Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 2010837, 10 pages
http://dx.doi.org/10.1155/2016/2010837

Research Article

Power Optimization of Multimode Mobile Embedded Systems with Workload-Delay Dependency

Hoeseok Yang[1] and Soonhoi Ha[2]

[1]*Department of ECE, Ajou University, Suwon 16499, Republic of Korea*
[2]*Department of CSE, Seoul National University, Seoul 08826, Republic of Korea*

Correspondence should be addressed to Soonhoi Ha; sha@snu.ac.kr

Received 24 March 2016; Accepted 14 June 2016

Academic Editor: Yuh-Shyan Chen

This paper proposes to take the relationship between delay and workload into account in the power optimization of microprocessors in mobile embedded systems. Since the components outside a device continuously change their values or properties, the workload to be handled by the systems becomes dynamic and variable. This variable workload is formulated as a staircase function of the delay taken at the previous iteration in this paper and applied to the power optimization of DVFS (dynamic voltage-frequency scaling). In doing so, a graph representation of all possible workload/mode changes during the lifetime of a device, Workload Transition Graph (WTG), is proposed. Then, the power optimization problem is transformed into finding a cycle (closed walk) in WTG which minimizes the average power consumption over it. Out of the obtained optimal cycle of WTG, one can derive the optimal power management policy of the target device. It is shown that the proposed policy is valid for both continuous and discrete DVFS models. The effectiveness of the proposed power optimization policy is demonstrated with the simulation results of synthetic and real-life examples.

1. Introduction

Today's mobile embedded systems often interact with physical processes or external environments, referred to as Cyber-Physical Systems (CPSs). Such systems are usually modeled with interactions between the physical world and the devices [1]. For instance, handheld or stationary embedded systems need to continuously interact with environments in the example of smart building [2]. The system performs a computational task and responds through an actuator to the physical side, while the resulting change at the physical side, in turn, makes a variation on the input (sensor) of the device. In order not to make this control loop unstable, it is common that the embedded system has a real-time constraint within which all the computation should be completed.

In a class of applications, the computational workload of the embedded systems depends on the variation of the sampled input value, while the computation delay, in turn, affects the input variation of the next iteration. Usually, if it invests more time at one iteration for processing information, it would have more work to do at the next iteration. One example of such delay-workload dependency can be found in an object tracking which is frequently used in drone, surveillance camera, or augmented reality [3–5]. The image obtained from the camera is processed by the object tracker to follow an object. As the object may continuously change its position meanwhile, the object tracker should reactively take an image from the adjusted position/angle to make the next decision. The more time the object spends in the tracker, the more distance the object will move by.

Such workload-delay relations can be popularly found in modern mobile embedded systems, which rely on computer vision algorithms to capture what happens in the external world. In those applications, it is typical that the *current internal state* is maintained to figure out the *difference* caused by what happened in the external world. The examples of such internal states range from a simple snapshot of a sensor reading to a complicated model of the scene obtained from camera. No matter what the model is, it is generally true that the longer execution delay between two consecutive invocations of the algorithm results in the larger workload in the successive iteration as the degree of the heterogeneity gets bigger.

The workload-delay dependency can also be found in many different types of applications. Real-time pattern matching over event streams [6], for instance, exhibits similar behavior: the queries can be handled either by small amount (shorter delay, less workload) or in an aggregated manner (longer delay, more workload). Similarly, haptic rendering in Human-Computer Interface (HCI) uses adaptive sampling techniques to deal with the stringent real-time constraint [7] and the rendering algorithm can be warm-started to exploit the temporal coherence [8]. In essence, applications which exploit temporal coherence have possible workload-delay dependencies. That is, any iterative algorithms that can be warm-started can lead to one.

Nowadays, most modern microprocessors used in mobile embedded systems support dynamic voltage-frequency scaling (DVFS) [9] for power-efficient operations. Generally, delay and energy in the systems with DVFS are in a tradeoff relationship for a given workload. That is, given a certain amount of work to be handled, a faster solution (with a higher frequency) is less energy efficient. Considering this control knob with the aforementioned delay-workload dependency, the power optimization problem gets very challenging. Conventionally, it has just been understood that "working as slow as possible" within the real-time constraint is the best discipline in terms of minimizing the power dissipation. However, with the existence of the workload-delay dependency, it is no longer valid since a slower execution may cause a bigger workload at the next iteration. On the other hand, "as fast as possible" is not optimal either, as the power consumption is a strong function of the operating speed [10].

The workload-delay dependency has been firstly modeled and applied to the DVFS optimization in [11]. It is assumed that the workload is a continuous and monotonically increasing function of the delay, under which a simple yet effective power management technique has been proposed. Specifically, it has been shown that staying in a certain DVFS mode is better than alternating between different DVFS modes dynamically. Later, the optimization is generalized to various power models and formally proven to be optimal [12].

This work differs from our previous work [12] in that we take different optimization approach tailored for discrete workload levels. We observed that the continuity assumption does not always hold true in reality. Rather, there are a number of applications that have discrete levels of workload. For instance, recall that many image or signal processing algorithms handle input data in the unit of macroblock or frame. In such application domains, the workload tends to grow in a discrete manner. In this paper, the workload is modeled as a staircase function of the delay taken in the previous iteration. Since the solution obtained by the previous work [11, 12] is no longer optimal or nonexisting at all in the staircase model, a new power management technique is proposed. The contributions of this paper can be summarized as follows:

(i) The workload-delay dependency is modeled in a staircase function generalizing the previous model and validated with a real-life example.

(ii) A novel data structure, Workload Transition Graph (WTG), is proposed to represent all possible operation workload/mode changes of a device.

(iii) Based on WTG, a power management policy is derived and shown to be optimal.

2. Related Work

Bogdan and Marculescu [13] observed that workloads from physical processes tend to be nonstationary but exhibit some systematic relationship in space and time. They proposed a workload characterization approach based on statistical physics and showed how the workload-awareness can improve the design of electronic systems. Zhang et al. [14] studied the relationship between the control stability and workload in inverted pendulum control. While enlarged invocation periods may lower the degree of stability, more inverted pendulums can be controlled by a system as the lengthened invocation periods lower the utilization of the algorithm. This can be seen as trading off the control stability for resource efficiency. In other words, they proposed to sacrifice the stability to accommodate more workload in a system. The proposed technique also deals with variable workload in electronic systems but differs from the above-mentioned works in that the effect of execution delay on workload is systematically considered.

