Energy-Efficient Scheduling of a Real-Time Task on DVFS-Enabled Multi-Cores

Wan Yeon Lee, Hyogon Kim, and Heejo Lee

CIC-CCS-TR 08-001

Korea University

Abstract

TV and video streaming on mobile devices is fast becoming reality, thanks to recent technical developments such as broadband wireless communication and packetized video services. In order to maximize the quality and lifetime of mobile video services, their energy-efficient processing is crucial. Even though processors on mobile devices are evolving into multi-core architectures, little work has been done for their energy-efficient scheduling. of real-time video tasks on a multi-core processor. In this paper, we propose an energy-efficient scheduling of a long-lived real-time video task running on DVFS-enabled multi-core platforms. The proposed scheduling minimizes the energy consumption by executing the task in parallel on an *appropriate* number of cores with the other cores powered off, and assigning as lower frequency as possible while meeting the deadline. Evaluation shows that the scheduling saves impressive amount of energy, up to 72% and 90% of energy consumed when executing the task on a single core and all cores respectively.

Index Terms

Energy efficiency, multi-core processor, dynamic voltage and frequency scaling, parallel processing, real-time video

I. Introduction

"Video on mobile" is taking mainstream as on-demand TV and video streaming services are deployed by many wireless carriers in Europe, South Korea, Japan and the U.S. As of early 2008,

W. Lee is with the Hallym University.

H. Kim and H. Lee are with the Korea University.

service offerings from carriers with more than 20 channels including live TV can be found easily and will be available in more countries. This trend is enabled and bolstered by recent technical developments such as broadband wireless communication, high-efficiency video compression and packetized video. With the increasing demand on video playing time, however, the limitation of battery life has become a pressing issue on mobile devices. The energy shortage of a mobile device relying on battery power prevents users from playing games, music and video more often [1]. Therefore, an energy-efficient solution specifically tailored to a long-running video task executed on mobile devices must be promptly explored.

In devising the proper solution, an emerging trend that demands our attention in processor design is the multi-core architecture. The physical limits of semiconductor-based microelectronics such as heat dissipation and data synchronization led to the adoption of multiple independent processing cores as a method to expedite the task execution. The multi-core architecture can exploit the parallelism of the given task better, without much architectural changes [2]. It is envisioned that processors with tens of cores will emerge, and it is believed to promise further performance and efficiency gains, especially in processing multimedia and networking applications [3]. In particular, multi-core system-on-chip processors have become increasingly popular in mobile systems because they have clear benefits in terms of performance and implementation costs [4].

Another processor design trend that combines well with the multi-core architecturing for the sake of improving energy efficiency is the advanced fine-grain or ultra fine-grain energy management with dynamic voltage and frequency scaling (DVFS) [5], [6]. It is an energy-saving technique that consists of changing the clock frequency and voltage supplied to a processor according to processing needs. The speed of the DVFS-enabled processor is typically proportional to the clock frequency, whereas the energy consumption increases along with a polynomial function of the clock frequency. The degree of this polynomial function is typically no smaller than 2, even though the exact relationship between the energy consumption rate and the clock frequency depends on the hardware [5], [6], [7]. As a consequence, putting more cores into a task while providing a lower operating frequency to them can open a rich possibility of reducing the total energy consumption for the task execution.

In this paper, we consider the problem of energy-efficient video processing on the basis of the aforementioned new trends in processor design. While the multi-core platform has been

intensively considered for energy-efficient processing of *many* real-time tasks, it has been never focused for a *single* real-time task due to its overabundant hardware. This paper addresses an issue how to exploit these overabundant resources with DVFS capability. Specifically, we propose a scheduling method designed for a single long-running real-time video task on the DVFS-enabled multi-core platform, as the video task represents the most popular and energy-demanding application for current and future mobile devices. We remark that the application of the proposal is not limited to the video tasks, however, and it should be applicable to other long-running real-time tasks.

The main contribution of this paper is to introduce an optimal off-line scheduling which minimizes the energy consumption of a single periodic real-time task by fully exploiting the architectural benefits of multiple cores and their DVFS capability. Considering the non-linear scaling property of parallel execution on various numbers of cores and the irregularly discrete energy consumption rates of available frequencies, the proposed method determines both the number of cores allocated to parallel execution and the frequency executing each computation cycle under the deadline constraint. This is analogous to the 3-D bin-packing problem of determining a triple input: number of cores, frequency value and completion time, whereas previous scheduling studies are restricted within the 2-D bin-packing problem of determining a duplex input: number of processors and completion time, or frequency value and completion time.

In order to determine the number of allocated cores and the lowest frequency meeting the deadline, the proposed method needs to know in advance the worst-case computation of the task and the speedup values of parallel execution on various numbers of cores. The worst-case computation and speedup of parallel execution can be estimated by existing analysis skills [8], [9] or profiling scheme [10]. The proposed method also requires at most one frequency transition per each image frame having deadline constraint. Simulation results show that the proposed scheduling consumes only 28% and 10% of the energy comparing with two greedy approaches: executing the task on a single core and turning off the other cores, and executing the task in parallel on all available cores, respectively.

Our work departs from previous works in one or more aspects. Many studies [5], [6], [7], [10] that minimize the energy consumption by reducing the clock frequency under various constraints exist, but these are in the context of a single processing core. Yang *et al.* [11] address an

energy-efficient intra-task scheduling of subtasks having dependency on a specific multi-core architecture, but this work does not consider the task assignment on varying numbers of cores. Our previous work [12] also addresses a greedy scheduling method on multiple cores, but it does not guarantee the minimum energy consumption. Furthermore, many of these works make generally impractical assumptions, for instance, an exact polynomial function between the energy consumption and the clock frequency, infinitely continuous clock frequency, or task preemption and reallocation on different number of cores at run-time without any additional overhead. Below, we present an energy-efficient real-time task scheduling method for mobile multi-core platforms with DVFS capability, that does not require these impractical assumptions.

The rest of the paper is organized as follows: Section II describes the system model and the definition of the problem handled in this paper. Section III explains the proposed method in detail. Section IV evaluates the proposed method through analysis and Section V gives the concluding remarks.

II. PRELIMINARY

A. Task Model

A real-time video task consists of consecutive image frames arriving from the network or being retrieved from the disk. As modern video compression schemes such as MPEG-4 and H.264 use variable bit rate coding, each video frame dictates different computation amount for the device to decode and play it. Because of the unpredictable computation amount of a frame, the device's scheduler must account for the worst case, *i.e.*, the maximum computation cycles. The processing speed of each frame is generally proportional to the clock frequency supplied to processing cores. In terms of execution time, the completion time is the required computation cycles divided by the clock frequency. The computation cycles of each frame must be completed within a given time limit, *i.e.*, deadline. Notice that the deadline depends on the type of media and coding. For instance, NTSC DVD quality MPEG-2 video can be transmitted at approximately 30 or 24 frames per second, in which cases the deadline is given by $D \approx 33.3ms$ and $D \approx 41.7ms$, respectively.

