Reducing Peak Power Consumption in Multi-Core Systems without Violating Real-Time Constraints

Jinkyu Lee, *Member, IEEE*, Buyoung Yun, *Student Member, IEEE*, and Kang G. Shin, *Life Fellow, IEEE*

Abstract—The potential of multi-core chips for high performance and reliability at low cost has made them ideal computing platforms for embedded real-time systems. As a result, power management of a multi-core chip has become an important issue in the design of embedded real-time systems. Most existing approaches have been designed to regulate the behavior of *average* power consumption, such as minimizing the total energy consumption or the chip temperature. However, little attention has been paid to the worst-case behavior of instantaneous power consumption on a chip, called *chip-level peak power consumption*, an important design parameter that determines the cost and/or size of chip design/packaging and the underlying power supply. We address this problem by reducing the chip-level peak power consumption at design time without violating any real-time constraints. We achieve this by carefully scheduling real-time tasks, without relying on any additional hardware implementation for power management, such as dynamic voltage and frequency scaling. Specifically, we propose a new scheduling algorithm FP_Θ that restricts the concurrent execution of tasks assigned on different cores, and perform its schedulability analysis. Using this analysis, we develop a method that finds a set of concurrent executable tasks, such that the design-time chip-level peak power consumption is minimized and all timing requirements are met. We demonstrate via simulation that the proposed method not only keeps the design-time chip-level peak power consumption as low as the theoretical lower bound for trivial cases, but also reduces the peak power consumption for non-trivial cases by up to 12.9 percent compared to the case of no restriction on concurrent task execution.

Index Terms—Peak power consumption, system design, multi-core systems, real-time scheduling

1 Introduction

With the advance of CMOS manufacturing technology, a single chip multi-processor has become widely deployed in embedded real-time systems due to its potential for achieving high performance at low cost. As the size of a CMOS transistor continuously shrinks, more cores and electronic devices are getting integrated on a single die. However, since the power density also rapidly grows on a modern multi-core chip, proper power management has become one of the most critical issues in minimizing its adverse effects on both chip cooling efficiency and chip reliability. To address this issue without violating any timing requirements, numerous power-management techniques have been proposed to minimize energy consumption [1], or to minimize chip temperature [2], [3], [4], by controlling the average chip power consumption.

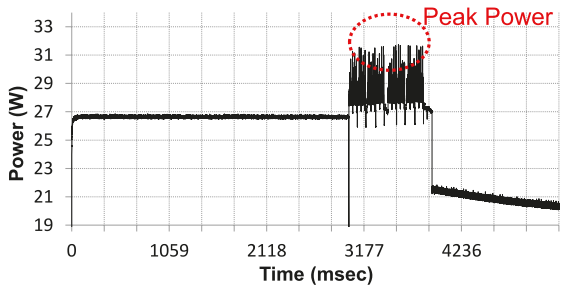
However, little has been done on the worst-case behavior of instantaneous peak power consumption on a chip—called the *chip-level peak power consumption*—in embedded real-time systems. The chip-level peak power consumption is an important design parameter as the chip should be designed for efficient and robust delivery of power to all components on the chip even in the case of worst-case instantaneous (i.e., chip-level peak) power consumption. To meet the chip-level peak power demand during runtime, power supply units with sufficient capacity are required, and the proper chip packaging solution should be applied at the chip design time. However, the chip packaging solution does not always guarantee robust and efficient power delivery, because the proper placement of decoupling capacitors is known to be a non-trivial problem [5]. Thus, the chip-level peak power consumption dictates the cost and the size of such power-related components, each of which is crucial to commercial embedded real-time products. So far, there have been studies on how to manage the chip-level peak power consumption for general-purpose systems [6], [7], [8], but it is unclear how to adapt them to meet the timing constraints of embedded real-time applications.

The goal of this paper is to minimize chip-level peak power consumption at design time without violating any timing requirements. With this goal achieved, one can optimize the chip design/packaging process and the power-related components according to the minimized chip-level peak power consumption (guaranteed at design time), and then reduce the cost and/or size of the process and components by avoiding/minimizing their over-design. To achieve the above goal, we only rely on scheduling of tasks (each of which has its own peak power characteristics shown in Fig. 1) at the software layer without requiring any

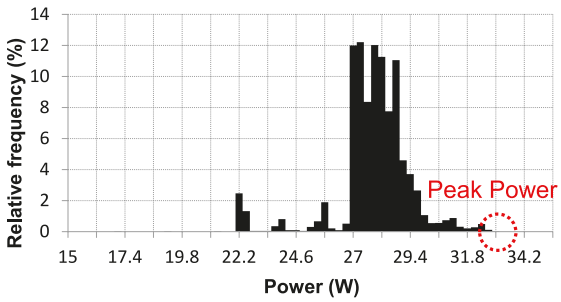
- *J. Lee is with Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Gyeonggi-Do, South Korea. E-mail: jinkyu.lee@skku.edu.*
- *B. Yun and K.G. Shin are with Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2121, U.S.A. E-mail: {buyoung,kgshin}@eecs.umich.edu.*

Manuscript received 4 Sept. 2012; revised 17 Apr. 2013; accepted 28 Apr. 2013; date of publication 12 May 2013; date of current version 21 Feb. 2014. Recommended for acceptance by M. Kandemir.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.131

1045-9219 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



(a) Sampled power dissipation during the execution of an application



(b) Histogram of the sampled power dissipation

Fig. 1. An example of power characteristics when "qsort" in MiBench benchmark suites is running on the selected target core model in Section 6.

hardware support, such as dynamic voltage/frequency scaling (DVFS) and power-gating. DVFS has been widely used for power management in microprocessors [1], [2], [3], but it suffers from the non-zero switching overhead of changing the operating voltage/frequency. Also, power-gating can be used to reduce leakage power efficiently [9], but due to the complexity of data-retention mechanism and the large latency in turning power on/off, it is unsuitable for embedded real-time systems. Thus, one of the main advantages of our approach is its simplicity in and applicability to any multi-core platform without requiring additional hardware support. Note that our scheduling-based approach does not limit the use of any hardware technique since it is orthogonal to hardware implementation. To the best of our knowledge, this is the first peak power-aware scheduling that addresses the problem of designing a multi-core chip with its power-related components in embedded real-time systems.

In this paper, our focus is confined to the partitioned scheduling—in which a task is executed on its designated core—for simplicity/less overhead (e.g., no task migration between cores) and/or for efficient utilization of heterogeneous cores (e.g., different tasks can be executed more efficiently on different cores). To reduce the design-time chip-level peak power consumption using partitioned scheduling, we need the worst-case instantaneous (i.e., task-level peak) power consumption of each task on its designated core, which will be discussed in Section 2. We also need to control concurrent execution of tasks on different cores; otherwise, the tasks with the highest task-level peak power consumption on each core (as well as the parts of task execution that dissipate the power as much as each task's peak power) can be executed at the same time, making no reduction of chip-level peak power consumption.

We extend the traditional partitioned fixed priority (FP) scheduling algorithm [10] that is widely deployed in many embedded real-time systems, and develop a new scheduling algorithm FP_Θ, which avoids the concurrent execution of tasks in a given list Θ in Section 4.1. To meet the timing requirements, we develop a schedulability test of FP_Θ in Section 4.2. The final step is then to find a list of concurrently executable tasks that minimizes the chip-level peak power consumption at design time without violating the timing constraints. By deriving some properties of the schedulability test of FP_Θ, we develop a method for finding such a list with low time-complexity in Section 5. The effectiveness of the proposed method has been demonstrated via simulation in terms of reduction of chip-level peak power consumption in Section 6; for trivial cases, our method keeps the peak power consumption as low as the theoretical lower bound, and for non-trivial cases, it reduces the consumption up to 12.9 percent over the case of no restriction on concurrent task execution.