Recently, Pant et al. [15] proposed a codesign of computation delay and control stability based on anytime algorithm. Anytime algorithm is a kind of algorithms that can be stopped at any point in time but still provides a decent solution. Typically, the quality of the solution is increasing function of the computation delay. In their work, it is the duty of the control algorithm to adaptively change the real-time deadline constraint and error bound (quality of control). On the contrary, the relationship between execution delay and workload is formally described in the form of workload-delay function; thus no explicit runtime monitoring/control is required in the proposed technique.

A design guideline for flexible delay constraints in distributed embedded systems was proposed by Goswami et al. [16], where some of the samples are allowed to violate the given delay deadline. They presented the applicability of the proposed approach using the FlexRay dynamic segment as a communication medium. This work is similar to the proposed approach in the sense that they do not stick to a given fixed real-time deadline. While they could avoid the resource overprovisioning by trading off the hard real-time constraints, the workload dependency to the delay has not been considered. Moreover, from a real-time standpoint, the proposed work is more rigorous as it allows no real-time constraint violations.

3. Problem Definition

This section presents the system model assumed in this paper, which is followed by the formulation of the power optimization problem.

3.1. System Model

3.1.1. Dynamic Voltage-Frequency Scaling. In this paper, we assume that a system has multiple operation modes due to DVFS feature, where the operating frequency and voltage can be modulated. For simplicity, we first assume that there are infinitely many operation modes available, among which one is chosen at each iteration. It will be shown that the proposed technique can be applied to a discrete DVFS as well in Section 5. The operation mode at the ith iteration is represented with the speed scaling factor k_i ranging from $k_{\min}$ to $k_{\max} = 1$ ($k_{\min} \le k \le k_{\max} = 1$). Then, the operating frequency of the ith iteration, f_i, is

$$f_i = k_i \cdot f_{\max}, \tag{1}$$

where $f_{\max}$ is the maximum frequency of the microprocessor.

3.1.2. Workload. The workload is defined to be a number of clock cycles elapsed to complete the given computation. We denote the number of cycles elapsed to handle the workload of the ith iteration at the full speed of the microprocessor ($k_i = 1$) as $t_{\text{ref},i}$. That is,

$$w_i = t_{\text{ref},i} = k_i \cdot t_i. \tag{2}$$

Note that the elapsed time t_i increases as the speed is scaled down ($k_i < 1$). Then, the delay t_i is automatically determined when a speed scaling factor k_i is chosen for the given workload ($t_i = w_i/k_i$).

3.1.3. Real-Time Constraint. The delay t_i cannot be unboundedly long as the system is associated with real-time constraint T. For all iterations, the elapsed time should be no more than T:

$$t_i \le T, \quad \forall i. \tag{3}$$

3.1.4. Delay-Workload Dependency. As stated earlier, the workload is dependent upon the previous execution delay. Usually, the workload is not a continuous function of the delay variation. Rather, the changes happen in a discrete manner. Therefore, the workload at the ith iteration is a monotonically increasing staircase function of the delay of the previous iteration, t_{i-1}: $w_i = W(t_{i-1})$. If the given system has n workload levels, the workload function W can be formulated as follows:

$$W(t) = \begin{cases} \text{wl}_1 & \text{if } 0 < t \le \text{th}_1 \\ \text{wl}_2 & \text{if } \text{th}_1 < t \le \text{th}_2 \\ \vdots & \\ \text{wl}_{n-1} & \text{if } \text{th}_{n-2} < t \le \text{th}_{n-1} \\ \text{wl}_n & \text{if } \text{th}_{n-1} < t, \end{cases} \tag{4}$$

in which the workload levels are $\text{wl}_1 < \text{wl}_2 < \cdots < \text{wl}_n$ and the delay thresholds (workload changing moments) are $\text{th}_1 < \text{th}_2 < \cdots < \text{th}_{n-1}$.

3.1.5. Execution Trace. At the ith iteration, the speed scaling factor k_i uniquely defines an execution *mode* as the delay is fixed accordingly by (2). The initial workload is assumed to be given as w_1. Then, an execution trace tr of length L is defined to be a sequence of the speed scaling factors of L iterations:

$$\text{tr} \coloneqq (k_1, k_2, \ldots, k_L). \tag{5}$$

3.1.6. Average Power Consumption. The dynamic power consumption of CMOS circuits is $cv_{dd}^2 f$, where c, v_{dd}, and f are capacitance, operating voltage, and frequency, respectively. As the operating frequency is proportional to $(v_{dd} - v_{\text{th}})^2/v_{dd}$ [10], the power consumption is an increasing function of f. It is worth noting that the proposed model is not dependent upon any specific DVFS model. We denote the energy consumption of a unit workload at the full speed ($k = 1$) as C_e and assume that energy dissipation grows linearly to the size of workload. Then, the reference energy of a workload w_i at the full speed is $C_e \cdot w_i$. Given a DVFS energy model S as a function of the speed scaling factor k, the energy consumption at the ith iteration EG_i is formulated as follows:

$$\text{EG}_i = C_e \cdot w_i \cdot S(k_i), \tag{6}$$

in which $S(0) = 0$ and $S(1) = 1$. Then, the average power consumption of a trace tr can be formulated as follows:

$$\text{AVG}(\text{tr}) = \frac{\sum_{i=1}^{|\text{tr}|} \text{EG}_i}{\sum_{i=1}^{|\text{tr}|} t_i}. \tag{7}$$

It is worthwhile to mention that the proposed technique is not specific to a certain workload-energy model. While we adopt linear model for the workload-energy relation for ease of presentation, any, possibly nonlinear, model can be used in (6).

3.2. Problem Formulation.

Our objective is to minimize the average power consumption of a given system as follows:

> Given the modeling constant C_e, DVFS energy modeling function S, workload function W, and the real-time constraint T, determine an execution trace tr such that the average power consumption formulated in (7) is minimized.

4. Proposed Technique

In this section, we describe the proposed operation management policy as an answer to the problem defined in the previous section. In doing so, we first derive the condition for feasible and schedulable systems. Then, we study when the workload changes and how it affects the power dissipation. Based on that, we propose a novel graph representation that captures all possible workload transitions in the power-optimal operation. Finally, we derive the power-optimal operation policy with the given workload function W.