Video decoding tasks can be partitioned into multiple computational components, *e.g.*, separate groups of image pictures and disjoint partitions of each image picture in coarse grained implementation and fine grained implementation, respectively [13], [14]. These components can be

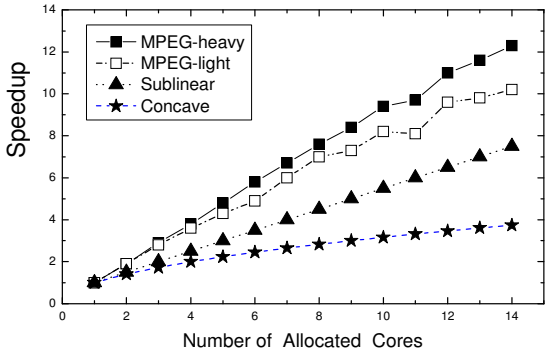

Fig. 1. Values of four speedup vectors against the number of cores allocated to the parallel execution of a task.

computed by multiple cores concurrently and independently. The speedup of parallel execution is approximately proportional to the number of allocated cores, but usually smaller than the number of allocated cores due to *inefficiency factors* of parallel execution, such as intrinsically sequential processing portion, conflict of concurrent memory accesses, and additional processing required for distributing subtasks, communication and synchronization. These inefficiency factors determine the speedup value of parallel execution on multiple cores. In order to evaluate the impact of various speedup data of parallel execution, we use four speedup models. The first two speedup models are drawn from the experimental data generated in the parallel MPEG-2 video task execution on Silicon Graphics Challenge multiprocessor with a shared memory [13]. They are 1408×960 and 352×240 resolution video tasks, and we call them *MPEG-heavy* and *MPEG-light*, respectively. We synthetically generate the other two to represent the tasks having severe inefficiency factors of parallel execution. The task, whose inefficiency factors occupy the half of the total computation, is called *Sublinear*. The speedup of Sublinear task with n allocated cores is $S[n]_{sublinear} = (n-1) \times 0.5 + 1$ for $n \geq 1$. In case that the portion of the inefficiency factors to the total computation increases along with the number of allocated cores, we call it the *Concave* task. There, $S[n]_{concave} = \sqrt{n}$ for $n \geq 1$, such that the speedup is proportional to the square root of the number of allocated cores. The four speedup models are depicted in Fig. 1. The speedup of parallel execution depends on the structure of video tasks, the hardware of multi-core processors, and multimedia softwares. Fast inter-processor communication inherited from on-chip nature of multi-core processors accelerates the speedup of parallel execution on

classic multi-processor platforms [2].

B. Processor Model

For the purpose of practical evaluation in DVFS, we use the data obtained from two well-known DVFS processors, *i.e.*, the Intel XScale and the IBM PPC405LP [7]. Given M discrete frequencies available to these processors, let us denote them as $f_1, \cdots, f_M$ in increasing order. It is assumed that an identical frequency is supplied to every core being activated in the processor. For a given frequency f_m where $1 \leq m \leq M$, the power consumption per unit time is denoted by p_m. Then, the energy consumption per each cycle is $\frac{p_m}{f_m}$. If $i < j$, then $f_i < f_j$ and $p_i < p_j$. Table I shows the available frequencies, their voltages and their power consumptions (energy consumption rates) of introduction execution[1].

TABLE I

PROCESSOR MODEL

(a) Frequency, voltage and power consumption of the XScale processor

m	1	2	3	4	5
f_m (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
p_m (mW)	80	170	400	900	1600

(b) Frequency, voltage and power consumption of the PPC405LP processor

m	1	2	3	4
f_m (MHz)	33	100	266	333
Voltage (V)	1.0	1.0	1.8	1.9
p_m (mW)	19	72	600	750

When a processor has no computation to execute, it is general to change its operation mode into the *idle status* where the processor is only executing the *no-operation* instruction with the minimum frequency f_1 [7]. The power consumption of the idle status is positive but strictly less than the value of p_1. Therefore, we denote the power consumption of the idle status as p_0 and set these values of the XScale and the PPC405LP processor models to 40 mW and 12 mW,

[1]The energy reduction of other system components such as memory and I/O is beyond the scope of this paper.

respectively [7]. For convenience, we additionally define a virtual frequency of the idle status as f_0 and set its value to zero, because its computation speed is semantically equivalent to zero.

The transition of frequency at runtime requires extra energy and time delay. The extra energy is roughly proportional to the difference between the squares of two frequencies [15]. Yuan and Nahrstedt [10] show that the extra energy and time penalties are relatively small compared to huge computation amount of multimedia tasks. The transition overhead of frequency is expected to become smaller with the advances in microprocessor design [5], [10].

C. Problem Definition

Let us consider a processor consisting of N homogeneous cores. If η cores are assigned to the parallel execution, the other $(N - \eta)$ cores are powered off to save energy. Suppose a task requires C^* cycles to be executed in the worst case, and the task should be completed within the deadline D. In case of continuous frequencies, the problem for finding a schedule consuming the minimum energy can be formulated as follows:

$$\text{Minimize } \int_0^{C'} e(f(c)) \cdot \eta(c)\, dc, \quad \text{subject to } \int_0^{C'} \frac{1}{f(c)}\, dc \leq D, \tag{1}$$

where C' denotes the reduced cycle length of C^* cycles on the parallel execution, and $f(c)$, $e(f(c))$, and $\eta(c)$ denote the supplied frequency, the energy consumption rate, and the number of allocated cores when executing c-th cycle, respectively. If there is an exact polynomial function relationship between $f(c)$ and $e(f(c))$, we can apply the Lagrange Multiplier method in order to find the schedule.

$$L(f(c), \lambda) = \textstyle\int_0^{C'} e(f(c)) \cdot \eta(c)\, dc + \lambda(D - \int_0^{C'} \frac{1}{f(c)}\, dc),$$
$$\tfrac{\partial L}{\partial \lambda} = D - \textstyle\int_0^{C'} \frac{1}{f(c)}\, dc = 0, \text{ and } \tfrac{\partial L}{\partial f(c)} = e(f(c)) \cdot \eta(c) - \lambda \cdot \tfrac{1}{f(c)} = 0.$$

Solving the above equations, we get

$$f(c) = E^{-1}(\tfrac{\lambda}{\eta(c)}) \text{ and } \textstyle\int_0^{C'} \frac{1}{E^{-1}(\frac{\lambda}{\eta(c)})}\, dc = D, \text{ where } E(f(c)) = f(c) \cdot e(f(c)).$$

From this, we can derive a deterministic $f(c)$ with regard to the value of c by calculating the function $E^{-1}(\frac{\lambda}{\eta(c)})$ and the value of λ.