In summary, this paper makes the following contributions.

- Introduction of a new multi-core chip design problem regarding the chip-level peak power consumption for embedded real-time systems;
- Development of a new scheduling algorithm FP_Θ that avoids concurrent execution of tasks on different cores in a given list Θ, by generalizing the traditional partitioned FP scheduling algorithm (i.e., the uniprocessor FP scheduling algorithm [10] for each core);
- Derivation of a schedulability test of FP_Θ, a generalization of that of the traditional partitioned FP scheduling algorithm (i.e., the exact schedulability test of the uniprocessor FP scheduling algorithm [11] for each core);
- Development of a method that reduces the design-time chip-level peak power consumption using FP_Θ and its schedulability analysis, without requiring any hardware support; and
- Demonstration of the effectiveness of the proposed method via in-depth simulation.

The rest of this paper is organized as follows. Section 2 presents our system model. Section 3 outlines how to achieve our design goal. Section 4 presents FP_Θ and its schedulability test, which will be used for a method to achieve the goal, presented in Section 5. Section 6 evaluates the proposed method, and Section 7 discusses the related work. Finally, Section 8 concludes the paper.

2 System Model and Assumptions

We consider a multi-core chip, which consists of m cores $\{\mathcal{S}_j\}_{j=1}^{m}$, and the cores may be identical or heterogeneous. We assume that each task τ_i is executed only on its designated core, and let Φ_j denote a set of tasks assigned to $\mathcal{S}_j$. Let Φ denote the entire task set to be executed on the multi-core chip (i.e., $\Phi \triangleq \cup_{j=1}^{m} \Phi_j$). Also, let $|A|$ denote the cardinality of the set A.

We focus on a sporadic task model [12] in which a real-time task $\tau_i \in \Phi$ is modeled as (T_i, C_i, D_i), where T_i is the minimum separation between two successive invocations, C_i is the worst-case execution time when it is executed on its designated core, and D_i is its relative deadline. We restrict our attention to the constrained deadline tasks, i.e., $C_i \leq D_i \leq T_i$, $\forall \tau_i \in \Phi$. Note that this sporadic task model is general since it can represent not only typical control systems that employ periodic sampling, but also event-triggered functions in computing systems with a specified minimum time separation between two consecutive events [12]. Also, the global priority P_i is given to each task τ_i, depending on its criticality; the smaller P_i, the higher priority. Every invocation of τ_i is called a *job*, and each job is separated from its predecessor/successor by at least T_i time units. In constrained deadline task systems, at most one active job per task can exist in any time slot, and hence, for simplicity of presentation, we use the term "task" also to refer "job of the task" in the rest of this paper. Like most existing studies on real-time scheduling, we focus on task sets in which all tasks are independent of each other. This task model is not as restricted as it may appear, since there are ways to transform a set of dependent tasks to independent ones, e.g., [13].

In addition, each task consumes a different amount of electric power during its execution, depending on its characteristics and computational load. To reduce the chip-level peak power consumption at design time, we need an upper bound of the instantaneous power dissipation of each task. For this purpose, we use one more parameter associated with each task τ_i: the worst-case instantaneous (i.e., task-level peak) power consumption (denoted by B_i) of τ_i. In other words, any job of τ_i does not dissipate more instantaneous power than B_i at any time during its execution. Let B_{max} be the highest (instantaneous) peak power consumed by the tasks in Φ (i.e., $B_{max} \triangleq \max_{j=1}^{m} \max_{\tau_i \in \Phi_j} B_i$).

When the task set Φ and the chip model are given at design time, B_i of each task $\tau_i \in \Phi$ can be estimated/obtained off-line from the readings of the power sensor attached/embedded in the processor, which is similar to the power measurement in [14], after the task has been executed continuously for a given set of its inputs. For this, the input of each task should be known, because the amount of power dissipation can differ depending on its task execution path determined by the input. However, the task input data can be collected empirically or estimated as is done for the estimation of its worst-case execution time (e.g., the statistics of the wheel/vehicle speed can be used to determine the set of inputs for the ABS control task in an automotive control system).

Fig. 1 shows the sampled power dissipation and the histogram of power consumption when the "qsort" application in MiBench benchmark suites [15] is executed on the Wattch power simulator [16], where a core is modeled as described in Section 6. To obtain the task-level peak power dissipation during its execution, 10 input sets were randomly generated using the dictionary database, and the power dissipation was sampled during each execution with different input sets.

One may think that the current estimation/modeling of the worst-case instantaneous power consumption of each task (i.e., B_i) can be overly pessimistic. However, the "worst-case" profile of power consumption is necessary since the power-related elements should be designed to accommodate all possible values of instantaneous power consumption. To lower the degree of pessimism, we may divide a task into different phases, provided the internal structure of the task is known. Then, it is possible to differentiate the worst-case peak power consumption of each phase, which may be effective when the worst-case peak power consumption significantly varies with different phases of a given task. In this paper, we focus on the modeling of the per-task peak power consumption; one may easily extend it to per-phase peak power consumption.

3 Overview

In this section, we first formally state our goal and identify issues in achieving the goal. Then, we outline how to address the issues, which will later be detailed in Sections 4 and 5.

The goal of this paper, as mentioned in Section 1, can be formally stated as follows:

> Given $\{\Phi_j\}_{j=1}^{m}$, each task to be executed on its designated core $\{\mathcal{S}_j\}_{j=1}^{m}$, respectively, minimize the design parameter $\mathcal{B}$, such that the following two conditions are guaranteed at design time: (i) the chip-level peak power consumption does not exceed $\mathcal{B}$; and (ii) there is no deadline miss for any job invoked by $\{\Phi_j\}_{j=1}^{m}$.

To achieve this goal, we need to address the following issues:

1. How to guarantee (i) for given $\mathcal{B}$;
2. How to guarantee (ii) for given $\mathcal{B}$; and
3. How to find the minimum $\mathcal{B}$ without compromising (i) and (ii).

We now outline how to address each issue, starting from issue 1. Let Θ^* denote a list of tuples of all combinations of at least two tasks assigned to different cores. Tuples in Θ^* are sorted in descending order of the sum of the peak power consumption of tasks (i.e., sum of B_i) in each tuple. Example 1 in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.131, illustrates Θ^*.

If all tasks in $\Theta^*(y)$ simultaneously execute on their designated cores, the parts of the tasks that dissipate the power as much as each task's peak power can also execute at the same time. In this case, the chip-level power consumption can be up to the sum of the peak power consumption of the tasks (i.e., sum of B_i) as shown in the second column of Table 1. Therefore, to guarantee (i) for given $\mathcal{B}$ at design time, we need to control the concurrent execution of tasks on different cores. Without such controls (e.g., any traditional partitioned scheduling algorithm in which the schedule of tasks in a core is independent of that of tasks in other cores), all tasks in $\Theta^*(1)$ can be executed at the same time, meaning that we cannot guarantee (i) if $\mathcal{B}$ is less than the sum of the peak power consumption of tasks in $\Theta^*(1)$.

To address issue 1, the following lemma provides a relationship between the control of concurrent task execution and an upper bound of the chip-level peak power consumption.