4.1. Feasibility.

In this subsection, we examine under which condition a given system is feasible. First, the system should

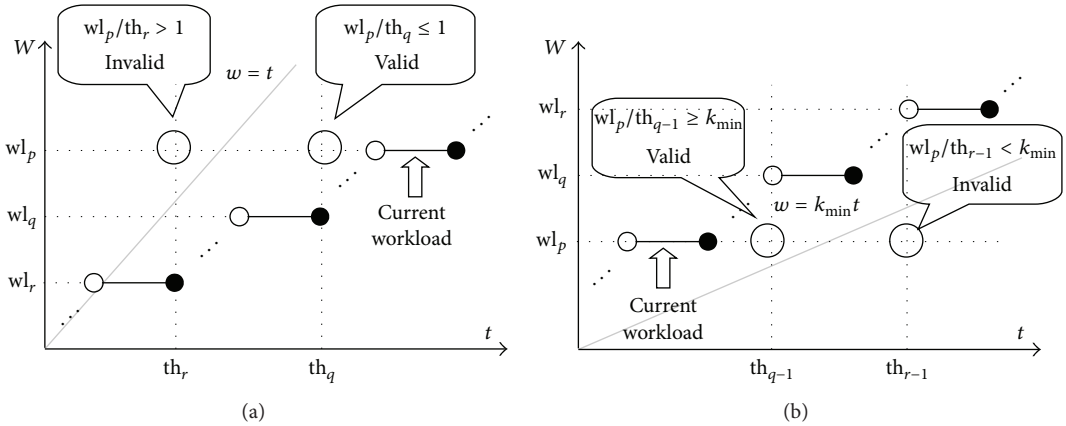

Figure 1: Examples of workload transitions: (a) transitions to lower workload levels and (b) transitions to higher workload levels. The transitions from wl_p to wl_q are valid, while the ones from wl_p to wl_r are not.

be schedulable within the given real-time constraint T at every iteration.

Theorem 1 (schedulability). *Given the workload function W and the real-time constraint T, the system is not schedulable if $W(t)/t > 1, \forall t \in (0, T]$.*

Proof. Suppose that the delay at the ith iteration is t_i. Then, $w_{i+1} = W(t_i) > t_i$ and $w_{i+1} = k_{i+1} \cdot t_{i+1}$. Since $k_{i+1} \leq 1, t_i < t_{i+1}$. That is, the delay is increasing as iteration goes by and will eventually reach the real-time constraint: $t_j = T$. At the next iteration, the system becomes unschedulable even with the full speed, as $W(t_j = T) = 1 \cdot t_{j+1} > T$ requires $t_{j+1} > T$. □

Once the workload gets bigger than T, the system is trivially not schedulable afterwards even with the full speed, $k_i = 1$. Thus, the workload must not be bigger than T at any time. Moreover, once the workload reaches T, k should remain the full speed $k = 1$ afterwards. We can make the upper bound of workload even tighter if there exists t' such that $W(t) > t$ for all $t \in (t', T]$. In this case, the workload larger than $W(t')$ is not allowable as it makes the execution delay longer and longer, eventually violating the deadline.

Given the workload function W and the initial workload w_1, one can calculate the lower bound of the workload as well. If a value $\tilde{t}$ exists which satisfies $W(\tilde{t}) = \tilde{t}$ and $W(\tilde{t}) < w_1$, the workload will never become smaller than $W(\tilde{t})$. In other words, even with the full speed, the execution delay never goes below $\tilde{t}$.

Then, the valid workload levels and the execution delay range during the lifetime of a given system can be formulated as below.

Definition 2 (valid ranges). Given the workload function W and the initial workload w_1, the minimum and maximum workload levels of a system are defined to be

$$\mathrm{wl}_{\min} = \begin{cases} W\left(\max\left(\tilde{t}\right)\right) \text{ s.t. } \tilde{t} = W\left(\tilde{t}\right) < w_1 & \text{if } \exists \tilde{t} \\ \mathrm{wl}_1 & \text{otherwise,} \end{cases} \tag{8}$$

$$\mathrm{wl}_{\max} = \begin{cases} W\left(\min\left(t'\right)\right) \text{ s.t. } W(t) > t, \ \forall t \in \left(t', T\right] & \text{if } \exists t' \\ W(T) & \text{otherwise.} \end{cases} \tag{9}$$

Then, the valid range of the execution delay is formulated as $[t_{\min}, t_{\max}]$ according to $\mathrm{wl}_{\min}$ and $\mathrm{wl}_{\max}$ with

$$\begin{aligned} t_{\min} &= \min\left(\tilde{t}\right) \text{ s.t. } W\left(\tilde{t}\right) = \mathrm{wl}_{\min}, \\ t_{\max} &= \min\left(T, \max\left(t'\right)\right) \text{ s.t. } W\left(t'\right) = \mathrm{wl}_{\max}. \end{aligned} \tag{10}$$

4.2. Workload Transitions. In this subsection, we examine when a workload transition between valid workload levels possibly occurs and how it affects the system.

As presented in (4), workload is a function of the delay taken at the previous iteration. If the delay taken at an iteration is t and $\mathrm{th}_{l-1} < \mathrm{t} \leq \mathrm{th}_l$. Then, if the system works fast enough to result in a shorter delay, $t' \leq \mathrm{th}_{l-1}$, the next workload will get smaller than wl_l. Similarly, in case that the delay gets longer ($t' > \mathrm{th}_l$), the system will need to handle a larger workload than wl_k at the successive iteration.

However, such workload transitions can occur only within limited ranges. Figure 1 depicts valid and invalid transitions from one workload level. Figure 1(a) shows two transitions from a workload level wl_p to lower ones wl_q and wl_r ($r < q < p$). To make the next workload level wl_q, the delay should be in the range of $(\mathrm{th}_{q-1}, \mathrm{th}_q]$. Given the current workload wl_p, the speed scaling factor should be larger than or equal to $\mathrm{wl}_p/\mathrm{th}_q$ from (2). If $\mathrm{wl}_p/\mathrm{th}_q \leq k_{\max} = 1$, this workload transition can possibly occur. In contrast, if $\mathrm{wl}_p/\mathrm{th}_r > 1$ for another workload level wl_r, the transition from wl_p to wl_r never happens because the delay never goes below th_r even with the full processing speed.

The same principle is also applied to the transition from a workload level to higher ones. If the delay can be lengthened properly with a speed scaling factor within the range $[k_{\min}, k_{\max}]$, the transitions are valid. Figure 1(b) illustrates that the transition from wl_p to wl_q is valid, while

the one to wl_r is not. One can tell if a transition can happen or not with the following definition.