In this paper, we do not consider the change of the number of cores allocated to a task during its execution, because it incurs severe overhead, such as subtask preemption, migrations

of suspended subtasks and synchronization. That is, $\eta(c)$ is replaced with a fixed natural number n ($1 \leq n \leq N$). We assume that speedup value $S[n]$ of parallel execution on n cores is given for $1 \leq n \leq N$. When C^* cycles are executed in parallel on n cores ($N - n$ cores powered off) with a speedup of $S[n]$, the task can be executed within $\lceil \frac{C^*}{S[n]} \rceil$ cycles. When there are M discrete frequencies, $f_1, \cdots, f_M$, available as described in Section II-B, $f(c)$ is replaced with the M frequencies such that $f(c) \in \{f_1, \cdots, f_M\}$. When C_k denotes the number of cycles executed with the frequency f_k for each $1 \leq k \leq M$, Eq. (1) is then reformulated as follows:

$$\begin{aligned} \text{Minimize} \quad & \textstyle\sum_{m=1}^{M} C_m \cdot \frac{p_m}{f_m} \cdot n \quad \text{for} \quad 1 \leq n \leq N \\ \text{subject to} \quad & \textstyle\sum_{m=1}^{M} \frac{C_m}{f_m} \leq D \quad \text{and} \quad \sum_{m=1}^{M} C_m = \lceil \frac{C^*}{S[n]} \rceil. \end{aligned}$$

This problem is referred to as *Minimum-Energy Multi-core Scheduling (MEMS)* and we design a scheduling algorithm, called *OPT-MEMS* algorithm, to solve MEMS in the next section.

If the completion time even when using the lowest frequency f_1 to execute all $\lceil \frac{C^*}{S[n]} \rceil$ cycles is less than the deadline, there exists slack time until the deadline. For this slack time $(D - \lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{f_1})$, the power p_0 of the idle status is consumed. We do not consider turning off the power of the activated cores during the slack time, because the time delay required to turn off/on the power [16] is relatively large, compared with the tight deadline of each image frame. This slack time is too small to justify the time delay and extra energy required to frequently turn off/on the power.

III. Energy-efficient Scheduling on Multi-core Processors

The main idea of the scheduling methods proposed in this paper comes from the following observation. If the speedup of parallel execution is proportional to the number of cores and the degree of an energy consumption function with the clock frequency is no smaller than 2, the parallel execution of a task on multiple cores can reduce the total energy required for completing the task before its deadline, as illustrated in Fig. 2. Unfortunately, allocating all available cores to the execution does not directly result in the minimum energy consumption. There are two main reasons for the consequence. One is the non-linear speedup of parallel execution with regard to the number of cores. The other is the lower bound in decreasing clock frequencies and the non-stationary relationship between available clock frequencies and their energy consumption rates. In this section, we introduce a scheduling algorithm which finds the frequency value

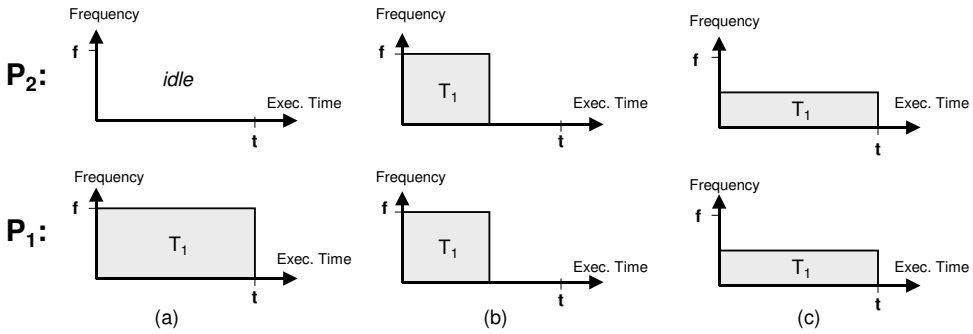

Fig. 2. Execution example of a task T_1 on two cores P_1 and P_2: (a) executing T_1 on P_1, (b) executing T_1 in parallel on P_1 and P_2, and (c) executing T_1 with low frequency in parallel on P_1 and P_2

minimizing energy consumption and meeting the deadline for each number of cores allocated to parallel execution, and selects the best number of allocated cores consuming the minimum energy among all candidate numbers.

A. *Proposed Scheduling*

The proposed method first checks whether there is a *defective frequency*, f_y, which violates the convex function of energy consumption per unit time. That is, $\frac{p_y - p_x}{f_y - f_x} > \frac{p_z - p_y}{f_z - f_y}$ for any $f_x < f_y < f_z$. According to the following Lemma 1, the defective frequency f_y is discarded henceforth.

Lemma 1: If $\frac{p_y - p_x}{f_y - f_x} > \frac{p_z - p_y}{f_z - f_y}$ for any $f_x < f_y < f_z$, the frequency f_y is not included in the solution of the MEMS problem.

Proof: See the Appendix. ■

We can select all defective frequencies by calculating $\frac{p_k - p_{k-1}}{f_k - f_{k-1}} > \frac{p_{k+1} - p_k}{f_{k+1} - f_k}$ for $1 < k < M$. The frequency f_3 in Table I (b) is a defective frequency and henceforth discarded, while there is no defective frequency in Table I (a).

Next, the proposed method searches the minimum energy consumption by examining each number of cores and selects the best one, denoted by n^*. To calculate the minimum energy for each number n of cores ($1 \leq n \leq N$), it chooses a frequency f_{m^*} which is nearest to and no smaller than $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{D}$. If $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{f_{m^*}} = D$, it is clear that processing C^* cycles with frequency f_{m^*} consumes the minimum energy within the time D for a fixed number of n. Minimizing the number of allocated cores leads to energy saving because we can turn off the power to the other

cores except the allocated cores [4] (or allocate them to other tasks).

In case of $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{f_{m^*}} < D$, the slack time $(D - \lceil \frac{C^*}{s(n)} \rceil \cdot \frac{1}{f_m})$ can be utilized to achieve further energy saving by using a combination of another frequency f_{m^*-1}. Namely, the proposed method assigns two different frequencies f_{m^*} and f_{m^*-1}, where f_{m^*} is the smallest frequency satisfying $f_{m^*} \geq \lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{D}$. It attempts to find the transition point C', which satisfies $\frac{C'}{f_{m^*}} + \frac{\lceil \frac{C^*}{S[n]} \rceil - C'}{f_{m^*-1}} = D$. The transition point C' can be found as follows:

$$C' = \frac{f_{m^*} \cdot (\lceil \frac{C^*}{S[n]} \rceil - D \cdot f_{m^*-1})}{f_{m^*} - f_{m^*-1}}.$$

Because $\frac{\lceil C' \rceil}{f_{m^*}} + \frac{\lceil \frac{C^*}{S[n]} \rceil - \lceil C' \rceil}{f_{m^*-1}} \leq D$ when $f_{m^*} > f_{m^*-1}$, it assigns the frequency f_{m^*} to execute $\lceil C' \rceil$ cycles and the frequency f_{m^*-1} to execute the rest $(\lceil \frac{C^*}{S[n]} \rceil - \lceil C' \rceil)$ cycles. This scheduling, which assigns f_{m^*} to $\lceil C' \rceil$ cycles and f_{m^*-1} to $(\lceil \frac{C^*}{S[n]} \rceil - \lceil C' \rceil)$ cycles, is called *Tight-Scheduling*.

The following theorem shows that the Tight-Scheduling method finds the schedule consuming the minimum energy among the schedules satisfying the deadline on any fixed number of allocated cores.

Theorem 1: The schedule found by Tight-Scheduling consumes the minimum energy among the schedules satisfying the deadline.