TABLE 1
An Upper-Bound for the Chip-Level Power Consumption When At Least Two Tasks on Different Cores Are Executed at the Same Time

$\Theta^*(y) =$ (task on $\mathcal{S}_1$, task on $\mathcal{S}_2$, task on $\mathcal{S}_3$, task on $\mathcal{S}_4$)	an upper bound on the chip-level power consumption (W)
$\Theta^*(1) = (\tau_1, \tau_3, \tau_4, \tau_6)$	$20 + 9 + 17 + 12 = 58$
$\Theta^*(2) = (\tau_2, \tau_3, \tau_4, \tau_6)$	$15 + 9 + 17 + 12 = 53$
$\Theta^*(3) = (\tau_1, \tau_3, \tau_5, \tau_6)$	$20 + 9 + 8 + 12 = 49$
$\Theta^*(4) = (\tau_1$, none, $\tau_5, \tau_6)$	$20 + 0 + 17 + 12 = 49$
$\Theta^*(5) = (\tau_1, \tau_3, \tau_4$, none)	$20 + 9 + 17 + 0 = 46$
$\Theta^*(6) = (\tau_2, \tau_3, \tau_5, \tau_6)$	$15 + 9 + 8 + 12 = 44$
...	...
$\Theta^*(27) =$ (none, τ_3, none, τ_6)	$0 + 9 + 0 + 12 = 21$
$\Theta^*(28) =$ (none, none, τ_5, τ_6)	$0 + 0 + 8 + 12 = 20$
$\Theta^*(29) =$ (none, τ_3, τ_5, none)	$0 + 9 + 8 + 0 = 17$

Lemma 1. *Let* $\Theta = [\Theta^*(1), \Theta^*(2), \ldots, \Theta^*(y)]$. *If we avoid the concurrent execution of tasks in all tuples in* Θ, *then we can guarantee that the chip-level peak power consumption does not exceed* $\max\big(B_{max}, \sum_{\tau_k \in \Theta^*(y+1)} B_k\big)$, *where* $B_{max} \triangleq \max_{j=1}^{m} \max_{\tau_i \in \Phi_j} B_i$.

Proof. Since we can prevent all the concurrent task execution whose upper bound on the chip-level power consumption is larger than that of $\Theta^*(y+1)$, we can guarantee that the chip-level peak power consumption does not exceed $\sum_{\tau_k \in \Theta^*(y+1)} B_k$, if at least two tasks are concurrently executed on different cores.

However, if only one task is executed on the multi-core chip, its power consumption amounts to at most B_{max}. Therefore, we choose the maximum between B_{max} and $\sum_{\tau_k \in \Theta^*(y+1)} B_k$.

Using Lemma 1, we can set Θ for given $\mathcal{B}$; note that $\mathcal{B}$ is infeasible if $\mathcal{B} < B_{max}$. Then, the remaining step is to develop a new scheduling algorithm that can avoid all the concurrent execution of tasks in each tuple of Θ. However, the maximum size of Θ (i.e., the size of Θ^*) is the number of all combinations of at least two tasks on different cores, which is $O(\sum_{j=2}^{m} n^j \cdot \binom{m}{j})$, where $n \triangleq \max_{j=1}^{m} |\Phi_j|$. Therefore, the maximum size of θ increases exponentially with m, e.g., if $n = 10$ and $m = 10$, the size of θ is at least in the order of 10^{10}.

For scalability, we logically divide a multi-core chip into several groups of cores, and achieve the goal for each group. In this paper, we consider each group with two cores, and then the maximum size of Θ for each group is only $O(n^2)$. Tables 2 and 3 show each Θ^* (maximum Θ) when the four-core chip in Example 1 is divided into two groups: $\{\mathcal{S}_1, \mathcal{S}_2\}$ and $\{\mathcal{S}_3, \mathcal{S}_4\}$. Then, the size of Θ^* is reduced from 29 (in Table 1) to 2 (in Table 2 or 3), and the scheduling algorithm for each group needs to maintain only a small number of pairs instead of a huge number of tuples.

TABLE 2
An Upper-Bound of the Power Consumption of $\{\mathcal{S}_1, \mathcal{S}_2\}$ When Two Tasks on Different Cores Are Executed at the Same Time

$\Theta^*(i) =$ (task on $\mathcal{S}_1$, task on $\mathcal{S}_2$)	an upper bound on the two-core-level power consumption (W)
$\Theta^*(1) = (\tau_1, \tau_3)$	$20 + 9 = 29$
$\Theta^*(2) = (\tau_2, \tau_3)$	$15 + 9 = 24$

TABLE 3
An Upper-Bound on the Power Consumption of $\{\mathcal{S}_3, \mathcal{S}_4\}$ When Two Tasks on Different Cores Are Executed at the Same Time

$\Theta^*(i) =$ (task on $\mathcal{S}_3$, task on $\mathcal{S}_4$)	an upper bound on the two-core-level power consumption (W)
$\Theta^*(1) = (\tau_4, \tau_6)$	$17 + 12 = 29$
$\Theta^*(2) = (\tau_5, \tau_6)$	$8 + 12 = 20$

In Section 4.1, we will develop FP_Θ, a new scheduling algorithm that generalizes the traditional partitioned FP scheduling algorithm with restriction of the concurrent execution of tasks in all pairs in given Θ.

Then, to address issue 2, we will, in Section 4.2, develop a schedulability test of FP_Θ, determining whether a task set is schedulable by FP_Θ or not.

Once FP_Θ and its schedulability test are developed, the remaining step for issue 3 is to develop a method for determining Θ that minimizes $\mathcal{B}$. In Section 5, we will derive some properties of the schedulability test of FP_Θ, and then develop such a method, by utilizing FP_Θ and its schedulability analysis to guarantee (i) and (ii), respectively.

In summary, we will achieve this by addressing issues 1, 2 and 3 as follows:

1. In Section 4.1, we will develop the FP_Θ scheduling algorithm that can avoid the concurrent execution of tasks in all pairs in given Θ;
2. In Section 4.2, we will develop a schedulability test of FP_Θ; and
3. In Section 5, we divide a multi-core chip in groups of two cores each, and find per-group Θ that minimizes $\mathcal{B}$ such that (i) is guaranteed by the FP_Θ scheduling algorithm itself and (ii) is guaranteed by the schedulability test of FP_Θ.

4 FP_Θ Scheduling Algorithm and Its Schedulability Analysis

As mentioned in Section 3, we need a new scheduling algorithm that can avoid the concurrent execution of tasks on different cores in all pairs in given Θ. To meet this need, we first present a new scheduling algorithm, FP_Θ, and then develop its schedulability test.

4.1 FP_Θ Scheduling Algorithm

In the traditional partitioned fixed priority scheduling algorithm (i.e., the uniprocessor FP scheduling algorithm [10] on each core), one of multiple ready tasks is scheduled on each core according to their (fixed) priority. To extend this algorithm with the restriction of concurrent task execution in all pairs in given Θ, we need to decide which task should be chosen when both tasks in a pair are ready. The (global) fixed priority of each task is also used for this decision, and we call this algorithm FP_Θ (fixed priority with the restriction of concurrent task execution in all pairs in Θ). Algorithm 1 shows a formal description of FP_Θ. In the beginning, $Q(t)$ and $E(t)$ are set to all active tasks and no task, respectively,

in Steps 1 and 2. Then, via Steps 3-7, each task in $Q(t)$ is either moved to $E(t)$ or deleted. Finally, in Step 8, $E(t)$ has a set of tasks to be executed at t. Note that we describe FP_Θ for any number of cores while illustrating it for a two-core group as mentioned in Section 3. The scheduling overhead of Algorithm 1 is not significant in that FP_Θ additionally needs to sort a set of active tasks, compared to the traditional partitioned FP scheduling. Example 2 in the supplementary file, available online, illustrates how FP_Θ in Algorithm 1 operates.