Definition 3 (valid transition). A workload transition from wl_o to wl_n is said to be *valid* if $\mathrm{wl}_o/\mathrm{th}_n \le k_{\max} = 1$ and $\mathrm{wl}_o/\mathrm{th}_{n-1} \ge k_{\min}$.

4.3. Workload Transition Graph. The essential difficulty of the presented power optimization problem lies in the fact that two conflicting forces should be handled at the same time. In order to minimize the power, on one hand, it tries to scale down the speed (thus lengthen the delay) as much as possible as described in (2) and (6). On the other hand, the lengthened delay is not desirable as it imposes a bigger workload in the successive iteration, as shown in (4).

Therefore, no one simple intuition can be exploited to solve the problem. Rather, we need to compare different modes in a comprehensive way. In order to be able to explore all possible execution modes and quantify their effects, we need to devise a data structure that includes elementary information on how workload transitions change the system status and power dissipation. In line with that purpose, we propose a graph representation of the workload evolution, Workload Transition Graph (WTG), which captures all possible transition scenarios of the workload changes during the lifetime of a system.

A valid workload transition from one workload level to another can be caused by any delay within the corresponding range. A transition from wl_p to wl_q in Figure 1(a), for instance, can be caused by any delay within the range of $(\mathrm{th}_{q-1}, \mathrm{th}_q]$. In other words, when handling a workload of wl_p, any scaling factor within the range of $[\mathrm{wl}_p/\mathrm{th}_q, \mathrm{wl}_p/\mathrm{th}_{q-1})$ can cause the transition. In the power-optimal execution trace, however, only one specific scaling factor is always chosen for a certain transition even though it happens many times. We show this in the next theorem.

Theorem 4 (optimal scaling factor). *In the power-optimal trace $tr = (k_1, k_2, \ldots, k_{|tr|})$, if the workload level handled by k_i is wl_o and the next workload level is wl_n,*

$$k_i = \min\left(\frac{wl_o}{th_n}, k_{\min}\right). \tag{11}$$

Proof. We prove this by contradiction. Let us suppose that there exists a power-optimal trace tr where k_i (which results in the transition from wl_o to wl_n) is not equal to $\min(\mathrm{wl}_o/\mathrm{th}_n, k_{\min})$. Then, we make a new execution trace tr' from tr by replacing k_i with $k_i' = \min(\mathrm{wl}_o/\mathrm{th}_n, k_{\min})$. Note that all other execution modes in tr' are the same as in tr. That is,

$$\mathrm{tr}(j) = \mathrm{tr}'(j), \quad \forall j \ne i. \tag{12}$$

By the definition of the scaling factor,

$$\begin{aligned} k_i &\ge k_{\min}, \\ k_i &\ge \frac{\mathrm{wl}_o}{\mathrm{th}_n}, \end{aligned} \tag{13}$$

since the transition is from wl_o to wl_n. Thus,

$$\mathrm{tr}(i) = k_i > \mathrm{tr}'(i) = k_i' \tag{14}$$

and accordingly we obtain

$$S(k_i) > S(k_i'). \tag{15}$$

Then, from (6) and (7), the average power in tr' is lower than that of tr and this contradicts the proposition. □

From Theorem 4, we know that only one speed scaling factor is associated with all transitions of a certain type in the power-optimal operation scenario. So, we define a scaling factor of a valid transition as follows.

Definition 5 (scaling factor of a transition). The scaling factor of a valid workload transition from wl_o to wl_n is

$$\mathrm{sf}(\mathrm{wl}_o, \mathrm{wl}_n) = \max\left(\frac{\mathrm{wl}_o}{\min(\mathrm{th}_n, T)}, k_{\min}\right). \tag{16}$$

Now, we define WTG.

Definition 6 (Workload Transition Graph). WTG is defined to be a graph (V, E), where V and E are the sets of vertices and edges, respectively. Each valid workload level forms a vertex

$$V = \{v_i = \mathrm{wl}_i \mid \mathrm{wl}_{\min} \le \mathrm{wl}_i \le \mathrm{wl}_{\max}\}, \tag{17}$$

while a valid transition from a workload level to another forms an edge between vertices. That is, there exists an edge from v_i to v_j corresponding to a valid transition from wl_i to wl_j:

$$E = \{(v_i, v_j) \mid \exists \text{ valid transition from } \mathrm{wl}_i \text{ to } \mathrm{wl}_j\}. \tag{18}$$

We denote the source and destination vertices of an edge $e \in E$ as $\mathrm{src}(e)$ and $\mathrm{dst}(e)$, respectively.

With these definitions, a power-optimal execution trace $\mathrm{tr} = (k_1, k_2, \ldots, k_{|\mathrm{tr}|})$ can be represented as a walk (a sequence of vertices where any pair of consecutive vertices are connected through an edge) of length $|\mathrm{tr}| + 1$ in WTG. In a different form, the execution trace is a sequence of edges in WTG of length $|\mathrm{tr}|(e_1, e_2, \ldots, e_{|\mathrm{tr}|})$, where the workload transition from $\mathrm{src}(e_i)$ to $\mathrm{dst}(e_i)$ is caused by the execution mode $\mathrm{sf}(\mathrm{src}(e_i), \mathrm{dst}(e_i)) = k_i$.

Algorithm 1 shows how WTG is generated out of the given workload function W and the initial workload w_1. After the initialization, all valid workload levels are added as vertices in lines (5)-(6). Then, for each permutation of two workload levels (lines (7)-(8)), it is checked if the in-between transition is valid or not in line (9). If valid, it is added as an edge in line (10).

Let us take Figure 2 as an example of workload function W. Once given the initial workload, one can easily get the valid workload levels according to (8) and (9). When $w_1 = \mathrm{wl}_2$, for instance, $\mathrm{wl}_{\min} = \mathrm{wl}_2$ and $\mathrm{wl}_{\max} = \mathrm{wl}_6$. Figure 3(b) illustrates the corresponding WTG of the workload function

```
(1) procedure GenerateWTG (W, w_1)
(2)    wl_min ← equation (8);                    ▷ Initializations
(3)    wl_max ← equation (9);
(4)    V ← φ, E ← φ;
(5)    for all wl_min ≤ wl ≤ wl_max do           ▷ Vertices
(6)       V ← V ∪ {wl};
(7)    for all wl_min ≤ wl_1 ≤ wl_max do         ▷ Edges
(8)       for all wl_min ≤ wl_2 ≤ wl_max do
(9)          if ∃ valid transition from wl_1 to wl_2 then
(10)            E ← E ∪ {(wl_1, wl_2)}
```

Algorithm 1: Generation of WTG.