Proof: See the Appendix. ■

In the schedule found by Tight-Scheduling, the total energy consumption of each core for the time D is $(C' \cdot \frac{p_{m^*}}{f_{m^*}} + (\lceil \frac{C^*}{S[n]} \rceil - C') \cdot \frac{p_{m^*-1}}{f_{m^*-1}})$, and its average power consumption (the average energy consumption rate) for the time D is

$$\frac{C' \cdot \frac{p_{m^*}}{f_{m^*}} + (\lceil \frac{C^*}{S[n]} \rceil - C') \cdot \frac{p_{m^*-1}}{f_{m^*-1}}}{D} = p_{m^*-1} + \frac{p_{m^*} - p_{m^*-1}}{f_{m^*} - f_{m^*-1}} \cdot (\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{D} - f_{m^*-1}).$$

In summary, the value of $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{D}$ determines the values of f_{m^*} and C', and the average power consumption of each activated core. We denote the value of $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{D}$ as L_n, and refer to it as *Core Load*. We also denote the average power consumption of each activated core as a function $P(L_n)$. Then $P(L_n)$ can be formulated as follows:

$$P(L_n) = \begin{cases} p_m & \text{if } L_n = f_m \quad \text{for } 0 \leq m \leq M \\ p_m + \frac{p_{m+1} - p_k}{f_{m+1} - f_m} \cdot (L_n - f_m) & \text{if } f_m < L_n < f_{m+1} \quad \text{for } 0 \leq m \leq (M-1) \end{cases}$$

which is a *convexly increasing* and *piece-wisely linear* function of L_n, as shown in Fig. 3. If $L_n > f_M$, there is no schedule to complete this task before the deadline. Exploiting the function

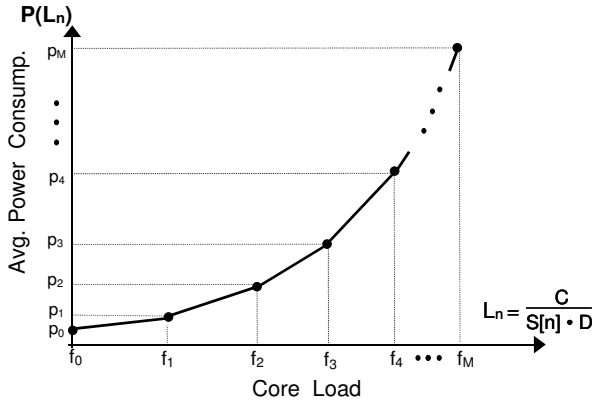

Fig. 3. Average power consumption of each core against the value of *Core Load*

of $P(L_n)$, we can easily find the best number of cores, n^*, minimizing the energy consumption for the time D. The value of n^* is determined to be

$$\min_{1 \le n \le N} \{\ P(L_n) \cdot n\ \}.$$

The following pseudo-code formally describes the proposed scheduling, called *Optimal Scheduling of the MEMS problem (OPT-MEMS)*, which minimizes the energy consumption of the real-time task by comparing the values of $P(L_n) \cdot n$ and exploiting the frequency assignment of Tight-Scheduling.

OPT-MEMS Algorithm

Input: vector of speedup values ($S[\ ]$), available cores (N), and computation cycles (C^*)

Output: allocated cores (n^*) and the frequency to execute each cycle (f_{m^*} or f_{m^*-1})

search for the number n^* minimizing the value of $P(\lceil \frac{C^*}{S[n^*]} \rceil \cdot \frac{1}{D}) \cdot n^*$ for $1 \le n^* \le N$;
select the lowest frequency f_{m^*} satisfying the constraint of $f_{m^*} \ge \lceil \frac{C^*}{S[n^*]} \rceil \cdot \frac{1}{D}$;
$C' \leftarrow \lceil \frac{f_{m^*} \cdot (\lceil \frac{C^*}{S[n^*]} \rceil - D \cdot f_{m^*-1})}{f_{m^*} - f_{m^*-1}} \rceil$ and $C'' \leftarrow \lfloor \frac{f_{m^*-1} \cdot (D \cdot f_{m^*} - \lceil \frac{C^*}{S[n^*]} \rceil)}{f_{m^*} - f_{m^*-1}} \rfloor$;
allocate n^* cores and turn off the power of the other cores;
assign frequency f_{m^*} to execute C' cycles and frequency f_{m^*-1} to execute C'' cycles;

The OPT-MEMS algorithm requires at most one frequency transition per each frame alternatively from f_{m^*} to f_{m^*-1} or from f_{m^*-1} to f_{m^*}. To minimize the number of frequency transitions,

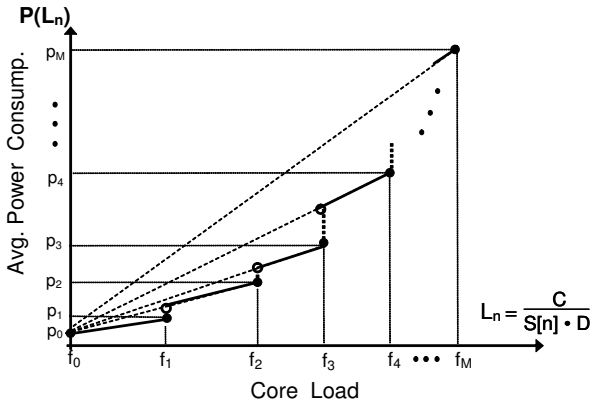

Fig. 4. In Loose-Scheduling, average power consumption of each core against the value of *Core Load*

if the previous frame completes its execution with f_{m^*-1}, the next frame starts its execution with f_{m^*-1} and completes it with f_{m^*} in a reverse manner.

B. Simplified Modification of the Proposed Method

In this subsection, let us consider the simpler execution environments where the change of the clock frequency supplied to cores is not allowed during execution. Then we need to modify the Tight-Scheduling method. The modified method, called *Loose-Scheduling* assigns only the frequency f_{m^*} to execute $\lceil \frac{C^*}{S[n]} \rceil$ cycles even if $\lceil \frac{C^*}{S[n]} \rceil \cdot \frac{1}{f_{m^*}} < D$. The Loose-Scheduling method does not use the frequency f_{m^*-1} in order to avoid the frequency transition during the execution.

In Loose-Scheduling, the average power consumption of each activated core is modified as follows:

$$P(L_n) = \begin{cases} p_m & \text{if } \ L_n = f_m \quad \text{for } \ 0 \le m \le M \\ p_0 + \frac{p_m - p_0}{f_m} \cdot L_n & \text{if } \ f_{m-1} < L_n < f_m \quad \text{for } \ 0 < m \le M. \end{cases}$$

Fig. 4 depicts the $P(L_n)$ function of the Loose-Scheduling method.

Also, the definition of the *defective frequency* is changed as follows:

$$\frac{p_x - p_0}{f_x} > \frac{p_y - p_0}{f_y} \quad \text{for any} \quad f_x < f_y.$$

We can select all defective frequencies by calculating $\frac{p_k - p_{k-1}}{f_k - f_{k-1}} > \frac{p_{k+1} - p_k}{f_{k+1} - f_k}$ for $1 < k < M$. Note that the frequency f_3 in Table I (b) is not a defective frequency, while it does in the Tight-Scheduling method. There is no defective frequency in Table I (a) and (b).