Algorithm 1 FP_Θ scheduling algorithm: the following steps are triggered whenever any task in $\{\Phi_j\}_{j=1}^m$ is released or finishes its execution on any core in $\{\mathcal{S}_j\}_{j=1}^m$,

1: $Q(t) \leftarrow$ {active tasks at t on any core in $\{\mathcal{S}_j\}_{j=1}^m$}.
2: $E(t) \leftarrow \emptyset$.
3: Choose the highest-priority task in $Q(t)$ by the given fixed priority, and denote the task by τ^*.
4: $Q(t) \leftarrow Q(t) - \tau^*$.
5: $E(t) \leftarrow E(t) \cup \tau^*$.
6: Remove each task τ' in $Q(t)$ if one of the following conditions holds:
   (a) τ' is designated on the same core as τ^*; or
   (b) (τ^*, τ') is in Θ.
7: Repeat Steps 3, 4, 5 and 6, until $Q(t) = \emptyset$.
8: Execute tasks in $E(t)$ on their designated cores.

Then, FP_Θ guarantees no concurrent execution of any pair in Θ, and the following lemma states it.

Lemma 2. *The FP_Θ scheduling algorithm guarantees that any pair of tasks in Θ is not executed at the same time.*

Proof. This trivially holds by Step 6(b) of Algorithm 1.

4.2 Schedulability Analysis of FP_Θ

Using Lemma 2, we address issue 1 in Section 3. However, we need to guarantee no job will miss its deadline (i.e., issue 2 in Section 3). For this, we first derive some properties of FP_Θ. Based on these properties, we develop a schedulability test of FP_Θ. This test will be used to find Θ that minimizes $\mathcal{B}$, the "design-time" chip-level peak power consumption (i.e., issue 3 in Section 3), to be presented in Section 5. Note that while Algorithm 1 is for online operation, other parts including the schedulability test and Section 5 work offline.

To determine whether a given task set is schedulable or not, we use the response time analysis which has been widely used for schedulability analysis (e.g., [11]). In this analysis, we calculate the maximum duration between the release time and the finishing time of any job of each task (i.e., called the *task response time*). To do this, we calculate the maximum duration for a task's job to be blocked by the jobs of higher-priority tasks. Then, the task's response time is upper-bounded by the sum of its execution time and the maximum blocking time. Finally, a task set is deemed schedulable if the upper-bound of the response time of each task is not greater than its relative deadline. We also follow this to develop a schedulability test of FP_Θ.

Let $\mathcal{H}_k$ be a set of higher-priority tasks than τ_k, designated on the same core, and Γ_k be a set of tasks that can block the execution of τ_k's jobs. Then, under the traditional partitioned FP scheduling algorithm, $\Gamma_k = \mathcal{H}_k$ holds since jobs of any higher-priority task on the same core can block the execution of a job of τ_k while jobs of any other task cannot. Then, the following lemma provides the properties of Γ_k for the traditional partitioned FP scheduling algorithm.

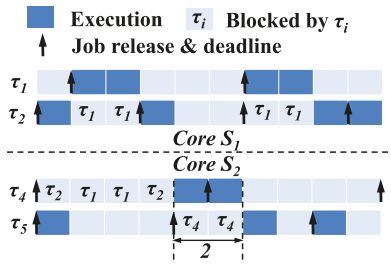


Fig. 2. The maximum blocking time of τ_5 by tasks in $\Gamma_5 = \{\tau_4\}$ in an interval of length 2; we do not include τ_3 since its execution is irrelevant to τ_5.

Lemma 3. *The traditional partitioned FP scheduling algorithm has the following properties:*

P1: If $\tau_i \in \Gamma_k$, then $\Gamma_i \subset \Gamma_k$.
P2: The maximum blocking time of τ_k by tasks in Γ_k in an interval of length l is upper-bounded by

$$\sum_{\tau_i \in \Gamma_k} \left\lceil \frac{l}{T_i} \right\rceil \cdot C_i. \tag{1}$$

Proof. Since a task is blocked by only higher-priority tasks on the same core, P1 trivially holds.

P2 is implicitly used in [11], and follows from the fact that the synchronous release of a task with higher-priority tasks is the critical instant [10]. The proof is also given in Lemma 4 to be presented later, and the lemma is a generalization of P2.

However, under FP_Θ, a job of τ_k can be blocked by the jobs of a higher-priority task τ_i designated on a different core, if Θ includes a pair of (τ_i, τ_k). Let $\mathcal{F}_k$ denote a set of higher-priority tasks (than τ_k) which belong to Θ as a pair with τ_k. Then, under FP_Θ, $\Gamma_k = \mathcal{H}_k \cup \mathcal{F}_k$ holds.

The two properties of the traditional partitioned FP scheduling algorithm presented in Lemma 3 do not necessarily hold for FP_Θ as shown in Fig. 2 in this paper and Example 3 in the supplementary file (available online).

As shown in Example 3, P1 and P2 in Lemma 3 do not hold for FP_Θ. However, P2 can hold for FP_Θ under some conditions related to P1, as stated in the following lemma.

Lemma 4. *Let Γ'_k denote $\{\tau_i \in \Gamma_k | \Gamma_i \subset \Gamma_k\}$. Then, the maximum blocking time of τ_k by tasks in Γ'_k in an interval of length l is upper-bounded by*

$$\sum_{\tau_i \in \Gamma'_k} \left\lceil \frac{l}{T_i} \right\rceil \cdot C_i. \tag{2}$$

Proof. We consider two cases: (a) any $\tau_i \in \Gamma'_k$ does not have its *carry-in* job[1] in an interval of length l; and (b) some $\tau_i \in \Gamma'_k$ does.

In Case (a), there are at most $\lceil \frac{l}{T_i} \rceil$ jobs of τ_i that can be executed within an interval of length l, and thus, the

1. A carry-in job in an interval means its release before the interval, but it finishes execution within the interval.

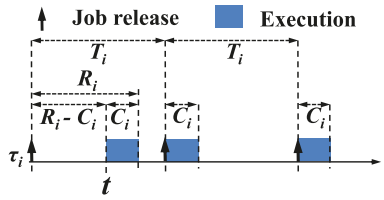


Fig. 3. An execution pattern of jobs of τ_i that maximizes the amount of execution of jobs of τ_i in an interval starting at t.

maximum blocking time of τ_k by tasks in Γ'_k in an interval of length l is upper-bounded by Eq. (2).

In Case (b), suppose $\tau_j \in \Gamma'_k$ has its carry-in job in $[t, t+l)$ of length l, and the job is released at $t-x$ ($x>0$). In $[t-x, t)$, τ_j itself or tasks in Γ_j are executed. Since $\Gamma_j \subset \Gamma_k$, the tasks ($\tau_j$ or tasks in Γ_j) belong to Γ_k. Therefore, the blocking time of τ_k by tasks in Γ'_k in $[t-x, t-x+l)$ is not smaller than that in $[t, t+l)$. This interval shift can be repeated until there is no carry-in job of tasks in Γ'_k in an interval of interest. Then, the blocking time of τ_k by tasks in Γ'_k in the final interval is not smaller than that in the original interval. Since there is no carry-in job in the final interval, this belongs to Case (a).