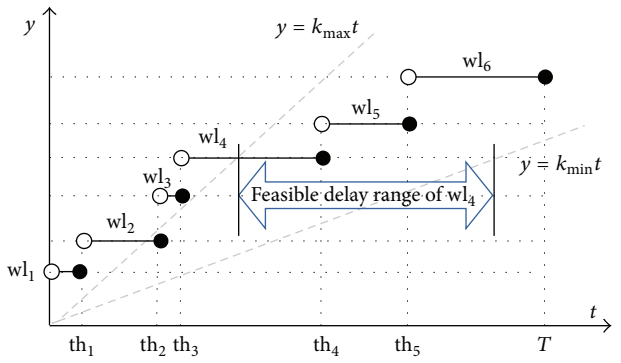

Figure 2: A workload staircase function example with six workload levels.

shown in Figure 2 with the initial workload of wl_2. It is not a complete graph as some pairs of vertices cannot be connected directly since it is not *valid* according to Definition 3.

The feasible delay range to handle workload wl_4 is highlighted in Figure 2, justifying that vertex wl_4 has three outgoing edges to wl_6, wl_5, and itself. To be more specific, when the workload is given as wl_4, the shortest possible delay is the case when the speed is chosen as k_{max}. Then, the delay (x coordinate of the cross point of $y = wl_4$ and $y = k_{max}t$) is between th_3 and th_4 as shown in Figure 2. This means that the lowest possible workload in the next iteration is wl_4. Similarly, the biggest possible workload can also be calculated as wl_6 as the biggest possible delay is between th_5 and T. Note also that some of the vertices may not have an edge directed to itself (self-loop) such as vertex wl_3. It has outgoing edges only to higher workload levels. This means that the computation burden of that state is so big that it only results in higher workload levels at the next iteration even with the full speed.

Different initial workload levels may result in different WTGs as shown in Figures 3(a)–3(c). The WTG derived from the initial workload level of wl_1 is illustrated in Figure 3(a). In contrast to Figure 3(b), vertex wl_1 is included in the graph. The WTG derived from higher initial workload levels, wl_4, wl_5, and wl_6, is shown in Figure 3(c). Note that the WTGs in Figures 3(a) and 3(c) are not *strongly connected*. Vertex wl_2 in Figure 3(b), for example, is not reachable from wl_4. However, from the definition of valid workload levels, all vertices are reachable from the initial workload level. This property is important for deriving the optimal operation policy that will be presented in the next subsection.

4.4. Proposed Operation Policy. In this subsection, we present the proposed operation policy that compromises the energy-delay tradeoff caused by the delay-workload dependency.

As stated earlier, a power-optimal execution trace tr can be represented as a walk of WTG. Then, we have the following definition.

Definition 7 (corresponding walk). Given the power-optimal execution trace $\mathrm{tr} = (k_1, k_2, \ldots, k_{|\mathrm{tr}|})$ and its initial workload w_1, the corresponding walk of the trace is denoted as

$$\mathrm{cw}_{\mathrm{tr},w_1} = (e_1, e_2, \ldots, e_{|\mathrm{tr}|}), \tag{19}$$

where $\mathrm{src}(e_1) = w_1$ and $\mathrm{sf}(\mathrm{src}(e_i), \mathrm{dst}(e_i)) = k_i$ (for simplicity, we also denote it as $\mathrm{sf}(e_i) = \mathrm{sf}(\mathrm{src}(e_i), \mathrm{dst}(e_i)) = k_i$ for the rest of the paper). The average power consumption of the walk can be formulated as follows:

$$\mathrm{AVG}\left(\mathrm{cw}_{\mathrm{tr},w_1}\right) = \frac{\mathrm{EG}\left(\mathrm{cw}_{\mathrm{tr},w_1}\right)}{\mathrm{DL}\left(\mathrm{cw}_{\mathrm{tr},w_1}\right)}, \tag{20}$$

where $\mathrm{DL}((e_1, e_2, \ldots, e_n))$ is the total delay elapsed for traversing the walk, $\sum_{i=1}^{n}(\mathrm{src}(e_i)/\mathrm{sf}(e_i))$, and $\mathrm{EG}((e_1, e_2, \ldots, e_n))$ is the total energy consumption for the walk, $C_e \sum_{i=1}^{n} \mathrm{src}(e_i) S(\mathrm{sf}(e_i))$.

A cycle (closed walk) of WTG is a walk whose starting and ending vertices are the same. That is, a walk $(e_1, e_2, \ldots, e_n)$ is a cycle if $\mathrm{src}(e_1) = \mathrm{dst}(e_n)$. Hence, if the corresponding walk of a trace in WTG is a cycle, the trace is ever repeatable. We argue that, in case that the length of the trace is sufficiently long ($|\mathrm{tr}| \to \infty$), the average power consumption is minimized when the cycle which minimizes (20) repeats over and over again in the trace.

Theorem 8 (optimal cycle of WTG). *Suppose that a cycle of WTG,*

$$cy_{opt} = (e_1, e_2, \ldots, e_n), \tag{21}$$

has the minimum value of (20) among all cycles of the WTG. Then, if the length of the trace is long enough, the average power consumption of the optimal trace converges to

$$\mathrm{AVG}\left(cy_{opt}\right) = \frac{\mathrm{EG}\left(cy_{opt}\right)}{\mathrm{DL}\left(cy_{opt}\right)}. \tag{22}$$

Proof. An arbitrary walk of a WTG, $(e_1, e_2, \ldots, e_n)$, can be decomposed into a path (a walk with distinct vertices) from $\mathrm{src}(e_1)$ to $\mathrm{dst}(e_n)$ and a set of cycles (see, e.g., Section 10.3 of [17]). Figure 4 depicts a walk example of length 8, where the initial workload level is wl_2 and the last vertex that it traverses is wl_6. If a path from wl_2 to wl_6, (e_1, e_2, e_5), highlighted in the dashed arrows, is removed from the walk, the remaining part is a set of cycles.