Finally, the OPT-MEMS algorithm is modified as follows:

Modified Algorithm of OPT-MEMS

Input: vector of speedup values ($S[\,]$), available cores (N), and computation cycles (C^*)

Output: allocated cores (n^*) and the frequency to execute all cycles (f_{m^*})

search for the number n^* minimizing the value of $P(\lceil \frac{C^*}{S[n^*]} \rceil \cdot \frac{1}{D}) \cdot n^*$ for $1 \leq n^* \leq N$;
select the lowest frequency f_{m^*} satisfying the constraint of $f_{m^*} \geq \lceil \frac{C^*}{S[n^*]} \rceil \cdot \frac{1}{D}$;
allocate n^* cores and turn off the power of the other cores;
assign the frequency f_{m^*} to execute $\lceil \frac{C^*}{S[n^*]} \rceil$ cycles;

IV. Performance evaluation

We compare the proposed method with two greedy methods: *single-core scheduling* and *all-cores scheduling*. The single-core method [5], [6] executes the real-time task on a single core and turns off the power of the other cores, while the all-cores method [11], [12] executes the task on all available cores. For fair comparison, these two greedy methods determine the frequency to execute each cycle on the basis of the Tight-Scheduling method similarly to the proposed method. Here, we do not take into account the time penalty and the energy penalty to change frequency at runtime, whose impact will be discussed in subsection IV-B.

A. Impacts of task workload, number of cores and speedup model

In the first set of comparison, we examine the performance effects of the relative computation amount to the deadline and the number of cores available in the system. To measure the relative computation amount to the deadline, we define the ratio of the completion time of the worst-case cycles under the highest frequency to the deadline as *Task Load*, *i.e.*, $\frac{C^*}{f_M \cdot D} \times 100 = \frac{L_1}{f_M} \times 100$.

Fig. 5 and Fig. 6 show the average power consumptions of the three methods for the deadline time on the XScale and the PPC405LP processor models, respectively. Remind that the frequency f_3 of the PPC405LP processor model is not used. In these figures, α in '$N = \alpha$' denotes the number of available cores in the system and the speedup model of the MPEG-heavy task is used. Given value of Task Load, the single-core scheduling consumes the same power regardless of the number of available cores. The proposed scheduling consumes less power as the number

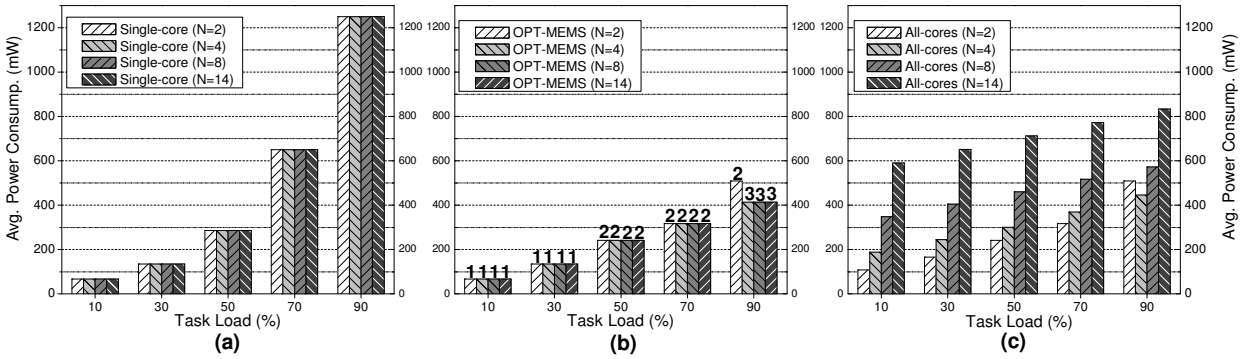

Fig. 5. On the XScale processor model, power consumption rates of (a) the single-core scheduling, (b) the proposed scheduling and (c) the all-cores scheduling

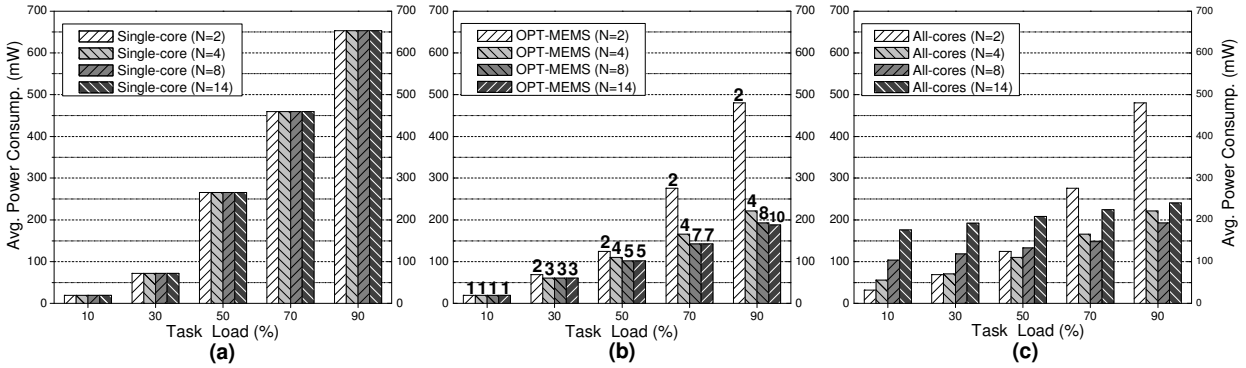

Fig. 6. On the PPC405LP processor model, power consumption rates of (a) the single-core scheduling, (b) the proposed scheduling and (c) the all-cores scheduling

of available cores increases but eventually hits the lower bound of power consumption at some number of cores. The value on the top of bar is the number of cores selected by the proposed scheduling, i.e., $n^* = \min_{1 \le n \le N} \{ P(L_n) \cdot n \}$. The all-cores scheduling consumes less power as the number of available cores becomes close to the value of n^*, but consumes more power as the number of cores increases beyond the value of n^*.