Using Lemma 4, we can upper-bound the blocking time of τ_k by tasks in $\{\tau_i \in \Gamma_k | \Gamma_i \subset \Gamma_k\}$. However, how to handle the other tasks in Γ_k still remains open. To address this issue, in the following lemma we present an upper bound on the blocking time of τ_k by any $\tau_i \in \Gamma_k$.

Lemma 5. *The amount of execution of τ_i's jobs in an interval of length l is upper-bounded by*

$$\left\lceil \frac{l + R_i - C_i}{T_i} \right\rceil \cdot C_i, \tag{3}$$

where R_i is an upper bound of τ_i's response time.

Proof. We consider the following execution pattern of τ_i which maximizes the amount of execution of jobs of τ_i in an interval starting at t, as shown in Fig. 3. Given an interval $[t, t+l)$ of length l, the first job of τ_i starts its execution at t and ends at $t+C_i$. Here $t+C_i-R_i$ is the job's release time, meaning that the job's response time is R_i. Thereafter, jobs of τ_i will be released and scheduled as soon as possible. In this case, the number of jobs of τ_i to be executed in $[t, t+l)$ is upper-bounded by $\lceil \frac{l+R_i-C_i}{T_i} \rceil$.

Using Lemmas 4 and 5, we develop a schedulability test of FP_Θ in the following theorem.

Theorem 1. *A task set $\Phi \triangleq \{\Phi_j\}_{j=1}^m$ is schedulable by FP_Θ on their designated core $\{\mathcal{S}_j\}_{j=1}^m$, if $R_k \le D_k$ holds for all $\tau_i \in \Phi$, where $R_k = R_k^x$ when the following formula converges (i.e., $R_k^{x+1} = R_k^x$) starting from $R_k^0 = C_k$:*

$$R_k^{x+1} \leftarrow C_k + \sum_{\tau_i \in \Gamma_k} \left\lceil \frac{R_k^x + \delta_{(k,i)}}{T_i} \right\rceil \cdot C_i, \tag{4}$$

where

$$\delta_{(k,i)} = \begin{cases} 0, & \text{if } \Gamma_i \subset \Gamma_k, \\ R_i - C_i, & \text{otherwise.} \end{cases} \tag{5}$$

Note that R_k is calculated from the highest-priority task to the lowest-priority one.

Proof. By Lemma 4, the maximum blocking time of τ_k by tasks in $\Gamma'_k \triangleq \{\tau_i \in \Gamma_k | \Gamma_i \subset \Gamma_k\}$ in an interval of length R_k^x is upper-bounded by $\sum_{\tau_i \in \Gamma'_k} \lceil \frac{R_k^x}{T_i} \rceil \cdot C_i$. By Lemma 5, the maximum blocking time of τ_k by a task $\tau_i \in \Gamma_k$ in an interval of length R_k^x is upper-bounded by $\lceil \frac{R_k^x + R_i - C_i}{T_i} \rceil \cdot C_i$. Therefore, the maximum blocking time of τ_k by tasks in Γ_k is upper-bounded by $\sum_{\tau_i \in \Gamma_k} \lceil \frac{R_k^x + \delta_{(k,i)}}{T_i} \rceil \cdot C_i$. This means, if the RHS of Eq. (4) is equal to R_k^x, τ_k finishes its execution within R_k^x in any case. Therefore, R_k resulting from this theorem is an upper-bound of the response time of τ_k's jobs.

We show in the supplementary file, available online, how Theorem 1 operates in Example 4.

5 System Design with the Lowest Chip-Level Peak Power Consumption

We would now like to minimize the chip-level peak power consumption using FP_Θ and its schedulability analysis developed in Section 4. For this, we first derive some properties of the schedulability analysis of FP_Θ. Then, we develop a method to find Θ that minimizes the chip-level peak power consumption at design time, and prove its correctness and optimality.

Lemma 6. *FP_Θ in Algorithm 1 and its schedulability analysis in Theorem 1 are generalizations of the traditional partitioned FP scheduling algorithm (i.e., the uniprocessor FP scheduling algorithm [10] for each core) and its exact schedulability analysis (i.e., the exact schedulability analysis of uniprocessor FP scheduling algorithm [11] for each core), respectively.*

Proof. We show that FP_Θ and its schedulability analysis are equivalent to their counterparts when $\Theta = \emptyset$.

If $\Theta = \emptyset$, then the FP_Θ scheduling algorithm is trivially equivalent to the traditional partitioned FP scheduling algorithm. Any task $\tau_i \in \Gamma_k$ thus satisfies $\Gamma_i \subset \Gamma_k$ by Lemma 3, implying that $\delta_{(k,i)}$ in Theorem 1 is always 0 regardless of k and i. Therefore, Theorem 1 is equivalent to the exact schedulability test of the uniprocessor FP scheduling algorithm [11] for each core.

Also, the schedulability analysis of FP_Θ in Theorem 1 is sustainable with respect to Θ, as stated in the following lemma.

Lemma 7. *Suppose $\Theta' \subset \Theta''$. If the schedulability test in Theorem 1 sees a task set unschedulable by FP_Θ with $\Theta = \Theta'$, then the test also sees the task set unschedulable by FP_Θ with $\Theta = \Theta''$. If the test sees a task set schedulable by FP_Θ with $\Theta = \Theta''$, then the test also sees the task set schedulable by FP_Θ with $\Theta = \Theta'$.*

Proof. Suppose $\Theta' \subset \Theta''$. Then, it is trivial that for any task τ_i, Γ_i when $\Theta = \Theta'$ is a subset of Γ_i when $\Theta = \Theta''$. Therefore, for any task τ_i, R_i when $\Theta = \Theta'$ is always smaller than or equal to R_i when $\Theta = \Theta''$.

Using the properties in Lemmas 6 and 7, we can efficiently find Θ which minimizes the design parameter $\mathcal{B}$ such that (i) the chip-level peak power consumption does not exceed $\mathcal{B}$ and (ii) no job will miss its deadline. The basic idea is to find such Θ using a binary search. That is, if FP_Θ

with $\Theta = [\Theta^*(1), \ldots, \Theta^*(y)]$ for given $y = x$ is deemed schedulable (unschedulable) by Theorem 1, we need not test $\Theta = [\Theta^*(1), \ldots, \Theta^*(y)]$ for given $y = x'$ if $x' < x$ ($x' > x$) thanks to Lemma 7. Algorithm 2 describes how to find such Θ, and requires only $O(\log(|\Theta^*|))$ iterations in Steps 7–10 since we use a binary search.

Algorithm 2 LOWESTPP $(\{\Phi_1, \Phi_2\}, \{\mathcal{S}_1, \mathcal{S}_2\})$: an algorithm of finding Θ that minimizes $\mathcal{B}$, the design-time chip-level peak power consumption

1: Construct a list Θ^* of all pairs, each of which is a combination of tasks on different cores (i.e., a pair is $(\tau_{i_1} \in \Phi_1, \tau_{i_2} \in \Phi_2)$), and sort Θ^* in descending order of the sum of the peak power consumption of tasks in each pair.
2: Remove pairs in Θ^*, whose sum of the peak power (B_i) of tasks in each pair is smaller than or equal to B_{max}.
3: Consider two extreme cases as follows:
  (a) If Theorem 1 deems $\{\Phi_j\}_{j=1}^m$ schedulable by FP_Θ with $\Theta = \emptyset$, return INFEASIBLE.
  (b) If Theorem 1 deems $\{\Phi_j\}_{j=1}^m$ schedulable by FP_Θ with $\Theta = \Theta^*$, return FEASIBLE with $\Theta = \Theta^*$ (in this case, $\mathcal{B}$ is set to B_{max}).
4: succ $\leftarrow 0$.
5: fail $\leftarrow |\Theta^*| + 1$.
6: curr $\leftarrow \lfloor(\text{succ} + \text{fail})/2\rfloor$.
7: **while** fail $>$ succ $+ 1$ **do**
8: If Theorem 1 deems $\{\Phi_j\}_{j=1}^m$ schedulable by FP_Θ with $\Theta = [\Theta^*(1), \cdots, \Theta^*(\text{curr})]$, succ $\leftarrow$ curr. Otherwise, fail $\leftarrow$ curr.
9: curr $\leftarrow$ (succ $+$ fail)$/2$.
10: **end while**
11: Return FEASIBLE with $\Theta = [\Theta^*(1), \cdots, \Theta^*(\text{succ})]$ (in this case, $\mathcal{B}$ is set to the sum of peak power of tasks in $\Theta^*(\text{succ}+1)$).