Now, consider a power-optimal trace tr of length N that starts from the workload level of w_1. The corresponding walk

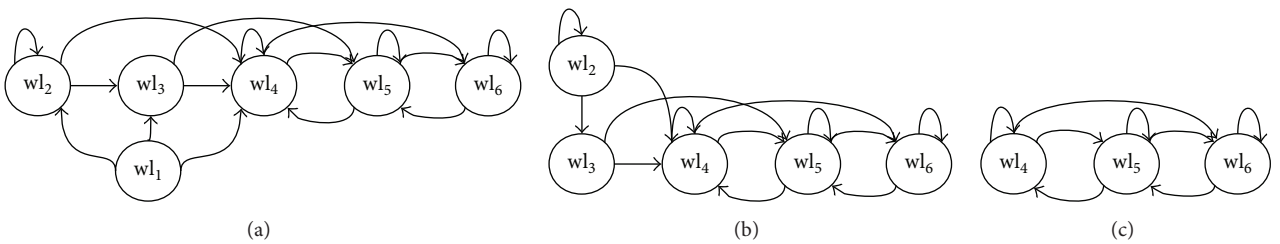

Figure 3: The corresponding WTG of the workload function W shown in Figure 2 with the initial workload of (a) wl_1, (b) wl_2 or wl_3, and (c) wl_4, wl_5, or wl_6.

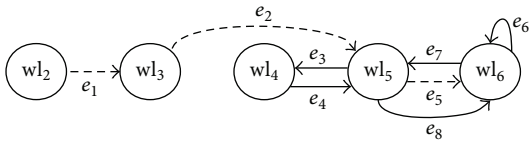

Figure 4: A walk example of the WTG shown in Figure 3(b) of length 8.

of tr can be decomposed into a path pt starting at w_1 and a set of cycles CY. Then, the average power consumption of the trace tr is

$$\mathrm{AVG}(\mathrm{tr}) = \frac{\mathrm{EG}(\mathrm{pt}) + \sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{EG}(\mathrm{cy})}{\mathrm{DL}(\mathrm{pt}) + \sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{DL}(\mathrm{cy})}. \quad (23)$$

Since pt contains at most $|V| - 1$ edges by definition, EG(pt) and DL(pt) can be upper bounded by a certain value. Then, if N is sufficiently large, $\mathrm{EG(pt)} \ll \sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{EG(cy)}$ and $\mathrm{DL(pt)} \ll \sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{DL(cy)}$. That is, if $|\mathrm{tr}| \to \infty$,

$$\mathrm{AVG}(\mathrm{tr}) \simeq \frac{\sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{EG}(\mathrm{cy})}{\sum_{\mathrm{cy}\in\mathrm{CY}} \mathrm{DL}(\mathrm{cy})}. \quad (24)$$

Note also that the average power consumption of every cycle in CY is not smaller than $\mathrm{EG}(\mathrm{cy}_{\mathrm{opt}})/t(\mathrm{cy}_{\mathrm{opt}})$ by definition:

$$\frac{\mathrm{EG}(\mathrm{cy})}{\mathrm{DL}(\mathrm{cy})} \geq \mathrm{AVG}\left(\mathrm{cy}_{\mathrm{opt}}\right) = \frac{\mathrm{EG}\left(\mathrm{cy}_{\mathrm{opt}}\right)}{\mathrm{DL}\left(\mathrm{cy}_{\mathrm{opt}}\right)}, \quad \forall \mathrm{cy} \in \mathrm{CY}. \quad (25)$$

Therefore, the average power consumption of the power-optimal trace tr, AVG(tr), will get infinitesimally close to $\mathrm{AVG}(\mathrm{cy}_{\mathrm{opt}})$. □

From Theorem 8, it is understood that changing the DVFS mode of the given system following the optimal cycle presented above results in asymptotically optimal average power consumption. Thus, we propose following rules for DVFS operation policy:

(i) If the current workload level vertex is in the optimal cycle $\mathrm{cy}_{\mathrm{opt}}$, just follow the cycle repeatedly ever; that is, at the ith iteration, the speed scaling factor is chosen to be $\mathrm{sf}(e)$ such that e is in the optimal cycle and $\mathrm{src}(e) = w_i$.

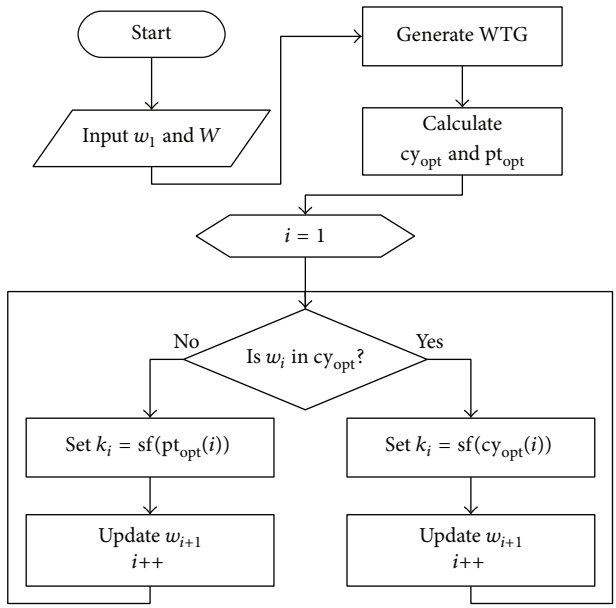

Figure 5: Flowchart diagram of the proposed operation policy.

(ii) Otherwise, take a path $\mathrm{pt}_{\mathrm{opt}} = (e_1, e_2, \ldots, e_n)$ that has the minimum $\mathrm{AVG}(\mathrm{pt}_{\mathrm{opt}})$ value and $\mathrm{dst}(e_n)$ is in the optimal cycle $\mathrm{cy}_{\mathrm{opt}}$. In other words, try to get in the optimal cycle with the minimum cost.

The optimal cycle $\mathrm{cy}_{\mathrm{opt}}$ of WTG can be searched by using an existing cycle enumeration algorithm. In this paper, we use the one proposed by Tarjan [18]. The minimum path to the optimal cycle, $\mathrm{pt}_{\mathrm{opt}}$, can also be searched by simple enumeration. It is worthwhile to mention that we cannot simply use the minimum weight cycle searching algorithm as the weight is not simply a summation of the weights but a complex function of them as presented in (20).

Figure 5 shows the proposed operation policy in a flowchart diagram. Given the initial workload and the workload function, we generate WTG, from which $\mathrm{cy}_{\mathrm{opt}}$ and $\mathrm{pt}_{\mathrm{opt}}$ are derived in the next steps. It is worth mentioning that this can be done in a tractable time and is just one-time effort taken offline. As long as the initial workload vertex is not included in $\mathrm{cy}_{\mathrm{opt}}$, the system follows the trace represented in $\mathrm{pt}_{\mathrm{opt}}$ until it reaches the optimal cycle of WTG. Then, it simply repeats the trace implied in $\mathrm{cy}_{\mathrm{opt}}$ from that iteration on.