Compared with the single-core scheduling, the proposed scheduling saves significant amount of power when Task Load is heavy. For instance, in Fig. 5, the power saving ratio when 'Task Load' = 90% is 67% on 4 or more cores and that when 'Task Load' = 70% is 52% on 2 or more cores. In Fig. 6, those when 'Task Load' = 90% and when 'Task Load' = 70% are 68%

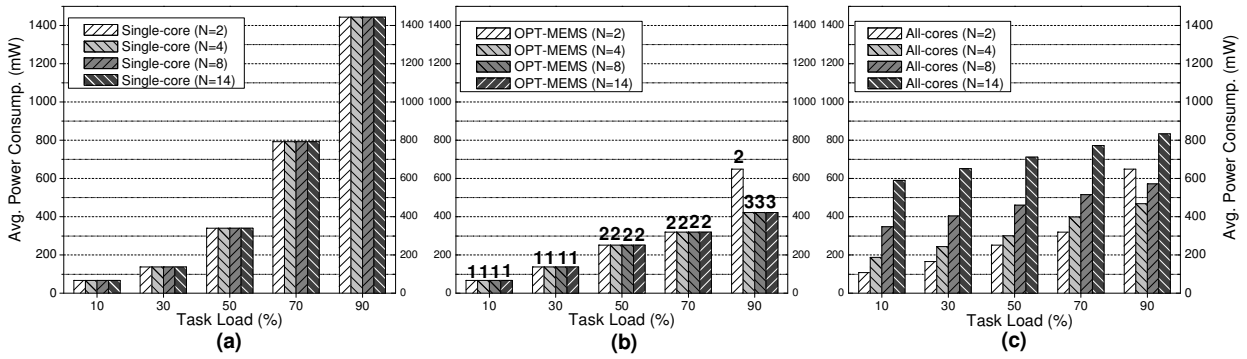

Fig. 7. On the XScale processor model, power consumption rates of the simplified versions of (a) the single-core scheduling, (b) the proposed scheduling and (c) the all-cores scheduling

on 8 or more cores and 72% on 14 core, respectively. The power consumption of the proposed scheduling is equal to that of the single-core scheduling, when Task Load is 10% or 30% in Fig. 5 and 10% in Fig. 6. Compared with the all-cores scheduling, the proposed scheduling saves significant amount of power when Task Load is light or the number of available cores is large. For instance, in Fig. 5, the power saving ratio when 'Task Load' = 10% is 89% on 14 cores and that in Fig. 6 is 90% on 14 core. And those when 'Task Load' = 70% are 59% and 37% on 14 cores in Fig. 5 and Fig. 6, respectively. The power consumption of the proposed scheduling is equal to that of the all-cores scheduling when $N = n^*$: when 'Task Load' $\geq$ 50% and $N = 2$ in Fig. 5, when 'Task Load' $\geq$ 50% and $N = 2$ or 4 in Fig. 6, and when 'Task Load' $=$ 30% and $N = 2$ in Fig. 6.

In the second set of comparison, we examine the average power consumptions of the three methods when they use the Loose-Scheduling method instead of the Tight-Scheduling method. Fig. 7 and Fig. 8 show the average power consumptions of the three methods for the deadline time on the XScale and the PPC405LP processor models, respectively. Note that the frequency f_3 of the PPC405LP processor model is used in this evaluation, because it is not a defective frequency in the Loose-Scheduling method. Compared with the single-core scheduling, the proposed scheduling saves significant amount of power when Task Load is heavy. For instance, in Fig. 7, the power saving ratio when 'Task Load' = 90% is 71% on 4 or more cores and that when 'Task Load' = 70% is 60% on 2 or more cores. In Fig. 8, those when 'Task Load' =

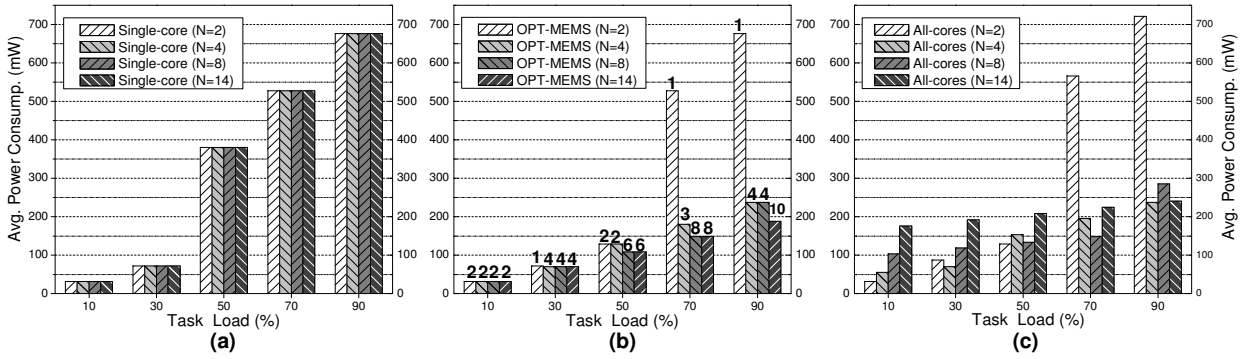

Fig. 8. On the PPC405LP processor model, power consumption rates of the simplified versions of (a) the single-core scheduling, (b) the proposed scheduling and (c) the all-cores scheduling

90% and when 'Task Load' = 70% are 73% on 10 or more cores and 72% on 8 or more core, respectively. Compared with the all-cores scheduling, the proposed scheduling saves significant amount of power when Task Load is light or the number of available cores is large. For instance, in Fig. 7, the power saving ratio when 'Task Load' = 10% is 89% on 14 cores and that in Fig. 8 is 83% on 14 core. And those when 'Task Load' = 70% are 60% and 34% on 14 cores in Fig. 7 and Fig. 8, respectively.

In the third set of comparison, we examine the performance effects of three different speedup models described in Section II-A: MPEG-light task, Sublinear task, and Concave task. For direct comparison of the proposed scheduling with the two greedy methods, we measure the ratio of the average power consumed by the proposed scheduling to that of a greedy method, referred to as *Normalized Power Consumption* (NPC). Fig. 9(a) and (b) show the values of NPC against the single-core scheduling and the all-cores scheduling, respectively. In these figures, β in '$TL = \beta$' denotes the value of Task Load and the Tight-Scheduling method is used on the XScale processor model.

In Fig. 9(a), the task with higher speedup shows a smaller NPC than the task having lower speedup on a given number of cores. If the task has heavier parallel execution overhead, *i.e.*, lower speedup value on the same number of allocated cores, its completion time becomes closer to the deadline and thus the proposed scheduling has little chance to assign a lower frequency to the cores. In Fig. 9(a), the NPC of the MPEG-light tasks reaches the bound of 34% at 4 cores

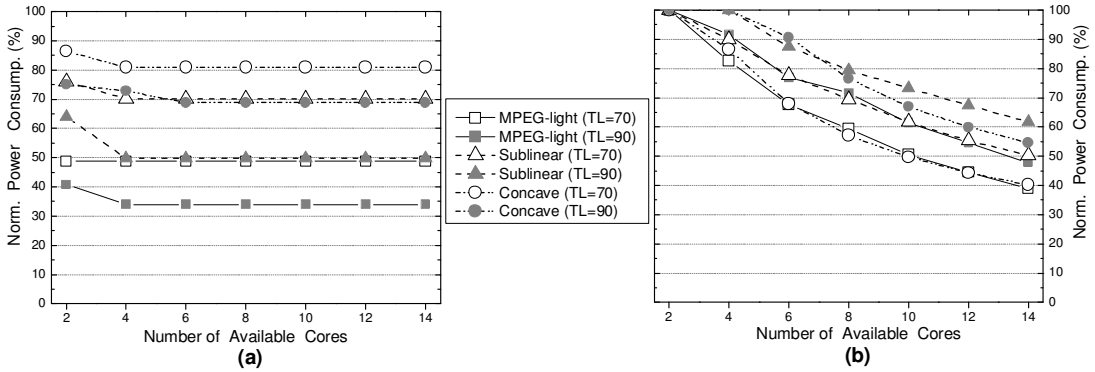

Fig. 9. Normalized Power Consumption of three speedup models on the XScale processor against (a) the single-core scheduling and (b) the all-cores scheduling