Example 5 in the supplementary file, available online, shows how Algorithm 2 works.

We now prove the correctness of Algorithm 2 and state its optimality in the following two lemmas.

Lemma 8 (Correctness of Algorithm 2). *If a task set is schedulable by the traditional partitioned FP scheduling algorithm, the task set is also schedulable by FP_Θ with Θ from Algorithm 2.*

Proof. In Step 3(a), we check if the task set is schedulable by the traditional partitioned FP scheduling algorithm (i.e., FP_Θ with $\Theta = \emptyset$). If schedulable, Algorithm 2 finally returns FEASIBLE in either Step 3(b) or 11(b). Then, FP_Θ with Θ from Algorithm 2 is schedulable by Theorem 1, which is tested in Step 3(b) or 8.

Lemma 9 (Optimality of Algorithm 2). Θ *produced by Algorithm 2 minimizes $\mathcal{B}$ (the design-time chip-level peak power consumption) with respect to the FP_Θ schedulability analysis in Theorem 1.*

Proof. Let Θ' and Θ'' denote Θ produced by Algorithm 2 and any arbitrary Θ, respectively. Let $\mathcal{B}'$ and $\mathcal{B}''$ denote $\mathcal{B}$ corresponding to Θ' and Θ'', respectively. We prove $\mathcal{B}' \leq \mathcal{B}''$ by contradiction.

Suppose $\mathcal{B}' > \mathcal{B}''$. We consider two cases: (a) $\mathcal{B}'' = B_{max}$ and (b) $\mathcal{B}'' > B_{max}$. In Case (a), it holds $\Theta'' \supset \Theta^*$; otherwise, $\mathcal{B}'' > B_{max}$. Then, by Lemma 7, Theorem 1 deems the task set schedulable by FP_Θ with $\Theta = \Theta^*$. Therefore, Step 3(b) of Algorithm 2 returns FEASIBLE with $\Theta = \Theta^*$ in which $\mathcal{B}' = B_{max}$ holds. This is a contradiction of $\mathcal{B}' > \mathcal{B}''$.

In Case (b), let $\Theta^*(y)$ denote a pair of tasks whose sum of the peak power consumption (i.e., sum of B_i) is equal to $\mathcal{B}''$. Then, $\Theta^*(x) \in \Theta'', \forall x = 1, \ldots, y-1$. Similar to Case (a), by Lemma 7, Theorem 1 deems the task set schedulable by FP_Θ with $\Theta = [\Theta^*(1), \ldots, \Theta^*(y-1)]$, meaning that Step 8 of Algorithm 2 sets $succ \leftarrow y - 1$. Therefore, in Step 11, $succ \geq y - 1$, meaning that $\mathcal{B}'$ is at most the sum of the peak power consumption of tasks in $\Theta^*(y)$ (i.e., equal to $\mathcal{B}''$) by Step 11. This contradicts the condition $\mathcal{B}' > \mathcal{B}''$.

6 Evaluation

In this section, we evaluate the effectiveness of the proposed design via simulation with a set of synthetic tasks running on a target multi-core chip. We first describe how the task sets are generated with realistic peak power parameters, and then demonstrate quantitative improvements by the proposed method.

6.1 Task Set Generation

We generated task sets using the technique in [17], which has been widely used in the real-time systems community (see the supplementary file, available online, for details).

Since we logically divide the cores in the chip by groups of two cores and apply our method to each group independently, the total improvement for a multi-core chip is just the sum of that when the proposed method is applied to each group. Therefore, we only show the result of a two-core chip model (i.e., $m = 2$); we generate 20,000 two-core chip task sets (i.e., Φ), by arbitrarily choosing two per-core task sets among the generated 20,000 ones without redundancy.

Then, for each per-chip task set, we determine the global priority of each task (P_i) according to the RM (Rate Monotonic) and DM (Deadline Monotonic) policies for implicit and constrained deadline task sets, respectively, i.e., the smaller period (T_i) or relative deadline (D_i), the higher the priority. Note that RM and DM are known as the optimal static priority assignment, respectively, for implicit and constrained deadline task sets in uniprocessor platforms.

To determine the peak power of each task (B_i), we first modeled each core as a core in Intel Xeon Processor, 65nm CMOS technology, operating at 3.4 GHz [18]. To obtain the realistic power dissipation on the target core model, we ran several embedded benchmark programs with their various inputs in MiBench benchmark suites [15] on the Wattch power simulator [16], and obtained the sampled dynamic power dissipation during the execution of each application as shown in Fig. 1a. For each task, 10 inputs were randomly selected as task inputs to obtain the sampled power traces on different execution paths. Because the Wattch simulator does not support the selected target microprocessor and technology and the leakage power model, we used the McPAT power simulator [19] to model the leakage power and the peak power consumption on the selected target core. Then, the linear scaling methods in [20] were applied to the power traces obtained from the Wattch simulator, assuming that the power variation over time on the Wattch simulator is the same as the power variation on the target core.

TABLE 4
The Measured Task-Level Peak Power Consumption of Mibench Testbench Applications (W)

bitcnts	basicmath	dijkstra	fft	gsm	patricia
23.35	22.47	28.99	23.61	27.13	23.84
qsort	rijndael	sha	susan	jpeg	adpcm
33.09	28.81	30.77	32.43	29.19	20.74

So, based on the measurement shown in Table 4, we set the peak power consumption of each task between the minimum and maximum values of the 12 tasks in Table 4, i.e., B_i was uniformly distributed in $[20.74, 33.09)$. Note that while we use Intel Xeon processor, such a peak power difference between different applications is a general phenomenon and hence applicable to other processors. For example, different applications with a 2.0 GHz Power PC 970 processor have different peak power (roughly ranging from 28 to 50 W), as shown in [21, Fig. 10]; likewise, Intel Sandy Bridge and PXA255 processors have different peak power depending on different underlying applications [22], [23]. Since the variation of B_i depends on processors and applications and schedulability varies greatly with the variation of B_i, rather than the number of different tasks, we fixed the 12 tasks and then explored two more settings for the distribution of B_i for each task set: half variation ($B_i \in [20.74, 26.92)$) and double variation ($B_i \in [20.74, 45.55)$).

