different system models are presented. Evaluations show that the proposed heuristic algorithm can reduce the overall system energy consumption by 8.73% in average (up to 28.44%).

Acknowledgement

This work was partially supported by grants from City University of Hong Kong [Project No. CityU 117913] and [Project No. CityU 9231168].

References

[1] S. Albers and A. Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. SODA, pages 1266–1285, 2012.

[2] S. Albers, F. Mller, and S. Schmelzer. Speed scaling on parallel processors. In *IN: PROC. SPAA*, pages 404–425, 2007.

[3] B. Bingham and M. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *ISPA*, pages 153–161, 2008.

[4] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. RTAS, pages 408–417, 2006.

[5] G. Dhiman, R. Ayoub, and T. Rosing. Pdram: A hybrid pram and dram main memory system. DAC, pages 664–469, 2009.

[6] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4), Nov. 2007.

[7] C.-G. Lyuh and T. Kim. Memory access scheduling and binding considering energy minimization in multi-bank memory systems. DAC, 2004.

[8] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem. Energy aware scheduling for distributed real-time systems. In *IPDPS*, page 21, 2003.

[9] S. J. E. Wilton and N. Jouppi. Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[10] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. DATE, pages 468–473, 2005.

[11] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS*, pages 374–382, 1995.

[12] X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: Lower bound and approximation. *ACM Trans. Embed. Comput. Syst.*, 7(3):28:1–28:24, May 2008.

[13] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *DAC*, pages 628–631, 2005.