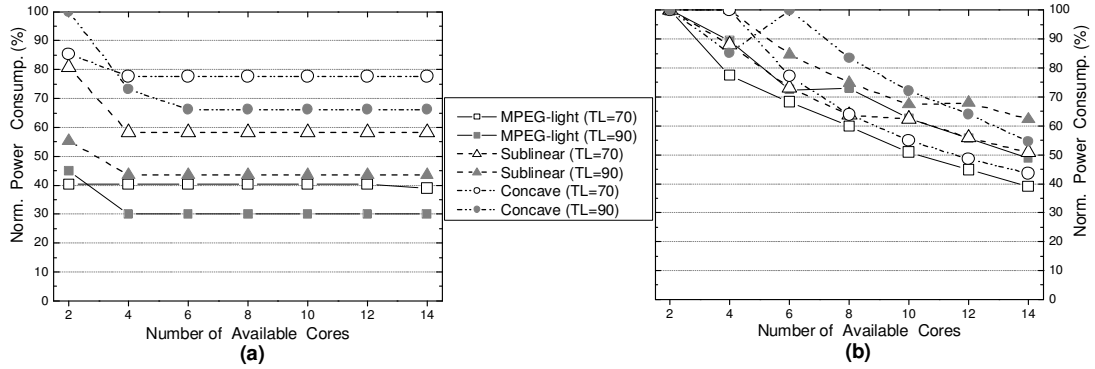

Fig. 10. In the simplified version, Normalized Power Consumption of the three speedup models against (a) the single-core scheduling and (b) the all-cores scheduling

when 'Task Load' = 90% and the bound of 48% at 2 cores. This is because the number of cores activated by the proposed method is n^* even when $N > n^*$. For the Sublinear task, the NPC reaches the bound of 70% at 4 cores when 'Task Load' = 70% and the bound of 49% at 4 cores when 'Task Load' = 90%. The Concave task hits 75% at 4 cores when 'Task Load' = 70% and 68% at 6 cores when 'Task Load' = 90%. In Fig. 9(b), the NPC of the three tasks decreases as the number of available core increases, while it hits the bound in Fig. 9(a). The NPC values of the MPEG-light, the Sublinear and the Concave tasks on 14 cores are 38%, 50% and 40% when 'Task Load' = 70%, and are 47%, 61% and 54% when 'Task Load' = 90%, respectively. The

results in Fig. 9 verify that, for the task having low speedup of parallel execution, the proposed scheduling can save manifest amount of energy required for the execution of the real-time task.

Fig. 10 shows NPC values of the three different speedup models when the Loose-Scheduling method is used instead of the Tight-Scheduling method. The results in Fig. 10 are similar to those in Fig. 9.

B. Complexity and remaining issues

The operation of eliminating all defective frequencies requires $O(M)$ steps. The time complexity of the Loose-Scheduling and Tight-Scheduling methods is $O(N \cdot \log M)$, because the operation of finding f_{m^*} requires $O(\log M)$ steps. It is a one-time cost, incurred at compilation time. The Loose-Scheduling requires one frequency transition for the entire execution of the task, whereas the Tight-Scheduling requires at most one frequency transition for the execution of each frame. Let us consider the time penalty and the energy penalty to change frequency at runtime. To accommodate the time penalty, referred to as τ, the deadline D is replaced with $D^{'} = D - \tau$. Finally, note that these penalties are also required on both the single-core method and the all-cores method.

In terms of implementation, the proposed method needs the information about speedup of parallel execution on various numbers of cores, the worst-case cycles of the task to be computed within each frame, and the time limit between two adjacent frames in advance. Approximate values of the speedup, computation cycles, and the deadline are obtained from the type of media codec, analysis tools [8], [9] and accumulated statistics [10], [5]. For safety, the proposed method adopts their worst-case values. For more accurate model, the one-dimensional vector $S[n]$ of the speedup values against the number of allocated cores is replaced with a multiple-dimensional vector, which does not change the framework of the proposed method. For example, two-dimensional vector $S[n, f_m]$ of the speedup values against the number of allocated cores and the frequency value provides better estimation of the worst-case, where the speedup of parallel execution varies along with the frequency value as well as the number of allocated cores. Instead of the conservative approach based on the worst-case computation cycles, the aggressive approach [7], [10] exploiting statistical distribution of varying computation cycles can achieve further energy saving. Similar to the OPT-MEMS algorithm, this aggressive approach first searches a schedule consuming the minimum energy for each fixed number of cores, and

next selects the best number of cores.

V. Conclusions

We devised two scheduling algorithms that harness the multi-cores with DVFS capability. The proposed method minimizes the energy consumption of a real-time task on a DVFS-enabled multi-core platform by considering the non-linear scaling property of parallel execution speedup and the finitely discrete energy consumption rates of available frequencies. It is shown that the proposed scheduling can engineer the multi-core processor to operate at a highly energy-efficient manner for a single long-running real-time task. Compared with the counterparts of assigning either a single core or all cores to the execution of the task, the scheduling achieve as high as 72% and 90% energy saving, respectively. As long-running multi-media tasks such as video playing are becoming increasingly popular among mobile device users, such high potential gain warrants a serious investigation along this venue for the emerging multi-core mobile processor architecture. We believe that these results clearly point to the multi-core as the architecture of choice for future mobile devices. In future work, we will test the effectiveness of the proposed method in dealing with various real-life video streaming tasks, for which some central parameters to the proposed algorithm can only be approximately estimated.

References

[1] L. D. Paulson, "TV comes to the mobile phone," *Computer*, vol. 39, no. 4, pp. 13–16, 2006.

[2] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino, "Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 606–621, 2007.

[3] D. Geer, "Industry trends: Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, 2005.

[4] M. Nikitovic and M. Brorsson, "An adaptive chip-multiprocessor architecture for future mobile terminals," in *Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, 2002, pp. 43–49.

[5] J. H. Anderson and S. K. Baruah, "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms," in *Proc. of the 24th Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2004, pp. 428–435.

[6] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*, 2005, pp. 468–473.

[7] R. Xu, C. Xi, R. Melhem, and D. Mossé, "Practical PACE for embedded systems," in *Proc. of the 4th ACM Int'l Conf. on Embedded Software (EMSOFT)*. ACM Press, 2004, pp. 54–63.

[8] A. Maxiaguine, S. Künzli, and L. Thiele, "Workload characterization model for tasks with variable execution demand," in *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*, 2004, p. 21040.

[9] S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, S.-M. Moon, and C. S. Kim, "An accurate worst case timing analysis for RISC processors," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, pp. 593–604, 1995.

[10] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*. ACM Press, 2003, pp. 149–163.

[11] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-aware runtime scheduling for embedded-multiprocessor SOCs," *IEEE Design & Test of Computers*, vol. 18, no. 5, pp. 46–58, 2001.

[12] W. Y. Lee and H. Lee, "Energy-efficient scheduling for multiprocessors," *Electronics Letters*, vol. 42, no. 21, pp. 1200–1201, 2006.