6.2 Reduction of Chip-Level Peak Power Consumption

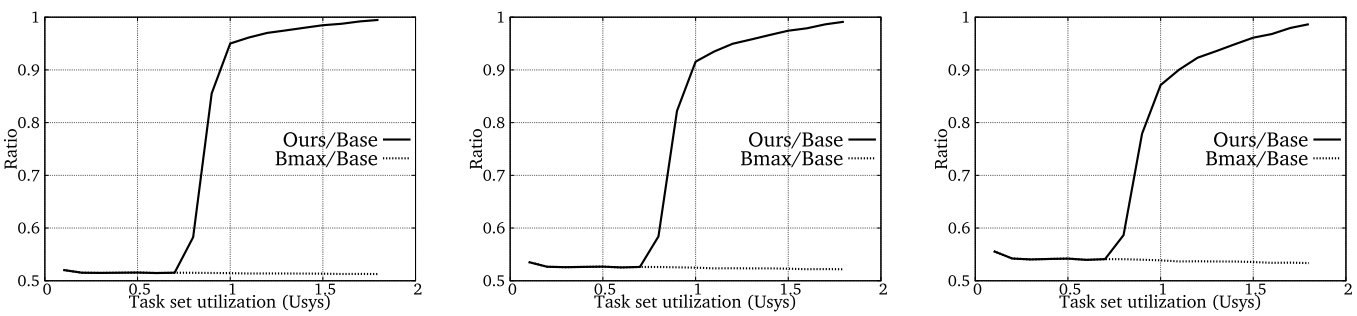
In this section, we compare the design-time chip-level peak power consumption of the proposed method (denoted by "$Ours$") with that without controlling concurrent task execution (denoted by "$Base$"), which is the sum of the largest task-level peak power consumption on each core (i.e., $\max_{\tau_i \in \Phi_1} B_i + \max_{\tau_i \in \Phi_2} B_i$). In other words, $Base$ is the design-time chip-level peak power consumption by any traditional partitioned scheduling algorithm, because without controlling concurrent task execution, two tasks that yield the largest peak power in each core eventually execute at the same time. We also include B_{max} as a reference (denoted by $Bmax$); since B_{max} means the maximum of the task-level peak power consumption on the core (i.e., $B_{max} \triangleq \max_{j=1}^{2} \max_{\tau_i \in \Phi_j} B_i$), no method can make the design-time chip-level peak power consumption lower than B_{max} even if the timing requirements are not guaranteed (assuming every task will eventually be executed).

Figs. 4a, 4b and 4c show the ratios of $Ours$ to $Base$, and of $Bmax$ to $Base$ for implicit deadline task sets in which B_i is uniformly distributed within intervals with small, medium and large variations, respectively. In these figures, each plot shows an average ratio for task sets with utilization ($U_{sys} \triangleq \sum_{\tau_i \in \Phi} C_i/T_i$) in $[U_{sys}, U_{sys} + 0.1)$. For example, it is shown in Fig. 4c that the average ratio of $Ours$ to $Base$ is 0.871 when $1.0 \le U_{sys} < 1.1$. This means that the proposed method reduces the design-time chip-level peak power consumption by 12.9 percent on average, compared to the base approach which does not control concurrent execution. Therefore, a smaller ratio implies a larger reduction of the peak power by the proposed method. We did not include results for constrained deadline task sets since their behavior is similar to that of implicit deadline task sets.

First, if we focus on $U_{sys} > 1.0$, $Ours/Base$ grows as U_{sys} increases. This is consistent with an intuition: it is more difficult to add constraints of concurrent task execution as the system gets overloaded. While our method can reduce the design-time chip-level peak power consumption up to 12.9 percent compared to $Base$, the amount of reduction varies with the variation of B_i; a larger variation results in a larger reduction. For example, when $1.0 \le U_{sys} < 1.1$, $Ours/Base = 0.950$, 0.916 and 0.871, respectively, in Figs. 4a, 4b and 4c (i.e., 5.0 percent, 8.4 percent and 12.9 percent reduction, respectively). We expect a more variation of B_i to yield a more reduction.

The second observation is that $Ours/Base$ is the same as $Bmax/Base$ when $U_{sys} \le 0.69$. Since $\lim_{n\to\infty} n \cdot (2^{\frac{1}{n}} - 1) \approx 0.69$ is the sufficient utilization bound of RM on a single-core chip [10], a certain optimal method can schedule all task sets with $U_{sys} \le 0.69$ on a two-core chip without concurrent execution of tasks on different cores. The proposed method achieves the same optimality, so $Ours$ is equal to $Bmax$.

One may wonder why the behavior of $Ours/Base$ is so different when $U_{sys} \le 0.69$ and $U_{sys} > 1.0$. This is because (i) it is inevitable for some pair of tasks on different cores to execute at the same time when $U_{sys} > 1.0$, while (ii) it is possible to avoid all concurrent task executions on different cores when $U_{sys} \le 0.69$. For example, consider a task set in Fig. 4a with $\max_{\tau_i \in \Phi_1} B_i = 26$, $\max_{\tau_i \in \Phi_2} B_i = 25$, $\min_{\tau_i \in \Phi_1} B_i = 22$, and $\min_{\tau_i \in \Phi_2} B_i = 21$. Then, $Base = 26 + 25 = 51$ and $Bmax = 26$ hold. If some pair of tasks on different cores should be executed concurrently (in case of $U_{sys} > 1.0$), $Ours$ is at least $22 + 21 = 43$. Thus, there is inherently a large gap between the lower bound of $Ours/Base$ for



(a) B_i is uniformly distributed in $[20.74, 26.92)$ (b) B_i is uniformly distributed in $[20.74, 33.09)$ (c) B_i is uniformly distributed in $[20.74, 45.44)$

Fig. 4. Ratio of the chip-level peak power consumption of no control of concurrent task execution to that of our method.

$U_{sys} \leq 0.69$ and $U_{sys} > 1.0$ (i.e.,$26/51 = 0.51$ versus $43/51 = 0.84$). For this reason, when $0.69 < U_{sys} \leq 1.0$ in which task sets satisfying (i) and those satisfying (ii) are mixed, we observe a sharp increase of $Ours/Base$.

7 Related Work

A large body of research has been done on power/energy management. For embedded real-time systems, most prior research has mainly focused on energy/temperature management for uniprocessor or multi-core platforms without considering instantaneous peak power consumption. Fisher et al. [3] analyzed the global thermal-aware scheduling of sporadic tasks on a multi-core chip to minimize the peak temperature. Based on dynamic voltage and frequency scaling and the heat transfer model of the chip, the optimal speed of cores is derived to meet both the thermal and the timing constraints. Likewise, Wang et al. [4] proposed a scheduling analysis for minimization of the energy consumption on a multi-core chip under the thermal constraint without violating the tasks' timing constraints. However, because the scheduling decisions in these algorithms were intended to reduce the energy or temperature by controlling the average power consumption, it is not easy to modify/adapt them to real-time systems in which both the timing constraints and the chip-level peak power constraints should be guaranteed at design time.

For general-purpose computing systems, how to guarantee the chip-level power constraints has been studied in recent years, but most of them have not dealt with the timing constraints. Isci and Martonosi [6] proposed the global power-management schemes using DVFS which minimize the peak power consumption at runtime subject to some performance loss constraints. Meng et al. [8] also explored power management by reconfiguring the processor speed or the size of cache adaptively at runtime to guarantee the peak power constraints. However, these global runtime schemes treated the peak power requirement as a soft constraint, whereas we treat it as a hard constraint since our goal is to reduce the peak power consumption design time. Similarly, Kontorinis et al. [7] proposed a reconfigurable micro-architecture to meet the peak power constraints. By adaptively changing the size of micro-blocks of the processor at runtime, the peak power constraint violation was avoided at the expense of considerable reconfiguration overhead. However, this adaptive microprocessor cannot be used for real-time systems either, due to unpredictable task execution times during a processor reconfiguration. In contrast with these peak power control techniques for general-purpose systems, our approach relies only on a software-layer scheduling algorithm without any hardware support to reduce the chip-level peak power consumption at design time while guaranteeing the timing requirements.