5. Extension to Discrete DVFS

Whilst we assume a continuous DVFS model for ease of presentation and generality, modern microprocessors in reality have finite DVFS modes with a set of predefined operation voltages and frequencies. In this section, we show that the continuity of the model presented in (1) can be relaxed by modifying Definitions 3 and 5 without harming the effectiveness of the proposed technique.

Let us suppose that we now have a system with a discrete and finite DVFS model, where only $k_i \in K$ can be chosen as a speed scaling factor at the ith iteration. Valid transitions, in Definition 3, are redefined: the transition is valid if there is $k \in K$ that meets the same requirement.

Definition 9 (valid transition in discrete DVFS). Given a set of feasible scaling factors K, a workload transition from wl_o to wl_n is said to be *valid* if

$$\mathrm{th}_{n-1} < \frac{\mathrm{wl}_o}{k} \leq \min\left(\mathrm{th}_n, T\right), \quad \exists k \in K. \tag{26}$$

Likewise, the scaling factor of a valid transition, in Definition 5, is also reformed to be minimum $k \in K$ that keeps the elapsed delay fallen into the range which results in the same transition. One has the following definition.

Definition 10 (scaling factor in discrete DVFS). Given a set of feasible scaling factors K, the scaling factor of a valid workload transition from wl_o to wl_n is

$$\begin{aligned} &\mathrm{sf}\left(\mathrm{wl}_o, \mathrm{wl}_n\right) \\ &\quad = \arg\min_{k \in K} \left\{ \mathrm{th}_{n-1} < \frac{\mathrm{wl}_o}{k} \leq \min\left(\mathrm{th}_n, T\right) \right\}. \end{aligned} \tag{27}$$

6. Experiments

In this section, we validate the proposed model and operation policy with experimental analysis and simulations.

6.1. A Case Study: Object Tracking. It is firstly shown that the proposed workload-delay dependency is evidently observed in an object tracking application. The performance of a commonly used object tracking method [19] is profiled, as a tracking solution, using the publicly available implementation [20]. We choose Exynos5422 [21] as the target mobile embedded computing platform, which has 2 GB main memory running Linux operating system. The actual power dissipations of the processor are measured individually for five different DVFS modes. That is, $K = \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and at the maximum speed the core is operating at 2 GHz. Leveraging on a priori knowledge on the maximum speed of the object, the maximum distance that the object could have moved between two iterations is calculated. In this experiment, it is assumed that the object's speed never exceeds 10 pixels/ms. If an iteration takes t ms, for instance, the search area for the next iteration is given as a square with the side length of $(10 \cdot 2 \cdot t + 125)$ pixels. This search area is growing in a discrete manner at every 5 ms. The real-time constraint T is set to 25 ms. The workload function in a shape of staircase is illustrated in Figure 6 with five guiding lines, each of which denotes $k \cdot t$ for $k \in K$.

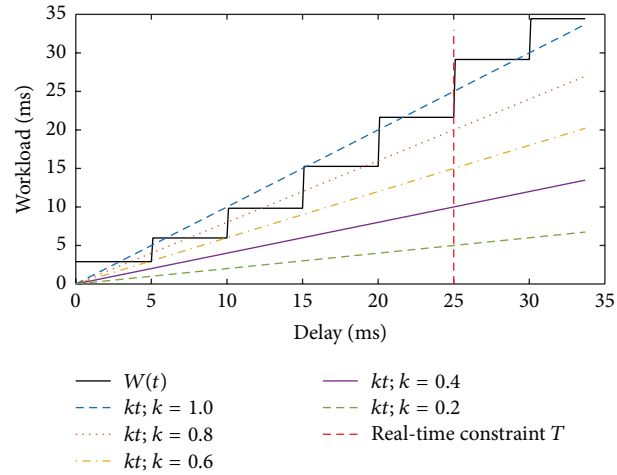

Figure 6: Workload-delay dependency function W of Lucas-Kanade object tracking algorithm.

We compare the proposed power management policy with two others. The first one is ALAP, where the speed is chosen to be the slowest one with respect to the real-time constraint. The other comparison is made against a stable trace with the maximum speed as another extreme (ASAP). When the initial workload is wl_3, that is, $w_1 = \mathrm{wl}_3 = 9.833$, ASAP and ALAP result in the average power consumption of 2.5032 W and 1.2623 W, respectively. The average power consumption of the proposed power management policy outperforms the others as 0.7592 W. The optimal cycle of its WTG is the self-loop of node wl_2, which implies a stable operation mode, $\forall i,\ k_i = 0.6$.

6.2. Stable versus Alternating Operation Modes. There are two kinds of cycles in WTG: the first one is a self-loop which implies a *stable* operation mode, where no mode changes happen over the edge. Other than these self-loops, the WTG has one non-self-loop cycle as well. From the perspective of the operation policy, this non-self-loop cycle implies a predefined sequence of mode changes that can repeat over and over again. We call this *alternating* operation mode. Many examples including a real-life application shown in the previous subsection, as well as one presented in [11], tend to be optimal in a stable operation mode. However, in principle, the optimal solution cannot be achieved in a stable mode in some configurations. In order to illustrate this, we show a counter example shown in Figure 7 and apply the proposed technique to the example, with a DVFS power modeling function of $S(k) = 5 \cdot k^3$ and $K = \{0.4, 0.8, 0.9\}$. The delay threshold and the real-time constraint are $\mathrm{th}_1 = 0.2$, $\mathrm{th}_2 = 0.6$, $\mathrm{th}_3 = 0.7$, and $T = 1.0$, respectively. Figure 7(b) shows the derived WTG for the synthetic example.