[13] A. Bilas, J. Fritts, and J. P. Singh, "Real-time parallel MPEG-2 decoding in software," in *Proc. of the 11th Int'l Symp. on Parallel Processing (IPPS)*. IEEE Computer Society, 1997, pp. 197–203.

[14] G. W. Cook and E. J. Delp, "An investigation of scalable SIMD I/O techniques with application to parallel JPEG compression," *J. Parallel Distrib. Comput.*, vol. 30, no. 2, pp. 111–128, 1995.

[15] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," in *Proc. of the Int'l Symp. on Low Power Electronics and Design (ISLPED)*. ACM Press, 2000, pp. 9–14.

[16] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 3, pp. 299–316, 2000.

APPENDIX

***Lemma** 1:* If $\frac{p_y-p_x}{f_y-f_x} > \frac{p_z-p_y}{f_z-f_y}$ for any $f_x < f_y < f_z$, the frequency f_y is not included in the solution of the MEMS problem.

Proof: Let us assume that the frequency f_y is used to execute C cycles in the solution of the MEMS problem. When $C = C_a + C_b$, the amount of energy consumed when using f_y to execute C cycles is $\frac{p_y}{f_y} \cdot (C_a + C_b)$, and that when using f_x to execute C_a cycles and using f_z to execute C_b cycles is $(\frac{p_x}{f_x} \cdot C_a + \frac{p_z}{f_z} \cdot C_b)$. If $\frac{C_a+C_b}{f_y} = \frac{C_a}{f_x} + \frac{C_b}{f_z}$, then $C_b \cdot \frac{f_x}{f_x-f_y} = C_a \cdot \frac{f_z}{f_y-f_z}$. As well, if $\frac{p_y-p_x}{f_y-f_x} > \frac{p_z-p_y}{f_z-f_y}$ and $C_b \cdot \frac{f_x}{f_x-f_y} = C_a \cdot \frac{f_z}{f_y-f_z}$, then $C_a \cdot (\frac{p_x}{f_x} - \frac{p_y}{f_y}) - C_b \cdot (\frac{p_y}{f_y} - \frac{p_z}{f_z}) = (\frac{p_x}{f_x} \cdot C_a + \frac{p_z}{f_z} \cdot C_b) - \frac{p_y}{f_y} \cdot (C_a + C_b) < 0$. Because $\frac{p_y}{f_y} \cdot (C_a + C_b) > (\frac{p_x}{f_x} \cdot C_a + \frac{p_z}{f_z} \cdot C_b)$, the amount of energy consumed when using f_y to execute C cycles is larger than that when using f_x to execute C_a cycles and using f_z to execute C_b cycles. Consequently, the assumption on the minimum energy of the solution is a contradiction. This means that the frequency f_y is not used in the solution of the MEMS problem. ■

***Lemma** 2:* If $\frac{p_y-p_x}{f_y-f_x} \leq \frac{p_z-p_y}{f_z-f_y}$ for any $f_x < f_y < f_z$, the amount of energy consumed when using f_y to execute C cycles is no larger than that when using f_x to execute C_a cycles and using f_z to execute C_b cycles, where $C_a + C_b = C$ and $\frac{C_a}{f_x} + \frac{C_b}{f_z} \leq \frac{C}{f_y}$.

Proof: The completion time when using both f_x and f_z is $(\frac{C_a}{f_x} + \frac{C_b}{f_z})$ and that when using only f_y is $\frac{C}{f_y}$. If $\frac{C_a}{f_x} + \frac{C_b}{f_z} < \frac{C}{f_y}$, it is possible that some cycles from C_b cycles are executed with f_x, instead of f_z, for less energy consumption while satisfying $\frac{C_a}{f_x} + \frac{C_b}{f_z} \leq \frac{C}{f_y}$. Hence, the minimum energy consumption when $\frac{C_a}{f_x} + \frac{C_b}{f_z} = \frac{C}{f_y}$ is smaller than that when $\frac{C_a}{f_x} + \frac{C_b}{f_z} < \frac{C}{f_y}$. From now, we need to consider only the case $\frac{C_a}{f_x} + \frac{C_b}{f_z} = \frac{C}{f_y}$.

In case of $C = C_a + C_b$, the amount of energy consumed when using f_y to execute C cycles is $\frac{p_y}{f_y} \cdot (C_a + C_b)$, and that when using f_x to execute C_a cycles and using f_z to execute C_b cycles is $(\frac{p_x}{f_x} \cdot C_a + \frac{p_z}{f_z} \cdot C_b)$. If $\frac{p_y-p_x}{f_y-f_x} \leq \frac{p_z-p_y}{f_z-f_y}$ and $\frac{C_a}{f_x} + \frac{C_b}{f_z} = \frac{C}{f_y}$, then $\frac{p_y}{f_y} \cdot (C_a + C_b) \leq (\frac{p_x}{f_x} \cdot C_a + \frac{p_z}{f_z} \cdot C_b)$ by similar reason of Lemma 1. That is, the amount of energy consumed when using f_y to execute C cycles is no larger than that when using f_x and f_z for C_a and C_b cycles, respectively. ■

***Theorem** 1:* The schedule found by Tight-Scheduling consumes the minimum energy among the schedules satisfying the deadline for the execution on any fixed number of cores.

Proof: Let us assume that there is another schedule which satisfies the deadline and consumes less energy than the schedule found by Tight-Scheduling. This schedule is referred to as *New Schedule* and the schedule found by Tight-Scheduling is referred to as *Original Schedule*. Then, the New Schedule does not use the frequency f_y such that $\frac{p_y - p_x}{f_y - f_x} > \frac{p_z - p_y}{f_z - f_y}$ for any $f_x < f_y < f_z$ by Lemma 1.

Instead of f_{m^*} and f_{m^*-1}, the New Schedule may assign other frequency f_{m_1} such that $f_{m_1} < f_{m^*-1}$ to execute some C_a cycles among $C^{'}$ cycles and C_b cycles among $C^{''}$ cycles, respectively. In this case, the energy consumption of the New Schedule is larger than that of the Original Schedule. If $f_{m_1} < f_{m^*-1}$, then $\frac{C_a}{f_{m_1}} + \frac{C^{'} - C_a}{f_{m^*}} + \frac{C_b}{f_{m_1}} + \frac{C^{''} - C_b}{f_{m^*-1}} > D$.

Now, let us consider the case that the New Schedule assigns other frequencies f_{m_1} such that $f_{m_1} < f_{m^*-1}$ and f_{m_2} such that $f_{m_2} > f_{m^*}$ to execute C_a and C_b cycles, respectively. When $\frac{p_y - p_x}{f_y - f_x} \leq \frac{p_z - p_y}{f_z - f_y}$ for any $f_x < f_y < f_z$, the energy consumption when using f_{m_1} to execute C_a cycles and f_{m_2} to execute C_b cycles is no smaller than that when using f_{m^*-1} or f_{m^*} to execute $(C_a + C_b)$ cycles by Lemma 2.

Consequently, this contradicts our assumption that New Schedule consumes less energy and meets the deadline. Thus, the Tight-Scheduling method finds a schedule consuming the minimum energy among the schedules meeting the deadline. ■