8 Conclusion

In this paper, we have addressed the problem of controlling chip-level peak power consumption on a multi-core chip in embedded real-time systems. We have developed a new scheduling algorithm FP_Θ and its schedulability analysis, and demonstrated its ability to effectively reduce the chip-level peak power consumption at design time without any hardware support.

While this is the first peak power-aware scheduling that addresses the design problem of controlling peak power consumption on a multi-core chip in embedded real-time systems, several directions we can take to improve its effectiveness. First, we may relax some of the assumptions used. For example, instead of assuming global task priority is given, we may develop and incorporate optimal priority assignment (OPA). The traditional approach of efficient OPA [24] cannot be applied to our current method, thus needing an efficient way of OPA. More broadly, we may also consider relaxation of the assumption of a partitioned, fixed-priority approach. It would then be interesting to develop a new global scheduling algorithm that can further reduce the peak power consumption beyond FP_Θ. Second, we may consider a multi-core chip in which hardware implementation for power management such as DVFS is supported. We may then compare the performance of a hardware control method (e.g., DVFS) with that of a software control (i.e., scheduling) method such as our present work, and develop a method that exploits both hardware and software controls.

Acknowledgments

The work reported in this paper was supported in part by the NSF under grants CNS-0930813 and CNS-1138200. The work reported in this paper was also supported in part by National Research Foundation of Korea Grant funded by the Korean Government (Ministry of Education, Science and Technology) [NRF-2011-357-D00186].

References

[1] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP)*, pp. 89-102, 2001.

[2] J.-J. Chen, S. Wang, and L. Thiele, "Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints," *Proc. 15th IEEE Real-Time Technology and Applications Symp. (RTAS)*, pp. 141-150, 2009.

[3] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-Aware Global Real-Time Scheduling on Multicore Systems," *Proc. 15th IEEE Real-Time Technology and Applications Symp. (RTAS)*, pp. 131-140, 2009.

[4] S. Wang, J.-J. Chen, Z. Shi, and L. Thiele, "Energy-Efficient Speed Scheduling for Real-Time Tasks under Thermal Constraints," *Proc. 15th IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 201-209, 2009.

[5] M.D. Pant, P. Pant, and D.S. Wills, "On-Chip Decoupling Capacitor Optimization Using Architectural Level Prediction," *IEEE Trans. Very Large Scale Integration Systems*, vol. 10, no. 3, pp. 319-326, June 2002.

[6] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proc. 36th IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 93-104, 2003.

[7] V. Kontorinis, A. Shayan, R. Kumar, and D. Tullsen, "Reducing Peak Power with a Table-Driven Adaptive Processor Core," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 189-200, 2009.

[8] K. Meng, R. Joseph, R.P. Dick, and L. Shang, "Multi-Optimization Power Management for Chip Multiprocessors," *Proc. 17th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT)*, pp. 177-186, 2008.

[9] B. Calhoun, F. Honore, and A. Chandrakasan, "A Leakage Reduction Methodology for Distributed MTCMOS," *IEEE J. Solid-State Circuits*, vol. 39, no. 5, pp. 818-826, May 2004.

[10] C. Liu and J. Layland, "Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.
[11] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach," *Proc. IEEE Workshop on Real-Time Operating Systems and Software*, pp. 133-137, May. 1991.
[12] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," PhD dissertation, Massachusetts Inst. of Technology, 1983.
[13] S. Kodase, S. Wang, Z. Gu, and K.G. Shin, "Improving Scalability of Task Allocation and Scheduling in Large Distributed Real-Time Systems Using Shared Buffers," *Proc. Ninth IEEE Real-Time Technology and Applications Symp. (RTAS)*, pp. 181-188, 2003.
[14] R. McGowen, C. Poirier, C. Bostak, J. Ignowski, M. Millican, W. Parks, and S. Naffziger, "Power and Temperature Control on a 90-nm Itanium Family Processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 229-237, Jan. 2006.
[15] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Fourth IEEE Ann. Workshop on Workload Characterization*, pp. 3-14, 2001.
[16] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA)*, pp. 83-94, 2000.
[17] T.P. Baker, "Comparison of Empirical Success Rates of Global vs. Partitioned Fixed-Priority and EDF Scheduling for Hand Real Time," Technical Report TR-050601, Dept. of CS, Florida State Univ., 2005.
[18] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang, "A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache," *Proc. IEEE Int'l Solid State Circuits Conf. (ISSCC)*, pp. 315-324, 2006.
[19] S. Li, J.H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 469-480, 2009.
[20] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-ISA Heterogeneous Multi-Core Architectures: the Potential for Processor Power Reduction," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 81-92, 2003.
[21] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems," *Proc. 19th Ann. Int'l Conf. Supercomputing*, pp. 293-302, 2005.
[22] J. Dongarra, H. Ltaief, P. Luszczek, and V.M. Weaver, "Energy Footprint of Advanced Dense Numerical Linear Algebra Using Tile Algorithms on Multicore Architectures," *Proc. Second Int'l Conf. Cloud and Green Computing*, pp. 274-281, 2012.
[23] G. Contreras, M. Martonosi, J. Peng, G.-Y. Lueh, and R. Ju, "The XTREM Power and Performance Simulator for the Intel Xscale Core: Design and Experiences," *ACM Trans. Embedded Computing Systems*, vol. 6, no. 1, pp. 1-25, 2007.
[24] N. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times," Technical Report YCS164, Dept. of CS, Univ. of York, 1991.

Jinkyu Lee is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University, South Korea, where he joined in 2014. He received the BS, MS, and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2004, 2006, and 2011, respectively. He has been a research fellow/visiting scholar in the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

Buyoung Yun received the BS degree in electrical engineering from POSTECH, Korea, in 2005, and the MS degree from the University of Michigan in 2008. He is currently working toward the PhD degree at the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. Since 2008, he has been a research assistant at the Real-Time Computing Laboratory in the EECS Department, University of Michigan. His research interests include fault-tolerant real-time systems, power and temperature management in real-time and embedded systems. He has been awarded the Korea Institute of Energy Technology Scholarship in 2006-2008 for outstanding graduate students studying abroad. He is a student member of the IEEE.

Kang G. Shin is the Kevin & Nancy O'Connor Professor of Computer Science in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His current research focuses on QoS-sensitive computing and networking as well as on embedded real-time and cyber-physical systems. He has supervised the completion of 74 PhDs, and authored/coauthored more than 800 technical articles (more than 300 of these are in archival journals), a textbook and more than 20 patents or invention disclosures, and received numerous best paper awards, including the Best Paper Awards from the 2011 ACM International Conference on Mobile Computing and Networking (MobiCom '11), the 2011 IEEE International Conference on Autonomic Computing, the 2010 and 2000 USENIX Annual Technical Conferences, as well as the 2003 IEEE Communications Society William R. Bennett Prize Paper Award and the 1987 Outstanding *IEEE Transactions of Automatic Control* Paper Award. He has also received several institutional awards, including the Research Excellence Award in 1989, Outstanding Achievement Award in 1999, Distinguished Faculty Achievement Award in 2001, and Stephen Attwood Award in 2004 from University of Michigan (the highest honor bestowed to Michigan Engineering faculty); a Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002; 2003 IEEE RTC Technical Achievement Award; and 2006 Ho-Am Prize in Engineering (the highest honor bestowed to Korean-origin engineers). He is a life fellow of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.