The average power consumption of all self-loops in the synthetic example is tied to 2.5600 W. On the other hand,

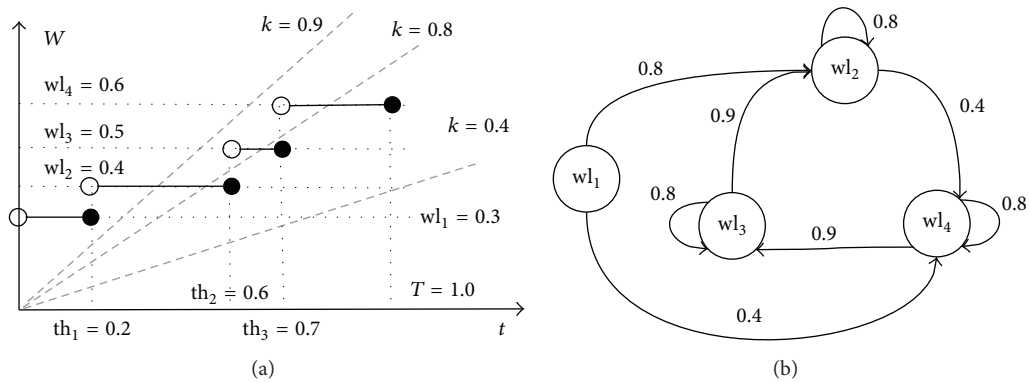

FIGURE 7: (a) A synthetic example, $W(t)$ with four workload levels, $\text{wl}_1 = 0.3$, $\text{wl}_2 = 0.4$, $\text{wl}_3 = 0.5$, and $\text{wl}_4 = 0.6$, and (b) its WTG representation, where the edges are annotated with scaling factors.

an alternating operation mode implied in $\text{wl}_2 \rightarrow \text{wl}_4 \rightarrow \text{wl}_3 \rightarrow \text{wl}_2$ shows the least average power consumption in the discrete model. This is due to the fact that a computer system cannot simply operate on an ideal design point. In case that the theoretically optimal design point cannot be captured by a commodity hardware, the proposed technique is particularly useful. It can effectively explore design space and find out the best one in alternating modes.

In principle, a stable operation mode is the case that the staircase workload function $W(t)$ has a crossing point with $k \cdot t$. If this k is sufficiently small, it is likely to be a near-optimal operation mode. However, it does not always result in a near-optimal power consumption. Particularly, in case that only a limited number of DVFS modes are available in a microprocessor, this k which crosses the current workload level may not exist in K.

7. Conclusion

This paper formulates the delay-workload dependency in power optimization problem of embedded systems as a staircase function of the delay taken at the previous iteration. In applying it to the power optimization of DVFS-enabled electronic devices, a novel graph representation, called WTG, is proposed for exploring all possible workload/mode changes. Then, it is shown that the power optimization problem is equivalent to finding a cycle of the graph that has the minimum average power consumption. The effectiveness of the proposed operation policy is proven by the power simulations of synthetic and real-life examples. It has been observed that staying in a low speed scaling factor in a stable operation mode is often the best discipline (self-loop in WTG). However, alternating modes, where the DVFS modes change over a predefined pattern, sometimes outperform the stable ones.

Disclosure

A preliminary version of this paper appeared in July 2015 at the International Symposium on Low Power Electronics and Design (ISLPED), under the title of "Modeling and Power Optimization of Cyber-Physical Systems with Energy-Workload Tradeoff" [11].

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by ICT R&D program of MSIP/IITP (B0101-15-0661, the research and development of the self-adaptive software framework for various IoT devices), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2013R1A2A2A01067907), and the new faculty research fund of Ajou University.

References

[1] E. A. Lee, "Cyber physical systems: design challenges," Tech. Rep. UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Calif, USA, 2008.

[2] J. Kleissl and Y. Agarwal, "Cyber-physical energy systems: focus on smart buildings," in *Proceedings of the 47th Design Automation Conference (DAC '10)*, pp. 749–754, ACM, Austin, Tex, USA, June 2010.

[3] J. S. Kim, D. H. Yeom, and Y. H. Joo, "Fast and robust algorithm of tracking multiple moving objects for intelligent video surveillance systems," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1165–1170, 2011.

[4] D. K. Park, H. S. Yoon, and C. Sun Won, "Fast object tracking in digital video," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 3, pp. 785–790, 2000.

[5] J. Pestana, J. L. Sanchez-Lopez, P. Campoy, and S. Saripalli, "Vision based GPS-denied object tracking and following for unmanned aerial vehicles," in *Proceedings of the 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR '13)*, pp. 1–6, IEEE, Linkoping, Sweden, October 2013.

[6] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of the ACM SIGMOD International Conference on Management of*

Data (SIGMOD '08), pp. 147–160, ACM, Vancouver, Canada, June 2008.

[7] J. Barbič and D. James, "Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '07)*, pp. 171–180, Eurographics Association, San Diego, Calif, USA, August 2007.

[8] J. Barbič and D. L. James, "Six-DoF haptic rendering of contact between geometrically complex reduced deformable models," *IEEE Transactions on Haptics*, vol. 1, no. 1, pp. 39–52, 2008.

[9] P. Padmanabhan and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS—Operating Systems Review*, vol. 35, no. 5, pp. 89–102, 2001.

[10] T. Mudge, "Power: a first class design constraint for future architectures," in *High Performance Computing—HiPC 2000*, pp. 215–224, Springer, 2000.

[11] H. Yang and S. Ha, "Modeling and power optimization of cyber-physical systems with energy-workload tradeoff," in *Proceedings of the 20th IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED '15)*, pp. 315–320, Rome, Italy, July 2015.

[12] H.-C. An, H. Yang, and S. Ha, "A formal approach to power optimization in cpss with delay-workload dependence awareness," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 750–763, 2016.

[13] P. Bogdan and R. Marculescu, "Cyberphysical systems: workload modeling and design optimization," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 78–87, 2011.

[14] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Proceedings of the Real-Time Systems Symposium (RTSS '08)*, pp. 47–56, Barcelona, Spain, December 2008.

[15] Y. V. Pant, H. Abbas, K. Mohta, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of anytime computation and robust control," in *Proceedings of the 2015 IEEE Real-Time Systems Symposium (RTSS '15)*, pp. 43–52, IEEE, San Antonio, Tex, USA, December 2015.

[16] D. Goswami, R. Schneider, and S. Chakraborty, "Co-design of cyber-physical systems via controllers with exible delay constraints," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pp. 225–230, IEEE Press, Yokohama, Japan, January 2001.

[17] A. Schrijver, "Combinatorial optimization: polyhedra and efficiency," *Discrete Applied Mathematics*, vol. 146, pp. 120–122, 2005.

[18] R. Tarjan, "Enumeration of the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 2, no. 3, pp. 211–216, 1973.

[19] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," vol. 5, pp. 1–10, 2001.

[20] G. Bradski, "The opencv library," *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120–126, 2000.

[21] Samsung Exynos, Octa 5422, 2015, http://www.samsung.com.